Knowledge-Based Systems Manufacturing With Viability Ensuring

Valeriya V. Gribova

Institute of Automation and Control Processes

Elena A. Shalfeeva (✉ shalf@dvo.ru)

Institute of Automation and Control Processes <https://orcid.org/0000-0001-5536-2875>

Research Article

Keywords: knowledge-based system, interpretable knowledge base, viability, ontology-oriented manufacturing, cloud-platform

Posted Date: August 23rd, 2021

DOI: <https://doi.org/10.21203/rs.3.rs-814383/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Abstract

With highly increased competition, intelligent product manufacturing based on interpretable knowledge bases has been recognized as an effective method for building applications of explainable Artificial Intelligence that is the hottest topic in the field of Artificial Intelligence. The success of product family directly depends on how effective the viability mechanisms are laid down in its design. In this paper, a systematic cloud-based set of tool family is proposed to develop viable knowledge-based systems. For productive participation of domain and cognitive specialists in manufacturing, the knowledge base should be declarative, testable and integratable with other architectural components. Mechanisms to ensure KBS viability are provided in an ontology-oriented development environment, where each component is formed in terms of domain ontology by using the adaptable instrumental support.

Due to the explicit separation of ontology from knowledge, it became possible to divide competencies between specialists creating an ontology and specialists creating a knowledge base. We rely on the fact that the activity of creating an ontology is significantly different from the activity of creating a knowledge base. Creating an ontology is a creative process that requires a systematic analysis of the domain area in order to identify common patterns among its knowledge.

The characteristic properties of knowledge-based systems related to viability are described. It is explained, how these properties are provided in development environments implemented on cloud platform. The concept of a specialized manufacturing environment for knowledge-based system is introduced. The necessary set of tools for such ontology-oriented environment construction is determined.

The example of tools for creating specialized manufacturing environments is the instruments implemented on the «IACPaaS» platform. The IACPaaS is already used for collective development of thematic cloud knowledge portals with viable knowledge-based systems. This specialized manufacturing environment has enabled the creation of multi-purpose medical software services to support specialist solutions based on knowledge being remotely improved by experts.

1. Introduction

The developer of any Software System (SS) has the task of create a software product, which meets all current requirements, but in addition he must also providently designate and put the mechanisms into it, which will provide further maintenance to meet the changeable user requirements to the interface, functionality, subject knowledge and service conditions.

According to the experts' estimations for the maintenance process can take a significant share, from 50 % up to 80%, in software system life cycle, which is quite significant amount (Selby 2007; Burki and Vogt 2014; Koskinen 2015; Marounek 2012; Nguyen 2010). To try to reduce this figure the maintenance process should be simplified by the special mechanisms used at the SS architectural design stage. These mechanisms simultaneously help to solve maintainability problems and problems encountered in the

implementation of «continuous delivery» (a software is always kept releasable). Common problems relate to build design (Betz and Walker 2014; Laukkanen et al. 2017), system design (Claps et al. 2015; Bellomo et al. 2014; Chen 2015) and integration (Laukkanen et al. 2017; Claps et al. 2015) stages.

Some of these known mechanisms (solutions to the problems) are (Pressman 2010; Dehaghani and Hajrahimi 2013; Dodonov and Landje 2011; Chernikov 2012):

- Architectural solutions (these are separated weakly coupled components with logical understanding and functions);
- Declarative representation of SS components (e.g. by the structural and graphical editors), and “4GL techniques” using, which shift the accent “from the programming to design”;
- Automation tools for architectural design models construction and code fragments automatic generation (DBMS circuits generation, program code generation);
- The tools to link the components to each other (for example, classes, which use each other, or stored database tables with rendered query results);
- Standard architectural solutions application, which are “imposed” by so-called shells (from DBMS of MS Access to 1C platform);
- Competences separation between specific area developers such as interface designers, architects, database designers, network interaction specialists, Big Data specialists, database fillers, programmers and others.

Knowledge-Based Systems (KBS) represent the special class of systems with the additional architectural component - knowledge base, which requires specialized tools for their creation and maintenance by the domain experts and cognitive specialists (Musen 2015; Mortensen et al. 2015; Kryazhych 2015). This specificity requires the additional specialized mechanisms to ensure KBS maintenance and viability.

The methods and technologies proposed in (Fikes and McGuinness 2000; Musen 2015; Mortensen et al. 2015; Rybina et al. 2011; Rybina 2008) make it possible to create knowledge bases and knowledge-based systems, test them and put them into operation. However, not enough attention is paid to the methods and tools for improvement of their components, especially in process of their operation. An approach with a universal representation of knowledge in the form of rules processed by a single "output machine" is suitable for areas where the number of rules is measured in the hundreds. Further, their creation and development, especially by experts, becomes almost impossible. The use of ontologies made it possible to abstract some "descriptive" part of knowledge and begin to apply deductive inference, using the concepts and concepts introduced in ontologies in the premises and conclusions of the production rules, that is, to reason in terms of ontology ("ontological reasoning") (Gavrilova et al. 2020). New environments that combine ontologies and logic programming (for example, "ontological logic programming") are more effective for implementation, but in terms of subsequent maintenance, they have made only a small step

- a declarative representation of some part of knowledge (where an properties for entities, binary relationships and class hierarchies of such entities can be defined). So, despite significant progress in the field of creating of such class of tools, it should be noted that the methods and tools for creating and, above all, KBS maintenance, are not given sufficient attention. According to Gartner, Inc., AI Engineering is included in the top of strategic technology trends in 2021. Gartner research shows only 53% of projects make it from artificial intelligence (AI) prototypes to production. "CIOs and IT leaders find it hard to scale AI projects because they lack the tools to create and manage a production-grade AI pipeline" (Gartner 2020).

Thus, despite significant advances in software engineering, known methods aimed at ensuring the viability of software, the proposed solutions for KBS do not provide the necessary level of viability.

The goal of this article is to represent the generalized viable KBS architectural model and the tool set for its evolving (viability) ensuring.

The Chap. 2 and 3 discuss a concept and requirements for viable software; the Chap. 4 analyzes why traditional maintenance methods are not enough for KBS. The Chap. 5 presents a model of a viable KBS and a method for building it in accordance with the concept and requirements presented in Chaps. 2 and 3. The 6 and 7 Chaps. 6 demonstrates implementation a set of tools using IACPaaS cloud platform, their use for building viable KBS, and gives the comparative analysis of labor costs.

2. Software System Maintenance And Viability Concepts

With complex software systems advent, the development of which has begun to require the efforts measured by man-years, the problem of their maintenance has now become to be more essential.

Traditionally, SS maintenance means the possibility of its correction, restructuring (refactoring), adaptation (to equipment and system software, to new types of human-machine interfaces and users), and expansion (new functions' adding on the users' requests).

But the following decades have identified SS new useful properties such as the viability as sustainability to changeable environment (maintaining of usefulness and operability) (Izurieta and Bieman 2013), and ability to evolve as ability to adapt with the least possible cost to requirements' variability, maintaining the architectural integrity (Breivold et al. 2008).

We'll specify "the viability" as SS resilience to some functioning environment changes (the maintaining of working capacity), and the ability to develop over the "life" (evolvability), which are provided through the adaptability of environment changes (the automatic change), and through the adaptivity to new requirements (the interactive change).

The adaptability is implemented through:

- The declarative components representation, providing, from one hand, the components' software-processed representation, and, from another hand, the representation, corresponding to visual and understandable developer's terms' concepts system and language communication;
- Availability of structural and graphical editors to support all SS components' construction;
- Components' construction automation (DB schema generation and program code generation), with their architectural integrity checking.

The adaptivity is provided by the following mechanisms:

- The Machine learning, which improves the data processing algorithms by adding input data sets;
- User and environment feedback, which provides the interface configuration and SS interaction scenario.

Sometimes, thus, viability extends the traditional maintenance concept, first of all by the adaptive mechanisms presence.

3. Different Classes Of Requirements For Development Of Software

As mentioned above, SS life cycle maintenance average costs are about 50% (Braude and Bernstein 2010; Glass and Noiseux 1981), but according to some reports they could reach even 80%-90% (Islam and Katiyar 2014). Although it is known that some types of SS (such as firmware), do not require the maintenance (Pressman 2010).

SS modification is due to changes of operating conditions, user requirements, subject area etc., but whether it will be required or not depends on SS assignment, its uniqueness or from type of information being processed.

If consider the applied SS (including KBS), and also SS applications shells and tool software, the required SS modification could be estimated as follows.

Applied SS are created and purchased for professional activity support, customer service improving or for training purpose.

In the process of applied SS operating there the need in mistakes correction, new user-defined functions adding, adaptation to new instruments/devices/platforms, user interface changing and information format clarification.

The same thing is to the tools for applied SS development.

In case of the applied KBS related to traditional intelligent tasks such as diagnostics, planning, forecasting etc, the situation is different. Here it is rather expected the knowledge variability and new

solutions appearance such as new diagnostic methods creation and new influencing factors identification than just the user functions expanding.

The reason is that many known tasks statements in diagnostics, planning, etc. are quite stable, and new intellectual challenges are just the concretization or refinement of the existing ones.

The applied SS development or modifiability also depends on the development tools.

For software systems with the predetermined architecture, provided by frameworks and shells, the different components' maintaining and combining is markedly easier. It's because the problem of adaptability to devices, platforms, User Interface (UI), flexibility, and to the stored or new information format is "transferred" to the mentioned tools.

The frameworks, shells, tool environments themselves are subjects to adaptation and expansion within their lower level language environments. At the same time, the required tools components can be both "own" and borrowed or integrated from other tools. For example, the architectural essential elements such as the object model itself, the data model, the component model are designed at "Rational Rose" tools environment, but the user interface is designed at the "Delphi" environment with conversion to "Rational Rose" Model program code.

The approach to applied software systems maintenance is similar to development tools maintenance, i.e. tool systems supporting.

4. The Features Of Knowledge-based Software Systems

KBS is a class of software (one of the classes of AI systems), the specific features of which require specialized development and maintenance tools, which makes traditional maintenance methods ineffective. Unlike other types of systems (including AI systems), KBS explicitly distinguishes between information-data and information-knowledge.

- Data is the operational input information for a task that describes the objects, processes, and phenomena of the domain, as well as their properties (can be stored in databases/facts).
- Knowledge is the result of a person's theoretical and practical activities, reflecting the accumulation of previous experience. Knowledge is the regularities of the domain (principles, relations and regulations) necessary for solving problems based on new data (facts).
- If the knowledge is isolated and formed into an independent architectural components - knowledge base, then the SS that uses it becomes KBS. Knowledge should be formed and maintained by domain experts (possibly with knowledge engineers and cognitive scientist), who become, in addition to programmers, designers, and other traditional types of software developers, full participants in the development and maintenance process. This requires them to be represented in the form understandable to domain experts.

In explainable Artificial Intelligence (AI), an explanation of solution should be understandable and sufficiently detailed, and formalized knowledge should be interpretable.

The specification of a task based on knowledge bases is the predicate $P(x, k, y)$, where $x \in X$, $k \in K(X, Y)$, $y \in Y$. A feature of the specification of the problem is that the statement about the existence of solutions is not valid (Gribova and Kleshchev 2014). It is assumed that there is a "correct" knowledge base $k^* \in K(X, Y)$ such that $\forall x \in X \exists y \in Y P(x, k^*, y)$; however, this assumption cannot be proved. In the general case, this "correct" knowledge base is unknown, and for any other knowledge base the statement about the existence of a solution does not have to be true (Gribova and Kleshchev 2014). But assuming the validity of the "weak" statement (about the existence of the "correct" knowledge base), it is possible to develop an algorithm for solving the problem (implementation in the algorithmic language of a partially defined functional mapping $A: \langle X, K(X, Y) \rangle \rightarrow Y$) (Gribova and Kleshchev 2014).

Such problems in modern KBSs are solved "using heuristics, including empirical induction, analogy and deduction" (Finn 2014), often using more than one method of imitating human intellectual activity (Gavrilova et al. 2020). Modern KBSs offer a convenient user interface, they can receive data from measurement instrumentation and are often integrated with statistical and other software components that do not use formalized knowledge (including standard software packages) (Rybina et al. 2011; Rybina 2008).

The viability for a knowledge-based system is the providing its efficiency and usefulness in conditions when the domain (more often - knowledge, less often - ontology, functioning environment and user interface) are changed.

Since the viability of KBSs is manifested, first of all, in the conditions of variability of domain knowledge, and "the ability to adapt in accordance with a change in the set of facts and knowledge" is considered one of the aspects of intelligence (Finn 2014), for most domains associated with solving intellectual problems, it implies the evolution of knowledge. Moreover, in domains where the influence of factors and events on the state of the system (object, situation), their change in time, the influence of individual characteristics of the system and some of its processes on others is important, the development / evolution of knowledge bases is the main "challenge" of modern "conditions" in relation to KBSs. Continuous improvement of the knowledge base allows us to hope for a "reference" knowledge base $k^* \in K(X, Y)$. Its relevance (compliance with current knowledge) and quality will determine the success of the use of KBS (to obtain an explanation of $y \in Y$ for any $x \in X$).

5. The Viability Model Of Declarative Knowledge Systems

The main task required for the solution is how to ensure that the knowledge base corresponds to the current knowledge in this domain and how to ensure their continuous improvement.

The relevance of knowledge (knowledge base) is achieved in three main ways (Dodonov and Landje 2011; Kryazhych 2015; Gribova and Kleshchev 2016): adaptability (interactive change of the knowledge

base), adaptability - using KBSs machine learning methods (means of inductively generating knowledge from selected precedents, means of knowledge discovery from “big data”, or a combination thereof).

The “success” of adaptability (interactive change) depends on the domain ontology and the user interface of the knowledge base editor, which must meet the requirements and expectations of domain experts. At the same time, the ontology is a structural basis of both tools for experts (editing tools) and software components of KBSs (solvers of intellectual problems and user interfaces).

Thus, the traditional architecture of KBSs - “knowledge base (KB) + fact base + intellectual problem solver + intellectual GUI” (Finn 2014) - expands with a new component: “ontology + knowledge base + fact base + ontology-oriented problem solver + intellectual GUI”.

5.1 Architectural properties of systems based on declarative knowledge

In the domain ontology, the types of statements (allowing solving domain problems) and restrictions on the interpretation of the meaning of concepts (terms) are defined. For example, for the diagnosis of processes, one of the most common types of sentences (statements) is $\langle \text{deviation class}_k, \text{characteristic}_j, \text{range of values}_{kj} \text{ of characteristic}_j \rangle$; sentences like “a necessary condition for the existence of the process”, “a variant of the process of changing the values of the attribute”, etc.).

The knowledge base contains domain knowledge that is presented explicitly. They are generated manually or inductively, including from training samples from archives and databases (inductive generalization in machine learning (Finn 2014), Bayesian classifiers, clustering algorithms, and reinforced learning (Nikolenko and Tulupiev 2009; Golenkov et al. 2018)).

The base of facts contains a set of facts (statements) observed or objectively measured in the situation under consideration, regarding which the problem is being solved.

The explicit representation (and grouping) in the ontology of all structural types of statements (and the separation of the ontology from the knowledge base) necessitates the “replacement” of decision-makers, interpreting production rules, with specialized ontology-based algorithms.

An ontological-oriented algorithm (ontological reasoner) searches for or refutes hypotheses bypassing the (declarative) knowledge base. Such an ontology-oriented problem solver “sorting through” the KB statements of each type related to the hypothesis. He compares these sentences of input information (about the object). The algorithm for bypassing the knowledge base and comparing its statements with the elements of the structure of domain objects is generally simple, because the number of “matching rules” is determined by the types of KB statements corresponding to the problem being solved.

The implementation of the bypassing the declarative knowledge base mainly consists in searching for the values of input data in the knowledge base. (Example: the implementation of the bypassing the

declarative Diagnostic Knowledge Base usually consists in searching for the values of the signs (symptoms) of the diagnosed object and comparing them with the areas of (expected) values in order to reject or confirm the corresponding hypotheses about the classes of deviations). Thus, the “structural” complexity of an ontology-oriented algorithm is usually small. Its complexity is determined by:

- The number and complexity of the types of axioms (statements) of the ontology,
- The length of the chains of cause-effect relations between the hypotheses and observations – the elements of the description of the domain objects,
- The number of observations (description elements) of the objects and/or the number of restrictions / target conditions, for example, in the designing problem, the conditions will relate to the sizes and different properties of the constructed object (bridge, hardware complex, ...); in the treatment task restriction is “the object must become healthy”). The processing time (calculation) depends on the size of the knowledge base.

Such solvers can become reusable, because, firstly, a set of traditionally solved problems (diagnostics, forecasting, control, designing, planning ...) is known (Clancey 1985; Kleschev and Shalfeeva 2015), and secondly, a traditional set of types of relationships in domains with cause-effect relations (for solving the problems of diagnostics, forecasting, and object state control) (Clancey 1985; Kleschev and Shalfeeva 2015) is limited. Reuse consists in choosing a task in accordance with the classification of tasks and their correction in accordance with the domain characteristics (additional types of relations in the ontology), finding a problem solver and, possibly, adapting it to the format of the result of the work (explanation).

Using an ontology to form a knowledge base, data (facts), as well as an explanation structure makes it possible to generate a user interface (GUI) in terms (concepts) that are understandable to a specialist, and also, as a rule, generate a dialogue scenario corresponding to the structure of explaining the results.

KBSs as a variety of software can also:

- Include databases for storing reference or other information,
- Accept files or databases with operational information (about the situation regarding which a decision is required),
- Create as a result files or databases with the results of the work and an explanation of solution.

Thus, viable system of explainable AI should include: knowledge base described in terms of ontology, ontology-based solver (it can to put forward and explain hypotheses), ontology-dependent adaptive and adaptable user interface. It is high-level architectural design model of KBS.

The most characteristic properties of KBSs related to viability are:

- Regular updating of knowledge base;
- The admissibility of the improvement of the decision-making method;
- The permissibility of changing or adding functions (for example, the formation of additional results);
- Adaptability of the user interface due to changes in the input data;
- The permissibility of expanding the ontology (adding concepts, relationships).

Changes in the ontology, as a rule, lead to the adjustment of all components of KBSs; in this case, it is advisable to talk about a new version of the KBS.

As a result, the structure of KBS and its components does not require changes due to the current maintenance and sustainable development (Chhabra 2017).

5.2 Ontological approach to support and evolving of KBSs

Ontology more often corresponds to one class of problems being solved or to one class of problems in one subject area. To design applied KBSs, not one ontology may be required, but their set of ones for interconnected problems. The examples of related problems are diagnosing and fix-planning or predicting.

A toolkit for constructing application systems with declarative knowledge is based on the ontology of the subject area, since algorithms process information from subject area. An formalized information being processed in their algorithms is a laws of the subject area, a real facts, an archived data and documents stored.

Knowledge and other information should be formed in terms of unified terminological concepts of subject area and in accordance with the structure determined by ontology of domain - ontology of knowledge and of reality's data. Ontological agreements on the rules of matching facts to knowledge should be known to developers of algorithms for solving problems.

Let's call the toolkit that allows create KBS components on base of domain ontology the *specialized development environment* for KBSs.

A specialized environment for manufacturing and development of knowledge-based systems should at least provide:

- KB Editor,
- Database Editor,
- Editor of solution results' explanations structure,
- A library of reusable software solvers corresponding to the classes of tasks to be solved

- A library of software units (the components of software solvers),
- Search and selection tools for KBS components - KBs and software solvers,
- Tools for integration of knowledge bases and solvers,
- Specialized UI editor.

Note

In a situation where the possibility of making changes to domain ontology is not ruled out even after building knowledge bases, manufacturing environment of KBSs should provide an ontology editing tool. The introduction of additional concepts or new relationships between concepts in the ontology does not violate operability of AI software solvers. The ontology-oriented solver has this property because it processes only concrete types of sentences from Knowledge Base.

To form the solvers you will need:

- Coding tools for new software units (SU) or their new versions,
- Tools \ means of Cataloging of SU for reuse,
- Tools of integrating of reusable units and new ones into new solvers or their new versions.

Note

The ability to create program units for the solver may turn out to be unnecessary if the ready-made solver is according with the statement of the problem, and “building up” additional functionality is not supposed.

The complete architecture framework of knowledge based system is < set of KBs, set of software solvers, set of UI- components, set of factBases, set of software units>.

Due to the importance of developing and evolving knowledge bases, only those KBSs that are integrated with knowledge base management system (KBMS) should be seriously considered. So the software components for evolving and checking of knowledge bases are desirable to be the part of the “integrated architecture”. Then the manufacturing environment has to provide:

- Tools of checking and assessing of knowledge bases,
- Tools of evaluating quality of knowledge bases by archives of solved problems,
- Tools of inductive formation of knowledge bases or fragments of knowledge bases.

KBS acquire the aforementioned vitality properties with these KBMS tools from specialized environment for its development.

The property 1 - support for updating knowledge.

The property 2 - support for changing configuration of an ontology-oriented solver (in connection with the replacement of a component that implements a different decision strategy or method of obtaining the result, or in connection with the addition of a component that implements an additional function, for example, generating an additional result).

The property 3 - Support for improving the UI in connection with a change in functions.

The property 4 - support for change in ontology.

The property 5 - support for improving UI of the expert (and UI of the user) in connection with updating the ontology.

The property 6 - support for coding new versions of software units of an ontology-oriented solver or support for solver code changes. The example of changes is adaptation of algorithm of processing Knowledge Base in connection with updating an ontology of knowledge.

5.3 An example of the implementation of KBSs manufacturing environments

The example of tools for creating environments for manufacturing of KBSs is the instruments implemented on the IACPaaS platform for processing ontological information resources (such resources are being generated in terms of an explicitly presented ontology or “meta-information”) (Gribova et al. 2018). The ontology defines the rules for forming information, and the limitations of its interpretation. The structure of the information generated and a number of interpretation restrictions are determined by the users of the IACPaaS platform for their own information resources formation. Usually a cognitologist makes this work for the community of experts, specialists, and users.

KBSs is a special case of applied software services on the IACPaaS platform (IACPaaS services) (Fig. 1).

This toolkit also contains tools for managing software components of KBSs (Fig. 2):

- “Master” of the formation of declarative parts of SUs (IACPaaS agents),
- Solver constructor from the root IACPaaS agent and processing agents (they are being represented only by declarative part),
- Generator of code workpieces (blanks) for new IACPaaS agents (or new versions),
- Byte code loader integrated with declarative part of agent,
- Tools for testing SU and preparing them for the library of SUs (agents).

The set of tools for formation of KBS' information components is as follows:

- IACPaaS ontology editor (Fig. 3a),
- IACPaaS-editor of knowledge base, generated in terms of ontology with a self-adaptive UI (when changing an ontology) (Fig. 3b),
- IACPaaS Data Base Editor (with self-adaptive UIs) (Fig. 3c).

This toolkit also contains tools for managing software components of KBSs:

- "Master" of the formation of declarative parts of SUs (IACPaaS agents),
- Solver constructor from the root IACPaaS agent and processing agents (they are being represented only by declarative part),
- Generator of code workpieces (blanks) for new IACPaaS agents (or new versions),
- Byte code loader integrated with declarative part of agent,
- Tools for testing SU and preparing them for the library of SUs (agents).

IACPaaS agents are encoded in Java. The Java Development Kit (JDK) for Java SE 8, 9, ... 13 is recommended for writing source codes and creating agent byte code. The environment for KBS manufacturing implies coding of new versions of agents (when it is necessary to process new terms or new relations of terms) or coding of new IACPaaS agents (when it is necessary to make a change in the decision-making process). JDK can be local to the programmer (or hosted on the IACPaaS platform).

Note

On the IACPaaS platform, the "master" of formation declarative parts of solver and SU are both the Editors generated in terms of corresponding meta-information. This meta-information is the part of ontology of the IACPaaS technology. A self-adaptive UI for forming data and knowledge and for viewing the results of KBSs provides three visualization views, one of which is graphical.

Thus, the manufacturing of KBSs using ontology-oriented environments fundamentally increases their viability by significantly increasing the role and share of controls (for making changes to declarative components) vs. coding and factoring tools (for making changes to the source code).

The KBS design technology in the proposed environment provides for a sequence of activities.

1. Create a problem ontology or search for ready-made ontologies of knowledge base about a particular task and data ontologies. If the IACPaaS platform does not yet have a portal for the problem under consideration, then the knowledge engineer uses the Ontology editor to create a resource "Knowledge ontology" — a description of a set of concepts, relationships, and restrictions used by specialists when

solving problems (and / or when transferring knowledge between specialists). It should be sufficient to provide all the necessary knowledge about the decision process. One of the most important works is the formation of the domain thesaurus (names of objects, their attributes, and possible values for each attribute or ranges of values). Often, an engineer, together with an expert, fixes a set of rules for making decisions, called "agreements".

2. Formation of the knowledge base by the domain experts. It is important to note that the IACPaaS platform has a Knowledge Base Editor Generator that automatically generates expert-oriented editors based on the knowledge ontology. As a rule, the creation of KB is a collective process..

3. Designers and programmers create new software components (if a portal with an already created solver is not found on the IACPaaS platform). To do this, the designer uses the Ontology editor to create an Ontology of explanation (the result of KBS's work must be an explanation). Next, the solver components (agents) are declared and implemented using software engineering methods.

Note

Creating the solver and knowledge bases are parallel processes. Some library may already contain a set of solvers or program units (agents) for the problem to be solved. Then creating a solver is not required. The main work is reduced to the formation of new knowledge. The ontological framework ensures the compatibility of components within the formed portals.

6. Ensuring Kbss Viability With Iacpaas Environment Tools

The above-described characteristic properties of KBSs related to viability are provided in development environments implemented on the IACPaaS platform. Consider this with the example of the Knowledge Updatability property and the support for updating knowledge.

If updating the knowledge base is required in connection with obtaining new knowledge (statements), it is natural to modify them manually (and evaluate the consistency with the available facts). For this process (in the development environment) Editors are required for all knowledge bases.

On the IACPaaS platform, the regular generator of Information Resource Editors is always operating, therefore knowledge base editors are always available in manufacturing environment. Currently a tool for checking the quality of the new version of the knowledge base vs. last (or current) one under development. It must check non-decrease of set of correctly solving tasks when replacing the version of the knowledge base (according to the importance of monotonous improvement of knowledge bases (Kleschev et al. 2013)). The procedure of checking of non-deterioration of knowledge is: to enter the input conditions of reference task to the solver, integrated with the new version of the knowledge base, to obtain the result (explanation) and to compare the obtained explanation with the output of this reference task.

If you need to update the knowledge base in connection with obtaining "facts" (precedents, decisions) that are not consistent with knowledge (when the precedent contains the correct result of solving the problem, which does not correspond to the result obtained using KBSs), then it is effective to form a new

version of the KB automatically (by inductive methods). The example of such an update of knowledge “from practice” is: after a certain period of time, the correct result of diagnosis or treatment “comes” (from a medical institution), it is compared with the result from KBS: are they contradictory. Then it is preferably to evaluate the consistency of work result of updated KBSs with existing precedents.

So, to implement the process of monotonous improvement of knowledge bases (based on the methods of inductive formation of knowledge base fragments (Kleschev 2013)), the following means are required:

- Tools of inductive knowledge formation for each solved intellectual problem (diagnostics, planning, forecasting ...);
- Tools of supporting the choice of precedents (correctly solved problems in one statement);
- Tools of verifying the correctness (quality) of the new version of the KB (the same as described above).

The checking of the “Knowledge Updatability” property is to check the availability and performance of the above tools. There is always a regular (self-adaptive) Editor on the IACPaaS platform; other tools are added to the development environment one after another.

We compared the process of building a complex of interconnected developing knowledge bases to build a medical decision support system for the diagnosis and treatment of diseases using the Protege (Musen 2015) and IACPaaS tools (Gribova et al. 2019). For this purpose, we have created classifiers of diseases, symptoms and medicines on the Protege platform. Next, using the Object Property mechanism, we needed to link diseases with symptoms (some acute diseases required a dynamic description of the clinical picture). However, these mechanisms did not provide an opportunity to describe knowledge in a form that is required and understandable to doctors, and the description of the dynamics of the development of diseases was particularly difficult. It should be noted that the Protege tools are incomprehensible and difficult for doctors, they are intended for knowledge engineers (although the description of the dynamics for them turned out to be difficult work). Similar difficulties were caused by attempts to describe treatment protocols taking into account the specifics of the use of medicines and the characteristics of patients. Protege's mechanisms did not allow the patient's history to be formed as a single document.

In the second case (using the IACPaaS platform), domain ontology was formed by domain experts and knowledge engineers with several types of relations:

< diagnosis_j, {symptom complex_{kj}}, [necessary condition_j] >;

< symptom complex_k, {feature_{kj}, range_{kj} of values of feature_j} >;

< symptom complex_k, {sign_{jk}, {period_{ik}, duration of period_i, range_{ijk} of values of sign_j in period_i}} > and so on.

In a network of such relations experts then began to create knowledge bases without the participation of knowledge engineers. To date, doctors have created and maintained knowledge bases on a wide range of nosologies (see Fig. 4). The total number of vertices in the knowledge bases is more than 100,000.

7. Using Iacpaas Environment Tools To Create Viable Kbs

The toolkit is already being used for collaborative development of cloud-based thematic Knowledge Portals and viable KBS. The domains are technical diagnostics of autonomous underwater vehicles (Inzartsev et al. 2016), agriculture (Romanov et al. 2015), proof of theorems (Kleschev et al. 2019), medicine (Gribova et al. 2019) and others.

The formation of a Medical Portal Med-IACPaaS (<https://iacpaas.dvo.ru/>) began with the development of medical ontologies and editing tools. The formation process included the development of the ontology of medical case records history (about 200 concepts - nodes of the semantic network). It intended for describing information about characteristics of an organism, facts, events, and being observed manifestations of diseases of the patients:

- a glossary of terms (more than 25,000) for description of the anamnesis of life, the current state of patients, their complaints, the results of objective, laboratory and instrumental research, etc. The glossary includes commonly used and specific terms, for example, the base of symptoms of cardiological pathologies, neurology terminology and so on;
- ontology of medical diagnostics (about 70 concepts and 100 relationships between concepts) as one of knowledge ontologies, and the diagnosis knowledge base, formed in its terms, currently including more than 150,000 concepts describing diagnosis of 25 diseases from six groups;
- ontology of knowledge about the nomenclature and effects of Medicines on the human body with various impaired functioning (about 80 concepts) and knowledge base formed in its terms, currently including more than 50,000 concepts;
- ontology of treatment regimens of diseases (about 80 concepts), and a knowledge base (for several nosology groups) formed on the basis the ontology;
- formats of structured reports with analysis and explaining hypotheses on decision on the base of knowledge.

Then development of knowledge bases and intelligent software components for viable intelligent medical software services began in parallel. Such services are intended to automate medical teams and institutions through the provision of "cloud" tools for support all the tasks of intellectual activity solved there.

Often, such "labour unit" has more than one activity profile (m profiles) and solves several intellectual problems (diagnostics, treatment, prognosis). The "cloud" implementation of KBSs makes it possible to use a single knowledge base for several classes of tasks and profiles. General knowledge bases

accumulate in addition to universal knowledge the experience of several professional communities and teams.

Knowledge bases were created on various nosologies: viral, diseases of oral cavity, salivary glands and jaws; diseases of the gallbladder, biliary tract and pancreas; bowel disease; hemorrhagic fevers; coronary heart disease; diseases characterized by increased blood pressure; chronic rheumatic heart disease. Specialists in mucopolysaccharidoses formed knowledge, tested them on real examples of patients from different countries, and proceeded to refine knowledge. Usually, experts form a clinical picture of diseases with dozens of dynamic symptoms (Fig. 5).

Ontology-based solvers are reusable components of KBSs. They were integrated with different knowledge bases of a wide or, on the contrary, narrow spectrum in order to provide a multitude of software services for supporting solutions for different categories of users. The standard service assembly assumes a standard GUI. Then a specialized, improved GUI is created, being customized to such set of terms from a glossary of terms to which specialists of their profile are limited.

The assembly of services from the knowledge base and the corresponding solvers was carried out and is carried out at workplaces depending on the needs of specialists. Users get access to the necessary services in their Personal office areas on a cloud platform. This allows them to accumulate archives of case records for each profile on the cloud server during operation (for next training, for verification). The "cloud" implementation of KBS allows use common solvers and common knowledge base control system.

The development and implementation process showed that the labor costs for automation of all user teams (for m activity profiles in each) are close to the costs for automation of one institution: the volume of work is performed in one "place" (instead of performing it at each workplace). For m activity profiles, $n * m$ knowledge bases (and their management systems) are required. There is no need for each institution to specifically engage in the improvement of knowledge (enough designate responsible persons).

Today specialists from various medical fields use the portal tools to develop formalized knowledge and its use. KB is large and maintained by a large number of specialists. To update current diagnostic knowledge, tools for search precedents, tools for editing knowledge bases, and tools for it checking are used. Update activities are carried out in accordance with procedures that allow users services themselves to remain operational. So, knowledge about some diseases of the digestive system was expanded, knowledge about region-specific manifestations of fever caused by rodents was clarified, and new unique knowledge about mucopolysaccharidoses was added. And neurology specialists formed the terminological base of symptoms for digitizing existing archives, for preparing them for the inductive generation of knowledge about neurology diagnosis, treatment, and prognosis of recovery.

A comparative analysis of labor costs (number of employees) to maintain the relevance of the KBS is carried out: in the case of cloud-based implementation of the KBS with the separation of competencies and in the traditional development of the KBS.

Let N conditional medical institutions working on P medical profiles with $M = 3$ tasks (and K development teams involved in automation, presumably working on different technologies). The diagram below (Fig. 6) shows the estimated number of specialists (vertically) involved in the development (maintenance of relevance and improvement) of the KBS in the traditional approach, the lower level shows the situation in the cloud implementation of the KBS with the separation of competencies. In horizontal direction, some principal stages and results of work performed by different categories of participants are presented. It is shown that in the new KBS cloud technology with the division of competencies, $P * 2$ specialists will be required to improve the diagnostic knowledge base (compared to $N * P$ - with the traditional approach). Similarly, improving the knowledge base about treatment ($P * 2$ pers.) Vs. ($N * P$); improving Diagnostic Solver With Interface – 2 People Vs. $K * P$ persons.

Another case (experiment) of checking the system (viability model) was in early 2020 a request to expand the service for the diagnosis and treatment of viral diseases with the possibility of differentiating diseases caused by COVID-19 among many respiratory diseases (Fig. 5).

In comparison with other service providers who presented updated versions a few months after the appearance of diagnostic guidelines (Infermedica, klinika.com.ua, medicase.pro), in our technology, the addition to existing knowledge base by description of several known variants of manifestation and course and diagnosing methods of new disease took several days.

For such KBS extension, Med-IACPAAS manufacturing environment tools provides: the medical expert-oriented knowledge base editor, the Med-CaseHistories base editor, focused on doctors, CaseHistory loader, the assessing tool. We assess correctness of the updated knowledge base on the basis of exemplary solutions and a declarative linker of service (new version) on base of ready-made IACPaaS solver.

So a week later, a new demanded cloud service was deployed to search for hypotheses about a patient's possible viral disease and differential diagnostics.

The Service is an example of explanatory AI. It provides a rationale for the proposed solutions and the recommendations issued (as opposed to klinika.com.ua, medicase.pro). The service informs which signs of the disease are / are not included in the clinical picture of the diseases in question, and also, whether additional information is needed to confirm or refute. At the same time the service prompts which values of which signs need to be obtained additionally.

The new cloud service and a declarative way to get it (based on the existing solver) demonstrate:

- feasibility of the method and approach to evolving of KBSs,
- the adequacy of the proposed infrastructure for manufacturing of KBSs and support their evolving.

Thus, the use of the proposed approach ensured the construction of multi-user medical software services to support the decisions made by specialists in different profiles based on knowledge remotely supported

by experts.

8. Conclusion

Through cloud-platform family of tools, the proposed *ontology-oriented manufacturing* method has been demonstrated to provide an effective means to produce knowledge-based products family and achieving scalability, viability and adequacy to professional level of users. The contribution of this paper is presented in five aspects.

1. The proposed approach implements explicit separation between main KBS components: knowledge base and problem solver with user interface, That is, a very important requirement of software engineering is implemented, which provides the possibility of independent (but consistent) creation and maintenance of each architectural component. So, in the proposed approach, domain knowledge is concentrated in the knowledge base (including various types of causal, spatial, and temporal knowledge), the solver does not contain domain knowledge. Recall that this requirement could not be implemented either in the production model or in object-oriented ontological models of knowledge. In them, some of the knowledge is included in the problem solver, which means that when you change the knowledge, you need to change both the knowledge base and the solver.

2. Explicit separation between the ontology and the knowledge base. In other systems based on the ontological approach, the issue of a clear division between ontology and knowledge has not yet been resolved, as a result, the division of competencies between specialists who create an ontology and specialists who create a knowledge base is quite difficult to implement. We rely on the fact that the activity of creating an ontology is significantly different from the activity of creating a knowledge base. Creating an ontology is a creative process that requires a lot of analytical work, a systematic analysis of the domain area in order to identify common patterns in the formation of knowledge, its structure, and integrity constraints. This activity should be carried out by cognitive scientists (knowledge engineers) together with experts. When creating an ontology, it is important to create it in such a way that it "covers" a class of tasks (for example, an ontology of knowledge on the diagnosis of diseases regardless of their etiology [42], an ontology of knowledge on the modes of laser additive manufacturing [43]. The ontology is generally not changeable throughout the KBS lifecycle. The knowledge base is an ever-changing component. It should be formed and modified by experts of the domain independently (without cognitive scientists and knowledge engineers) based on the created ontology (it is mandatory to have knowledge editors controlled by the ontology).

3. Tools for estimating the created knowledge base. We believe that this is a very important requirement of software engineering. We automatically evaluate the formal completeness of the knowledge base, a number of integrity and correctness constraints (based on the ontology). In addition, an important requirement is the availability of tools for estimating the knowledge base at any change using the accumulated cases (precedents). This ensures a monotonous improvement of the knowledge base, we can always argue on which base of cases the created knowledge base works correctly.

4. Generating detailed explanations is a very important element of KBS. In our approach, the explanation is generated based on the explanation ontology, which is built using the knowledge base ontology (in some cases, it partially repeats its structure). Having explanations is also useful when debugging the knowledge base.

5. The ontological solver, together with the user interface, is repeatedly used for a variety of systems with knowledge bases. It does not depend on the content of knowledge bases (it uses a knowledge base as a parameter), so their completeness and quality can be improved infinitely by integrating with the solver into new and different versions of useful systems.

Unlike most existing methods of knowledge-based product constructing, that assume either presence of a sufficient set of data to extract the necessary knowledge from them, or the presence of an expert who is able to formulate a sufficient set of knowledge, our research attempts to address the problem across the compatibility and complementarity of these paths, rather than interchangeability, including providing an explanation of the generated solutions.

The method was tested for sufficiently computational efficiency for a large-scale knowledge-based products family design problem.

The presented cost model can provide insight on what can be achieved from building cloud systems from shared or reused ontological solvers and being evolved bases.

The current research work focused on an emerging concept in KBSs Manufacturing and their controlled improvement.

The goal of this study was to provide a manufacturing environment for producing viable systems.

The proposed methodology proved to be easy to learn and convenient for teamwork.

For a medical diagnostic system, each significant expansion of knowledge (more than 20 such acts were performed in total) required from 5 hours to 2 working days for an expert, 2–3 hours for quality control, 10 minutes for architector, and without programmer, which in another production environment would be unattainable.

The results from the analysis of product characteristics after each update showed that the results were consistent with case-samples received from real practice.

The flexibility of the proposed methodology and tool can also be a benefit in cases where a quick reconfiguration of the production is required to adapt to new circumstances, as was required to recognize COVID-19 in patients. A limitation of the current approach is that the solver is explanation content-specific (depends on the specific composition of the generated explanation): if the degree of detail changes, the software components that record the explanation must be modified.

Further research steps should focus mainly on the integration of the developed tools with textual facts and knowledge parsers and with third-party diagnostic and predictive tools.

A detailed study is required to demonstrate whether the components working with structured information, with verbal text, with images and digital arrays, can be combined into a single complex.

This approach would save valuable time users from critical areas of activity.

Work is currently underway to further expand the capabilities of the approach. Today, the «bottleneck» for us is an adaptable user interface. The technology allows you to generate three types of user interfaces based on the explanation ontology, but these features are not enough. We are currently working on creating tools for automatically generating an interface based on the user model, taking into account the requirements of usability.

Declarations

Funding

This work was supported in part by the Russian Foundation for Basic Research (projects nos. 20-07-670 and 19-07-00244).

Conflicts of interest

We wish to confirm that there are no known conflicts of interest associated with this publication.

Availability of data and material

'Not applicable' for that section.

Code availability

'Not applicable' for that section.

References

1. Bellomo, S., Ernst, N., Nord, R., Kazman, R. (2014). Toward design decisions to enable deployability: Empirical study of three projects reaching for the continuous delivery holy grail. *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE. <https://doi.org/10.1109/DSN.2014.104>.
2. Betz, R.M., Walker, R.C. (2014). Streamlining Development of a Multimillion-Line Computational Chemistry Code. *Computing in Science & Engineering*, 16(3), pp. 10-17.
3. Braude, E.J., Bernstein, M.E. (2010). *Software Engineering: Modern approaches (2nd edition)*, Hoboken, NJ: John Wiley & Sons, Inc., 800 p.

4. Breivold, H.P., Crnkovic, I., Eriksson, P.J. (2008). Analyzing software evolvability. *COMPSAC 2008: 32nd Annual IEEE International Computer Software and Applications Conference*. Turku, Finland, 327–330.
5. Burki, C.J., Vogt H.H. (2014). How to save on software maintenance costs An Omnnext white paper on software quality November 2014. <http://asq.org/public/wqm/how-to-save-on-software-maintenance-costs.pdf>.
6. Chen, Lianping. (2015). Continuous delivery: Huge benefits, but challenges too. *IEEE Software* 32.2, 50-54.
7. Chernikov, B.V. (2012). *Quality management of software*. Moscow, Russia: ID Forum, 240 p. (In Russ.).
8. Chhabra, J. K. (2017). Improving modular structure of software system using structural and lexical dependency. *Information and Software Technology*, 82, 96-120. <https://doi.org/10.1016/j.infsof.2016.09.011>
9. Clancey, W.J. (1985). Heuristic Classification. *Artificial Intelligence*, 27, 289–350.
10. Claps, G.G., Svensson, R.B., Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software Technology*, 57, 21-31.
11. Dehaghani, S.M.H., Hajrahimi, N. (2013). Which Factors Affect Software Projects Maintenance Cost More? *Acta Informatica Medica*, 21(1), 63-66. DOI: 10.5455/aim.2012.21.63-66.
12. Dodonov, A.G., Landje, D.V. (2011). *Viability of information systems*. Kiev, Ukraine: Naukova dumka, 256 p. (In Russ.).
13. Fikes, R., McGuinness, D. (2000). Creating, Maintaining, and Integrating Understandable Knowledge Bases; *DARPA RKF Program Kickoff Meeting; New Orleans, Louisiana; June 7, 2000*.
14. Finn, V.K. (2014). About data mining. *Artificial Intelligence News*, 3, 3–18. (In Russ.).
15. *Gartner Identifies the Top Strategic Technology Trends for 2021*. (2020). STAMFORD, Conn., October 19, 2020. <https://www.gartner.com/en/newsroom/press-releases/2020-10-19-gartner-identifies-the-top-strategic-technology-trends-for-2021>.
16. Gavrilova, T.A., Kudrjavcev, D.V., Muromcev, D.I. (2020). *Knowledge engineering. Models and methods*. Saint-Petersburg, Russia: Publishing house—Lan', 324 p. (3rd edition). (In Russ.).
17. Glass, R.L., Noiseux, R. (1981). *Software Maintenance Guidebook*. Prentice Hall, 193 p.
18. Golenkov, V.V., Gulyakina, N.A., Grakova, N.V., Nikulenkа, V.Y., Ereemeev, A.P., Tarasov, V.B. (2018). From training intelligent systems to training their development means. *Open Semantic Technologies for Intelligent Systems*, 2, 81-98. ISSN: 2415-7740.
19. Gribova, V.V., Kleshchev, A.S. (2014). What has to be a paradigm of the solution of tasks on the basis of knowledge bases? *Open Semantic Technologies for Intelligent Systems*, 4, 131–136.
20. Gribova, V.V., Kleshchev, A.S. (2016). Paradigm for controlled intelligent systems. *Management systems and information technologies*, 3(65), 32–38. (In Russ.).

21. Gribova, V.V., Kleschev, A.S., Moskalenko, F.M., Timchenko, V.A., Shalfeeva, E.A. (2018). Extensible Toolkit for the Development of Viable Systems with Knowledge Bases. *Software Engineering*, 9(8), 339–348. <https://doi.org/10.17587/prin.9.339-348>. (In Russ.).
22. Gribova, V., Kulchin, Y., Nikitin, A., Timchenko, V. (2020). The Concept of Support for Laser-Based Additive Manufacturing on the Basis of Artificial Intelligence Methods. In: Kuznetsov S.O., Panov A.I., Yakovlev K.S. (eds) *Artificial Intelligence. RCAI 2020. Lecture Notes in Computer Science*, 12412, 403–415. Springer, Cham. https://doi.org/10.1007/978-3-030-59535-7_30.
23. Gribova, V., Moskalenko, Ph., Petryaeva, M., Okun, D. (2019). Cloud environment for development and use of software systems for clinical medicine and education. *Advances in Intelligent Systems Research*, 166, 225–229. <https://doi.org/10.2991/itids-19.2019.40>.
24. Gribova, V., Shalfeeva, E., Petryaeva, M. (2019a). Cloud Infrastructure for Creation of Interpretable Diagnostic Knowledge Bases of Diseases Regardless their Etiology. *Atlantis Highlights in Computer Sciences*, 3, 79–82. <https://doi.org/10.2991/csit-19.2019.13>.
25. Inzartsev, A., Pavin, A., Kleschev, A., Gribova, V., Eliseenko, G. (2016). Application of Artificial Intelligence Techniques for Fault Diagnostics of Autonomous Underwater Vehicles. *OCEANS 2016 MTS/IEEE Monterey, IEEE*. 1-6. ISBN 978-1-5090-1527-6. <https://doi.org/10.1109/OCEANS.2016.7761098>.
26. Islam, M., Katiyar, V. (2014). Development of a software maintenance cost estimation model: 4th GL perspective. *International Journal of Technical Research and Applications*, 2(6), 65–68.
27. Izurieta, C., Bieman, J.M. (2013). A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal*, 21(2), 289–323.
28. Kleschev, A.S., Chernyakhovskaya, M.Y., Shalfeeva, E.A. (2013). The paradigm of an intellectual professional activity automation. Part 1. The features of an intellectual professional activity. *Ontology of Designing*, 3(9), 53–69. (In Russ.). ISSN 2223-9537.
29. Kleschev, A.S., Shalfeeva, E.A. (2015). An ontology of intellectual activity tasks. *Ontology of Designing*, 5(2), 179–205. (In Russ.). ISSN 2313-1039.
30. Kleschev, A.S., Moskalenko, Ph.M., Timchenko, V.A. (2019). A Concept of Software Shell for Interactive Mathematical Proof Verification Systems. *CEUR Workshop Proceedings*, 2426, 153–160. ISSN 1613-0073.
31. Koskinen, J. (2015). Software Maintenance Costs. *Jyväskylä studies in computing*: University of Jyväskylä. Updated: April 30, 2015.
32. Kryazhych, O.O. (2015). Ensuring the viability of the information in time under its processing in the DSS. *Mathematical Machines and Systems*, 2, 170–176. (In Russ.).
33. Laukkanen, Eero, Itkonen, Juha, Lassenius, Casper (2017). Problems, causes and solutions when adopting continuous delivery—A systematic literature review. *Information and Software Technology*, 82, 55-79.
34. Marounek, P. (2012). Simplified approach to effort estimation in software maintenance. *Journal of Systems Integration*, 3, 51-63.

35. Mortensen, J., Minty, E., Januszyk, M., Sweeney, T., Rector, A., Noy, N., et al. (2015). Using the wisdom of the crowds to find critical errors in biomedical ontologies: A study of SNOMED CT. *Journal of the American Medical Informatics Association*, 22(3), 640–648. DOI: 10.1136/amiajnl-2014-002901.
36. Musen, M. (2015). The Protégé Project: A Look Back and a Look Forward. *AI Matters*, 1(4), 4–12.
37. Nguyen, Vu. (2010) Improved Size and Effort Estimation Models For Software Maintenance. *IEEE International Conference on Software Maintenance*. – IEEE, 1-2.
<https://doi.org/10.1109/ICSM.2010.5609554>.
38. Nikolenko, S.I., Tulupiev, A.L. (2009). *Self learning systems*. Moscow, Russia: MCNMO, 288 p. (In Russ.).
39. Pressman, R.S. (2010). *Architectural Design, Software Engineering: Practitioner's Approach*, 7th ed. New York, NY, USA: McGraw-Hill, pp. 242-275.
40. Romanov, V., Gribova, V., Galelyuka, I., Voronenko, O. (2015). Multilevel sensor networks for precision farming and environmental monitoring. *Information Technologies & Knowledge*, 9(1), 3–10. ISSN 1313-0455.
41. Rybina, G.V. (2008). The training integrated expert systems: some results and prospects. *Artificial intelligence and decision making*. 1, 22–46. (In Russ.).
42. Rybina, G.V., Blohin, Ju.M., Ivashhenko, M.G. (2011). Intelligent technology of integrated expert systems construction. *Artificial intelligence and decision making*. 3, 48–57. (In Russ.).
43. Selby, R.W. (2007). *Software engineering: Barry W. Boehm's lifetime contributions to software development, management, and research*. – John Wiley & Sons, vol. 69, 832 p.

Figures

Figure 1

Basic components of environment for development of basic components of KBS

Figure 2

The model of the system "KBSs manufacturing environment"

Figure 3

Some editing tools on the IACPaaS platform: a) IACPaaS ontology editor, b) IACPaaS-editor of knowledge base, c) IACPaaS Data Base Editor

Figure 4

The fragment of description of the disease in the diagnosis knowledge base

Figure 5

The fragment of the extended diagnosis knowledge base

Figure 6

Graphical representation of the comparison of efforts for different approaches to automation of activities