Knowledge presentation methods development in the intelligent business analytics systems based on ontologies and models

Victoria Vysotska[1,†], Yevhen Burov[1,*,†], Lyubomyr Chyrun[2,†], Sofia Chyrun[1,†], Oksana Brodyak[1,†], Valentyna Panasyuk[3,†], Dmytro Karpyn[4,†], Liubomyr Pohreliuk[1,†] and Nadiya Shykh[4,†]

[1] *Lviv Polytechnic National University, Stepan Bandera 12, 79013Lviv, Ukraine*
[2] *Ivan Franko National University of Lviv, University 1, 79000Lviv, Ukraine*
[3] *West Ukrainian National University, Lvivska 11, 46004Ternopil, Ukraine*
[4] *Ivan Franko Drohobych State Pedagogical University, Ivan Franko 24, 82100Drohobych, Ukraine*

Abstract

The use of knowledge presentation models to solve the management problems of complex ontologies is a promising direction in the development of ontology management systems and will increase the efficiency of the expert's work. We will demonstrate ways of using knowledge models to solve the problems of ontology management, in particular, creating, tracking the origin of elements, and validating the ontology. We will consider ways of presenting and using algorithmic models using the example of an intelligent system for automated testing of software products.

Keywords

Semantic, knowledge, ontology, business analytics, intelligent system, model, validation

1. Introduction

The central element of intelligent semantically oriented systems, which are developed within the scientific direction of the semantic web, is an ontology - a formal, declarative model of a defined subject area. An ontology is created by an expert - an ontology engineer.

CLW-2024: Computational Linguistics Workshop at 8th International Conference on Computational Linguistics and Intelligent Systems (CoLInS-2024), April 12–13, 2024, Lviv, Ukraine

* Corresponding author.
† These authors contributed equally.

victoria.a.vysotska@lpnu.ua (V. Vysotska); yevhen.v.burov@lpnu.ua (Y. Burov); Lyubomyr.Chyrun@lnu.edu.ua (L. Chyrun); sofiia.chyrun.sa.2022@lpnu.ua (S. Chyrun); brodyakoksana1976@gmail.com (O. Brodyak); v.panasiuk@edu.wunu.ua (V. Panasyuk); dmytro.karpyn@gmai.com (D. Karpyn); liubomyr@inoxoft.com (L. Pohreliuk); shykh.nadiya@gmail.com (N. Shykh)

0000-0001-6417-3689 (V. Vysotska); 0000-0001-8653-1520 (Y. Burov); 0000-0002-9448-1751 (L. Chyrun); 0000-0002-2829-0164 (S. Chyrun); 0000-0002-9886-3589 (O. Brodyak); 0000-0002-5133-6431 (V. Panasyuk); 0000-0002-0476-3406 (D. Karpyn); 0000-0003-1482-5532 (L. Pohreliuk); 0000-0003-0059-7137 (N. Shykh)

Due to the high complexity of subject area (SA), it is impossible to display all the concepts and dependencies of this area in the ontology. Therefore, as a rule, it is advisable to include in the ontology only concepts and dependencies that are necessary for presenting and solving problems that are planned to be solved in the subject area. This approach has a lot in common with well-known software design methodologies, such as RUP [1] or object-oriented using UML [2], because the first stage of program design is also the analysis and formalization of SA, the definition of program usage scenarios. At the same time, for a certain SA, as a rule, a significant number of various problems are solved. Each of these tasks uses part of the entities of the ontology. At the same time, the ontology generally plays the role of a single formal model, a set of concepts, and a common language for presenting all tasks and creates a foundation for a unified understanding of the concepts underlying the essences of the ontology by various tasks. All this leads to an increase in the number of ontology components and their connections. Known higher-level ontologies (general ontologies) reflect a significant number of concepts, for example, CYC contains 2 million, and Wordnet - has about 207 thousand entities [3, 4].

Complexity creates significant problems in solving ontology management problems. This class of tasks includes the creation, updating, modification, visualization and validation of the ontology, documenting the origin of its elements. It is known [4, 5] that a person can keep a relatively small number of objects in focus at the same time (4-7). Therefore, with the increase in the size of the ontology, it becomes increasingly difficult to solve the tasks of its management manually. One of the consequences of this is the contradiction between the number of entities (ontology width) and the amount of information provided for each element (ontology depth). It is not surprising that, for example, Wordnet is currently used mainly only as an extremely developed linguistic resource - a dictionary [3, 4]. Difficulties in the management of ontologies ultimately lead to a deterioration in the quality of the ontology. In [6], the quality of the ontological model is determined by fulfilling the requirements for its completeness, correctness, and stability. Errors by an expert during the processing of a complex ontology lead to the failure to take into account essential concepts and software relationships in the ontology, which in turn leads to the creation of an incomplete and incorrect ontology [4].

Research on the problem of management of complex ontologies is carried out in several directions. In particular, metrics and ways of measuring ontology complexity are being developed. In [7], by analogy with the definitions of the concept of software complexity, ontology complexity is defined as difficulties in performing such tasks as development, reuse, and modification of ontology. In this work, a multidimensional set of metrics was developed that reflects both the complexity of the ontology in general and its complexity at the level of classes and relations [4].

The work [8] proposed the O2 meta-ontology, which defines an ontology as a semiotic object. Using this ontology, three metrics of ontology complexity were developed: structural metrics, functional metrics, and usability profiling metrics. In addition, a significant number of potential metrics are analysed in this work. Some of these metrics are qualitative and cannot be automatically calculated. Metrics for a previously normalized ontology are proposed in [9]. Normalization of the ontology includes such steps as naming classes, facts, materialization of the hierarchy of inheritance, unification of names, and normalization of attributes. Such normalization aims to transform different ontologies into a semantically equivalent form to subsequently create semantic complexity metrics [4].

Ontology management functions are integrated into ontology development and conceptual modelling tools, such as Protégé [10] and TopBraid [11]. There are ontology management systems focused on industrial use [4, 12].

The development of ontology visualization tools is aimed at increasing the efficiency of the work of an ontology management expert. The work of many of these tools boils down to displaying and bringing into focus a certain part of the ontology with which the expert works. Existing data visualization tools (Information Visualization) allow you to apply visual metaphors to a defined set of data, analyse multidimensional data, time slices, etc. They use combinations of textual, tabular, diagrammatic, and graphical data display [13] Ontology tools have capabilities for visualizing ontologies and facts from an information base. For example, Protégé has an OntoViz application that allows you to display ontology entities and relations in the form of a graph, the commercial ontology modelling tool TopBraid can display not only the structure of ontology classes but also geoinformation data. At the same time, the existing means of visualization of ontologies, for example, do not allow to graphical display of complex relationships or to display of the ontology and facts in the context of problems that are solved using the ontology. An important task in ontology management is tracking the origin of ontology elements and facts. Completion of this task is necessary to validate and ensure the correctness of the ontology, because the subject area changes, and to maintain the correctness of the ontology, it is necessary to monitor the dependencies between the elements and facts of the ontology and the corresponding objects of the subject area. Today, four levels of origin are defined [14]: static (permanent data), dynamic (variable data), unclear (the origin of this data is by its very nature unclear, unclear), expert (expert assessment is required to obtain information about the origin). In [15], it is proposed to track the origin of facts by recording the history of their changes, as well as a more detailed description of the events that led to the changes. At the same time, the task of finding the elements of the ontology scheme, which depend on a certain fact of the software, remains largely unsolved [4].

The use of such knowledge presentation models to solve the management problems of complex ontologies is a promising direction in the development of ontology management systems and will increase the efficiency of the expert's work. We will demonstrate ways of using knowledge models to solve the problems of ontology management, in particular, creating, tracking the origin of elements, and validating the ontology.

2. Related works

2.1. Use of executable models of ontology management

2.1.1. Using models to create, modify and validate ontologies

One of the problems faced by the SA-defined ontology developer is the multivariate construction of the ontology [4]. Due to the complexity and vagueness of SA, it is generally not possible to represent all aspects of this domain in an ontology. The choice of concepts to be included in the ontology is influenced by both the developer's experience and his subjective ideas about the importance of certain concepts in SA. As a result, concepts that will never be used may be included in the ontology, and some important concepts, on the contrary, are not included. Moreover, mistakes made during the conceptualization of SA can significantly complicate the further development of the ontology. Under these conditions, it is important to define the decision-making criteria regarding the inclusion of SA concepts in the ontology. A similar problem in the field of software design is solved by constructing and analysing software use-cases. It is appropriate to use a similar approach to the creation of an ontology - to build an ontology in the process of analysing problems that are solved using the ontology. At the same time, the problem is formalized in the form of a model. In the model, the entities participating in the solution of the problem, their attributes, and all relations and restrictions necessary for solving the problem are displayed. We will consider any of its constituent parts to be a component of the Cm ontology. Ontology components form the $CmOn$ set:

$$CmOn = \{x|Type(x) \in \{Cl, SlCl, LnCl, RuCl, CsCl\}\}.$$

Thus, by analysing the ontology components included in the $ScMd$ problem model scheme, it is possible to determine those components that need to be added to the ontology. At the same time, the following components Cm' are added to the ontology, for which: $Cm' \notin On, \quad Cm' \in ScMd$ [4].

In [5] such requirements for the quality of the ontology as the requirement of completeness and correctness are given. In particular, the requirement of completeness is formulated as follows: "all aspects of SA relevant for the defined task code must be reflected in the ontology." An ontology is considered correct if the knowledge defined in it is correct for the defined SA and relevant to the functions performed by the IS using the ontology. By analogy with these definitions, we will consider a **complete** model if it contains all the entities, relations, restrictions and operations necessary to solve the given problem. We will consider the model that solves the task according to the specified efficiency criteria to be **correct**. Let the set of complete and correct models $M(Md)$ be defined for a given SA. Then the ontology built based on this set is complete if: $\forall Cm_i \in On\colon Cm_i \in Md_i \in M(Md)$. An ontology is correct if it is built based on correct models. Solving the task of tracing the origin of ontology components is important for maintaining the correctness of the ontology over time. A change in the facts of the subject area, on which the ontology is based, leads to the need for its modification. At the same time, a model containing a reference to a changed fact immediately points to all components of the ontology dependent on this fact and thus will simplify the process of its modification [4].

2.1.2. Adaptive ontologies

To be able to build metrics on ontologies, it is proposed to expand the ontology model by introducing two scalar values - the weight of the importance of knowledge (KB) concepts and the semantic connections between them [16]. These weights allow us to adapt the ontology of the knowledge base to the specifics of the subject field, and determine the elements embedded in its structure and the mechanisms of its optimization (more precisely, adaptation) with the help of self-learning during operation. Coefficients of the importance of concepts and connections must meet the following basic requirements [17, 18, 19]:

- reflect the semantic importance of SA concepts, in which this intellectual system will be applied;
- to be formed during KB filling and to be adjusted according to defined rules;
- ensure KB integrity control;
- meet the requirements of the metric when they are used to compare the semantic and characteristic proximity of concepts.

There is a task to formulate an appropriate set of rules for assigning important coefficients (informational importance) to concepts and statements in the KB model, which will ensure an assessment of the actual value of its information content and the current situations under investigation (for example, the inclusion of text documents in classes according to the UDC).

We will show the possibility of carrying out the formulated task by introducing some simplifications and assumptions. Let's present the knowledge base in the form of a weighted conceptual graph, the numerical semantic characteristics of vertices and edges which are determined according to certain rules. It is an oriented weighted multigraph with the following properties [18, 19]:

- each element (vertex) can have any number of links;
- each element can have a connection with any number of other elements;
- each connection (edge) in the model corresponds to a certain direction and coefficient of the importance of the connection of the corresponding statement, each concept (vertices) - coefficients of the importance of the concept.

The coefficient of importance of a concept (connection) is a numerical measure that characterizes the importance of a certain concept (connection) in a specific subject area and dynamically changes according to certain rules during system operation [18-21]. Our approach to the presentation of knowledge in the form of weighted conceptual graphs is that any possible generalization, that is, a complex, complex concept is always clearly articulated, named and appears as a separate concept in the knowledge base. Therefore, if some generalization has common properties or ways of functioning, they can be physically implemented through the properties and processing of events of the corresponding generalizing concept. So, let's expand the concept of ontology by introducing coefficients of importance of concepts and relations into its formal description. Then such an ontology is defined as $O = \langle C', R', F \rangle$, where $C' = \langle C, W \rangle$, $R' = \langle R, L \rangle$, where, in turn, W is the importance of concepts C, L is the importance of relations R. Generally speaking, W is the vector of the dimension of the number of different precedents if the ontology is used for an intelligent system of searching for a relevant precedent or the dimension of the number of tasks that are solved by the AI of activity planning. The ontology defined in this way will be called adaptive, i.e. one that adapts to SA due to modification of concepts and coefficients of importance of these concepts and connections between them [21]. The methods of determining the coefficients of the importance of concepts will be considered at the end of this section, and the values of the coefficients of the importance of relations and the development of weights for the entire ontology will be considered in detail in the third section when we will analyse the structure of the ontology and the types of relations between concepts. Here we only note that the change of these coefficients occurs by the modification of knowledge by the methods of intellectual data analysis or knowledge engineering, which are aimed at extracting knowledge. The goal of data mining technology is the production of new knowledge that the user can further apply to improve the results of their activities. The following methods of identifying and analysing knowledge can be distinguished: classification; regression; clustering; association analysis; forecasting of temporal sequences (series); aggregation (generalization); detection of deviations; processing of text documents; and dialogue with an expert. The first seven belong to the methods of intelligent data analysis, and the last two to the methods of knowledge engineering.

2.1.3. Concepts and properties of the knowledge base ontology graph

The hierarchical multi-link structure of the semantic network of the KB ontology frames of the intelligent system can be represented as a directed weighted multigraph. Since KB is a semantic network of frames, each vertex C of the graph of the network G contains some set of elements characterizing the object corresponding to this vertex. The edges of the graph that correspond to connections (statements in the KB itself) are defined by ordered pairs of vertices $\langle i, j \rangle$. A path is a sequence of arcs (oriented edges) such that the end of one arc is the beginning of another arc, and we will use it to find the distance between two graphs. A graph is called connected if, for any pair of vertices, there is a path between them. The connectivity of an ontology's semantic network graph is a property that means that all elements of the network are within the reach of the ISBA and can be involved when generating a response to an appeal to it.

We will describe the relationship between the structure of the ontology links and the reasoning implementation mechanisms. The model should contain reasoning mechanisms, which will act as attached frame procedures using established relationships (assertions) to produce the necessary decision. According to the object paradigm and the frame model of knowledge representation, the parent frame class contains attached procedures for setting specific values of its property slots and slots of new instances/subclasses during their generation. The term "contains" means the presence of corresponding instances of the class of attached procedures (event handlers) in the corresponding slots of the address frame. These attached procedures for the newly created class/object are generated by the attached procedures class in response to a signal from the parent class, returning the address of the generated procedure instances. Therefore, each instance of a class contains only a basic procedure for generating calls to other instances, all other procedures are placed externally as instances of the procedure class, and their addresses are placed in the slots of the instance that can call this procedure. The procedure responds to the call with the parameters known to it, processes them and returns the result, which can be, in particular, the address of a new class generated by this procedure or an instance of an existing class. Therefore, the connections in the semantic network of frames are implemented through the exchange of messages between their attached procedures. Our approach to the presentation of knowledge in the form of a weighted semantic network (conceptual graphs) is that any possible generalization, that is, a complex, complex concept is always clearly articulated, named and appears as a separate concept in the KB. Therefore, if some generalization has common properties or ways of functioning, they can be physically implemented through the properties and event handlers of the corresponding generalized concept, according to the principle of inheritance.

2.2. Presentation of knowledge in tasks of verification of ontological models

Models for solving problems are created by a specialist in the subject area, who must not only reflect in the model all the essences and relationships of the subject area that are essential for the given task, but also determine the limitations on the use of models, the necessary conditions regarding the availability of input data, and determine the range of problems for which the method is implemented by the model is relevant and display relevance conditions through properties of domain entities. Accuracy, non-contradiction and completeness of models is a key factor that determines the quality of decisions made using models. The development of knowledge models requires a high level of developer competence and is a difficult formalized task. On the other hand, an error in the model will lead to errors in solving all problems in which this model is used. Therefore, an important stage in the development of models is their verification, both initial, which is carried out by the author of the model, and additional, which is carried out by a commission of experts in the subject area. The introduction of redundancy, that is, the use of a group of experts for additional verification of the model serves as a means of increasing the reliability of the knowledge reflected in the model at the expense of some increase in the cost of development. A certain problem is that the experts themselves can make mistakes, and even the usual expert voting, that is, identifying the position of the majority of experts, also does not guarantee against error. The credibility of experts' decisions can be improved by introducing an additional stage in decision-making, which consists of comparing expert opinions to identify obvious errors and removing from the decision-making process expert opinions in which errors are found.

The proposed approach is based on the introduction of the phase of expert verification of the developed knowledge models. For model verification, a group of experts (reviewers) is organized, each of whom checks the model and makes a reasonable assessment, indicating the advantages and disadvantages of the model. We will assume that there is no conflict of interest among the experts. The reviewer can make mistakes during model verification. A reviewer who made a mistake may be removed from the expert group, with a further reduction in his rating, which is used when creating new expert groups. The task of the expert group is to reach a consensus on the model being evaluated.

Consensus is defined as obtaining an agreed estimate, which is based only on estimates that do not contain errors. It is worth noting that the task of obtaining consensus was solved within the framework of the approach to building fault-tolerant computing systems. Two approaches can be distinguished here [22]. The first approach is developed within the framework of building fault diagnosis models at the system level (system-level fault diagnosis [23]) and is based on the classic work [24]. The second approach is developed within the analysis of the possibility of achieving the so-called "Byzantine agreement" and comes from classical works [25]. Both approaches have a similar goal - to determine the correct result in conditions of failures due to the introduction of redundancy. The first approach is based on the detected defective components of the system due to the organization of mutual checks. At the same time, it is believed that each functional module of the system correctly determines the failure of the module it is testing, and each defective module gives an unpredictable result about the state of the module being tested. The obtained results of mutual checks (syndrome) are analysed by a global arbiter (global observer or centralized arbiter [22]). To determine the conditions for centralized decoding of the syndrome, the parameter t is introduced - the maximum number of defective modules [26]. The article shows that to decipher the syndrome, the condition $n \geq 2t+1$ must be fulfilled, where n is the number of intelligent system (IS) modules.

The second approach is based on decentralized decision-making about the correctness of calculation results. Here, communication and protocol redundancy are introduced, which makes it possible to achieve a "Byzantine agreement" under the condition $n \geq 3t+1$, where t is the number of system components that distort the results of calculations [27]. It should be noted that the assumption of the presence of a centralized deciphering of the syndrome is quite acceptable for the case of organizing the verification of the knowledge model of an intelligent information system since such a task can be integrated into the decision support model itself by consensus.

Let's consider the method of verification of knowledge models of an intelligent system, which is based on reaching a consensus of a group of experts and determining how to use this method as a separate knowledge model - a component of an intelligent system.

3. Models and methods

We define the ontology On as a set of symbols of entities $\bar{E}$ and relations between them $\bar{R}$: $On = \{\bar{E}, \bar{R}\}$. Each relation $R \in \bar{R}$ is defined on the set of roles $\{P_1, P_2, \ldots, P_n\}$: $R(P_1, P_2, \ldots, P_n)$ [28-30]. In the general case, an initialization function ${F_{in}}^k$ is defined for each role P_k, which defines a subset of entities whose elements are allowed to initialize the role: ${F_{in}}^k: P_k \rightarrow {E_{in}}^k \subseteq \bar{E}$. In the simplest case, when $\forall k: |{E_{in}}^k| = 1$ for each role, there is only one entity type that can be initialized. In this case, it is possible to replace the roles with the corresponding entities of the ontology in the notation of the relationship: $R(E_1, E_2, \ldots, E_n)$. Entities of the ontology form a hierarchical structure (taxonomy) using the inheritance relationship (ISA-relation) R_{isa}. The binary relation R_{isa} is defined on an ordered pair of *Descendant-Ancestor* roles: $R_{isa}(P_{ch}, P_{pr})$. The inheritance relation is transitive, so that if $R_{isa}(E_1, E_2)$ and $R_{isa}(E_2, E_3)$ then $R_{isa}(E_1, E_3)$ is valid. We define the function F_{pr} which for each entity E_j determines the ordered list of its ancestors $(E_1, E_2, \ldots, E_k)$, so that $R_{isa}(E_i, E_{i+1})$ and $R_{isa}(E_j, E_1)$ are fulfilled. We also define the function ${F_{pr}}^1(E_j)$ which returns the immediate ancestor of the entity E_j or the empty set $\emptyset$. An important type of relation is the *Has-Parts* relation: $R_{has}(P_{wh}, P_{pt})$, where P_{wh} is the role of the whole, and P_{pt} is the role of the parts of this whole. The subtype of this relation R'_{has} defines specific entities for the whole and sets of allowed entities for parts [28-30]: $R'_{has}(E'_{wh}, \{E^1_{pt}, E^2_{pt}, \ldots, E^n_{pt}\}')$.

For simplicity, when presenting such a relation, we will use the notation: $E_{w?} = \{E_{pt}^1, E_{pt}^2, \ldots, E_{pt}^n\}$ [28-30]. Let us define an algebraic system (set) of abstract data types $\bar{T}$, in which for each element of T there is a mutually unique correspondence with a certain entity of the ontology: $\forall i \exists^1 j: T_i \to E_j,\ \forall j \exists^1 i: E_j \to T_i$. The $TypeEn()$ function returns for each data type T_i from $\bar{T}$ the corresponding ontology entity E_j: $TypeEn(T_i) = E_j$ and thus determines the semantic interpretation of this data type. According to the approach from [31], the abstract data type is algebraically represented by a triple [28-30]:

$$T = (Name, \Sigma, Ex), \tag{1}$$

where *Name* is the name of the type, Σ is the signature of the multivariate algebra, *Ex* is the set of equations in the signature Σ that specifies the defining relations of the abstract data type. The signature is as Σ=(*S*, *OP*), where *S* is the set of names of basic sets, and *OP* is the set of names of operations [28-30].

$$S = \{S_1, S_2, \ldots, S_n\}, OP = \{F_1, F_2, \ldots, F_k\}.$$

Each operation F_i defines a mapping [28-30]:

$$F_i: S_{a(1,i)} \times \ldots \times S_{a(n,i)} \to S_{m(i)}.$$

The relation *R* is a type of algebraic operation that acts on a defined set of argument types and defines a mapping into a Boolean set: $R_i: S_{a(1,i)} \times \ldots \times S_{a(n,i)} \to S_{BOOL} = \{true, false\}$. The *Md* model can also be considered as a kind of operation that acts on a certain set of argument types and defines a mapping into a set of results, which in general have different types [28-30]:

$$Md_i: S_{a(1,i)} \times \ldots \times S_{a(n,i)} \to \{S_{r(1,i)}, \ldots, S_{r(l,i)}\}.$$

On the other hand, relations and models at the ontology level are separate entities. These entities in the algebraic system of types $\bar{T}$ correspond to data types whose instances store data about these relationships and models (metadata) [32]. An important component of the definition of relations in ontology is the integrity constraints imposed on the entities connected by the relation. In the defined system of types, this restriction corresponds to a set of Boolean expressions $Cs_{T_{REL}}$ for each type of relation T_{REL}, which must be true: $\forall {cs^i}_{T_{REL}} \in Cs_{T_{REL}}: ev({cs^i}_{T_{REL}}) = true$, where $ev()$ is an expression value evaluation function. Expressions that define integrity constraints are a component of the set of defining relations of the abstract data type: $Cs_{T_{REL}} \subseteq Ex_{T_{REL}}$ [28-30]. In some cases, restrictions are also imposed on the values of entity attributes [28-30]. Consider such constraints as integrity constraints for unary relations. Similarly to ontology entities, data types from $\bar{T}$ form a type hierarchy in which the subtype T_{pid} inherits the properties of the supertype T_{sup} if and only if the relation R_{isa} is defined between the ontology entities corresponding to the subtype and the supertype, i.e.:

$$\exists R_{isa}(TypeEn(T_{pid}), TypeEn(T_{sup})).$$

Similar to ontology entities, data types are built on top of other, simpler types. At the same time, these more complex types are structures containing elements of simpler types. We denote the type-structure as T_{wh}, and the types – constituent parts as T_{pt}^{i}. Then $T_{wh} = \{T_{pt}^{1}, T_{pt}^{2}, \ldots, T_{pt}^{n}\}$, if the ontology entities corresponding to the data types are connected by the relation $R_{\square as}^{'}$ [28-30]:

$$\exists R'_{has}(TypeEn(T_{wh}), \{TypeEn(T_{pt}^{1}), TypeEn(T_{pt}^{2}), \ldots, TypeEn(T_{pt}^{n})\}').$$

Let's define the *TypeParents*() function, which for each type *T* will return an ordered set of its supertypes and is a transitive closure of the inheritance relation: $TypeParents(T) = \{T^{1}, T^{2} \ldots, T^{n}\}$, where T^{i+1} is a supertype relative to T^{i}. Let's define the function *TypeName*(), which for each type of data *T* returns a unique identifier (description, name) of this type [28-30]. For example, for a T_{MD} data type corresponding to a model: $TypeName(T_{MD})$ = "$Model$". For instances t of an arbitrary type T, we define the functions that return the type and the corresponding entity of the ontology:

$$Type(t) = T, Entity(t) = TypeEn(Type(t)) = E.$$

We denote the multi-set of all instances of type *T* by $Population(T)$.

$$Population(T) = \{t | Type(t) = T\}.$$

We denote the multi-set of instances of a certain type T_i as $\hat{t}_i$:

$$\hat{t}_i | \forall t_i \in \hat{t}_i : Type(t_i) = T_i, \quad \hat{t}_i \subseteq Population(T_i).$$

We denote the abstract data type corresponding to the multiset of instances $\hat{t}_i$ by $\hat{T}_i$. In the general case, an arbitrary software object o can be identified as an object of many types. Let's define the *TypeId*() function, which will return a set of types that can be used to identify the object [28-30].

$$TypeId(o) = \{T_o{}^{1}, T_o^{2}, \ldots, T_o^{m}\}.$$

Since there is an isomorphic mapping between ontology entities and data types, it is appropriate to name data types in the same way as the corresponding ontology entities. Thus: $TypeName(T) = Name$. In those cases when it is necessary to emphasize the semantic interpretation of a certain type of data, we will indicate the abbreviated name of the type in the form of an index. For example, let's denote the model data type as T_{MD}, a specific element of this type as t_{MD}, or a multiset of model elements as $\hat{t}_{MD}$ [28-30]. Given the given notation, we define the knowledge base type as an abstract data type T_{BKN}, represented by a set (structure) containing the fact base type T_{BFC}, the ontology type T_{ON}, and the multiset type of models $\hat{T}_{MD}$: $T_{BKN} = \{T_{BFC}, T_{ON}, \hat{T}_{MD}\}$. An instance of the knowledge base t_{BKN} at each moment is represented by the structure: $t_{BKN} = \{t_{BFC}, t_{ON}, \hat{t}_{MD}\}$, in which $t_{BFC}, t_{ON}, \hat{t}_{MD}$ correspond to instances of the fact base, ontology, and multiset models. The t_{BFC} fact base is a set of facts about objects and events of the external world and the relationship between them, i.e., $t_{BFC} = \{\hat{t}_{FC}, \hat{t}_{RF}\}$. Elements of the multiset $\hat{t}_{FC}$ are data instances having the fact type T_{FC}, $TypeName(T_{FC})$ = "$Fact$" [28-30]. On the other hand, these elements also have one of the types T_{pid}, which is derived from T_{FC} [28-30]: that is,

$\exists R_{isa}(TypeEn(T_{pid}), TypeEn(T_{FC}))$. Each fact and relation is semantically interpreted, that is, its type is defined in the On ontology: $\forall t_{FC}^{i}: Entity(t_{FC}^{i}) \neq \emptyset, \forall t_{RC}^{i}: Entity(t_{RC}^{i}) \neq \emptyset$.

The ontology type contains the types of the multiset of class definitions $\hat{T}_{CL}$, and the relations between them – $\hat{T}_{RCL}$, i.e. $T_{ON} = \{\hat{T}_{CL}, \hat{T}_{RCL}\}$. Each class T_{CL} is defined by a set of attributes $\hat{T}_{SL}$, rules $\hat{T}_{RU}$, restrictions $\hat{T}_{CS}$ defined on these attributes: $T_{CL} = \{\hat{T}_{SL}, \hat{T}_{RU}, \hat{T}_{CS}\}$. At the ontology level, the j-th attribute A_i^j of the entity E_i is given by an attribute relation connecting this attribute with another entity E_k: $R_{atr}(A_i^j, E_k)$ so that for each instance of the attribute a_i^j its value is determined by the instance e_k. Each attribute is specified by the type of its values T_{VSL}, multisets of rules and restrictions acting at the attribute level – $\hat{T}_{RUS}, \hat{T}_{CSS}$, i.e., $T_{SL} = \{T_{VSL}, \hat{T}_{RUS}, \hat{T}_{CSS}\}$. The value of the t_{SL} attribute belongs to the set of valid $RgVSL$ values: $\forall i: t_{SL}^{i} \in RgSL$. The constraint T_{cs} determines the mapping of a set of attribute values into a set of Boolean values $\{true, false\}$: $T_{cs}: \hat{T}_{SL} \to \{true, false\}$ [28-30]. The T_{RU} rule defines one subset mapping of attribute values to another. Each rule T_{RU}^{j} is associated with two non-intersecting subsets A_{bas}^{j}, A_{der}^{j} of class attributes [28-30]:

$$T_{RU}^{j}: A_{bas}^{j} \to A_{der}^{j}, \;\; A_{bas}^{j} \cap A_{der}^{j} = \emptyset.$$

The type of relationship between T_{RCL} classes are set on the ordered sequence of class types that connect: $(T_{CL}^{1}, T_{CL}^{2}, \ldots, T_{CL}^{n})$ [28-30].

The relationship between classes is itself a class in the sense that it is defined by a set of attributes, rules and restrictions [28-30]:

$$T_{RCL} = \{(T_{CL}^{1}, T_{CL}^{2}, \ldots, T_{CL}^{n}), \hat{T}_{SL}, \hat{T}_{RU}, \hat{T}_{CS}\}.$$

This approach allows you to consider relations as separate types of data and predict the possibility of forming relational structures. The KB fact type T_{FC} is a supertype for the class types T_{CL} and relations T_{RCL} [28-30]:

$$\exists R_{isa}(TypeEn(T_{CL}), TypeEn(T_{FC})), \exists R_{isa}(TypeEn(T_{RCL}), TypeEn(T_{FC})).$$

The models form a network of type T_{NMD}, which is defined as a set that includes multisets of the type of models $\hat{T}_{MD}$ and their relation $\hat{T}_{RMD}$, i.e. $T_{NMD} = \{\hat{T}_{MD}, \hat{T}_{RMD}\}$. Unlike ontology classes, models do not form a clear hierarchy but form a dynamic network in which connections and the models themselves can change, reflecting learning processes, changes in the external world, or the process of solving a certain problem [28-30].

Each model can be in one of two states - active or passive. Accordingly, we divide the set of model instances into subsets of active and passive models that do not intersect [28-30]:

$$Population(T_{MD}) = \hat{t}_{MD}^{ac} \cup \hat{t}_{MD}^{ps}, \;\; \hat{t}_{MD}^{ac} \cap \hat{t}_{MD}^{ps} = \emptyset,$$

where $\hat{t}_{MD}^{ac}, \hat{t}_{MD}^{ps}$ are multisets of instances of active/passive models.

An active model is a model initialized with information from a certain context. Models enter the active state at the request of other models or when certain events occur. Active models are used to solve current problems of the system and to interpret knowledge in the system. If the need for the model has disappeared (a result has been obtained, the goal has been achieved), then the model leaves the active state. Model interaction T_{RMD} is a data type that represents the activation relationship used to decide whether to activate the model(s). Let's consider the process and structure of interaction and activation of models in more detail. The activator model acts as the initiator of establishing a connection between models. The need to establish a connection does not always arise, but only when to solve the main problem, it is necessary to solve auxiliary problems presented in other models. For example, if the input data received by the activator from the context is incomplete it is necessary to activate other models to define the data. Such a determination may consist of searching for the necessary information in a database, the Internet, contacting a consultant, etc. [28-30].

Each type of T_{RMD}^{i} corresponds to a class of problems that must be solved by T_{PR}^{j} as a result of interaction [28-30]. In turn, the class of problems T_{PR}^{j} corresponds to a set of models $\hat{T}_{MD}^{j}$ that can be applied to solve problems in this class. During the interaction of models, the tasks of determining relevance, optimal selection among relevant models, and initialization of the selected model are successively solved. The relevance function is a mapping of the current context t_{CON} and the set of alternatives $\hat{t}_{MD}$ into the set $\{true, false\}$, i.e. $F_{rel}: t_{CON}, \hat{t}_{MD} \rightarrow (true, false)$. Determining the relevance of models allows you to select only relevant models for the selection procedure: $\hat{t}_{MD}^{re} \subseteq \hat{t}_{MD}$, i.e. models t_{MD}^{re} for which $F_{rel}(t_{CON}, t_{MD}^{re}) = true$. In the absence of relevant models, the modelling system returns a message to the activator about the impossibility of solving the problem [28-30]. The problem of optimal selection determines one t_{MD}^{op} from a set of relevant models, the application of which maximizes the selection function F_{ch} taking into account the selection criteria $\hat{t}_{CR}$ and the context t_{CON}, i.e. $F_{ch}(t_{MD}^{op}, \hat{t}_{CR}, t_{CON}) \rightarrow max$ [28-30]. The initialization function F_{in} maps the current context t_{CON} into a set of attribute values of the selected model – t_{VSL}, where $F_{in}: t_{CON} \rightarrow t_{VSL}$. Summarizing, we define T_{RMD} as a set [28-30]: $T_{RMD} = \{T_{PR}, \hat{T}_{MD}, F_{rel,} F_{ch}, F_{in}\}$. The model type T_{MD} consists of the schema types T_{SCM} and the implementation T_{IMD}: $T_{MD} = \{T_{SCM}, T_{IMD}\}$.

The model scheme describes its structure, and constituent elements, defines rules and restrictions on the use of the model, as well as a list of possible operations and requests. A schema is a component of a model that is visible to the outside world. It is used to interact with the model. The model scheme consists of slots $\hat{T}_{SLM}$, relations between them $\hat{T}_{RSM}$, rules $\hat{T}_{RUM}$, constraints $\hat{T}_{CSM}$, and operations $\hat{T}_{OPM}$: $T_{SCM} = \{\hat{T}_{RO}, \hat{T}_{RRO}, \hat{T}_{RUM}, \hat{T}_{CSM}, \hat{T}_{OPM}\}$.

A model slot is an attribute – role. For each slot, the function F_{RG} is defined, which specifies the set of classes $\hat{T}_{CL}^{RG}$, the objects which are allowed to initialize the slot: $F_{RG}: T_{SLM} \rightarrow \hat{T}_{CL}^{RG}$. In addition, the rules and restrictions operating at the slot level are defined for the model slot – $\hat{T}_{RUSM}$, $\hat{T}_{CSSM}$, operations on the $\hat{T}_{OPSM}$ slot values: $T_{SLM} = \{\hat{T}_{CL}^{RG}, \hat{T}_{RUSM}, \hat{T}_{CSSM}, \hat{T}_{OPSM}\}$. The slot relation T_{RESM} is specified by the set of slots that it connects $\hat{T}_{SLM}^{resm}$, by the set of ontology classes used for semantic interpretation of the relation – $\hat{T}_{CL}^{resm}$, by the set of models used to understand and carry out operations with the relation – $\hat{T}_{MD}^{resm}$ [28-30]: $T_{RESM} = \{\hat{T}_{SLM}^{resm}, \hat{T}_{CL}^{resm}, \hat{T}_{MD}^{resm}\}$. At the same time, for each instance of the relation, the slots connected by it belong to the slots of the model: $\hat{t}_{SLM}^{resm} \subseteq \hat{t}_{SLM}$. The relation of models corresponds to one of the types of relations defined in the On ontology [28-30]: $TypeEn(T_{RESM}) = E_{RESM} \in \bar{E}$.

The model describing the relationship is an element of the general set of models: $t_{MD}^{resm} \in Population(T_{MD})$. Let t'_{BFC} be a certain situation, state, or snapshot of the fact base. Let's define the goal data type T_{GL}, each instance of which is a specification of a certain set of states of the fact base, each of which corresponds to the achieved goal: $t_{GL} = \hat{t}_{BFC}^{GL}$. To determine the goal, it is useful to set the goal function, which allows you to determine whether in a certain state t'_{BFC}, the goal *GL* is achieved:

$$F_{gl}(t'_{BFC}) = \begin{cases} true | t'_{BFC} \in t_{GL} \\ false | t'_{BFC} \notin t_{GL} \end{cases}$$

One of the possible ways of assigning the objective function is its assignment in the form of an ordered list of statements-requirements $\hat{t}_{ASR}$ regarding objects of the information base that can be checked: $F_{gl}(t'_{BFC}) = \hat{t}_{ASR}(t'_{BFC})$, where $t_{ASR}(t'_{BFC})$ is a requirement - assertion regarding the values of properties of objects and their connections in the situation t'_{BFC}. Each statement $t_{ASR}(t'_{BFC})$ is a function defined on the set {*true*, *false*} [28-30]:

$$Range(t_{ASR}(t'_{BFC})) = \{true, false\},$$

where the function *Range*(*f*) specifies the value range of the function *f*. So,

$$F_{gl}(t'_{BFC}) = true \Leftrightarrow \forall t_{ASR}(t'_{BFC}) \in \hat{t}_{ASR}(t'_{BFC}) : t_{ASR}(t'_{BFC}) = true.$$

Executed ontological models are designed to solve specific problems specified in the form of a goal [28-30]. For the convenience of finding the necessary models, it is advisable to organize information about models in the form of the $OnGl \subset On$ ontology, with entities and relations. In the ontology, we will define categories of models according to the classes of problems they solve. For example, we will separately define classification models, algorithmic models, object and service models, access control models, and situational models. Information about objects that describe individual implementations of models is used by the intelligent system (IS) modelling service. The model interaction broker uses the target ontology to search for the model needed to solve the given task.

4. Experiments, results and discussions

4.1. Knowledge models verification tools for an intelligent system

4.1.1. Method and algorithm for obtaining a consensus decision in the process of model verification

Let each expert (reviewer) evaluate the verified knowledge model using an integer positive scale of the form $S = \{i_{min}, ..j,.. i_{max}\}$. Then each grade *j* corresponds to an integral review (positive or negative), which we obtain using the expression $r_j = |(i_{max} - i_{min})/2| - j$. The result of this expression translates the expert's assessment into an integral review (or simply a review), which is represented by a Boolean variable with a value of 1 for negative r_j, and 0 otherwise. We represent the set of reviews by a weighted graph $G(V, E)$ without cycles, the set of vertices of which represents the set of reviews, and each edge $e_{ij} \in E$ corresponds to the operation of comparing reviews v_i, v_j. The weight of such an edge

(Boolean variable a_{ij}) gives the result of the comparison of reviews. Let's call it a review comparison graph. Let's call the vertex of this graph workable if the corresponding review is not erroneous, and unworkable otherwise. The algorithm for comparing reviews is presented in [33].

4.1.2. Construction of knowledge models for verification of conceptual models of decision support

Solving the problem of model verification is a repetitive business procedure in an intelligent model-based system. For its implementation, it is important to follow uniform approaches, recommendations and rules, which will ensure uniform requirements for the set of models used. Such approaches reflect the technical policy of the organization in the issue of model verification and are a component of the corporate standards of this organization. The task of model verification is complex and difficult to formalize and is therefore solved by expert evaluation. At the same time, the model evaluation system should be flexible enough and provide the opportunity to use different evaluation methods. The use of one or another method depends on the content and purpose of the model, and available resources. To develop such a unified approach, it is necessary to solve some problems, in particular:

- ensure the formalization of a set of knowledge models and, thus, create the possibility of their automated execution;
- to provide the possibility of using different methods of evaluating models and for this purpose create a taxonomy (ontology) of types of models, united by a common task – verification of models through their evaluation;
- provide the manager managing the process of implementation and use of the models with the means to control and manage the evaluation process;
- implement the possibility of reuse of models and knowledge provided by models, avoid duplication of knowledge in models.

4.1.3. Hierarchy of verification methods and models

Evaluation models form a hierarchy, at the root of which is the evaluation model, presented in the most general, abstract form - the general model. This model is used in the modelling system to form a request to solve a problem and to present the most general parameters of this request. Based on these parameters, the component of the modelling system - the model interaction broker - chooses a method of solving the given task that is adequate to the request, provided by one or another model.

The *Md* model consists of the *ScMd* circuit and the *RlMd* implementation:

$$Md = (ScMd, RlMd).$$

The scheme of the model is directly visible to the user and the designer of the model - a specialist in the specified subject area. The designer can modify the scheme by creating derivative models. The implementation of the model is created by a specialist programmer and ensures the execution of the model. The general model acts only as a scheme, that is, it has no implementation. Thus, each model is considered an integral part of a certain hierarchical system of models that implement different methods of solving similar problems, complementing or replacing each other depending on the specifics of a specific problem. Let's consider the method of using the expert consensus model on the example of the model hierarchy, which includes the general model, the voting decision-making model, and the expert consensus model. We present the general Evaluation model as a set of the following entities and relationships (Fig. 1):

- *Entity EvaluatedObject*. Defines the evaluated subject. In general, any subject can be evaluated.
- *Type*(*EvaluatedObject*) = *Thing*, where *Type*() is a function that returns the ontology type to which the entity argument corresponds, and Thing is the root element of the ontology.
- *Evaluator entity*. Defines the entity that forms the assessment of the subject of assessment.
- *Evaluate relationship*. Defines the evaluation operation itself and combines the subject and the evaluation object (*Evaluator* and *EvaluatedObject*). An important attribute of the Evaluate relation is the entity Value, the definition of which is the result of the execution of the model.

As can be seen from the definition of the *Evaluation* model, it corresponds to a wide class of tasks. At the same time, the problem of model evaluation is a partial case of the general evaluation problem. For it, we will define additional restrictions on the elements of the general model, which are transferred when accessing it during the activation process. So, the subject of evaluation is the model, i.e. *Type*(*EvaluatedObject*) = *Model*. In addition, it is assumed that the rating of the model is chosen from a discrete rating scale ($true, false$). So, $Range(Value) = \{true, false\}$, where *Range*() is a function that returns a range of values for the *Value* attribute entity.

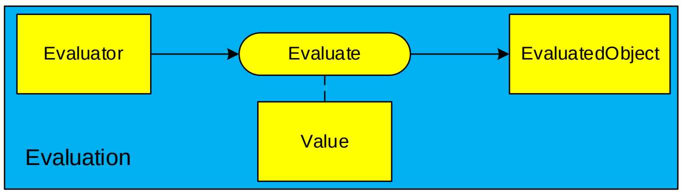


Figure 1: Scheme of the general evaluation model

A sufficient condition for the activation of the general model is the determination of the *Evaluator* and *EvaluatedObject* roles. If the evaluation request does not explicitly specify an evaluator, it can be determined by the model interaction broker based on the available information about the evaluation subject and evaluation requirements. A condition for the successful execution of the model is the definition of the attribute entity *Value*. The model of consensus assessment has two successive stages of implementation (Fig. 12).

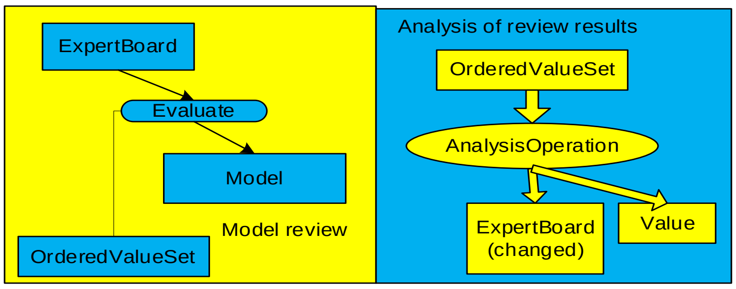


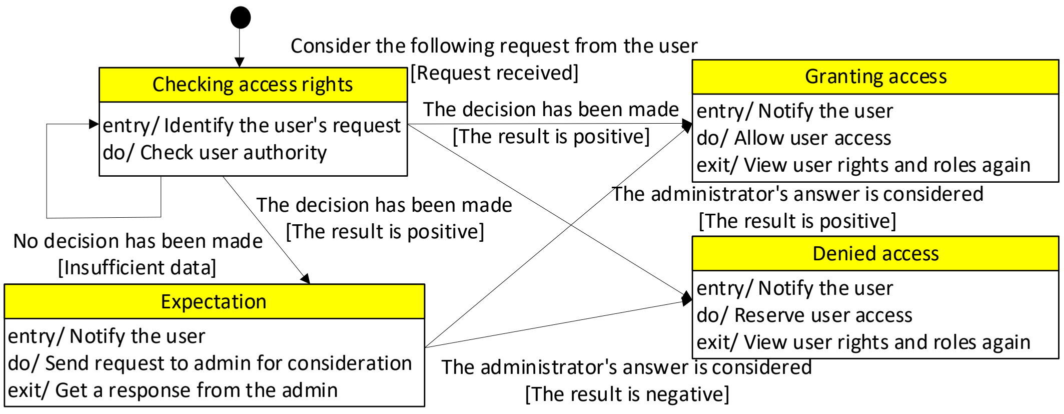
Figure 2: Outline of the consensus assessment model

In the first stage, an initial review of the evaluated object provided by the *Model* entity is performed. The prerequisite for activating the model and starting the first stage is the determination of the evaluation object (model) and the expert commission. As a result of the first stage, an ordered set of *OrderedValueSet* values is formed. The condition for the completion of the first stage of model execution is the completion of the formation of its results (rectangles with grey filling). The first phase of the model may take longer to complete, as experts need time to analyse the model and create estimates. IS modelling monitors the completion of individual stages and informs the manager about their status and completion. The second stage is performed automatically, without the participation of experts. The operation of analysing the results determines the group of experts who did not make mistakes in the assessments and determines the decision made by the consensus of the experts. As a result of the execution of the stage, a consensus decision of *Value* is formed and the rating of the experts who participated in the evaluation is updated.

4.2. Analysis and research of knowledge presentation methods in tasks of controlling access to information resources

Due to the constantly growing number of information systems, the problem of managing user identification information and access rights to information resources for them arises. This problem inevitably affects the information security of the enterprise and creates some complications and risks in various spheres of activity [34].

The world's leading consulting companies recognize that the topic of managing access rights to corporate information resources is fundamental in companies' information security strategies and is most relevant in enterprises with a developed IT infrastructure and a large number of employees, such as banks and financial organizations, telecommunications companies, large holdings, oil and gas and energy companies. In the absence of mechanisms for centralized automated management of employee access rights, large companies may face many problematic factors, including a low level of information protection, long-term coordination and provision of access rights to information resources, and significant labour costs for changing employee access rights. The desired behaviour of the automated module of the system, which is responsible for making automatic decisions about granting or denying access rights to information objects to the user, can be described using an object-oriented approach and UML notation, giving the example of a system state diagram that describes important aspects of functioning system and possible sequences of states and transitions, which collectively characterize the dynamic behaviour of the system module during its life cycle (Fig. 3) [34].



Consider the following request from the user
[Request received]
Checking access rights
entry/ Identify the user's request
do/ Check user authority
Granting access
entry/ Notify the user
do/ Allow user access
exit/ View user rights and roles again
The decision has been made
[The result is positive]
The decision has been made
[The result is positive]
No decision has been made
[Insufficient data]
The administrator's answer is considered
[The result is positive]
Denied access
entry/ Notify the user
do/ Reserve user access
exit/ View user rights and roles again
Expectation
entry/ Notify the user
do/ Send request to admin for consideration
exit/ Get a response from the admin
The administrator's answer is considered
[The result is negative]

Figure 3: Status diagram of the access rights management module

From the initial formal state, the system switches to the state of checking access rights. This state involves the identification of the received user request, and its consideration. In this state, the access rights management system checks the user's authority and makes certain decisions based on this check. This state is characterized by reflexivity, that is, it can pass into itself. This happens when the next request is received from some user and therefore needs to be considered and checked. The verification decision can be made independently by the system or with the participation of the administrator. If the decision is not made independently, it means that the knowledge base of the system has little or not enough data about decision-making, that is, its informational component is incomplete in some aspects. As a result, the system goes into a waiting state, notifies the user about this and sends a request to the administrator so that he can make a decision. The main feature of this state is it's a priori indefinite duration. This does not mean that the system is stuck on processing one request. In parallel, it processes other requests, that is, there are as many established sessions with users to process their requests as necessary (a kind of multi-session). To exit this state (in the context of some session), the system must receive and consider a response from the administrator [34].

If the administrator, by his decision, allows the user to gain access, the system goes into the state of granting access, notifies the user about this and allows unhindered use of the granted access rights [34]. If the administrator, by his decision, has forbidden the user to gain access, the system goes into the state of access denial, again notifies the user about this and does not allow the use of certain information resources requested by the user. However, the access rights management system can independently decide whether to grant or deny access. These actions are similar to those described above. The only difference is that the waiting state is bypassed and the administrator does not participate in the decision. The system can make a positive decision and grant access, or a negative decision and deny access. This model (diagram) does not take into account (does not show) the fact that access may not be full, but limited, that is, the user may be allowed to perform only some operations on information objects (for example, only viewing and reading operations). This fact significantly improves access control methods based on RBAC roles. The considered behaviour of the system leads to the implementation of the system in the form of a demonstration prototype of the expert system, based on which it will be possible to decide the suitability of the expert system for solving the tasks set before it. In this situation, the expert can be an administrator who is well aware of the process of granting or revoking access rights to the company's information resources. That is, this is a highly qualified specialist in the problem area who can determine the knowledge characterizing the problem area, as well as ensure the completeness and correctness of the knowledge entered into the system [34]. The developed implementation of the process of expert communication with the expert system is as follows [34]. The expert (here - the administrator) describes the problem area in the form of a set of facts and rules. Facts define the objects (files, users, roles, etc.), their characteristics and values that exist in the domain of expertise. The rules define methods of data manipulation that are specific to the problem area. The expert, using the knowledge acquisition component, fills the system with knowledge that allows the expert system in decision mode to independently (without an expert) solve problems in the problem area, as well as to self-learn (in this context, it means to derive new knowledge according to certain rules, using already existing in the knowledge base) and thus go on the path to a certain automation, which will eliminate the routine participation of the administrator in interaction with users when granting or denying access rights to the organization's network resources.

Since it is about the creation of a method of automating the work of an administrator (an expert in his field) regarding the management of access rights, the question of the application of expert systems in their classical sense is appropriate. That is, in this sense, an expert system is a computer system that contains the knowledge of specialists from a certain problem area and is capable of making independent expert (in this case, administrative) decisions within this area. The structural diagram of the intelligent access rights management information system is shown in Fig. 4. The knowledge base is the central part of the expert system. In our case, the knowledge base will be stored separately from the expert system (on disk or other media) in XML format. Saving and describing the knowledge base in XML format is relevant and perhaps the best option among all others today. Still, the preservation and description of the knowledge base in Prolog today is far from the optimal option compared to the past decades [34]. A possible disadvantage of the system can be considered the fact that the initial significant knowledge is acquired by the system implicitly by modifying the file with the knowledge base by an expert or engineer of knowledge through a third-party XML editor of the knowledge base. To present knowledge in the knowledge base, a production model is selected, an element of which is a production rule. Since the task completion time is not a critical indicator of the development of a given system and the generic hierarchy of concepts, although difficult, can be presented, and the modularity and ease of modification are the best fit for the delivered system, the production model is a fairly good option to choose. As an alternative, you can choose a frame model. For the inference algorithm to be able to operate with facts, and values of facts, take into account their relationship in a certain rule and draw conclusions corresponding to this set of facts, the KB of the expert system is presented in the form of certain structures [34]: list of targets

<Targets>, an array of variables (objects) <Objects>, the array of questions <Questions>, and a set of rules <Rules>.

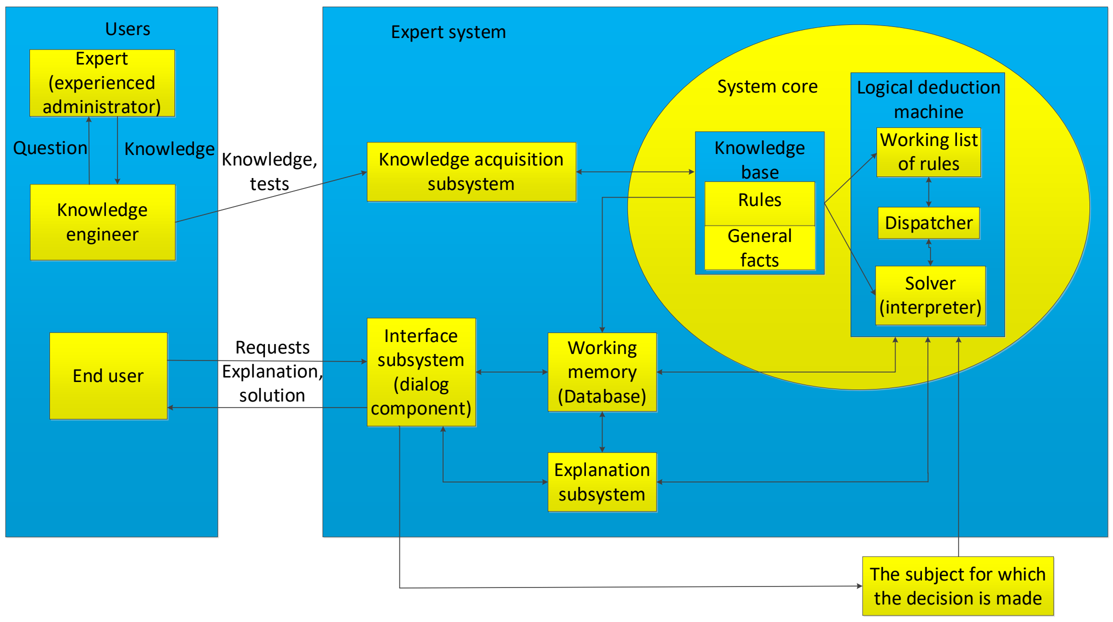


Figure 4: Structural diagram of IR access rights management

The list of goals is those goals that are set before the system for their solution [34]. For example, in KB, the goal can be written as follows:

```
<Targets>
            <Target name="Ability_to_grant_access_rights" />
</Targets>
```

Due to the ease of making changes to the knowledge base, adding new goals is not difficult [34]. The main thing is to ensure appropriate processing and sequence of actions in the system to achieve the entered goal. The main variables (objects) that appear in the knowledge base include employees, their positions, which can be associated with roles in the production unit, information resources (important files and directories that need protection from unwanted intervention), operations on resources, as well as duplicate goals, which structurally store possible alternative responses when the given goals occur. All these variables are separated from each other. For example, an employee can be entered in the KB using the following construction:

```
<Object name="Workers">  <Value> Brenych_Andrii </Value> </Object>
```

Employee last names and all possible values of other variables are written in alphabetical order, which identifies the value entries in the KB. For example, the set of positions in the knowledge base is described as follows [34]:

```
<Object name="Positions (roles)">
 <Value> Designer </Value>
 <Value> Engineer </Value>
 <Value> Project_Manager </Value>
</Object>
```

In all systems based on expert technologies, there is a dependency between the input data stream and the data in the KB [34]. During consultation with the system, input data is compared with data in the knowledge base. The result of the comparison is a negative or positive answer. In a rule-based system, an affirmative response is the result of one of the production rules. These production rules are determined by the input data. So, the main variables are divided into input (or explicit, which is set by the system user) and hidden (or implicit, which are initialized by the system based on input variables). Input variables include employee selection, information resources, and resource operations. And to the hidden - the choice of position (role). It is obvious that the person who will use the created system, or the system itself, when making independent decisions in the future, does not need to know the position of the employee. The system independently determines his position (role) based on the established production rules in KB. The question array <*Questions*> stores questions for the system user, which are related to the purpose of obtaining the necessary input knowledge from the user. These questions are: "*Select an employee*", "*Select an object to access*", "*Select possible operations on the object*" and a special question designed to select a target. The products are stored in the <*Rules*> array. It is determined in advance that the number of conditions in the rule is not limited. Let there be *N* rules of a similar structure. Each *i*-th rule in the knowledge base has the following structure [34]:

```
<Rule id="i"> <Condition name="Variable ID (name)" value1="Value of variable"/>
 [<Condition name=" Variable ID (name)" value2=" Value of variable " /> , ...]
  <Consequence name=" Variable ID (name)" value3=" Value of variable " />
       <Reason text=" comment (explanation) to the rule " />
```

Here, *Rule id* is the number of the rule, Condition name is the facts of the *i*-th rule, $value_1$, $value_2$ is the values of the facts, square brackets mean optionality (since the rule can contain from one to m facts), Consequence name is the name of the *i*-th output rules, value3 - the content or value of the output, Reason text - an explanation that indicates the existing rule. Now we will show how the system automatically recognizes the employee's position. Let's take for example some rule, which is recorded in the KB as follows [34]:

```
<Rule id="1">
 <Condition name="Selection_employee" value="Brenych_Andrii" />
 <Consequence name="Select_position" value="Programmer" />
 <Reason text="the first explanation" />
</Rule>
```

This rule uniquely identifies an employee's position (role) based on his or her last name and first name. That is, if the input value when selecting an employee is Andriy Brenych, then this rule will work and as a result, a fact will be obtained that certifies that his position is a programmer. Since the user is only asking about one person (i.e. himself), there is no need to worry about the position not being explicitly tied to an employee. Working memory will store copies of facts that are related to each other. This is possible by creating a separate session for each user request. We will demonstrate how the KB uniquely identifies the denial of access to a file. Let us take as a basis the following rule [34]:

```
<Rule id="13">
 <Condition name="Select_position" value="Designer" />
 <Condition name="Select_object" value="ClassDiagram.uml" />
 <Consequence name="Ability_to_grant_access_rights" value="No, it should_not_be given" />
 <Reason text="explanation thirteen - the designer has nothing to do with UML diagrams" />
</Rule>
```

Suppose that the IS has determined the position of the employee and received knowledge from the user about the consideration of the *ClassDiagram.uml* object. Then, regardless of the possible operation on the file received from the user (without even resorting to considering the operation), the IS uniquely identifies the prohibition to obtain the requested rights, since the designer has nothing to do with the diagrams. This is how unambiguous permission to access a file and perform the necessary operations on it is implemented. The developed rules also provide the possibility of access control distributed by operations. Let us show this using the example of the following rule [34]:

```
<Rule id="22">
        <Condition name="Select_position" value="Designer" />
        <Condition name="Select_object" value="Readme.txt" />
        <Condition name="Select_operation" value="reading (viewing)" />
<Consequence name="Ability_to_grant_access_rights" value="Yes, it can_be provided" />
<Reason text="explanation 22 temporary file with instructions for each"/>
</Rule>
```

Thus, the developed logical structure of the knowledge base is quite flexible and easily amenable to modifications. As for the logical inference machine, it uses a deductive algorithm with a direct chain of reasoning (it corresponds to a data-to-goal strategy or a data management strategy) with the option of searching for a solution broadly or deeply. The logical inference machine (rule interpreter) in the developed expert system performs two functions [34]. First, review the rules from the knowledge base. Secondly, the application of the rules. In the created IS, the interpreter of production rules works cyclically. In each cycle, it reviews all the rules to find those conditions that match the known and current facts. Once selected, the rule is triggered, its output is stored in working memory, and then the cycle is repeated from the beginning [34]. During each cycle, many rules can be activated and placed in the working rule list. In addition, in the working list of rules, the results of the activation of rules from previous cycles remain, if there is no deactivation of these rules because their left parts are no longer executed [34]. In this way, during the execution of the program, the number of activated rules in the working list of rules changes. Only one rule can work in one cycle. If several rules are successfully matched with facts, then such a situation is called a conflict. The interpreter performs conflict resolution by choosing a single rule based on a certain criterion, depending on the choice of conflict resolution strategy (wide or deep). A significant advantage of the system is that it explains all its administrative decisions thanks to the built-in explanation subsystem. This is how events are logged [34]. The problem solved mathematically with the help of the developed system can be described through sets of relevant entities [34]: Users – multiple users *u*, Roles – multiple roles *r*, and Permissions– multiple sets of access rights *p*. Then the user-role relation (UR relation) is mathematically represented as follows [34]:

$$UR \subseteq Users \times Roles. \tag{1}$$

The formal presentation of the role-legal relation (RP relation) is as follows:

$$RP \subseteq Roles \times Permissions \tag{2}$$

The access control relation (AC relation) can be presented as a composition of relations (1) and (2) [34]:

$$AC = UR \circ RP, \tag{3}$$

In other words, AC is defined as the set of corresponding pairs:

$$AC = \{(u, p) \in Users \times Permissions \mid \exists\, r \in Roles, (u, r) \in UR \wedge (r, p) \in RP\}$$

In addition, you must specify access rights sets for specific files (F) and directories (D). This maximizes the scalability of the system and brings a certain novelty effect to the modern RBAC model. The set of files and directories ($F \cup D$) should be considered as a set of constituent elements of some workspace, which requires "vigilant", specifically specified supervision, and not as a set of all files and directories of the file system since this would be a significant redundancy from the administrator's point of view or access rights management systems. Therefore, it is worth correcting (4) and presenting it as follows:

$$AC = UR \circ RP \circ (F \cup D). \qquad (4)$$

From the point of view of the system, the self-learning process in (5) can be depicted as a reverse arrow, the essence of which is that the system will eventually be able to independently derive new knowledge and make adequate decisions based on existing rules and knowledge, thereby replenishing its knowledge base with new one's knowledge and carrying out effective control and management of access to files and their content depending on user requests and their real access rights. So [34]:

$$\begin{tikzcd} AC = (UR \circ RP \circ (F \cup D)) \circ \arrow[loop above] \end{tikzcd} \qquad (5)$$

The developed IS of access rights management contains several subsystems that ensure the proper functioning of the system. Such subsystems include access control, administration and knowledge engineering subsystems. It is convenient to display subsystem data in the form of an object (package) diagram. This diagram is shown in Fig. 5 [34].

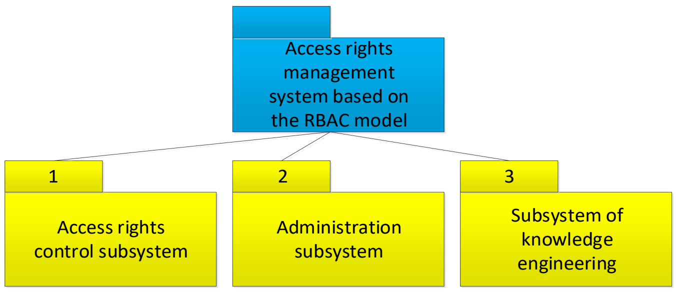

Access rights management system based on the RBAC model
1
Access rights control subsystem
2
Administration subsystem
3
Subsystem of knowledge engineering

Figure 5: Object diagram of the developed access rights management system

A significant difference of IS from all others is that it can self-learn and over time can become automated (in the sense that it will be deprived of the routine participation of the administrator in the processes of its functioning, related to the decision-making on granting or denying the rights of access of employees to informational corporate resources) [34]. The implementation of this access rights management system will allow for solving several problems, such as unauthorized access to corporate resources, as well as significant labour costs for granting and changing employees' temporary access rights to information objects.

4.3. Models and methods of presenting knowledge in tasks of automated testing of software products

One of the most important problems in the software industry is the high level of complexity of software systems and the related problems of complexity and high cost of administration, development and modification, a significant level of defects in such systems [35-36]. The National Institute of Standards and Technology (NIST) estimates that the annual cost of software defects to the US economy is $59.6 billion [37]. Product testing using a defined set of usage scenarios is traditionally used to detect defects [38]. Today, the cost of testing occupies a significant part of the total cost of product production. At the same time, the complexity of the software makes its exhaustive testing impossible [39]. To increase the efficiency of testing and reduce costs, automated testing is used [40]. Performing the task of automated testing involves performing various operations related to the preparation of the test environment, obtaining and installing software products, and configuring the operating system and testing tools. This task is usually performed by an automated testing expert and is a complex task because it requires careful consideration of a large number of interdependent factors. An error that occurs due to improper preparation of the test environment is expensive because then the test results have to be cancelled and the time (sometimes several hours) spent on such erroneous testing is lost. At the same time, time constraints are a significant requirement for automated testing itself, as developers and employees of the quality control department need to receive test results as soon as possible. In many firms that develop software products, the practice of nightly product layouts is adopted. At the same time, the code changes made by the developers during the day are integrated into the new version of the product at the end of the working day. This product is tested with a minimum set of tests to detect violations of basic functionality caused by new code. At the beginning of a new day, developers receive a list of defects that need to be fixed. As a rule, automated test sets are used to conduct such night testing. The testing system requires high reliability, the maximum level of automation, and - if possible - fully automatic execution. On the other hand, the automated testing system works in conditions of constant changes both in the tested product itself and in the testing environment. Testing specialists have to constantly adapt the testing code, and reconfigure the testing system under strict time constraints [35].

Organizing and conducting automated testing requires taking into account a large number of interdependent details and requirements [35]. At the same time, non-fulfilment of even one requirement, or the appearance of even one failure during testing, leads to a stop and rejection of test results. This places increased demands on the automation engineer. The constant change of the tested product, the change of its interface, and the appearance of new features and capabilities require constant changes and retesting of the test code itself to achieve compliance with the tested product. At the same time, there are strict time frames for completing these tasks.

To solve the above-mentioned problems for building the test code itself, such architectural solutions as using a test library, building tests based on keywords, a tabular approach, or data-driven tests [41], and tests based on models [42] are proposed. At the same time, existing architectural solutions do not allow automating the task of setting up and preparing the testing environment and conducting the testing itself, which is complex and performed manually [35]. The specified features of automated testing of software systems implement intelligent, knowledge-oriented methods for building a testing system promising [43-48]. At the same time, SA testing entities and dependencies provided in the relevant ontology are taken into account. The central place in such systems is occupied by algorithmic models because the testing process itself takes place as a sequence of operations given by a certain algorithm [49-54]. We will consider ways of presenting and using algorithmic models using the example of an intelligent system for automated testing of software products [35].

4.3.1. The architecture of the automated testing system

The second part of this work presents the architecture and principles of operation of an intelligent system that uses executable conceptual models [35]. Support for algorithmic models adds new components to the proposed modelling system (Fig. 6). The central element of the modelling system is the ontology of the subject area, which provides a semantic interpretation of all the facts of the information base [55-72]. Models are also built based on this ontology.

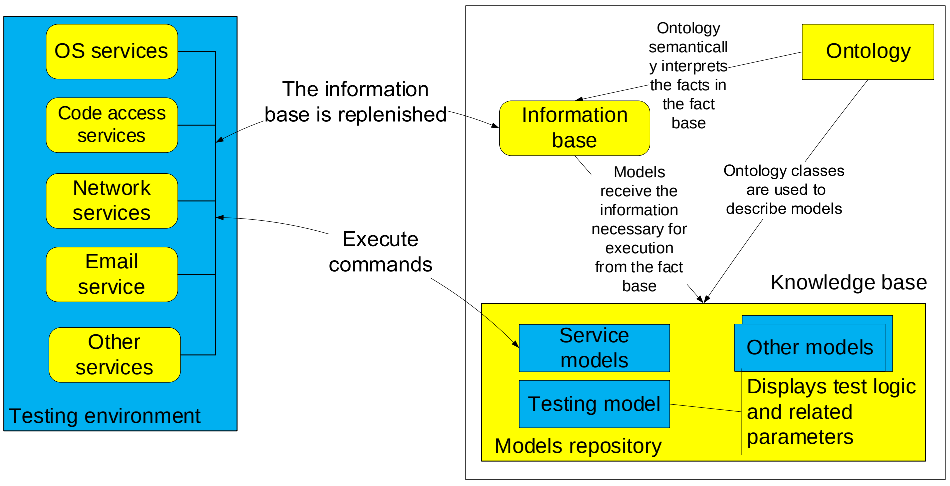

OS services
Code access services
Network services
Email service
Other services
Testing environment
The information base is replenished
Execute commands
Ontology semanticall y interprets the facts in the fact base
Ontology
Information base
Models receive the information necessary for execution from the fact base
Ontology classes are used to describe models
Knowledge base
Service models
Other models
Testing model
Displays test logic and related parameters
Models repository

Figure 6: Architecture of an intelligent automated testing system

The information base is constantly updated with new facts reflecting the state of the subject area [35]. Such facts are information about the product under test and the state and configuration of the simulation environment. An essential fact is the protocol of the current testing, which displays the status and results of the execution of all intermediate test results.

External services are an important element of the system. The purpose of the system is achieved by generating a sequence of commands by the testing system and their execution by external services. Among such important external services, we can mention operating system command services, file download services (FTP, HTTP), source code retrieval services (VSS), etc. Access to external services is provided through models of these services. Service models encapsulate entities that describe service configuration parameters, commands executed by the service, service states, dependencies between them, restrictions and operating conditions [35].

4.3.2. Formal specification of an algorithmic model

The central component of the intelligent automated testing system is the testing model, which belongs to the class of algorithmic models. The task of an algorithmic model is to represent a sequence of operations so that the execution of this model is the execution of this sequence of operations. Such a sequence of operations describes the algorithm for solving a certain problem, which explains the name of the model. Formally, an algorithmic model, like any other model, consists of a scheme and an implementation [35]:

$$AlgMd = (ScAlgMd, ReAlgMd).$$

The model diagram displays the parts of the model and the dependencies between them. The model implementation provides model execution. If conditions, requirements, or the test environment change, the SA changes the model schema, leaving its implementation unchanged. This approach provides flexibility and the ability to quickly adapt the model to changes. The model schema contains metadata sections, model bodies, and some other, auxiliary sections that allow you to initialize the model, verify it, define the necessary prerequisites for its execution, etc. The body of the algorithmic model is represented by an unordered set of operation models and an execution logic model: $BodAlgMd = (M(OpMd), FlMd)$. The $FlMd$ execution logic model, depending on the current state of the testing process and test environment, activates operation models, changes values in the information base, or deactivates the algorithmic model itself. This model is a set of situational models, each of which corresponds to a specific identified situation, and consists of signature and action specifications [35]:

$$FlMd = M(SitMd), \qquad SitMd = (SgSit, AcSit).$$

If the signature of the situation given by the set of conditions is true, then the actions specified in the model are performed. The execution of each action corresponds to one step of the algorithm [35]. Operation models are activated during an action. We believe that each operation after completion analyses the success of its execution, updates the testing status and enters appropriate messages in the test protocol. After the execution of the operation, the execution logic model is executed again. Execution of the $OpMd$ operation model involves sequential execution of tasks: initialization and verification of prerequisites; operating (applying to a model or service); analysis of the results and update of the test protocol: $OpMd = (InOpMd, ExOpMd, AnOpMd)$. The initialization and validation of $InOpMd$ prerequisites are specified in the model initialization section, and the execution of the $ExOpMd$ operation and the analysis of the $AnOpMd$ results are specified in the body of the operation model. During initialization, all the facts necessary for the operation are determined. They analyse the sufficiency of the facts and the necessary prerequisites. For example, in the case of downloading a file from a remote FTP server, check the availability of this server in the network, and the availability of all the data necessary to establish a connection. At the same time, the FTP service model can be used. If the prerequisites of the operation are not fulfilled, a message about the detected error is entered in the test protocol and the test is stopped or postponed, depending on the detected error. The execution of the operation consists of contacting an external service through the model of this service, or in the execution of another model that processes data in the information base, for example, classifies the state of the testing process. As a result of the operation, there is a change in the testing environment [35]. At the stage of analysing the results of the operation, changes in the testing environment [35] are analysed, for example, the presence of downloaded files, installed programs, and success (or failure) in the operation. To obtain the data necessary for analysis, it is often necessary to contact external services again with requests to perform operations and display their results in the test protocol of the information base.

4.3.3. Essences and models of the automated testing system

Let's consider in more detail the models and entities of the ontology used in the testing system, their interdependence and interaction. Basic information about testing is stored in the *AutomatedTesting* entity instance as attributes or references to other entities. Such information includes a reference to the entity, (server, role) of the test server, and sources of installers [35]. There is also a link to the general algorithmic model of automated testing.

The general algorithmic model of testing is presented as a set of problems that must be solved in the process of automated testing. It contains links to general models of relevant operations that reflect the process of solving each problem. Such operations, for example, will be *GetInstaller*, *InstallProduct*, *Test*, and *UninstallProduct*. Each general model, in addition to the specification of entities and operations, contains the definition of methods for checking the success of the task [35]. For example, after the *GetInstaller* task ends, the model provides a check for the presence of the installer file in the specified directory and, if it is, determines the result of the model execution as successful. Otherwise, the model execution result is defined as unsuccessful with an additional specification of the reasons for the error. In addition to general operation models, the general algorithmic model contains references to the next operation determination model and error handling models. The model for determining the next operation, depending on the result of the previous operation, determines the next or one of the error handling models. At the same time, the classification of errors adopted in the system can be used. Let's take a closer look at the *GetInstaller* operation model. The purpose of this pattern is to retrieve the installer in the specified directory. The execution of the model consists of checking the availability and copying the installer file from the remote server to the local test server using a certain file copying service. The general model of such an operation (Fig. 7) contains entities such as *RemotePlacement*, *LocalPlacement*, and *CopyService*.

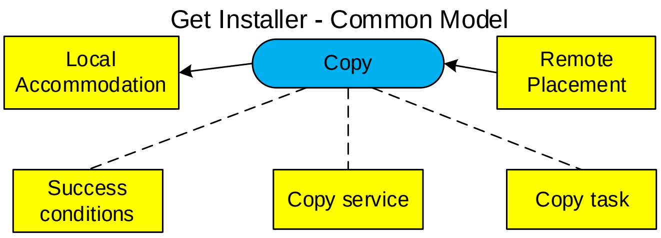


Figure 7: The general model of the GetInstaller task

The use of such a general model, which presents the task in the most general form, gives the system the necessary flexibility. With the use of such a general model, the task can be solved in various ways, simply by defining another specification of the general model. For example, this general model corresponds to the operation of downloading the installer using FTP, and http protocols, from VSS or another local network computer. To change the method of solving the given problem, it is enough to simply replace the detailed model [35].

Let's consider as a specification of the general *GetInstaller* model the model of getting a file using the FTP protocol - the *GetInstallerThroughFtp* model (Fig. 8). The *RemoteHosting* entity in this model corresponds to the *FTPServer* entity, and the *LocalHosting* entity to *TestServer*. The attributes of the *ServerFtp* entity are the network address (URL) of the server and authentication parameters. Similarly, for the test server, its URL is defined. Working directories are defined for both entities. The *CopyService* entity corresponds to the *FtpService* entity. Restrictions are associated with this entity that reflects the specifics of using this service for copying files [35]. In the *GetInstallerByFtp* model, additional prerequisites for the operation are defined (for example, network availability of the FTP server and the test server) and additional procedures for checking the success of the operation. So, in addition to checking the presence of the file on the test server, the hash sum of the received file is additionally checked.

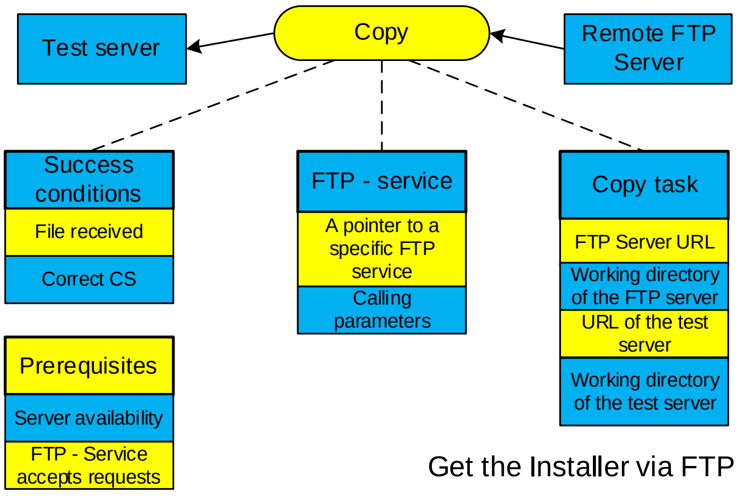


Figure 8: Scheme of the model GetInstallerViaFTP

The *FtpService* entity contains a reference to the FTP service model – *FtpModel*. This model reflects the current state of a specific FTP service, and the commands it accepts, and contains a model of the functioning of this service, which is used to organize interaction and make decisions [35]. The model of the functioning of the FTP service will be defined by a finite state machine with the definition of states and transitions between them. Based on the knowledge reflected in this model, the user of the service can decide to wait for the release of the service if this service is currently downloading, decide to establish a connection if it is not established, or, conversely, use an already established connection.

4.3.4. Presentation of the algorithmic model in the XML language

Algorithmic models, models of operations and services are created using the "Model Editor" tool and saved in XML format. Let's consider examples of presentation of the algorithmic model of automated testing and the model of the operation of obtaining the installer via the FTP service. For the sake of simplicity and clarity, we will skip sections [35] that are not important for illustrating the working principles of the model.

```
<Model>
		<ModelMetadata>
				<GeneralInfo>
						<ModelId> id </ModelId>
						<ModelType> AlgorithmicModel </ModelType>
						<ModelName>OvernightAutomatedTestingModel</ModelName>
						<OntologyURI>
www.acme.org/AutomatedTestingOntology</OntologyURI>
		<ModelRepositoryURI>www.acme.org/ModelRepository</ModelRepositoryURI>
				</GeneralInfo>
				<ActivationInfo>
						<Condition>
								<ConditionBd> Was not active during</ConditionBd>
								<ConditionParameter>15 min<ConditionParameter>
						</Condition>
						<StartState>InstallerChecking</StartState>
				</ActivationInfo>
		</ModelMetadata>
		<ModelBody>
				<Operations>
						<Operation>
						<OperationName>CheckInstallerAvailabilty<OperationName>
								<OperationModel>ModelId1<OperationModel>
						</Operation>
						<Operation>
								<OperationName>GetInstaller<OperationName>
								<OperationModel>ModelI2d<OperationModel>
						</Operation>
						<Operation>
								<OperationName>Install<OperationName>
								<OperationModel>ModelId3<OperationModel>
						</Operation>
						<Operation>
								<OperationName>Test<OperationName>
								<OperationModel>ModelId4<OperationModel>
						</Operation>
						<Operation>
								<OperationName>UnInstall<OperationName>
								<OperationModel>ModelId5<OperationModel>
						</Operation>
						<Operation>
								<OperationName>FinishAndClean<OperationName>
```

```
                        <OperationModel>ModelId5<OperationModel>
                </Operation>
                <Operation>
                        <OperationName>InformByEmail<OperationName>
                        <OperationModel>ModelId6<OperationModel>
                </Operation>
        </Operations>
        <ProcessFlow>
                <Signature>
                        <Condition>
                <IB_Entity Type="Test_Status">ReadyForTesting<IB_Entity>
                        </Condition>
                        <Execute>
<SetIB_InstanceValue Type="Test_Status">
CheckingInstallerAvailability<SetIB_InstanceValue>
                <ExecuteModel>CheckInstallerAvailabilty<ExecuteModel>
                        </Execute>
                </Signature>                    ……
        </ProcessFlow>
    </ModelBody></Model>
```

A model consists of metadata sections and a model body. The metadata section contains information about the ontology, model identifier and name, and model repository address. The model activation information is used by the simulation service – the Model Launch Manager – and determines that the model is automatically activated 15 minutes after the previous activation ends [35]. In the body of the model, the operations performed in the algorithmic model are specified and the execution logic model is defined. At the beginning, they check the availability of the installer for download. If the installer is not ready, the model is deactivated. Otherwise, the testing process is started, in particular the next operation - obtaining the installer. Before deactivating the model, the test environment cleaning operation is performed. The model of receiving the installer via the FTP service looks like this [35]:

```
<Model>
    <ModelMetadata>
        <GeneralInfo>
                <ModelId> id </ModelId>
                <ModelType> OperationModel </ModelType>
                <ModelName>GetInstallerFromFtp</ModelName>
                <OntologyURI>
www.acme.org/AutomatedTestingOntology</OntologyURI>
    <ModelRepositoryURI>www.acme.org/ModelRepository</ModelRepositoryURI>
                <GenericModel>GetInstaller</GenericModel>
        </GeneralInfo>
        <ActivationInfo>
                <Condition>
                        <ConditionBd> Server accessible</ConditionBd>
                <ConditionParameter Name= "ServerAddress">Server Ip address<ConditionParameter>
        <ConditionParameter Reference= "ServerAccessibilityCheckModel">Model Id<ConditionParameter>
                </Condition>
                <Condition>
                        <ConditionBd> Service available</ConditionBd>
                        <ConditionParameter Name= "Service name">FtpService<ConditionParameter>
                        <ConditionParameter    Reference=    "Service    model">FtpServiceModel
Id<ConditionParameter>
                </Condition>
        </ActivationInfo>
    </ModelMetadata>
    <ModelBody>
        <Entities>
                <Entity>
```

```
                    <EntityName>RemoteFtpServer</EntityName>
                    <EntityType>FtpServer</EntityType>
               <GenericEntityRef>RemoteLocation</GenericEntityRef>
                    <DefineParametersFromEntity Name="InstallSource">
                         ……        </DefineParametersFromEntity>
               </Entity>
               <Entity>
                    <EntityName>LocalTestServer</EntityName>
                    <EntityType>TestServer</EntityType>
               <GenericEntityRef>LocalLocation</GenericEntityRef>
               </Entity>
          </Entities>
          <Relations>
               <RelationName>GetFromFtp</RelationName>
               <RelationType>GenericRelation</RelationType>
               <GenericRelationRef>CopyFile</GenericRelationRef>
               <OperationRef>CopyFtpOperation</OperationRef>
          </Relations>
          <Operations>
           <Operation Name ="CopyFtpOperation">
             <ServiceRef>FtpServiceModelId</ServiceRef>
             <Command>GetFile</Command>
           </Operation>
          <Operations>
          <SuccessConditions>
           <Condition>
             <ConditionType>FileExists</Condition>
             <DefineParametersFromEntity Name="LocalTestServer">
                  <IpAddr>ipaddr<IpAddr>
                  <AccessCredentials>Credentials</AccessCredentials>
                  <FilePath>FilePath<FilePath>
             </DefineParametersFromEntity>
           </Condition>
          </SuccessConditions>
          <IB_Update>…</IB_Update>
     </ModelBody>
</Model>
```

The prerequisites for its activation are specified in the metadata of the model - checking the network availability of the server and the readiness of the FTP service. In the body of the model, separate sections define entities, relations, operations, success conditions of execution and determination of the next operation. Some attributes of entities contain references to attributes of other entities. Each entity and relationship contain a reference to the corresponding elements of the general model. The execution of the operation model consists of the execution of the operation specified in the operations section. After the operation, the success conditions defined in the relevant section of the model are checked. The operation is considered successful if all success conditions are met. In the last section of the body, the model updates the testing protocol in the information base [35].

5. Conclusions

Algorithmic models are created in the "Model Editor" software tool [35]. Ready models in XML format are stored in the model repository of the modelling system. In an automated testing system, the launch order of the models is monitored by the Launch Manager, which periodically checks the launch conditions and, if the conditions are met, activates the model. The activated algorithmic model is interpreted by a component of the modelling system – the Model Interpreter. The implementation of such an interpreter is a fairly simple task, it performs parsing of the XML file of the model, checks the truth of the situation signatures and initializes the activation of the corresponding operation models. The interpreter of the operation model checks the prerequisites for the execution of the operation and, if they are met, initiates the execution of the operation. After an operation is executed, the execution logic model checks the status of the test process and, depending on it determines and activates the next operation. From the description of the testing system, it is clear that the overall goal of the system is achieved through the interaction of many models. So, the algorithmic model uses the situational model to define the execution logic. The situation model, in turn, activates the operation models. Each model in the process of execution displays the results of the execution in the information base in such a way that the status of the information base reflects the status of the testing process. Using a situational model to specify test logic allows you to use the general state of the subject area, not just the objects represented in the model, to test conditions. Compared to alternatives, for example, finite-automatic representation of execution logic, this method provides flexibility and ease of expanding the list of situations.

Using the described approach, an automated system for testing software products was created, tested and successfully functions [35]. The VBScript scripting language was chosen for the IS implementation. As a software tool for automated testing, HP QuickTest Professional is used, which also uses VBScript as a programming language. Automated testing IS periodically checking for a new installer file on the FTP site of the product developer. If it is available, the installer file is downloaded and unzipped to the specified test directory. After that, the installation process starts in automatic mode, which does not require user input. After its completion, the IS checks the success of the installation and configures and restarts some system services. If the product is successfully installed, the automated testing tool is launched to run the selected tests. The results of the testing are recorded in the protocol, which is forwarded by e-mail to the developer's representatives after the testing is completed. Employees of the quality control department are also informed by e-mail in the event of a failure of the testing process. After testing is completed, the tested software product is uninstalled, the test directory is cleaned, and the environment is prepared for new testing. The developed IS made it possible to systematically retest several software products of considerable size (installer file size 500-700 Mbytes) in one night. Operation of the testing system proved its high reliability, flexibility, ease of modification and development. The use of algorithmic models for automating the testing of software products allows you to create IS testing that can quickly adapt to changes in the functionality of the tested product, as well as take into account and adequately react to changes in the hardware and software test environment.

References

[1] M. Alexandrou, Fundamentals Unified Process (RUP) Methodology, Risk Management, 2009, pp. 1-3.

[2] UML specification. URL: http://www.omg.org/spec/UML.

[3] O. Medelyan, C. Legg, Integrating Cyc and Wikipedia: Folksonomy meets rigorously defined common-sense, WIKIAI Wikipedia and AI Workshop 8 (2008) 13-18.

[4] Y. Burov, Knowledge Based Situation Awareness Process Based on Ontologies, CEUR Workshop Proceedings 2870 (2021) 413-423.

[5] O. Oborska, V. Andrunyk, L. Chyrun, R. Hasko, A. Vysotskyi, S. Mushasta, O. Petruchenko, I. Shakleina, The Intelligent System Development for Psychological Analysis of the Person's Condition, CEUR Workshop Proceedings 2870 (2021) 1390-1419.

[6] V. Lytvyn, A. Dmytriv, A. Berko, V. Alieksieiev, T. Basyuk, J. Rainer Noennig, D. Peleshko, T. Rak, V. Voloshyn. Conceptual model of information system for drone monitoring of trees' condition, CEUR Workshop Proceedings, 2604 (2020) 695–714.

[7] V. Hryhorovych, Analysis of Scientific Texts by Semantic Inverse-Additive Metrics for Ontology Concepts, CEUR Workshop Proceedings 3171 (2022) 801-816.

[8] A. Gangemi, C. Catenacci, M. Ciaramita, J. Lehmann, A theoretical framework for ontology evaluation and validation, Semantic Web Applications and Perspectives 166 (2005) 16.

[9] D. Vrandečić, Y. Sure, How to design better ontology metrics, in: Proceedings of the Semantic Web: Research and Applications: 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007. Proceedings 4, pp. 311-325. Springer Berlin Heidelberg.

[10] The Protégé Ontology Editor and Knowledge Acquisition System, URL: http://protege.stanford.edu.

[11] TopBraid Composer, URL: http://www.topquadrant.com/products/TB_Composer.html.

[12] A. Das, W. Wu, D. L. McGuinness, Industrial Strength Ontology Management, The Emerging Semantic Web 75 (2001).

[13] M. Lanzenberger, J. Sampson, M. Rester, Visualization in Ontology Tools, in: Proceedings of the International Conference on Complex Intelligent and Software Intensive Systems, 2009, pp. 705-711.

[14] J. Huang, M. S. Fox, Dynamic knowledge provenance, in: Proceedings of Business Agents and Semantic Web Workshop, 2004, pp. 11-20.

[15] S. Ram, J. Liu, Understanding the semantics of data provenance to support active conceptual modeling, Active Conceptual Modeling of Learning: Next Generation Learning-Base System Development 1 (2007) 17-29.

[16] Y. Burov, K. Mykich, I. Karpov, Intelligent systems based on ontology representation transformations, in: Proceedings of the Conference on Computer Science and Information Technologies, 2020, September, pp. 263-275. Cham: Springer International Publishing.

[17] V. Lopez, V. Uren, E. Motta, M. Pasin, AquaLog: An ontology-driven question answering system for organizational semantic intranets, Journal of Web Semantics 5(2) (2007) 72-105.

[18] Y. Burov, V. Vysotska, V. Lytvyn, L. Chyrun, Software Based on Ontological Tasks Models, in: Proceedings of the International Scientific Conference "Intellectual Systems of Decision Making and Problem of Computational Intelligence", 2022, May,pp. 608-638). Cham: Springer International Publishing.

[19] V. Lytvyn, V. Vysotska, D. Dosyn, Y. Burov, Method for ontology content and structure optimization, provided by a weighted conceptual graph, Webology 15(2) (2018) 66-85.

[20] E. Kaufmann, A. Bernstein, R. Z. Querix, A Natural Language Interface to Query Ontologies Based on Clarification Dialogs, in: Proceedings of the 5th International Semantic Web Conference (ISWC 2006), Athens, GA, November, 2006, pp. 980–981.
[21] V. Lytvyn, V. Vysotska, D. Dosyn, O. Lozynska, O. Oborska, Methods of Building Intelligent Decision Support Systems Based on Adaptive Ontology, in: Proceedings of the 2nd International Conference on Data Stream Mining and Processing, DSMP, 2018, pp. 145-150. doi: 10.1109/DSMP.2018.8478500.
[22] M. Barborak, A. Dahbura, M. Malek, The consensus problem in fault-tolerant computing, ACM Computing Surveys (CSur) 25(2) (1993) 171-220.
[23] A. D. Friedman, L. Simoncini, System-level fault diagnosis, Computer 13(03) (1980) 47-53.
[24] F. P. Preparata, G. Metze, R. T. Chien, On the Connection Assignement Problem of diagnosible systems, IEEE transactions on electronic computers 6 (1967) 848-854.
[25] L. Lamport, R. Shostak, M. Pease, The Byzantine generals problem, ACM Books: Concurrency: the works of leslie lamport (2019) 203-226. doi: 10.1145/3335772.3335936 10.1145/3335772.3335936.
[26] P. Kravets, V. Vysotska, V. Lytvyn, L. Chyrun, Adaptive decision-making strategies in the game with environment, Lecture Notes on Data Engineering and Communications Technologies 149 (2023) 286–301.
[27] M. Pease, R. Shostak, L. Lamport, Reaching agreement in the presence of faults, Journal of the ACM (JACM) 27(2) (1980) 228-234.
[28] P. Kravets, Y. Burov, V. Lytvyn, V. Vysotska, Gaming Method of Ontology Clusterization, Webology 16(1) (2019) 55-76.
[29] Y. Burov, The Introduction of Attentional Mechanism in the Situational Awareness Process, CEUR Workshop Proceedings 3171 (2022) 1076-1086.
[30] P. Kravets, V. Lytvyn, V. Vysotska, Y. Burov, I. Andrusyak, Game Task of Ontological Project Coverage, CEUR Workshop Proceedings 2851 (2021) 344-355.
[31] E. Bertino, B. Catania, G.P. Zarri, Intelligent Database Systems, Addison-Wesley, 2001.
[32] K. Mykich, Y. Burov, Algebraic model for knowledge representation in situational awareness systems, in: Proceedings of the Computer Sciences and Information Technologies Conference, CSIT, 2016, pp. 165-167.
[33] Y. Burov, K. Mykich, I. Karpov, Building a versatile knowledge-based system based on reasoning services and ontology representation transformations, in: Proceedings of the IEEE 15th International Conference on Computer Sciences and Information Technologies (CSIT), vol. 2, 2020, September, pp. 255-260.
[34] Y. Burov, K. Mykich, The approach of granular computing and rough sets for identifying situations, Econtechmod: An International Quarterly Journal on Economics of Technology and Modelling Processes 6(2) (2017) 45-50.
[35] E. Burov, Complex ontology management using task models, International Journal of Knowledge-Based and Intelligent Engineering Systems 18(2) (2014) 111-120.
[36] K. Mykich, Y. Burov, Algebraic framework for knowledge processing in systems with situational awareness, Advances in Intelligent Systems and Computing 512 (2017) 217-227.

[37] Software Bugs Cost U.S. Economy $59.6 Billion Annually, RTI Study Finds, URL: http://www.nist.gov/director/prog-ofc/report02-3.pdf.
[38] T. Badgett, G. J. Myers, The Art of Software Testing. Wiley-Blackwell, 2023.
[39] W. E. Lewis, G. Veerapillai, Software Testing and Continuous Quality Improvement, Auerbach publications, 2004.
[40] E. Dustin, J. Rashka, J. Paul, Automated software testing: introduction, management, and performance, Addison-Wesley Professional, 1999.
[41] Kelly M Choosing test automation framework. URL: http://www.ibm.com/developerworks/rational/library /591.html.
[42] P. Baker, Z. R. Dai, J. Grabowski, I. Schieferdecker, C. Williams, Model-driven testing: Using the UML testing profile, Springer Science & Business Media, 2007.
[43] V. Vysotska, V. Lytvyn, Y. Burov, P. Berezin, M. Emmerich, V. B. Fernandes, Development of Information System for Textual Content Categorizing Based on Ontology, CEUR Workshop Proceedings 2362 (2019) 53-70.
[44] Y. Burov, I. Karpov, Contextual Concept Meaning Alignment Based on Prototype Theory, CEUR Workshop Proceedings 3403 (2023) 137-146.
[45] V. Lytvyn, Y. Burov, V. Vysotska, Y. Pukach, O. Tereshchuk, I. Shakleina, Abstracting Text Content Based on Weighing the TF-IDF Measure by the Subject Area Ontology, in: Proceedings of the IEEE International Conference on Smart Information Systems and Technologies (SIST) 2021, pp. 1-7.
[46] V. Lytvyn, D. Dosyn, V. Vysotska, A. Hryhorovych, Method of ontology use in OODA, in: Proceedings of the IEEE 3rd International Conference on Data Stream Mining and Processing, DSMP, 2020, pp. 409-413. doi: 10.1109/DSMP47368.2020.9204107.
[47] Y. Burov, V. Lytvyn, V. Vysotska, I. Shakleina, The Basic Ontology Development Process Automation Based on Text Resources Analysis, in: Proceedings of the IEEE 15th International Scientific and Technical Conference on Computer Sciences and Information Technologies, CSIT, 2020, 1, pp. 280-284. Gg\doi: 10.1109/CSIT49958.2020.9321910.
[48] V. Lytvyn, V. Vysotska, Y. Burov, V. Hryhorovych, Knowledge novelty assessment during the automatic development of ontologies, in: Proceedings of the IEEE 3rd International Conference on Data Stream Mining and Processing, DSMP, 2020, pp. 372-377. doi: 10.1109/DSMP47368.2020.9204124.
[49] O. Pashchetnyk, V. Lytvyn, V. Zhyvchuk, L. Polishchuk, V. Vysotska, Z. Rybchak, Y. Pukach, The ontological decision support system composition and structure determination for commanders of land forces formations and units in Ukrainian armed forces, CEUR Workshop Proceedings 2870 (2021) 1077-1086.
[50] V. Vysotska, V. Lytvyn, M. Bublyk, A. Demchuk, L. Demkiv, Y. Shpak, Method of ontology quality assessment for knowledge base in intellectual systems based on ISO\IEC 25012, in: Proceedings of the IEEE 15th International Scientific and Technical Conference on Computer Sciences and Information Technologies, CSIT, 2020, 1, pp. 109-113. doi: 10.1109/CSIT49958.2020.9321871.
[51] V. Lytvyn, D. Dosyn, V. Vysotska, A. Demchuk, L. Demkiv, I. Lytvyn, Intellectual agent construction method based on the subject field ontology, in: Proceedings of the IEEE 15th International Scientific and Technical Conference on Computer Sciences and

Information Technologies, CSIT, 2020, 1, pp. 40-46. DOI: 10.1109/CSIT49958.2020.9322032.

[52] V. Lytvyn, M. Bublyk, V. Vysotska, V. Panasyuk, O. Brodyak, M. Luchkevych, Modelling of the Intelligent Agent's Behavior Scheduler Based on Petri Nets and Ontological Approach, in: Proceedings of the IEEE International Conference on Smart Information Systems and Technologies (SIST), Nur-Sultan, Kazakhstan, 2021. https://ieeexplore.ieee.org/document/9465994.

[53] A. Berko, I. Pelekh, L. Chyrun, M. Bublyk, I. Bobyk, Y. Matseliukh, L. Chyrun, Application of ontologies and meta-models for dynamic integration of weakly structured data, in: Proceedings of the IEEE 3rd International Conference on Data Stream Mining and Processing, DSMP 2020, 2020, pp. 432–437. doi: 10.1109/DSMP47368.2020.9204321.

[54] V. Lytvyn, V. Vysotska, Y. Burov, O. Brodyak, Approach to Automatic Construction of Interpretation Functions during Ontology Learning, in: Proceedings of the IEEE 15th International Scientific and Technical Conference on Computer Sciences and Information Technologies, CSIT, 2020, 1, pp. 267-271. doi: 10.1109/CSIT49958.2020.9321920.

[55] O. Oborska, M. Teliatynskyi, D. Dosyn, V. Lytvyn, S. Kostenko, An Intelligent System Based on Ontologies for Determining the Similarity of User Preferences, CEUR Workshop Proceedings 3403 (2023) 283-292.

[56] V. Hryhorovych, Calculation of the Semantic Distance between Ontology Concepts: Taking into Account Critical Nodes, CEUR Workshop Proceedings 3396 (2023) 206-216.

[57] V. Hryhorovych, Construction of Semantic Metric for Measuring the Distance between Ontology Concepts, CEUR Workshop Proceedings 2870 (2021) 498-510.

[58] T. Batiuk, L. Chyrun, O. Oborska, Ontology Model and Ontological Graph for Development of Decision Support System of Personal Socialization by Common Relevant Interests, CEUR Workshop Proceedings 3171 (2022) 877-903.

[59] T. Basyuk, A. Vasyliuk, Approach to a subject area ontology visualization system creating, CEUR Workshop Proceedings 2870 (2021) 528–540.

[60] D. Dosyn, Y. Ibrahim Daradkeh, V. Kovalevych, M. Luchkevych, Y. Kis, Domain Ontology Learning using Link Grammar Parser and WordNet, CEUR Workshop Proceedings 3312 (2022) 14-36.

[61] N. Khairova, A. Kolesnyk, O. Mamyrbayev, G. Ybytayeva, Y. Lytvynenko, Automatic Multilingual Ontology Generation Based on Texts Focused on Criminal Topic, CEUR Workshop Proceedings 2870 (2021) 108-117.

[62] T. Basyuk, A. Vasyliuk, Approach to a Subject Area Ontology Visualization System Creating, CEUR Workshop Proceedings 2870 (2021) 528-540.

[63] V. Shynkarenko, L. Zhuchyi, Ontological Harmonization of Railway Transport Information Systems, CEUR Workshop Proceedings 2870 (2021) 541-554.

[64] N. Kunanets, H. Matsiuk, Use of the Smart City Ontology for Relevant Information Retrieval, CEUR Workshop Proceedings 2362 (2019) 322-333.

[65] J. Chen, D. Dosyn, V. Lytvyn, A. Sachenko, Smart data integration by goal driven ontology learning, Advances in Intelligent Systems and Computing 529 (2017) 283-292.

[66] M. Davydov, O. Lozynska, Mathematical method of translation into Ukrainian sign language based on ontologies, Advances in Intelligent Systems and Computing 871 (2018) 89-100.
[67] O.H. Lypak, V. Lytvyn, O. Lozynska, R. Vovnyanka, Y. Bolyubash, A. Rzheuskyi, D. Dosyn, Formation of Efficient Pipeline Operation Procedures Based on Ontological Approach, Advances in Intelligent Systems and Computing 871 (2019) 571-581.
[68] V. Lytvyn, The similarity metric of scientific papers summaries on the basis of adaptive ontologies, in: Proceedings of the 7th International Conference on Perspective Technologies and Methods in MEMS Design, MEMSTECH, 2011, p. 162.
[69] V. Pasichnyk, et al., Ontological approach in the formation of effective pipeline operation procedures, in: Proceedings of the 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies, CSIT, 2018, pp. 80-83.
[70] V.V. Lytvyn, An approach to intelligent agent construction for determining the group of bank risk basing on ontology, Actual Problems of Economics (7) (2011) 314-320.
[71] N. Schahovs'ka, Y. Syerov, Web-community ontological representation using intelligent dataspace analyzing agent, in: Proceedings of the Experience of Designing and Application of CAD Systems in Microelectronics, CADSM, 2009, 479-480.
[72] O. Veres, O. Oborska, A. Vasyliuk, Y. Brezmen, I. Rishnyak, Problems and peculiarities of the IT project managment of ontological engineering for person psychological state diagnosing, CEUR Workshop Proceedings 2565 (2020) 162–177.