Software platform for next-generation intelligent computer systems

Nikita Zotov
Belarusian State University of Informatics and Radioelectronics
Minsk, Belarus
Email: nikita.zotov.belarus@gmail.com

***Abstract*—This paper describes the methodology of designing semantically compatible computer systems and ensuring their independence from the implementation of platforms for designing such systems. The article demonstrates the significance of designing and implementing next-generation platforms, and also proposes the solution to the problem in the form of designing and developing universal interpreters of logical-semantic models of systems according to the principles of the OSTIS Technology. This article is also a formal specification of the first software implementation of the ostis-platform.**

***Keywords*—computer aided design (CAD), ontological design, automation tools for the design and development of computer systems, graph database, graph knowledge base, database management system (DBMS), knowledge base management system, universal interpreter, graph storage, ostis-platform**

I. Introduction

The main result of research in the field of design and development of computer systems (c.s.) is not so much the existing c.s., but the development of technologies and tools based on the principles of these technologies that allow to quickly and in large quantities generate a wide variety of c.s. [1] that have great practical value. Right now, there are a wide variety of solutions in the field of automated design and development of c.s., which allow solving problems of a fairly serious level of complexity [2]. However, none of these systems are able to provide platform independence, and hence the semantic compatibility and interoperability of the created computer systems. The urgency of the problem is explained by the need to create next-generation computer systems capable of solving problems of any kind quickly and adequately [3]. To achieve this, it is necessary to design the *Standard of design and developing of c.s.* and implement such tools according to this standard [4], [5], [6].

II. Problems of the current state of computer systems and platforms for their implementation

Modern computer systems, as well as automation tools for the design and development of such systems, have a number of significant disadvantages:

1) Computer systems to a great extent remain dependent on the implementation of specific platforms on which they are designed, which, in turn, leads to significant costs for integrating the methods and tools for system design in case of transition to new platforms.
2) The design and development of the implementation of a particular system is carried out using different methods and models for designing software c.s. Thus, the description of the target state of the system and the description of the current implementation may not correspond to each other, and the integration of such solutions is difficult to achieve. This problem is well described when designing a platform for practical applications [7].
3) Specification of software systems is relegated to the background, and sometimes it is not done at all by the development project of a specific c.s. Consequently, the costs of maintaining the process of permanent re-engineering of such systems increase [8].
4) When developing modern systems, there is no understanding of the need to develop and describe the design methods for these systems, including descriptions of the implementation process, directions of use, etc. For this reason, developers of modern software c.s. do not use already existing accumulated experience but re-invent the same or similar solutions [8], [9].
5) There are no unified universal tools for the development [10] and re-engineering of other systems that allow not only to automate their design, but also to minimize the development process itself by way of unifying the representation models of these systems and having a semantically powerful complex library of reusable components.
6) Even highly specialized software c.s. must have a good level of intelligence and a good level of trainability to solve more complex problems. Next-generation software c.s., unlike modern c.s., must operate on the meaning of what they know and process: they must understand each other [11]. [12], find common ground and form teams to solve problems of any class [13], [14].

7) Modern computers are poorly adapted for effective implementation of even existing models of knowledge representation and models for solving problems that are hard to formalize, which requires the development of fundamentally new platforms and computers that ensure the unification of the representation of this knowledge (!) [15], [16].

Typically, systems of this kind are designed and developed to solve narrow applied problems. As a result, systems with the problems described above are created. So, most of the described problems, unfortunately, were not solved when creating CADs described in the following papers [17], [18], [19], [20], [21], [22].

When designing next-generation c.s., first of all, it is necessary to take into account the shortcomings of modern c.s. This means that the design of next-generation c.s. should be reduced to solving the problems that exist in modern c.s. Thus, the following possible solutions can be identified:

1) Implementation of next-generation c.s. should not be inferior to the implementation of modern c.s. Such a task is reduced to the choice of means for storing, representing and processing data in these systems. But still, when speaking of next-generation c.s., it is necessary to lean towards a higher form of data – knowledge [14]. They must be unified in their representation, semantically integral, connected, unambiguous, etc. Thus, next-generation c.s. should be organized in such a way that the form of representation of different types of knowledge is the same (!). And this, in turn, means that the design and development tools for other systems should be organized in such a way that these systems can be easily integrated with each other, striving to increase their degree of convergence [6].
2) Since modern c.s. are platform-dependent, which in turn complicates the development of such systems, it is necessary to create such tools that allow you to create c.s. independently (!) of the implementation of these tools. At the same time, this should be done in such a way that the process of designing, developing, documenting and using such systems is carried out using the same tools and methods [6], [23], which are part of these tools, and in such a way that the quality of such systems is determined by the degree of their deep integration with each other. This can be solved with the help of general ontologies for the development of such systems, that is, with the help of an ontological [24] and component [25] approaches (!) not only to their design, but also to their implementation.

Specific solutions will be discussed and described below. This work develops the ideas and solves the problems described in the previous work [26].

III. Approaches to designing automation systems used in c.s. design and implementation

Implementation of data storages used in the vast majority of c.s. is based on the relational data model. Examples of such systems for processing unstructured and semi-structured data are the SMILA platform (SeMantic Information Logistic Architecture), Teradata Aster Discovery Platform, which implement relational, columnar and hybrid models for storing records in a database with a massively parallel MPP architecture [27], CYC platform [28], Semantic Web tools [29].

However, information systems are currently undergoing intensive intellectualization. First of all, this is due to an increase in the level of complexity of the tasks being solved. The intellectualization of information systems, like any technology for developing software systems, requires taking into account weakly formalized, possibly not completely defined, fuzzy, temporal, spatially distributed information and, as a result, obtaining structured, semi-structured and unstructured data [30]. The increase in the number of intellectual tasks of processing large amounts of data in all spheres of human activity leads to the need to create universal means of storing, presenting and processing multistructured information.

The presence of such tasks stimulates the transition from conventional databases to graph counterparts. This is explained not so much by the efficiency of memory organization and data processing in graph databases, but by the importance of representing the configurations of relationships (i.e., meaning) between them [31]. A detailed explanation of the principles of organizing graph data in databases can be found in the work of the authors of the popular Neo4j graph DBMS [32].

For a general understanding of the whole problem associated with the representation and processing of data and knowledge, we will consider several modern implementations of graph data models in the form of software products. Any of the databases described below is designed for convenient storage and access to data presented in the form of super-large graphs. According to the organization of memory and the process of data processing, graph databases can be classified into the following types:

1) databases with local storage and processing of graphs (Neo4j, HyperGraphDB, AllegroGraph);
2) databases with distributed data storage and processing (Horton, InfiniteGraph);
3) databases in "key-value" format (Trinity, CloudGraph, RedisGraph, VertexDB);
4) document-oriented databases (OrientDB);
5) add-ons for SQL-oriented databases (Filament, G-store);
6) graph databases with MapReduce model (Pregel, Apache Giraph, GraphLab).

A detailed description of each of the presented databases can be found in the works devoted to comparing relational and graph databases [33], [34], [35], [36].

The motivation for moving from conventional graph databases is due to the advantages of organizing a memory model and processing in them:

1) Data processing performance improves by one or more orders of magnitude when data is represented as graph structures, which is explained by the properties of the graph itself. Unlike relational databases, where query performance degrades as the dataset grows with increasing query intensity, graph database performance tends to remain relatively constant even as the dataset grows. This is due to the fact that data processing is localized in some part of the graph. As a result, the execution time of each request is only proportional to the size of the part of the graph traversed to satisfy this request, and not to the size of the entire graph [37].
2) Graph structures have tremendous expressive power. Graph databases offer an extremely flexible data model and way of representing [38], [39]. Graph structures are additive, which provides the flexibility to add new data relationships, new nodes, and new subgraphs to an existing structure without violating its integrity and connectivity.

As mentioned earlier, next-generation c.s., by virtue of their properties, must operate not just with data, but with knowledge. In order to understand the meaning of knowledge, it is necessary to present this knowledge in an understandable form for everyone: both for a person and for a machine. Speaking about the unification of the representation of all types of knowledge, it is important to use graph databases not just as a means for storing structured data, but for storing semantically holistic and interconnected knowledge [40]. In the context of designing next-generation c.s., we will talk about knowledge bases designed according to the principles of graph databases.

It should also be noted that the emphasis in this work is on the development of c.s. support for the design of other c.s., and for the development of complex tools to support the automatic design of next-generation intelligent computer systems, which are knowledge-driven. Such tools can be compared with knowledge base management systems [41], [42], [43], [44].

IV. Suggested solution

Despite the vast variety of classical technologies used by mankind, there is no general solution that allows solving the problem in a complex manner. At the moment, the described problems can only be solved with the help of a general and universal solution – the OSTIS Technology. The OSTIS Technology is based on a unified version of information encoding based on semantic networks with a basic set-theoretic interpretation, called SC-code. The language of semantic representation of knowledge is based on two formalisms of discrete mathematics: set theory – determines the semantics of the language – and graph theory – determines the syntax of the language. Any types and models of knowledge can be described using SC-code [6].

The platform for interpreting the semantic models of ostis-systems will simply be called the ostis-platform, which denotes the interpreter of the logical-semantic models of ostis-systems. The logical-semantic model (sc-model) of the osits-system is a formal model (formal description) of the functioning of this osits -system, consisting of (1) a formal model of information stored in the memory of the osits-system and (2) a formal model of a team of agents that process the specified information. The sc-model of the ostis-system includes the sc-memory, the sc-model of the knowledge base, the sc-model of the problem solver, and the sc-model of the interface. Each ostis-system designed using the OSTIS Technology must include a platform for interpreting the semantic models of ostis-systems (in a particular case, sc-memory) and a logical-semantic model of the ostis-system, represented using SC-code (sc-model of the ostis-system) [6], [23].

For the convenience of knowledge representation, there are three external knowledge representation languages based on the SC-code:

1) SCg-code with which knowledge is displayed in the form of graph structures understandable to the average user;
2) SCs-code in which knowledge is represented as a linear text;
3) SCn-code for displaying sc-constructions as hypertext. This representation is close to natural, understandable to the average user [6].

No other classical technology used by the engineers of the modern information society has a clearly defined, strict, formal conceptual system that could be used to solve various types of problems, not to mention the means necessary to solve the tasks set in the direction of increasing efficiency, interoperability, semantic compatibility of systems developed on the basis of these technologies. The OSTIS technology provides all the necessary capabilities and tools for developing next-generation ostis-platforms that provide efficient and high-quality interpretation of logical-semantic models of semantically compatible interoperable ostis-systems [6], [12].

The ostis-platform also means a family of software-based semantic associative computer emulators [6].

Platforms for developing other c.s. (not necessarily intelligent) should provide:

- determinism and uniqueness of interpretation of systems built on the basis of such ostis-platforms,
- availability of common language tools for formal description of designed components at different

levels of detail [45], [46],
- clear separation of the process of developing formal descriptions of components and the process of their implementation according to given formal descriptions [47], [48],
- creation and use of powerful and accessible libraries of formal descriptions of reusable ostis-platforms components,
- quality and a high level of applicability of reusable ostis-platform components.

The need for such properties in implemented ostis-platforms is justified by their purpose. Computer complexes that allow the development of other c.s. must be implemented and described in such a way that any intelligent computer system implemented on it is compatible with another similar system, so that the future interpretation of its logical-semantic model remains correct, unambiguous and independent of the means and solutions by which the ostis-platform is implemented. From this point of view, the implemented ostis-platform is only a means of mass creation of other systems and can be easily replaced by its functionally equivalent analogue that meets all the requirements for platforms [23].

In general, next-generation platforms should provide platform independence (!) of the logical-semantic models of ostis-systems implemented and interpreted on them. Different options for implementing the ostis-platform should not affect the process and result of designing ostis-systems, that is, the process and result of building logical-semantic models of the developed ostis-systems. That is, the designed ostis-systems should not depend on the specific platform for their interpretation. Thus, the platform independence of systems from specific ostis-platforms means the functional completeness of these platforms for creating systems, the simplicity and flexibility of expanding their functionality and the range of tasks to be solved, and, ultimately, the level of high intelligence of computer systems implemented on them.

The implementations of the ostis-platform designed using the OSTIS Technology *Implementations* should be based on the following fundamental principles:

- All texts represented in SC-code are graph constructions. Therefore, the task of developing a *Software implementation of the ostis-platform* is reduced to developing means for storing and processing such graph structures. In other words, the future platform should provide functionally complete and unambiguous interpretation of stored graph constructions.
- Designing a platform for interpreting sc-models of computer systems, including its components, must be clearly specified and formulated in terms of models, methods and tools for describing complex systems offered by the OSTIS Technology. It is the ontological approach to the design, operation and re-engineering of such a subclass of computer systems that will make it possible to effectively and universally develop other ostis-systems for various purposes [49], [50].

One of the ways to test, develop, and, in some cases, introduce new models and technologies, regardless of the availability of appropriate hardware, is the development of software models of this hardware that would be functionally equivalent to the hardware itself, but at the same time interpreted on the basis of traditional hardware architecture (in this paper, we will consider the von Neumann architecture as the dominant traditional architecture now). Obviously, the performance of such software models in the general case will be lower than the hardware solutions themselves, but in most cases it turns out to be sufficient to develop the corresponding technology in parallel with the development of hardware and to gradually transfer existing systems from a software model to hardware.

The popularity and development of graph databases leads to the fact that, at first glance, it seems expedient and effective to implement the *Software implementation of the ostis-platform* based on one of these tools. However, there are a number of reasons why this is not possible. These include the following:

- To ensure the efficiency of storage and processing of information structures of a certain type (in this case, SC-code structures, sc-constructions), the specificity of these structures should be taken into account. In particular, the experiments described in [51] showed a significant increase in the efficiency of their own solution compared to those existing at that time.
- Unlike classical graph constructions, where an arc or an edge can only be incident to a graph vertex (this is also true for rdf-graphs), in SC-code, it is quite typical that an sc-connector is incident to another sc-connector or even two sc-connectors. In this regard, the existing means of storing graph constructions do not allow explicit storage of sc-constructions (sc-graphs). This problem can also be solved by passing from an undirected graph to a digraph [52].
- Information processing within the framework of the OSTIS Technology is based on a multi-agent approach [53], within which agents for processing information stored in sc-memory (sc-agents) respond to events occurring in sc-memory and exchange information by specifying the actions they perform in sc-memory [54]. In this regard, one of the most important tasks is the implementation within the *Software implementation of the ostis-platform* of the possibility of subscribing to events occurring in *Implementation the sc-memory*, which at the moment is practically not supported within modern tools for storing and processing graphs structures.
- SC-code also allows describing external information

structures of any kind (images, text files, audio and video files, etc.), which are formally treated as the contents of *sc-elements*, which are signs of *external files of ostis-systems*. Thus, the *Software implementation of the ostis-platform* component should be a file memory implementation that allows storing the indicated constructions in any generally accepted formats. The implementation of such a component within the framework of modern means of storing and processing graph structures is also not always possible.

Due to the combination of the above reasons, it was decided to implement the *Software implementation of the ostis-platform* "from scratch", taking into account the peculiarities of storing and processing information within the framework of the OSTIS Technology.

V. Current Software implementation of the ostis-platform

The current *Software implementation of the ostis-platform* is web-oriented, so from this point of view, each ostis-system is a website accessible online through a regular browser. This implementation option has an obvious advantage – access to the system is possible from anywhere in the world where the Internet is available, and no specialized software is required to work with the system. On the other hand, this implementation option provides the possibility of several users operating the system in parallel. The implementation is cross-platform and can be built from source on various operating systems. At the same time, the interaction between the client and server parts is organized in such a way that web-interface can be easily replaced with a desktop or mobile interface, both universal and specialized.

Software implementation of the ostis-platform
- ∈ *specialized ostis-platform*
- ∈ *web-based implementation of the ostis-platform*
  - := [implementation of the platform for interpreting sc-models of computer systems that involves the interaction users with the system via the Internet]
- ∈ *multi-user ostis-platform implementation*
- ∈ *non-atomic reusable ostis-systems component*
- ∈ *dependent reusable ostis-systems component*
- ⇒ *software system decomposition**:
  - {• *Implementation of the sc-memory*
  - • *Implementation of the interpreter of user interface sc-models*
  - • *Implementation of a basic set of platform-specific sc-agents and their common components*
  - }
- ⇒ *component dependencies**:
  - {• *Implementation of the sc-memory*
  - • *Implementation of the interpreter of user interface sc-models*
  - }

The core of the platform is *Implementation of the sc-memory*, which can simultaneously interact with both *Implementation of the interpreter of user interface sc-models*, and with any third-party applications using the appropriate networking languages (network protocols). From the point of view of the overall architecture *Implementation of the interpreter of user interface sc-models* acts as one of many possible external components that interact with *Implementation of the sc-memory* over the network. It is worth noting that the current version of the ostis-platform implementation is specialized, that is, it does not include the implementation of the SCP base language interpreter.

VI. General description of Implementation of the sc-memory

Within the framework of the current *Implementation of the sc-memory*, *sc-storage* is understood as a program model component that stores sc-constructions and accesses them through the program interface. In general, *sc-storage* can be implemented in different ways. In addition to *sc-storage* itself, *Implementation of the sc-memory* also includes *Implementation of the file storage*, designed to store the contents of *internal files of ostis-systems*.

Implementation of the sc-memory
- := [Implementation of the sc-machine]
- ⇐ *software model**: *sc-memory*
- ∈ *software model of the sc-memory based on linear memory*
- ∈ *non-atomic reusable ostis-systems component*
- ∈ *dependent reusable ostis-systems component*
- ⇒ *software system decomposition**:
  - {• *Implementation of the sc-storage*
  - • *Implementation of the file storage*
  - • *Implementation of the subsystem of interaction with external environment using networking languages*
  - • *Implementation of auxiliary tools for working with sc-memory*
  - }
- ⇒ *component dependencies**:
  - {• *GLib library of methods and data structures*
  - • *C++ Standard Library for Methods and Data Structures*
  - • *Implementation of sc-storage*
  - • *File storage implementation*
  - }

It should be noted that when switching from *Implementation of the sc-memory* to its hardware implementation, it would be advisable to implement the file memory of the ostis-system based on traditional linear memory (at least at the first stages of *semantic computer* development). The current version of *Implementation of the sc-memory* is open and available at [55].

Within this *Implementation of the sc-storage*, *sc-memory* is modeled as a set of *segments*, each of which is a fixed-size ordered sequence of *sc-storage* elements, each of which corresponds to specific sc-element. Currently, each segment consists of $2^{16} - 1 = 65535$ *sc-storage elements*. Each segment consists of a set of data structures describing specific *sc-elements* (sc-storage elements). Regardless of the type of sc-element being described, each *sc-storage element* has a fixed size (currently 36 bytes), which ensures convenient storage. Thus, the maximum size of the knowledge base in the current sc-memory software model can reach 180 GB (excluding the contents of *internal files of the ostis-system* stored on the external file system).

VII. Implementation of the sc-storage

A. Selected solution and its rationale

Implementation of the sc-storage must meet the following requirements:

- high performance – minimizing the time spent on adding, deleting and accessing stored information;
- minimal memory and disk space for storing sc-texts;
- scalability – the ability to easily add computing power as the load increases.

The sc-storage consists of sc-segments of elements that correspond to some sc-elements of the abstract SC-code. Each segment of the sc-storage has a number relative to the sc-storage itself, and each element of some sc-storage sc-segment has a number relative to that sc-segment.

Allocation of *sc-storage segments* makes it possible, on the one hand, to simplify address access to *sc-storage elements*, and on the other hand, to realize the possibility of unloading a part of sc-memory from RAM to the file system if necessary. In the second case, the sc-storage segment becomes the minimum (atomic) paged part of the sc-memory. The segment unloading mechanism is implemented in accordance with the existing principles of virtual memory organization in modern operating systems.

The maximum possible number of segments is limited by the settings of the software implementation of the sc-storage (currently the number of sc-segments is $2^{16} - 1 = 65535$ by default, but in the general case it may be different). Thus, technically, the maximum number of stored sc-elements in the current implementation is about 4.3×10^9 sc-elements. By default, all segments are physically located in RAM, if there is not enough memory, then a mechanism is provided for unloading some of the sc-segments to the hard disk (virtual memory mechanism).

The current version of the *Implementation of the sc-memory* assumes the possibility of saving the memory state (imprint) to the hard disk and subsequent loading from the previously saved state. This feature is necessary to restart the system in case of possible failures, as well as when working with the source code of the knowledge base, when the assembly from the source code is reduced to the formation of a snapshot of the memory state, which is then placed in the *Implementation of the sc-memory*.

B. General description of the current implementation of sc-storage

Implementation of the sc-storage
∈ *implementation of sc-storage based on linear memory*
∈ *non-atomic reusable ostis-systems component*
∈ *dependent reusable ostis-systems component*
⇐ *software model**:
sc-storage
⇐ *subset family**:
sc-storage segment
:= [sc-storage page]
⇐ *subset family**:
sc-storage element
⇒ *component dependencies**:
{• *GLib library of methods and data structures*
• *C++ Standard Library of Methods and Data Structures*
}
⇒ *used method representation language**:
• *C*
• *C++*
⇒ *internal language**:
• *SCin-code*

Each sc-storage element in the current implementation can be uniquely specified by its address (sc-address), which consists of the sc-segment number and the *sc-storage element* number within the sc-segment. Thus, the *sc-address* serves as the unique coordinates of an *sc-storage element* within the framework of the *Implementation of the sc-storage*.

For each sc-address, it is possible to assign one-to-one correspondence to some hash obtained as a result of applying a special hash function on this sc-address. The hash is a non-negative integer and is the result of converting the number of the sc-storage segment si, in which the sc-element is located, and the number of this sc-element of the sc-storage *ei* within this sc-segment *si*. The sc-storage framework uses a single hash function to get the hash of the sc-address of the sc-element and is specified as $f(si, ei) = si << 16 \vee ei \wedge 0xffff$, where

the operation $<<$ is the operation logical bit shift left of the left argument by the number of units specified by the right argument, relative to of this operation, the $\vee$ operation is a logical *OR* operation, the $\wedge$ operation is a logical *AND* operation, the number $0xffff$ is the number 65535, represented in hexadecimal form and denoting the maximum number of sc-elements in one sc-storage segment.

sc-address
:= [address of the sc-storage element corresponding to the given sc-element within the current implementation of the sc-storage as part of software model of sc-memory]
∈ *32-bit integer*

The sc-address is not taken into account in any way when processing the knowledge base at the semantic level and is only necessary to provide access to the corresponding data structure stored in linear memory at the *Implementation of the sc-storage* level. In general, sc-address of the sc-storage element corresponding to the given sc-element may change, for example, when rebuilding the knowledge base from source texts and then restarting the system. At the same time, the sc-address of the sc-storage element corresponding to the given sc-element cannot change directly during the system operation in the current implementation. For simplicity, we will say "sc-address of the sc-element", meaning *sc-address* of the *sc-storage element* that uniquely corresponds to the given *sc-element*.

The specification of such complex software objects must be represented in some kind of knowledge representation language, in this case SC-code. From the point of view of SC-code itself, the language that should describe the *Software implementation of the ostis-platform* is a sublanguage of SC-code, that is, it inherits all the properties of the syntax and denotational semantics of SC-code, and a metalanguage for describing the representation of the SC-code constructions in the memory of a software emulator of a semantic associative computer. Such a model for presenting the specification of a c.s., which is a platform for the creation, use and development of other c.s., certainly provides strong advantages over other options for presenting c.s. specifications:

1) The language, the texts of which the system stores and processes, and the language of the specification of how the system represents the texts of the first language in the memory of itself, are subsets of the same language. This simplifies not only the understanding of a developer who develops a complex software system, due to the fact that the form of representation of the language processed by this system and the language of its specification is unified, but also allows you to open new functionality for this system in knowing itself. Thus, this approach makes it possible to fully realize the properties of an intelligent system, for example, reflexivity.
2) It is impossible to design and implement intelligent c.s. on a computer system that is not itself intelligent. Presenting the specification of a system in this form makes it possible to increase the level of its intelligence.
3) Since the form of representation of the language describing the system is unified with the language it processes, there is no need to create additional tools for verification and analysis of the system operation.

C. The concept of SCin-code

We will call such a language the language of the internal representation of SC-code, or, briefly, *SCin-code (Semantic Code interior)*. Sc-storage of SC-code texts can be considered as a subset of scin text.

SCin-code
:= [Semantic Code interior]
:= [Language of the internal semantic representation of the SC-code inside the memory of the ostis-system]
:= [meta-language for describing the representation of the SC-code inside the memory of the ostis-system]
⇒ *frequently used non-primary sc-element external identifier**:
[scin-text]
∈ *common noun*
∈ *abstract language*
∈ *metalanguage*
⊂ *SC-code*
⊃ *sc-storage*

SCin-code syntax is given by: (1) *SCin-code alphabet*, (2) one-to-one correspondence *sc-addresses**.

D. SCin-code alphabet

SCin-code alphabet^
:= [syntactic type of sc-storage element]
:= [Set of types of sc-storage elements]
⇐ *alphabet**:
SCin-code
= {• *sc-storage element corresponding to sc-node*
• *sc-storage element corresponding to sc-arc*
• *sc-storage element with null sc-address*
∈ *singleton*
}

*SCin-code alphabet^* consists of three syntactically distinguished types of sc-storage elements: an sc-storage

element corresponding to a general sc-node, an sc-storage element corresponding to a general sc-arc, and an sc-storage element, having a null sc-address. Such an alphabet not only allows you to set in memory the minimum set of objects with which you can perform computational operations, but, if necessary, is convenient for expansion. So, for example, the given alphabet of the language can be extended by adding to it *sc-storage element, corresponding to the internal file of ostis-system* or *sc-storage element, corresponding to sc-edge*.

sc-storage element corresponding to sc-element
∈ *sc-element*
:= [sc-storage element]
:= [sc-storage cell]
:= [sc-element image within sc-storage]
:= [data structure, each instance of which within sc-storage corresponds to one sc-element]
⇒ *subdividing**:
SCin-code alphabet^

The relation *sc-address** is defined as a one-to-one correspondence, the first component of each ordered pair of which is some element of the sc-storage corresponding to some sc-element, and the second component is the sc-address of this element of the sc-storage.

sc-address*
∈ *one-to-one correspondence*
⇒ *first domain**:
sc-storage element correspoinding sc-element
⇒ *second domain**:
16-bit integer

E. Syntax and syntactic rules of SCin-code

Within *Implementation of sc-storage* there must be a set of *syntactic and semantic classes of sc-storage elements* that:

1) define the element type at the platform level and does not have a corresponding sc-arc of membership (more precisely, a base sc-arc) explicitly stored in sc-memory (its presence is implied, but it is not explicitly stored, since it will lead to infinite increasing the number of sc-elements to be stored in sc-memory);
2) can be represented as parameters of the corresponding elements of the sc-storage, that is, a set of such elements, each of which has a "label" expressed by some numerical value;
3) can specify the type of elements of the sc-storage with the level of detail that is necessary so that, for example, when performing a search operation using such element classes, it is easy to determine the class of a particular element.

For this purpose, the basic syntactic classification of its elements is allocated in the SCin-code. In order to represent and store any constructions of the SC-code, it is enough to have only two base classes of sc-storage elements, while the remaining classes of sc-storage elements can be added in the extended version of the SCin-code and thereby implement the necessary logic at the level of sc-memory Implementation .

Syntactic classification of SCin-code elements
⊃=
{

sc-storage element corresponding to sc-element
⇒ *subdividing**:
{• *sc-storage element corresponding to sc-node*
• *sc-storage element corresponding to sc-arc*
}
}

It should be noted that all classes of sc-storage elements that are part of the syntactic classification of SCin-code elements are syntactically distinguished classes of SCin-code elements.

Although the *sc-addresses** relation makes it possible to completely describe the links between the elements of the sc-storage of the ostis-system, but for the specification of the representation of SC-code constructions inside the memory of the ostis-system, only one *sc-address** relation is not always enough to indicate completely exactly and clearly the relationships between the elements of the sc-storage corresponding to the sc-elements of these constructions. Therefore, in practice, when describing the representation of SC-code structures inside the memory of the ostis-system, it is necessary to use more particular relations of this basic relation, for example, such as *sc-address of the sc-storage element corresponding to the outgoing sc-arc from the given sc-element **, *sc-address of the sc-storage element corresponding to the incoming sc-arc in the given sc-element**, *sc-address of the sc-storage element corresponding to the incident sc-element of the sc-arc**.

sc-address*
⇒ *subdivinding**:
{• *sc-address of the sc-storage element corresponding to the outgoing sc-arc from the given sc-element**
• *sc-address of the sc-storage element corresponding to the incoming sc-arc in the given sc-element**
• *sc-address of the sc-storage element*

*corresponding to the incident sc-element of the sc-arc**

}

The *sc-address of the sc-storage element corresponding to the outgoing sc-arc from the given sc-element** is defined as a binary oriented relation, the first component of each oriented pair of which is some element of the sc-storage corresponding to some sc-element from which the given sc-arc comes out, and the second component is the sc-address of this outgoing sc-arc. Particular types of this relation are the relation *sc-address of the sc-storage element corresponding to the initial outgoing sc-arc from the given sc-element**, the relation *sc-address of the sc-storage element corresponding to the next outgoing sc-arc from of the given sc-element** and the relation *sc-address of the sc-storage element corresponding to the previous outgoing sc-arc from the given sc-element**.

sc-address of the sc-storage element corresponding to the outgoing sc-arc from the given sc-element*

⇒ *subdividing**:

{• *sc-address of the sc-storage element corresponding to the initial outgoing sc-arc from the given sc-element**

• *sc-address of the sc-storage element corresponding to the next outgoing sc-arc from the given sc-element**

• *sc-address of the sc-storage element corresponding to the previous outgoing sc-arc from the given sc-element**

}

The relation *sc-address of the sc-storage element corresponding to the incoming sc-arc in the given sc-element** is defined as a binary oriented relation, the first component of each oriented pair of which is some element of the sc-storage corresponding to some sc-element, in which this sc-arc enters, and the second component is the sc-address of this incoming sc-arc. Particular types of this relation are the relation *sc-address of the sc-storage element corresponding to the initial incoming sc-arc in the given sc-element**, the relation *sc-address of the sc-storage element corresponding to the next incoming sc-arc in the given sc-element** and the relation *sc-address of the sc-storage element corresponding to the previous incoming sc-arc in the given sc-element**.

sc-address of the sc-storage element corresponding to the incoming sc-arc in the given sc-element*

⇒ *subdividing**:

{• *sc-address of the sc-storage element corresponding to the initial incoming sc-arc in the given sc-element**

• *sc-address of the sc-storage element corresponding to the next incoming sc-arc in the given sc-element**

• *sc-address of the sc-storage element corresponding to the previous incoming sc-arc in the given sc-element**

}

The relation *sc-address of the sc-storage element corresponding to the incident sc-element of the sc-arc** is defined as a binary oriented relation, the first component of each oriented pair of which is some element of the sc-storage corresponding to some sc-element, which is sc- arc, and the second component is the sc-address of some sc-element incident to it. Particular types of this relation are the relation *sc-address of the sc-storage element corresponding to the initial sc-element of the sc-arc** and the relation *sc-address of the sc-storage element corresponding to the final sc-element of the sc-arc** .

sc-address of the sc-repository element corresponding to the incident sc-element of the sc-arc*

⇒ *subdividing**:

{• *sc-address of the sc-storage element corresponding to the initial sc-element of the sc-arc**

• *sc-address of the sc-storage element corresponding to the final sc-element of the sc-arc**

}

The following restrictions are imposed on the syntactic constructions of the SCin code:

- Each *sc-storage element corresponding to an sc-element*, has a one-to-one relation to its sc-address.
- For each *sc-storage element corresponding to the sc-node*, there is one and only one relation pair *sc-addresses of the sc-storage element corresponding to the initial outgoing sc-arc from the given sc-element** and one and only one relation pair *sc-addresses of the sc-store element corresponding to the initial incoming sc-arc in the given sc-element**.
- For each *sc-storage element corresponding to the outgoing sc-arc from the given sc-element* (*sc-storage element corresponding to the incoming sc-arc to the given sc-element*), there is at most one relation pair *sc-addresses of the sc-storage element corresponding to the next outgoing sc-arc from the given sc-element** (*sc-addresses of the sc-storage element corresponding to the next incoming sc-arc in the given sc-element**) and at most one relation pair *sc-addresses of the sc-storage element corresponding to the previous outgoing sc-arc from the given sc-element** (*sc-addresses of the sc-storage element corresponding to the previous incoming sc-arc to the given sc-element**).
- For each *sc-storage element corresponding to the*

sc-arc that is the second component of each pair of the *sc-address relation of the sc-store element corresponding to the initial outgoing sc-arc from the given sc-element** (*sc-addresses of the sc-storage element corresponding to the initial incoming sc-arc in the given sc-element**) there is only one pair *sc-addresses of the sc-storage element corresponding to the next outgoing sc-arc from the given sc-element** (*sc-addresses of the sc-storage element corresponding to the next incoming sc-arc in the given sc-element**).

F. Denotational semantics of SCin-code

According to the above, for each class of sc-elements of the SC-code, there must be a program model of the class of sc-store elements that satisfies all the listed requirements. Therefore, it is important that *SCin-code Alphabet* is initially complete in order to immerse not only sc-constructions *SC-code Core*, but also its extended version. For this, semantic classes of sc-storage elements have been developed, the specification of which is represented as *Semantic classification of SCin-code elements*.

Semantic classification of SCin-code elements
⊃=
{

sc-storage element corresponding to sc-element
⇒ *subdividing**:
Typology of sc-storage elements based on constantness^
= {• *sc-storage element corresponding to sc-constant*
• *sc-storage element corresponding to sc-variable*
• *sc-storage element corresponding to sc-meta-variable*
}
⇒ *subdividing**:
Typology of sc-storage elements based on permanency^
= {• *sc-storage element corresponding to permanent sc-element*
• *sc-storage element corresponding to temporary sc-element*
}
⇒ *subdividing**:
Typology of sc-storage elements based on accessibility^
:= [sc-storage element access level class]
= {• *sc-storage element corresponding to sc-element on which read access is allowed*
• *sc-storage element corresponding to sc-element on which write access is allowed*
}
⇒ *include**:
sc-storage element corresponding to internal ostis-system file

sc-storage element corresponding to generic sc-node
⇒ *subdividing**:
Structural typology of sc-storage elements corresponding to sc-nodes^
= {• *sc-storage element corresponding to sc-node denoting a non-binary sc-link*
• *sc-storage element corresponding to sc-class*
• *sc-storage element corresponding to sc-node denoting a class of classes*
• *sc-storage element corresponding to sc-structure*
• *sc-storage element corresponding to sc-node denoting the role relation*
• *sc-storage element corresponding to sc-node denoting a non-role relation*
• *sc-storage element corresponding to sc-node denoting the primary entity*
}
⇒ *subdividing**:
Structural typology of sc-storage elements corresponding to sc-arcs^
= {• *sc-storage element corresponding to sc-arc of membership*
• *sc-storage element corresponding to generic sc-arc*
}
⇒ *subdividing**:
Typology of sc-storage elements corresponding to sc-arcs of membership, according to the type of denoted membership^
= {• *sc-storage element corresponding to sc-arc of positive membership*
• *sc-storage element corresponding to sc-arc of fuzzy membership*
• *sc-storage element corresponding to sc-arc of negative membership*
}
}

All semantically and syntactically distinguished classes of sc-storage elements, as well as all possible subclasses of these classes, are instances (elements) of the class.

At the moment, sc-edges are stored in the same way as sc-arcs, that is, they have a start and end sc-element, the difference is only in the *sc-storage element syntactic type*. This leads to a number of inconveniences during processing, but sc-edges are currently used quite rarely.

G. SCin-code specification

The specification of a SCin-code is the union of the specification of its elements. For each element, links between elements and their properties, restrictions are imposed in the form of syntactic rules described above.

sc-storage element corresponding to sc-element
∈ *sc-storage element syntactic type*
⇒ *specification**:
{}
⊃ *relation narrowing by the first domain (sign specification*, sc-storage element corresponding to sc-node)**
⊃ *relation narrowing by the first domain (sign specification*, sc-storage element corresponding to sc-arc)**

sc-storage element corresponding to sc-node
⇒ *specification**:
{• *class of sc-storage element corresponding to sc-node*
• *sc-storage element access level class*
• *sc-address**
• *sc-address of the first sc-arc outgoing from the given sc-element**
• *sc-address of the first sc-arc incoming in the given sc-element**
}

sc-storage element corresponding to sc-arc
⇒ *specification**:
{• *class of sc-storage element corresponding to sc-arc*
• *sc-storage element access level class*
• *sc-address**
• *sc-address of the sc-storage element corresponding to the initial sc-element of the sc-arc**
• *sc-address of the sc-storage element corresponding to the final sc-element of the sc-arc**
• *sc-address of the sc-storage element corresponding to the initial outgoing sc-arc from the given sc-element**
• *sc-address of the sc-storage element corresponding to the initial incoming sc-arc in the given sc-element**
• *sc-address of the sc-storage element corresponding to the next outgoing sc-arc from the given sc-element**
• *sc-address of the sc-storage element corresponding to the next incoming sc-arc in the given sc-element**
• *sc-address of the sc-storage element corresponding to the previous outgoing sc-arc from the given sc-element**
• *sc-address of the sc-storage element corresponding to the previous incoming sc-arc in the given sc-element**
}

H. General algorithm for embedding the SC-code construction into the memory of an ostis-system

Loading an sc-construction into the memory of the ostis-system means translating each sc-element of this sc-construction and the incidence relations between these sc-elements into the memory of the ostis-system, i.e. translating the syntactic structure of the sc-construction into the corresponding representation inside the memory of the ostis-system. In the general case, the algorithm for loading any arbitrary sc-construction into the memory of the ostis-system consists of the following steps:

1) Selection of sc-nodes and internal files of the sc-construction and saving to the corresponding memory cells of the ostis-system;
2) Select all free sc-connectors (i.e. sc-connectors whose start and end sc-element is not another sc-connector), store all sc-connectors in the corresponding ostis-system memory cells and establish links between the initial and final sc-elements of these sc-connectors;
3) Return to step 2 if there are unloaded sc-connectors;
4) Loading the contents of all internal files of the ostis-system into its file storage.

I. Example of the specification of the representation of the SC-code construction in the memory of the ostis-system

The figure 1 shows an example of the specification of the representation of an sc-construction in the memory of an ostis-system implemented on the basis of the designed ostis-platform. Here, each sc-element of the given sc-construction is assigned an sc-element denoting the storage element. For each sc-element denoting a storage element of some sc-element of a given sc-construction, its own denotational semantics is described: links between sc-storage elements and syntactic and semantic classes of elements.

J. Advantages and disadvantages of SCin-code

This model of representation in memory of a system of syntactic and semantic classes of sc-elements in the form of syntactic and semantic classes of elements of

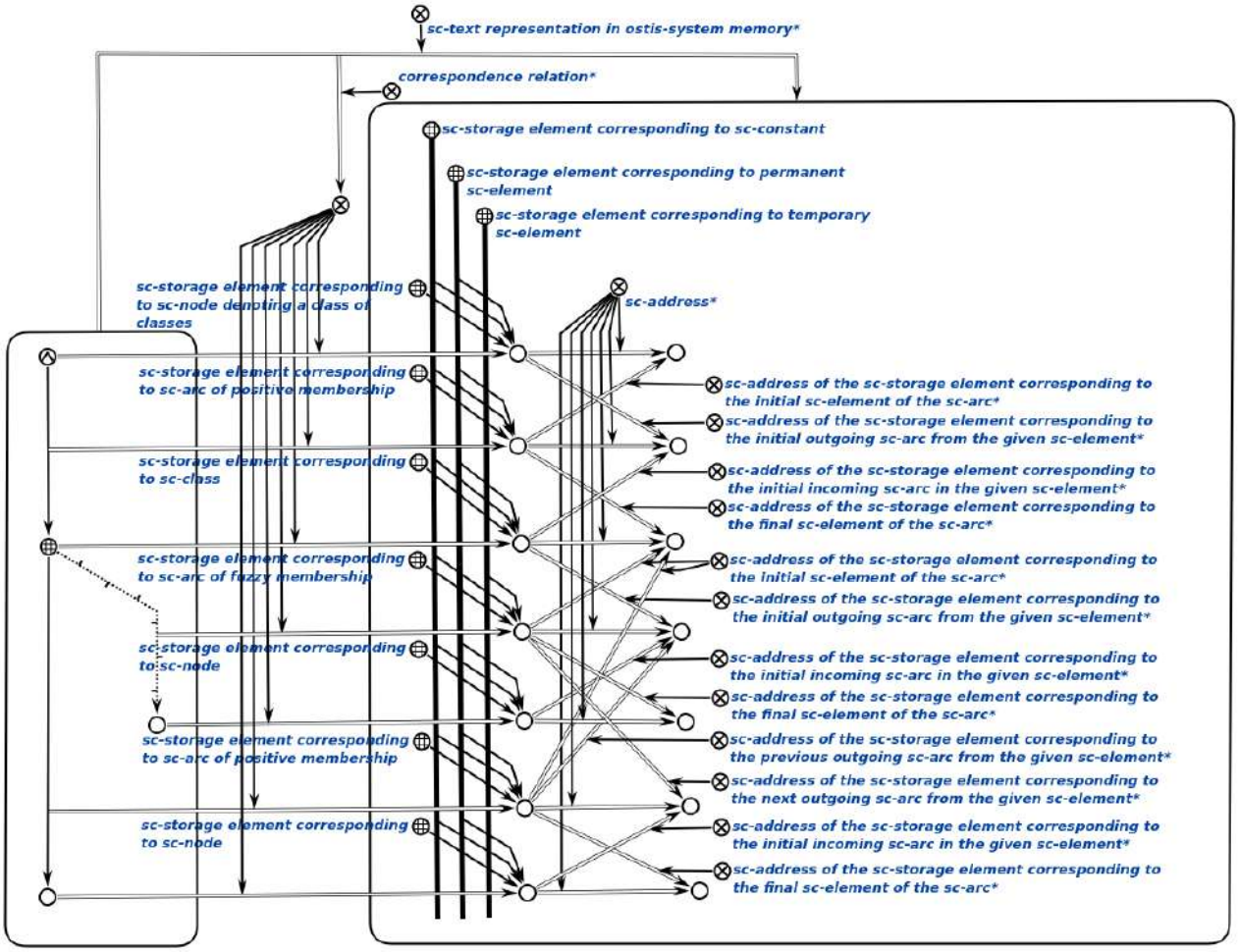

Figure 1. Example of the specification of the representation of the SC-code construction in the memory of the ostis-system

storage sc-elements that correspond to the first ones has a number of advantages:

- *Syntactic and semantic classes of sc-storage elements* can be combined with each other to obtain more specific classes. From the point of view of software implementation, such a combination is expressed by the operation of bit-wise addition of the values of the corresponding numerical expressions *classes of elements of the sc-storage* (here, in the specification on the SC-code, this can be done using the intersection of the corresponding classes). For example, bit-wise addition of numeric expressions of sc-storage element classes corresponding to sc-node and sc-constant results in a new sc-storage element class – *sc-storage element corresponding to constant sc-node*.
- Numeric expressions of some classes may match. This is done to reduce the size of the sc-storage element by reducing the maximum size of the numeric expression of the class of these elements. There is no conflict in this case, since such classes cannot be combined, for example *sc-storage element corresponding to the sc-node of the role relation* and *sc-storage element corresponding to the fuzzy membership sc-arc*.
- It is important to note that each of the selected classes of elements (except for classes obtained by combining other classes) uniquely corresponds to the ordinal number of a bit in linear memory, which can be seen by looking at the corresponding numerical expressions of these classes. This means that classes of elements are not included in each other (although this is not the case in the specification), for example, specifying membership in the class *of sc-store elements corresponding to an sc-arc of positive membership* does not automatically indicate the membership of *elements of the sc-storage corresponding to the sc-arc of membership*. At the implementation level, this makes label combination and comparison operations more efficient.

However, an increase in their number, although it improves the performance of the platform by simplifying

some operations for checking the class of an sc-storage element, it leads to an increase in the number of situations in which it is necessary to take into account the explicit and implicit representation of sc-arcs, which, in turn, complicates the development of the platform and development of program code for processing stored sc-construction. This model does not allow sufficient representation of the syntactic and semantic classes of sc-elements, since it has the following important disadvantages:

- At the moment, the number of *syntactic classes of sc-storage elements* is large enough, which leads to a fairly large number of situations in which it is necessary to take into account the explicit and implicit storage of sc-arcs belonging to the corresponding classes. On the other hand, changing the set of classes of elements for any purpose in the current implementation is a rather laborious task (in terms of the amount of changes in the program code of the platform and sc-agents implemented at the platform level), and expanding the set of classes without increasing the volume sc-storage element in bytes turns out to be completely impossible. The solution to this problem is to minimize the number of classes as much as possible, for example, to the number of classes corresponding to the *SC-code alphabet*. In this case, the membership of sc-elements to any other classes will be recorded explicitly, and the number of situations in which it will be necessary to take into account the implicit storage of sc-arcs will be minimal.
- Some class from the current set of *syntactic and semantic classes of sc-elements* are rarely used (for example, *sc-store element corresponding to a generic sc-edge* or *sc-store element corresponding to sc-arc of negative membership*), in turn, in sc-memory there can be classes that have quite a lot of elements (for example, *binary relation** or *number*). This fact does not allow us to fully use the efficiency of having classes. The solution to this problem is the rejection of a previously known set of classes and the transition to a dynamic set of classes (while their number can remain fixed). In this case, a set of classes expressed as numeric values will be formed based on some criteria, for example, the number of elements of this class or the frequency of calls to it.
- *base sc-arcs* denoting that sc-elements belong to some known limited set of classes in memory are presented implicitly. This fact must be taken into account in a number of cases, for example, when checking whether an sc-element belongs to a certain class, when searching for all outgoing sc-arcs from a given sc-element, etc. If necessary, some of these implicitly stored sc-arcs can be represented explicitly, for example, in the case when such an sc-arc must be included in some set, that is, another sc-arc must be drawn into it. In this case, it becomes necessary to synchronize changes associated with a given sc-arc (for example, its deletion) in its explicit and implicit representation. The current *Implementation of sc-storage* does not implement this mechanism. This problem is solved by one of the previous options for solving the problems of this model.

In the current *Implementation of sc-storage access-level classes* are used to provide the ability to restrict the access of some processes in sc-memory to certain elements stored in sc-memory. Each sc-store element belongs to one of two classes: the class *of sc-store elements corresponding to sc-elements on which the read right is allowed* and the class *of sc-store elements corresponding to sc-elements on which the right is allowed records*. Each of which is expressed as a number from 0 to 255.

Thus, the null value of the numeric expressions of the class *sc-storage elements corresponding to sc-elements on which read access is allowed* and the class *sc-storage elements corresponding to sc-elements on which write access is allowed* means that any process can get unrestricted access to this sc-storage element.

Each element of the sc-storage corresponding to some sc-element is described by its syntactic type (label), and, regardless of the type, the sc-address of the first sc-arc entering the given sc-element and the first sc-arc leaving the given sc-element is indicated (may be empty if there are no such sc-arcs). The remaining bytes, depending on the type of the corresponding sc-element (sc-node or sc-arc), can be used to store the specification of the sc-arc. Also, *sc-address of the first sc-arc outgoing from the given sc-element** and *sc-address of the first sc-arc entering the given sc-element** may generally be absent (be null, "empty"), but the size of the sc-element in bytes will remain the same.

From the point of view of software implementation, the data structure for storing the sc-node and sc-arc remains the same, but the list of fields (components) changes in it. In addition, as you can see, each sc-storage element (including *sc-storage element corresponding to sc-arc*) does not store a list of sc-addresses of associated sc-elements, but stores sc-addresses of one outgoing and one incoming arc, each of which in turn stores the sc-addresses of the next and previous arcs in the list of outgoing and incoming sc-arcs for the corresponding elements. All of the above allows you to:

- make the size of such a structure fixed (currently 36 bytes) and independent of the syntactic type of the stored sc-element;
- provide the ability to work with sc-elements without regard to their syntactic type in cases where it is necessary (for example, when implementing search queries like “Which sc-elements are elements of this set”, “Which sc-elements are directly related with the given sc-element”, etc.);

- provide the ability to access *sc-storage element* in constant time;
- provide the ability to place the *sc-storage element* in the processor cache, which in turn speeds up the processing of sc-constructions;

VIII. Implementation of the ostis-system file storage

A. Selected solution and its rationale

Often, the expressiveness of SC-code graph structures is not enough to represent and store linear sequences of texts, pictures, sound, video, and so on. Although SC-code is a universal tool for representing any kind of knowledge, there is not always a need to immerse something in the graph-dynamic memory of the ostis-system, at least in the early stages of development of the ostis-platform. This can also be explained by the fact that information constructions that do not belong to the SC-code are quite complex in syntax and volume. To solve such problems, an additional element is introduced at the *SC-code Alphabet^* level – the internal file of the ostis-system. With the help of ostis-system files, it is possible to represent, store, process and visualize information structures that do not belong to the SC-code using the SC-code.

Therefore, when implementing sc-memory, it is necessary to take into account the need to store information structures that do not belong to SC-code using SC-code. Such solution is the *Implementation of the file storage of the ostis-system.*

During the entire period of *Software implementation of the ostis-platform* development, there have been quite a few attempts to implement a fully functional and fast file storage based on popular databases. However, all these solutions did not take into account potential problems in the implementation of the search and navigation subsystem *Software implementation of the ostis-platform*. Now the file storage is implemented by its own means, as data structures for storing information structures that do not belong to the SC-code, prefix B-trees [56] and linear lists are used.

The choice is justified by the fact that:

- prefix structures are fairly easy to understand and minimal in their syntax;
- with the help of prefix structures, it is quite convenient to store and process key-value relations;
- accessing a value by key occurs in the worst case for the length of that key [57], [58];
- due to the fact that the prefixes stick together, there is a strong gain in memory usage.

To store the contents of internal files of ostis-systems, files are used that are explicitly stored on the file system, which is accessed by means of the operating system on which *Software implementation of the ostis-platform* is running.

Implementation of the ostis-system file storage
∈ *file storage implementation based on prefix tree*
⇐ *software model**:
ostis-system file storage
∈ *atomic reusable ostis-systems component*
∈ *dependent reusable ostis-systems component*
⇒ *component dependencies**:
{• *GLib library of methods and data structures*
}
⇒ *used method representation language**:
• *C*
⇒ *internal language**:
• *SCfin-code*

As in the case with the sc-storage, it is necessary to describe the language for representing information structures that do not belong to the SC-code inside the file storage of the ostis-system.

B. The concept of the SCfin-code

We will call such a language the language of internal representation of information constructions that do not belong to the SC-code, or, briefly, *SCfin-code (Semantic Code file interior)*. The file storage of texts that do not belong to the SC-code can be considered as a subset of the scfin-text.

SCfin-code
≔ [Semantic Code file interior]
≔ [Language of the internal semantic representation of information constructions that do not belong to the SC-code inside the memory of the ostis-system]
≔ [meta-language for describing the representation of the information constructions that do not belong to the SC-code inside the memory of the ostis-system]
⇒ *frequently used non-primary sc-element external identifier**:
[scfin-text]
∈ *common noun*
∈ *abstract language*
∈ *metalanguage*
⊂ *SC-code*
⊃ *ostis-system file storage*

The *SCfin-code syntax* is given by: (1) the *SCfin code alphabet*, (2) the *sequence in linear text** order relation.

C. SCfin-code alphabet

SCfin-code alphabet^
≔ [syntactic type of ostis-system file storage element]
≔ [Set of types of ostis-system file storage elements]

⇐ *alphabet**:
SCfin-code
= {• *element of ostis-system file storage corresponding to a substring of linear language text*
}

*SCfin-code alphabet^* consists of one syntactically distinguished type of file storage elements – *element of ostis-system file storage corresponding to a substring of linear language text*.

element of ostis-system file storage corresponding to a substring of linear language text
∈ *sc-element*
:= [ostis-system file storage element]
:= [ostis-system file storage cell]
:= [image of information construction substring that do not belong to the SC-code within the ostis-system file storage]

The relation *sequences in a linear text** is defined as a binary oriented order relation, the components of each ordered pair of which are elements of the ostis-system file storage corresponding to some substrings of the linear text, as a result of which, as a result of their concatenation, a substring belonging to the same linear text is formed.

D. SCfin-code syntax

The *SCfin-code syntax* is quite simple, since the information constructions on it are specified using the *SCfin-code alphabet*, whose cardinality is 1, and the single incidence relation *sequence in a linear text**. Hierarchies of syntactic elements are not distinguished as such, as this is not necessary.

E. Denotational semantics of SCfin-code

At the implementation level, it is important to single out the semantic classes e*elements of the ostis-system file storage, corresponding to a substring of the text of the linear language*, which denote some prefix or postfix part of the entire information construction.

Semantic classification of SCfin-code elements
⊃=
{

element of ostis-system file storage corresponding to a substring of linear language text
⇒ *subdividing**:
Typology of elements by substring location in linear text
= {• *element of the ostis-system file storage corresponding to the prefix substring of the linear language text*
• *element of the ostis-system file storage corresponding to the postfix substring of the linear language text*
}
}

F. Example of the specification of the representation of information constructions that doesn't belong to the SC-code in the memory of the ostis-system

In the SCfin-code, it is enough to simply set the information constructions of any linear texts. However, from the point of view of the implemented sc-memory model, there is a need to specify not so much the form of information structures that do not belong to the SC-code inside the file storage of the ostis-system, but rather the links between these external information structures, the files of the ostis-system, which are signs of the SC-code. At the same time, at the sc-memory level, both the method for obtaining ostis-system files that contain a given external information structure and the methods for obtaining external information structures from given ostis-system files must be implemented at the sc-memory level.

Figure 2 shows the representation of information constructions that do not belong to the SC-code and the correspondence between ostis-system files and information constructions. Using the relation *set of sc-addresses of ostis-system files by their content prefixes**, a binary oriented pair is specified, the first component of which is a prefix structure, the elements of which are substrings of external information constructions, and the second component is the set of corresponding sc-addresses ostis-system files. And using the relation *set of postfixes of the contents of ostis-system files by their sc-addresses**, a binary oriented pair is specified, the first component of which is a prefix structure, the elements of which are substrings of the sc-addresses of ostis-system files presented in string form, and the second component is the set of corresponding postfixes of external information structures of the prefix structure, which is the first component of each pair of the relation *set of sc-addresses of ostis-system files by their content prefixes**.

G. Advantages and disadvantages of SCfin-code

The used *Implementation of the ostis-system file storage* fully justifies itself when interacting with the system. Due to the use of prefix structures, the asymptotic complexities of the method for obtaining a set of external information constructions from given ostis-system files and the method for obtaining a set of ostis-system files from given external information constructions are linear, since it depends on the length of a given string and the structure of the prefix tree.

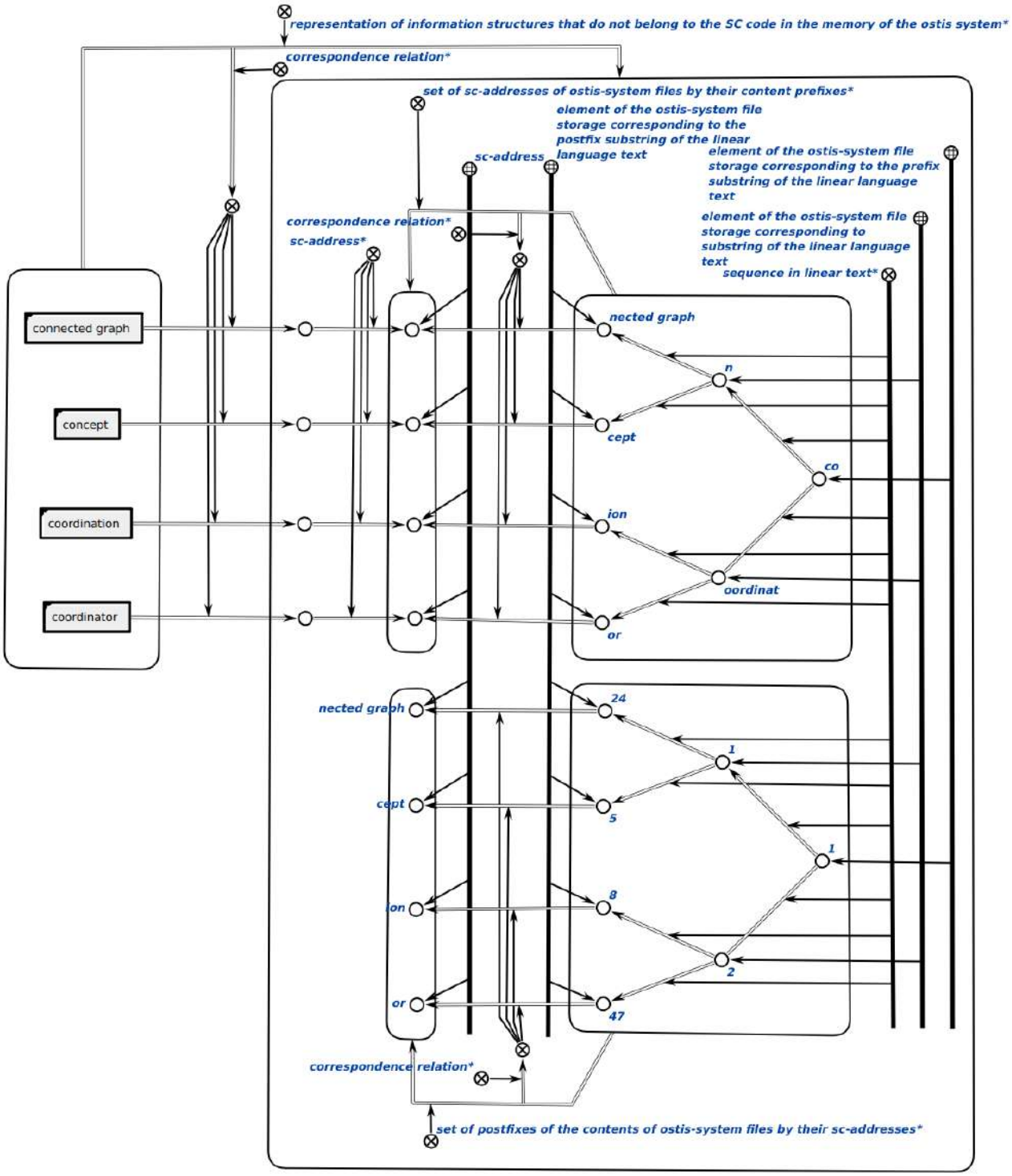

Figure 2. An example of a specification for the representation of information structures that do not belong to the SC-code in the memory of the ostis-system

- Information constructions that do not belong to the SC-code are still completely stored in RAM of the computer device on which the platform is deployed. This problem can be solved if only the first characters of substrings of information structures are stored in RAM, and the remaining parts of these substrings are stored at the file system level.
- At the moment, the information retrieval subsystem is not fully implemented. *Implementation of the ostis-system file storage* allows quickly solving the problem of searching for external information constructions by their prefix substrings, but does not allow quickly solving the problem of searching for information constructions by any substring, even for which some sample-template is specified.

The described problems will be solved within a future version of the *Software version of the ostis-platform*.

IX. General description of methods for Implementation of the sc-memory

The SCin-code and the SCfin-code are sufficient to represent the texts of the SC-code within the memory of the ostis-system. To translate some SC-code text into the ostis-system memory, it is necessary to use sc-memory methods (programs, procedures), which are elements of *Implementation of the sc-memory*.

sc-memory method
- ⊂ *method*
- ⊂ *Implementation of the sc-memory*
- ∋ *Method of creating an sc-storage element corresponding to the sc-node with a given type*
- ∋ *Method of creating an sc-storage element corresponding to the sc-arc with a given type*
- ∋ *Method of creating an sc-storage element corresponding to the ostis-system file with a given type*
- ∋ *Method of setting the information construction of the linear language in accordance with the given sc-storage element corresponding to the ostis-system file*

So, using the *Method of creating an sc-storage element corresponding to the sc-node with a given type*, the *Method of creating an sc-storage element corresponding to the sc-arc with a given type*, and the *Method of creating an sc-storage element corresponding to the ostis-system file with a given type*, it is possible to create all program elements of the *SCin-code alphabet^* corresponding to sc-elements of the *SC-code alphabet^*, and using the *Method of setting the information construction of the linear language in accordance with the given sc-storage element corresponding to the ostis-system file* to indicate the connections between the sc-storage elements corresponding to the files of the ostis-system and external information structures represented in the ostis-storage file systems as linear text.

There are other methods in *Implementation of the sc-memory*, but they will not be covered in this article.

X. Implementation of the subsystem of network interaction with Implementation of sc-memory

A. Selected solution and its rationale

The interaction of the sc-memory software model with external resources can be carried out through a specialized programming interface (API), however, this option is inconvenient in most cases, since:

- it is only supported for a very limited set of programming languages (C, C++);
- it requires that the client application accessing the sc-memory software model actually forms a single whole with it, thus eliminating the possibility of building a distributed collective of ostis-systems;
- as a consequence of the previous paragraph, the possibility of parallel work with sc-memory of several client applications is excluded.

In order to provide the possibility of remote access to sc-memory without taking into account the programming languages with which a particular client application is implemented, it was decided to implement the possibility of accessing sc-memory using a universal language that does not depend on the means of implementing one or another component or system.

Among the effective protocols used in the implementation of client-server systems, it is worth noting the application layer protocols of the TCP/IP stack – HTTP and WebSocket protocols [59], [60]. It is advisable to use the WebSocket protocol due to the following reasons:

- WebSocket is useful in web-based systems where data sent by the server is represented or stored on the client side. In WebSocket, data is constantly transferred over the same open connection, so WebSocket communication is faster than HTTP communication [61], [62]. This is very important in terms of designing the OSTIS Ecosystem, which can consist of tens of thousands of different ostis-systems kinds.
- Since ostis-systems are based on the idea of agent-oriented knowledge processing (asynchronous processing) and the memory of such systems must be both distributed and shared, it is necessary that each of them (in particular, an independent ostis-system) be able to communicate with other ostis-systems. Moreover, such communication can and should take place on the conditions of initiating events in the memory of these systems. This implies an unambiguous conclusion that the HTTP protocol cannot be used in advanced next-generation intelligent systems

due to the unidirectional nature of the connection it creates.

A string language based on the JSON language [63], [64] – *SC-JSON-code* – was developed as a system communication language. Such choice is explained by the flexibility of setting relation between the objects it describes.

B. Implementation of the subsystem for interaction with sc-memory based on the JSON language

Generally speaking, the subsystem of interaction with the external environment can be implemented in different ways. So, for example, before the implementation of the current subsystem, there was previously its analogue in the Python programming language, which used the HTTP protocol and a binary representation of commands and responses. Therefore, there can be a wide variety of such subsystems, that can build various *Implementations of the subsystem of interaction with the external environment using network languages*.

This *Implementation of the sc-memory interaction subsystem based on the JSON language* allows ostis-systems to interact with systems from the external environment based on the generally accepted JSON data transfer transport format and provides an API for accessing the sc-memory of the sc-model interpretation platform.

Implementation of the subsystem of interaction with the external environment using network languages
⇒ *software system decomposition**:
{•
Implementation of the subsystem of interaction with the external environment using network languages based on the JSON language
}

Implementation of the sc-memory interaction subsystem based on the JSON language
:= [Subsystem for interaction with sc-memory based on the JSON format]
∈ *non-atomic reusable ostis-systems component*
∈ *dependent reusable ostis-systems component*
∈ *client-server system*
⇒ *used method representation language**:
- *C*
- *C++*
- *Python*
- *TypeScript*
- *C#*
- *Java*

⇒ *used language**:
- *SC-JSON-code*

⇒ *software system decomposition**:
{
- *Websocket- and JSON-based server system providing network access to sc-memory*

{}
= {•
Implementation of the client system in the Python programming language
- *Implementation of the client system in the TypeScript programming language*
- *Implementation of the client system in the C# programming language*
- *Implementation of the client system in the Java programming language*

}
}

Interaction with sc-memory is provided by transferring information in the ***SC-JSON-code*** and is conducted, on the one hand, between the server, which is part of the ostis-system, written in the same implementation language of this ostis-system and having access to its sc-memory, and, on the other hand, a set of clients who are aware of the presence of a server within the network of their usage. Using the subsystem for interaction with sc-memory based on the JSON language, it is possible to interact with the ostis-system on the same set of possible operations as in the case if the interaction took place directly, in the same implementation language of the platform for interpreting sc-models of computer systems. In this case, the result of the work differs only in the speed of information processing.

C. Concept of the SC-JSON-code

As mentioned earlier, subsystems within the implemented software version of a specialized platform communicate using the external knowledge representation language – an SC-JSON-code. This language is string, i.e. linear, and easy to reverse, since there are a large number of facilities for processing its JSON superlanguage.

SC-JSON-code
:= [Semantic JSON-code]
:= [Semantic JavaScript Object Notation code]
:= [Language of external semantic representation of knowledge based on the JSON language]
⇒ *frequently used non-primary external identifier of an sc-element**:
[sc-json-text]
∈ *common noun*
∈ *abstract language*
∈ *linear language*
⊂ *JSON*

D. Syntax and syntactic rules of the SC-JSON-code

The *SC-JSON-code syntax* is specified by: the (1) *SC-JSON-code alphabet* and the (2) SC-JSON-code grammar. In the alphabet of the SC-JSON-code, the basic syntactic classification of its elements is distinguished.

Syntactic classification of SC-JSON-code elements

⊃=

{

SC-JSON-code

⇐ *subset family**:
sc-json-sentence
⊂ *json-list of json-pairs*
⇐ *subset family**:
*sc-json-pair**
⇐ *Cartesian product**:
⟨• *sc-json-string*
• *sc-json-object*
⟩
⇒ *subdividing**:
{• *SC-JSON-code command*
• *SC-JSON-code command response*
}

sc-json-object

⇒ *subdividing**:
{• *sc-json-list*
• *sc-json-pair*
• *sc-json-literal*
⇒ *subdividing**:
{• *sc-json-string*
• *sc-json-number*
}
}

}

The *SC-JSON-code alphabet*^ is a set of all possible characters in the SC-JSON-code. Since the *SC-JSON-code* is a linear string knowledge representation language, its alphabet includes the combination of the alphabets of all languages, the texts in which can represent external identifiers and/or the contents of ostis-system files, the set of all digits, and the set of all other special characters. Alphabet sequences can form sc-json-keywords, *sc-json-pairs*, *sc-json-sentences* from *sc-json-pairs*, and *sc-json-texts* from *sc-json-sentences*. At the same time, constructions on the SC-JSON-code are built according to the following syntactic rules:

- Each *SC-JSON-code grammar* rule describes the correct order of sc-json objects in an sc-json-sentence according to the *SC-JSON-code syntax*. The set of *SC-JSON-code grammar* rules describes the order of sc-json-sentences in sc-json-text that is correct in terms of the *SC-JSON-code syntax*. Each sc-json-sentence is an sc-json-list of sc-json-pairs, which represents a command or response to that command.
- Each *command (command response) in the SC-JSON-code* consists of a header that includes sc-json-pairs describing the command itself (command response) and a message that is different for each class of commands (command responses). The *command (command response) message in the SC-JSON-code* is usually a list of sc-json-objects, which may not be limited in size.
- Each sc-json-pair consists of two elements: a keyword and some other sc-json-object associated with that keyword. The set of keywords in sc-json-pairs is determined by a specific class of *commands (command response) in the SC-JSON-code*. The sc-json pair starts with an open brace "{" and ends with a close brace "}". The keyword and the sc-json object associated with it are separated by a colon character ":".
- Sc-json strings written in sc-json texts begin and end with the double-quoted character “.
- Sc-json-lists that do not consist of sc-json-pairs begin with an opening square bracket "[" and end with a close square bracket "]". Sc-json-objects in sc-json-lists are separated by commas ",".

E. Syntax and grammar of the SC-JSON-code. SC-JSON command and response examples

The grammar of the SC-JSON-code is the set of all possible rules used in building commands and responses to them in the SC-JSON code. Each *SC-JSON-code* command has a unique *SC-JSON-code* grammar rule. The *SC-JSON grammar* rules allow correctly representing commands in the SC-JSON-code. Each *SC-JSON-code* grammar rule is represented as a rule in the *ANTLR Grammar Description Language* and its natural language interpretation.

SC-JSON grammar

∋ *key sc-element'*:
Rule that specifies the syntax of SC-JSON-code commands
⇐ *syntax rule**:
SC-JSON-code command
∋ *key sc-element'*:
Rule that specifies the syntax of SC-JSON-code command responses
⇐ *syntax rule**:
SC-JSON-code command response
∋ *Rule that specifies the syntax of the command for creating sc-elements*
⇐ *syntax rule**:
command for creating sc-elements

∋ *Rule that specifies the syntax of response to the command for creating sc-elements*
⇐ *syntax rule**:
response to the command for creating sc-elements

The rule that specifies the syntax of the *SC-JSON-code command* means the following 3. The *SC-JSON-code command* class includes the *command for creating sc-elements*, *command for getting corresponding types of sc-elements*, *command for deleting sc-elements*, *command for processing key sc-elements*, *command for processing contents of ostis-system files*, *command for searching for sc-constructions isomorphic to a given sc-template*, *command for generating an sc-constructions isomorphic to a given sc-template*, and *command processing sc-event*. The *SC-JSON-code command* includes the command ID, type, and message.

```
sc_json_command
 : '{'
     '"id"' ':' NUMBER ','
     sc_json_command_type_and_payload
  '}'
 ;

sc_json_command_type_and_payload
 : sc_json_command_create_elements
 | sc_json_command_check_elements
 | sc_json_command_delete_elements
 | sc_json_command_handle_keynodes
 | sc_json_command_handle_link_contents
 | sc_json_command_search_template
 | sc_json_command_generate_template
 | sc_json_command_handle_events
 ;
```

Figure 3. Description of the Rule that specifies the syntax of the *SC-JSON-code command*

The rule specifying the syntax of the *SC-JSON-code command response* describes the syntax of command responses described by the previous rule. The *SC-JSON-code command response* class includes the *command response for creating sc-elements*, *command response for getting the corresponding types of sc-elements*, *command response for deleting sc-elements*, *command response for processing key sc-elements*, *command response for processing contents of ostis-system files*, *command response for searching for sc-constructions isomorphic to the given sc-template*, *command response for generating an sc-construction isomorphic to the given sc-template*, and *command response for sc-event processing*.

The *command for creating sc-elements* message contains a list of descriptions of the sc-elements to be created. Such sc-elements can be an sc-node, an sc-arc, an sc-edge, or an ostis-system file. The sc-element type is

```
sc_json_command_answer
 : '{'
     '"id"' ':' NUMBER ','
     '"status"' ':' BOOL ','
     sc_json_command_answer_payload
  '}'
 ;

sc_json_command_answer_payload
 : sc_json_command_answer_create_elements
 | sc_json_command_answer_check_elements
 | sc_json_command_answer_delete_elements
 | sc_json_command_answer_handle_keynodes
 | sc_json_command_answer_handle_link_contents
 | sc_json_command_answer_search_template
 | sc_json_command_answer_generate_template
 | sc_json_command_answer_handle_events
 ;
```

Figure 4. Description of the Rule that specifies the syntax of the *SC-JSON-code command response*

specified in pair with the "el" keyword: for an sc-node, the sc-json-type of element is represented as a "node", for an sc-arc and an sc-edge – an "edge", for ostis-system file – a "link". Type labels of sc-elements are specified in their corresponding descriptions in the command message, paired with the "type" keyword. If the sc-element being created is an ostis-system file, then the contents of this ostis-system file are additionally specified in pair with the "content" keyword; if the sc-element being created is an sc-arc or an sc-edge, then the descriptions of the sc-elements they go out and the sc-elements they come in are specified. Descriptions of such sc-elements consist of two pairs: the first pair indicates the method of association with the sc-element and is represented as "addr", or "idtf", or "ref" paired with the "type" keyword, the second pair represents what is associated with this sc-element: its hash, system identifier, or number in the array of created sc-elements – paired with the "value" 5 keyword.

The *command response for creating sc-elements* message is a list of hashes of created sc-elements corresponding to the *command for creating sc-elements* descriptions with status 1, in case of successful processing of the 6 command.

The *SC-JSON-code command* set is easily extensible due to the flexibility and functionality of the JSON language. The set of the *command responses in the SC-JSON-code* is also easily extensible, along with the *SC-JSON-code commands* extension.

command for creating sc-elements
≔ [create elements command]
⊂ *SC-JSON-code command*

```
sc_json_command_create_elements
  : '"type"' ':' '"create_elements"' ','
    '"payload"' ':'
    '['(
        '{'
            '"el"' ':' '"node"' ','
            '"type"' ':' SC_NODE_TYPE ','
        '}' ','
        |
        '{'
            '"el"' ':' '"link"' ','
            '"type"' ':' SC_LINK_TYPE ','
            '"content"' ':' NUMBER_CONTENT
                        | STRING_CONTENT
        '}' ','
        |
        '{'
            '"el"' ':' '"edge"' ','
            '"type"' ':' SC_EDGE_TYPE ','
            '"src"' ':' (
            '{'
                '"type"' ':' '"ref"' ','
                '"value"' ':' NUMBER
            '}' ','
            |
            '{'
                '"type"' ':' '"addr"' ','
                '"value"' ':' SC_ADDR_HASH
            '}' ','
            |
            '{'
                '"type"' ':' '"idtf"' ','
                '"value"' ':' SC_NODE_IDTF
            '}' ','
            )
            '"trg"' ':' (
            '{'
                '"type"' ':' '"ref"' ','
                '"value"' ':' NUMBER
            '}' ','
            |
            '{'
                '"type"' ':' '"addr"' ','
                '"value"' ':' SC_ADDR_HASH
            '}' ','
            |
            '{'
                '"type"' ':' '"idtf"' ','
                '"value"' ':' SC_NODE_IDTF
            '}' ','
            )
        '}' ','
        |
        scs_text ','
    )*']' ','
  ;
```

Figure 5. Description of the Rule that specifies the syntax of the *command for creating sc-elements*

```
sc_json_command_answer_create_elements
  : '"payload"' ':'
    '['
        (SC_ADDR_HASH ',')*
    ']' ','
  ;
```

Figure 6. Description of the Rule that specifies the syntax of the *command response for creating sc-elements*

⇒ *example**:
Example of the command for creating sc-elements
⇒ *command class**:
command response for creating sc-elements

The Websocket- and JSON-based server system providing network access to sc-memory will interpret the *Example of command for creating sc-elements* 7 as "Process command for creating sc-elements: an sc-node of type 1 (of an unspecified type), an ostis-system file of type 2 (of an unspecified type), and contents in the form of a floating point number 45.4, and an sc-arc of type 32 (of a constant type) between the sc-element located at the zero position in the array of created sc-elements, and an sc-element in the first position in the same array".

It should be noted that at the sc-memory interface level, the command is interpreted quickly due to the fact that templates for creating constructions isomorphic to them are not used. Also, the contents of the message of the *command for creating sc-elements* can be empty.

command response for creating sc-elements
≔ [create elements command response]
⊂ *SC-JSON-code command response*
⇒ *example**:
Example of the command response for creating sc-elements

An example of the command response for creating sc-elements is an example of a response to the previous command if this command was interpreted and executed successfully 8.

The formal text of the *Example of the command response for create sc-elements* is equivalent to the natural language text "Created sc-elements with hashes 323, 534, and 342, respectively. The command was processed successfully".

A detailed description of the syntax of commands and responses to these commands, as well as their examples, can be found in the OSTIS Standard [6].

```
{
 "id": 3,
 "type": "create_elements",
 "payload": [
  {
   "el": "node",
   "type": 1,
  },
  {
   "el": "link",
   "type": 2,
   "content": 45.4,
  },
  {
   "el": "edge",
   "src": {
    "type": "ref",
    "value": 0,
   },
   "trg": {
    "type": "ref",
    "value": 1,
   },
   "type": 32,
  },
 ],
}
```


Figure 7. An example of the *command for creating sc-elements*


```
{
 "id": 3,
 "status": 1,
 "payload": [
  323,
  534,
  342,
 ],
}
```


Figure 8. An example of the *command response for creating sc-elements*

F. Description of Implementation of The server system based on Websocket and JSON, providing network access to sc-memory

The *Server system based on Websocket and JSON, providing network access to sc-memory* is an interpreter of commands and responses of the *SC-JSON-code* for programm representation of sc-constructions in sc-memory using the *Library of software components for processing json texts (JSON for Modern C++)* and the *Library of cross-platform software components for implementing server applications based on Websocket (WebSocket++)*, and is also provided with comprehensive test coverage through the Google Tests and Google Benchmark Tests software frameworks. The *Library of software components for processing json-texts (JSON for Modern C++)* has a rich, convenient, and high-speed functionality necessary for the implementation of such components of ostis-systems, and the *Library of cross-platform software components for the implementation of server applications based on Websocket (WebSocket++)* allows elegantly designing server systems without using redundant dependencies and solutions. The software component is configured with the help of the *Software component for ostis-systems software components configuration*, as well as CMake and Bash scripts.

Implementation of the Server system based on Websocket and JSON, providing network access to sc-memory

:= [Implementation of the Websocket-based system that provides parallel-asynchronous multi-client access to sc-memory of the sc-model interpretation platform using the SC-JSON-code]

:= [sc-json-server]

⇒ *frequently used non-primary external identifier of the sc-element**:
[sc-server]

∈ *atomic reusable ostis-systems component*

∈ *dependent reusable ostis-systems component*

⇒ *used method representation language**:
- *C*
- *C++*

⇒ *used language**:
- *SC-JSON-code*

⇒ *component dependencies**:
{
- *Library of software components for processing json-texts JSON for Modern C++*
- *Library of cross-platform software components for implementing server applications based on Websocket WebSocket++*
- *Software component for ostis-systems software components configuration*
- *Implementation of the sc-memory*

}

It is worth noting that the current *Implementation of the Server system based on Websocket and JSON, providing network access to sc-memory* is not the first of its kind and replaces its previous implementation written in Python. The reason for this replacement is as follows:

- Previous *Implementation of the server system based on Websocket and JSON, providing access to sc-memory using SC-JSON-code commands*, implemented in the Python programming language, depends on the Boost Python library provided by

the C++ Language Development and Collaboration Community, as well as Python. The fact is that such a solution requires the support of the mechanism for interpreting the Python program code into the C++ language, which is redundant and unreasonable, since most of the *Software implementation of the ostis-platform* program code is implemented in the C and C++ languages. The new implementation of the described software system allows getting rid of the usage of capacious and resource-intensive libraries (for example, boost-python-lib, llvm) and the Python language.

- When implementing distributed subsystems, the speed of knowledge processing plays an important role, that is, the ability to quickly and urgently respond to user requests. The quality of access to sc-memory through the implemented *Subsystem for interacting with sc-memory based on the JSON language* should be commensurate with the quality of access to sc-memory using a specialized API, implemented in the same programming language as the system itself. The new implementation makes it possible to increase the processing speed of requests by the *JSON-based sc-memory interaction subsystem*, including knowledge processing, by at least 1.5 times compared to the previous implementation of this subsystem.

Implementation of the Server system based on Websocket and JSON, providing network access to sc-memory possesses the following common properties:

- From the point of view of its model, the server subsystem has the same specialized programming interface as the *Implementation of the sc-memory*, however, interaction with it using such an interface is carried out via the network. This makes it possible for client systems implemented in different programming languages to interact with the same shared memory.
- This subsystem can be considered as an interpreter of an external knowledge representation language (SC-JSON-code), which can be used by ostis-systems implemented on the basis of a specialized ostis-platform. Each command and response to a command of this language corresponds to a handler (potentially an agent at all), which is part of this interpreter. The SC-JSON-code language of external knowledge representation itself is independent of the platform implementation and is used only as a language of external knowledge representation, however, it can be used when implementing other tools and interpreters of sc-models of ostis-systems.
- The implemented software component provides multi-user asynchronous access to sc-memory. While testing the sc-server, it turned out that its implementation allows processing requests from at least 1000 client systems. Due to the need to provide parallel access to sc-memory, synchronization blocks were added at the implementation level of the software component. For example, in the implementation, it is possible to notice a queue of commands for processing by the system – regardless of the number of client systems and how many commands were sent for processing, all commands can queue up. This solution allows temporarily bypassing the problems of interaction of synchronization blocks at the sc-memory level when processing different types of commands over it (search, generative, destructive, etc.). However, the server system cannot be shut down as long as the command queue has any pending commands. Also, the server system continues to work if the list of client system identifiers still has non-disconnected ones. The need for these functions of the server subsystem is determined by the need to support the atomicity of requests processed by the system.
- In the process of testing the subsystem, an estimate of its speed of processing commands and responses was obtained. During load testing, a test client system was used, written in C++ and not possessing the functionality of processing texts of the SC-JSON-code. As a result of testing, it was found that when sending 1000 different commands – commands for creating sc-elements, commands for processing the contents of ostis-system files, and commands for deleting sc-elements – the time spent on their processing did not exceed 0.2 seconds. At the same time, in some cases, processing 1000 commands for creating sc-elements took no more than 0.14 seconds, commands for deleting sc-elements – no more than 0.07 seconds, commands for processing the contents of ostis-system files – no more than 0.27 seconds, commands search for sc-constructions isomorphic to a given sc-template – no more than 0.45 seconds.

The *Server system based on Websocket and JSON, providing network access to sc-memory* describes the necessary and sufficient programming interface for interacting with sc-memory. In the general case, it describes the functionality of not only the *Server system based on Websocket and JSON, providing network access to the sc-memory* but also the client systems interacting with it, since these client systems often include a specialized programming interface similar to the server system interface but implemented in a different programming language.

XI. Implementation of the interpreter of user interface sc-models

A. Concept of the interpreter of user interface sc-models

In most cases, user interface development in modern systems takes up most of the time spent on developing

the entire system. However, the effectiveness of using a software system depends on the developed user interface [65].

Along with the *Implementation of the sc-memory*, an important part of the *Software implementation of the ostis-platform* is the *Implementation of the interpreter of user interface sc-models*, which provides basic tools for viewing and editing the knowledge base by the user, tools for navigation through the knowledge base (asking questions to the knowledge base) and can be supplemented with new components depending on the problems solved by each specific ostis-system.

Implementation of the interpreter of user interface sc-models

∈ *non-atomic reusable ostis-systems component*

∈ *dependent reusable ostis-systems component*

⇒ *used method presentation language**:

- *JavaScript*
- *TypeScript*
- *Python*
- *HTML*
- *CSS*

⇒ *component dependencies**:

{•
- *Library of standard interface components in the JavaScript programming language*
- *Library for implementing server applications in the Python programming language, named Tornado*
- *Implementation of the client system in the TypeScript programming language*
- *Implementation of the client system in the Python programming language*

}

An important principle of the *Implementation of the interpreter of user interface sc-models* is the simplicity and uniformity of connecting any user interface components (editors, visualizers, switches, menu commands, etc.). To do this, the Sandbox software layer is implemented, within which low-level operations of interaction with the server part are implemented and which provides a more convenient programming interface for component developers. The current version of the *Implementation of the interpreter of user interface sc-models* is open and available at [66].

B. Main components of the interpreter of user interface sc-models

Implementation of the interpreter of user interface sc-models

⇒ *software system decomposition**:

{•
- *User interface command menu bar*
- *Component for switching the language of identification of displayed sc-elements*
- *Component for switching the external language of knowledge visualization*
- *Search field of sc-elements by identifier*
- *Panel for displaying the user dialog with the ostis-system*
- *Knowledge visualization and editing panel*

⇒ *software system decomposition**:

{•
- *Visualizer of sc.n-texts*
- *Visualizer and editor of sc.g-texts*

}

}

The *Component for switching the language of identification of displayed sc-elements* is an image of the set of natural languages available in the system. User interaction with this component switches the user interface to a mode of communication with a specific user with the help of *basic sc-identifiers* belonging to this *natural language* (Fig. 9). This means that when displaying sc-identifiers of sc-elements in any language, for example, SCg-code or SCn-code, *basic sc-identifiers* belonging to the given *natural language* will be used. This applies both to sc-elements displayed within the *Knowledge visualization and editing panel* and any other sc-elements, for example, command classes and even *natural languages* themselves, displayed within the *Component for switching the language of identification of displayed sc-elements of the ostis-meta-system*.

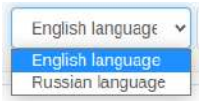

Figure 9. The Component for switching the language of identification of displayed sc-elements of the ostis-meta-system

The *Component for switching the external language of knowledge visualization* is used to switch the knowledge visualization language in the current window displayed on the *Knowledge visualization and editing panel*. In the current implementation, SCg-code and SCn-code (Fig. 10, as well as any other languages included in the *external SC-code visualization languages* set, are supported by default as such languages.

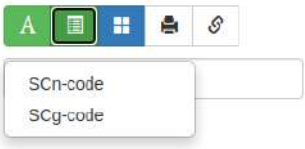

Figure 10. The Component for switching the external language of knowledge visualization of the ostis-meta-system

The *Search field for sc-elements by identifier* allows searching for sc-identifiers containing the substring en-

tered in this field (case sensitive). As a result of the search, a list of sc-identifiers containing the specified substring is displayed (Fig. 11), when interacting with them, the question “What is this?” is automatically set, the argument of which is either for the sc-element itself, which has the given sc-identifier (if the specified sc-identifier is the main or system identifier, and thus the specified sc-element can be uniquely determined), or for the internal file of the ostis-system that is the sc-identifier (in case when the given sc-identifier is not the main one).

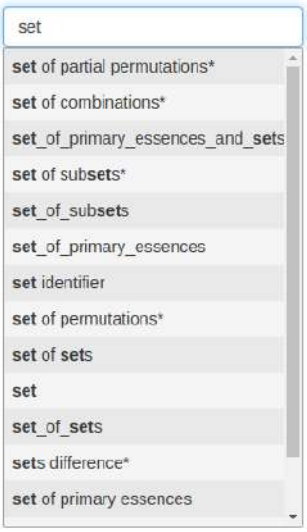

Figure 11. The Component for switching the external language of knowledge visualization of the ostis-meta-system

The *Panel for displaying the user dialog with the ostis-system* displays a time-ordered list of sc-elements (Fig. 12) that are signs of actions initiated by the user within the dialog with the ostis-system by interacting with images of the corresponding command classes (that is, if the action was initiated in another way, for example, by explicitly initiating it by creating an arc of membership to the *action initiated* set in the sc.g editor, then it will not be displayed on this panel). When the user interacts with any of the depicted action signs, the *Knowledge visualization and editing panel* displays a window containing the result of this *action* in the visualization language, in which it was displayed when the user viewed it in the last (previous) once. Thus, in the current implementation, this panel can work only if the action initiated by the user assumes the result of this action explicitly represented in memory. In turn, it follows from this that at present this panel, as well as the whole *Implementation of the interpreter of user interface sc-models*, allows working with the system only in the “question-answer” dialog mode.

The *Knowledge visualization and editing panel* displays windows containing sc-text, represented in some language from the set of *external SC-code visualization languages* and, as a rule, the result of some action initiated by the user. If the corresponding visualizer supports the ability to edit texts of the corresponding natural language, then it is also an editor at the same time. If necessary, the user interface of each specific ostis-system can be supplemented with visualizers and editors of various external languages, which in the current version of *Implementation of the interpreter of user interface sc-models* will also be located on the *Knowledge visualization and editing panel*. By default, two visualization and editing panels are available: the *Visualizer of sc.n-texts* (Fig. 13) and the *Visualizer and editor of sc.g-texts* (Fig. 14).

Figure 12. The Panel for displaying the user dialog with the ostis-meta-system

The *User interface commands menu bar* contains images of command classes (both atomic and non-atomic) currently available in the knowledge base and included in the *main user interface* decomposition (meaning the complete decomposition, which in may include several levels of non-atomic instruction classes in general) (Fig. 15). Interaction with the image of a non-atomic instruction class initiates a command for the image of instruction classes included in the decomposition of this non-atomic instruction class. Interaction with the image of an atomic command class initiates the generation of a command of this class with previously selected arguments based on the corresponding *generalized command class statement* (command class template).

The semantic models of the described user interface components are represented in more detail in [67].

C. Advantages and disadvantages represented in the current version of Implementation of the interpreter of user interface sc-models

The current implementation of the sc-interface model interpreter has a large number of shortcomings, namely:

- The idea of platform independence of the user interface (building the sc-model of the user interface) is not fully implemented. Fully describing the sc-model of the user interface (including the exact placement, size, design of components, their behavior, etc.) is currently likely to be difficult due to performance limitations, but it is quite possible to implement the ability to ask questions to all interface components, change them location, etc., however, these features cannot be implemented in the current version of the platform implementation.

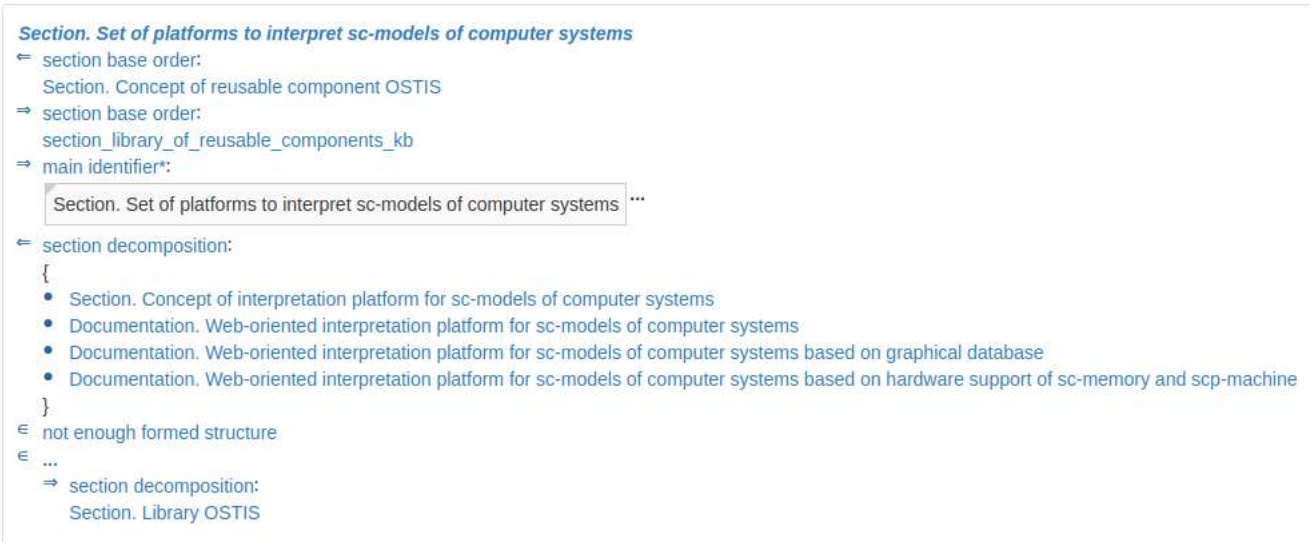

Figure 13. The Visualizer of sc.n-texts of the ostis-meta-system

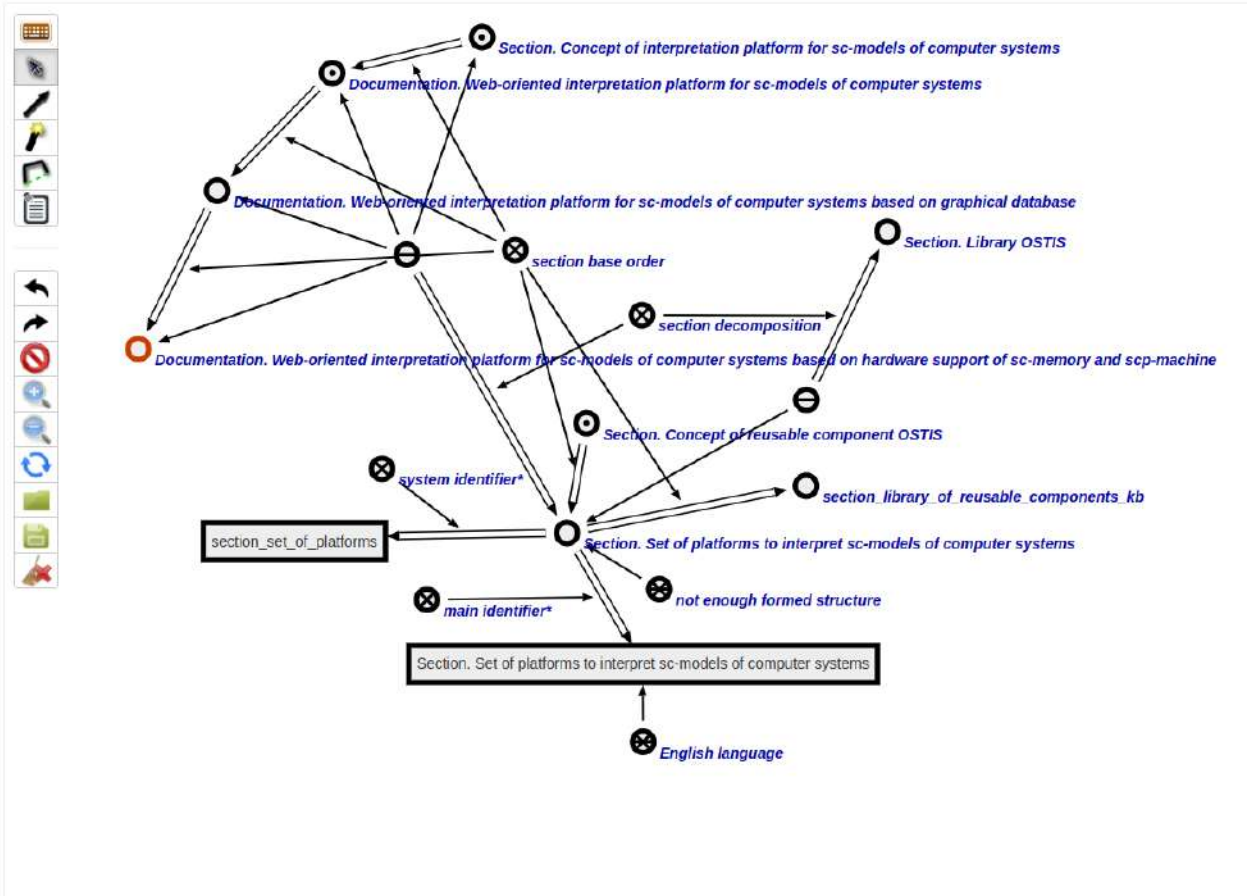

Figure 14. The Visualizer and editor of sc.g-texts of the ostis-meta-system

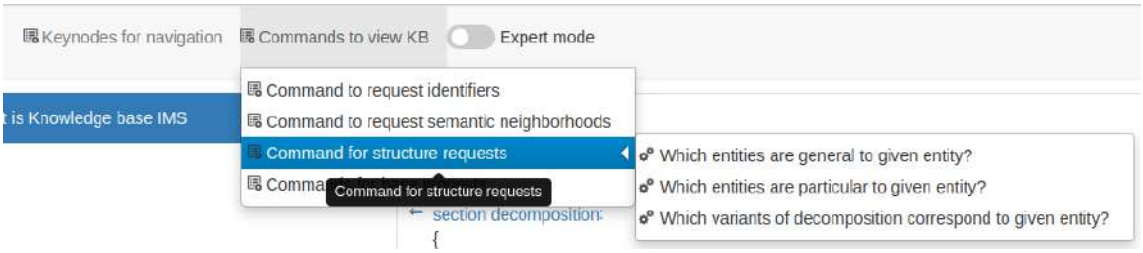

Figure 15. The User interface commands menu bar of the ostis-meta-system

- In addition, part of the interface actually works directly with sc-memory using WebSocket technology and part through an interlayer based on the tornado library for the Python programming language, which leads to additional dependencies on third-party libraries. Recently, the development of the current *Software implementation of the ostis-platform* has largely solved this problem, but there are still components implemented in Python.
- Some of the components (for example, the search field by identifier) are implemented by third-party tools and have almost nothing to do with sc-memory. This hinders the development of the platform.
- The current *Implementation of the sc-model interpreter for user interfaces* is focused only on dialog with the user (in the style of a user question – a system answer). Obviously, necessary situations are not supported, such as executing a command that does not expect a response; the occurrence of an error or lack of response; the need for the system to ask a question to the user, etc.
- Restricted user interaction with the system without the usage of special controls. For example, it is possible to ask the system a question by drawing it in the SCg-code, but the user will not see the answer, although it will be generated in memory by the corresponding agent. Most of the technologies used in the implementation of the platform are now outdated, which hinders the development of the platform.
- There is no inheritance mechanism implemented when adding new external languages. For example, adding a new language, even one that is very close to the SCg-code, requires physically copying the component code and making the appropriate changes, which results in two unrelated components that begin to develop independently of each other.
- Low level of documenting the current *Implementation of the interpreter of user interfaces sc-models*. Represented current specification only describes the key points of the *Implementation of user interfaces sc-models* but does not cover them.

Based on the shortcomings described, the following requirements are imposed on future implementation:

- Unify the principles of interaction of all interface components with the *Implementation of the sc-memory*, regardless of what type the component belongs to. For example, a list of menu commands should be formed through the same mechanism as a response to a user request, an editing command generated by the user, a command for adding a new fragment to the knowledge base, etc. It is necessary to improve the ways of using the interface for convenient and comfortable usage [68].
- Unify the principles of user interaction with the system, regardless of the mode of interaction and the external language. For example, it should be possible to ask questions and execute other commands directly through the SCg/SCn interface. At the same time, it is necessary to take into account the principles of editing the knowledge base so that the user cannot, under the guise of asking a question, enter new information into the agreed part of the knowledge base.
- Unify the principles of handling events that occur during user interaction with interface components – the behavior of buttons and other interactive components should not be set statically by third-party tools but implemented as an agent, which, nevertheless, can be implemented in an arbitrary way (not necessarily on platform-independent level). Any action performed by the user, at the logical level, must be interpreted and processed as the initiation of the agent.
- Provide the ability to execute commands (in particular, ask questions) with an arbitrary number of arguments, including without arguments.
- Make it possible to display the answer to a question in parts if the answer is very large and takes a long time to display.
- Each displayed interface component should be interpreted as an image of some sc-node described in the knowledge base. Thus, the user should be able to ask arbitrary questions to any interface components.
- Simplify and document the mechanism for adding new components as much as possible.
- Provide the ability to add new components based on existing ones without creating independent copies.

For example, it should be possible to create a component for a language that extends the SCg language with new primitives, redefine the principles for placing sc-texts, etc.
- Minimize dependency on third-party libraries.
- Minimize the usage of the HTTP protocol (bootstrap of the common interface structure), ensure the possibility of equal two-way interaction between the server and client parts.

Obviously, the implementation of most of the above requirements is associated not only with the implementation of the platform itself but also requires the development of the theory of logical-semantic models of user interfaces and the refinement of the general principles for organizing user interfaces of ostis-systems within it. However, the fundamental possibility of implementing such models should be taken into account in the *Implementation of the ostis-platform*.

XII. Plans for the development of the *Software implementation of the ostis-platform*

In the further development of the *Software implementation of the ostis-platform*, it will be important and correct to:

- maximally detail the specification of the components of the designed ostis-platform, including the languages used for external and internal knowledge representation, and clearly stratify the hierarchy of classes and relations used in describing the components of the ostis-platforms;
- eliminate and take into account the shortcomings in the implementation of new components in the designed ostis-platform, indicate possible options for their implementation;
- reduce the dependency of the ostis-platform components to a minimum, that is, if possible, implement them in the SCP language (for example, an interpreter of user interface sc-models);
- evaluate the quality of the designed system and its components as a whole.

In the direction of improving the quality and efficiency of the *Software implementation of the ostis-platform* components, the following problems will be solved first:

- The implemented sc-memory model is not intended for its usage in a multi-user mode, especially when there are more than 4 subjects of interaction with it. This, in turn, hinders the implementation of all the principles of the OSTIS Technology. All this is explained by the failure of the implementation of blocking mechanisms at the level of the memory itself. The sc-memory model will be revised and improved in such a way as to reduce the usage of blocking mechanisms and minimize the number of mutually exclusive situations for processes in sc-memory.
- Using the current *Software implementation of the ostis-platform* is quite difficult and resource-intensive. This is primarily due to the lack of the possibility for collectively developing knowledge bases. This is affected by the lack of the necessary interface components for easy editing and viewing the knowledge base. For example, the current scg-editor is quite primitive and inconvenient to use, and the tools for creating methods (programs) are not implemented at all.

XIII. Conclusion

The current implementation of the ostis-platform is a universal tool for designing next-generation computer systems. It acts as a software emulator of a semantic associative computer (!), focused on the semantic representation and processing of information of any kind. The ostis-platform acts as a program memory for any next-generation software c.s., implemented according to the principles of the OSTIS Technology, in which a logical-semantic model (knowledge) can be placed, regardless of its type and contents. Thus, on the basis of the ostis-platform, the sc-model of the OSTIS Metasystem is implemented [69], which acts as a software implementation of the OSTIS Standard [6].

Using the ostis-platform, it is possible to solve any information problems of human activity. In this sense of the word, the implemented ostis-platform is a design automation system not only for other systems but also for solving information problems of any kind in general.

In this article, the problems of ensuring the design of platforms for the design and development of other systems are considered. A comparative analysis of existing solutions in the field of design automation of c.s. and justified the chosen decision in detail. The work determines the solution of the problem in the form of designing and developing universal interpreters of logical-semantic models of systems according to the principles underlying the OSTIS Technology, named an ostis-platform. This article is also a formal specification of the first *Software implementation of the ostis-platform.*

Acknowledgment

The author would like to thank the research groups of the Departments of Intelligent Information Technologies of the Belarusian State University of Informatics and Radioelectronics and the Brest State Technical University for their help in the work and valuable comments.

References

[1] A. Iliadis, "The tower of babel problem: making data make sense with basic formal ontology," *Online Information Review*, vol. 43, no. 6, pp. 1021–1045, 2019.

[2] S. C. J. Lim, Y. Liu, Y. Chen *et al.*, "Ontology in design engineering: status and challenges," 2015.

[3] I. Ahmed, G. Jeon, and F. Piccialli, "From artificial intelligence to explainable artificial intelligence in industry 4.0: a survey on what, how, and where," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 8, pp. 5031–5042, 2022.

[4] Jeff Waters and Brenda J. Powers and Marion G. Ceruti, "Global interoperability using semantics, standards, science and technology (gis3t)," *Computer Standards & Interfaces*, vol. 31, no. 6, pp. 1158–1166, 2009.

[5] V. Golenkov, N. Guliakina, V. Golovko, and V. Krasnoproshin, "On the current state and challenges of artificial intelligence," in *International Conference on Open Semantic Technologies for Intelligent Systems*. Springer, 2022, pp. 1–18.

[6] Golenkov Vladimir and Guliakina Natalia and Shunkevich Daniil, *Open technology of ontological design, production and operation of semantically compatible hybrid intelligent computer systems*, V. Golenkov, Ed. Minsk: Bestprint [Bestprint], 2021.

[7] Sokolov A.P., Golubev A.O., "Computer-aided design system for composite materials. part 3. graph-oriented methodology for developing user-system interaction tools," *Izvestiya SPbGETU LETI*, pp. 43–57, 2021.

[8] T. S. Dillon, E. Chang, and P. Wongthongtham, "Ontology-based software engineering-software engineering 2.0," in *19th Australian Conference on Software Engineering (ASWEC 2008)*. IEEE, 2008, pp. 13–23.

[9] Dillon, Tharam and wu, Chen and Chang, Elizabeth, "Gridspace: Semantic grid services on the web-evolution towards a softgrid," in *3rd International Conference on Semantics, Knowledge, and Grid, SKG 2007*, 11 2007, pp. 7–13.

[10] V. Kabilan, "Ontology for information systems (04is) design methodology: conceptualizing, designing and representing domain ontologies," Ph.D. dissertation, KTH, 2007.

[11] A. M. Ouksel and A. Sheth, "Semantic interoperability in global information systems," *ACM Sigmod Record*, vol. 28, no. 1, pp. 5–12, 1999.

[12] F. W. Neiva, J. M. N. David, R. Braga, and F. Campos, "Towards pragmatic interoperability to support collaboration: A systematic review and mapping of the literature," *Information and Software Technology*, vol. 72, pp. 137–150, 2016.

[13] K. Lu, Q. Zhou, R. Li, Z. Zhao, X. Chen, J. Wu, and H. Zhang, "Rethinking modern communication from semantic coding to semantic communication," *IEEE Wireless Communications*, 2022.

[14] F. Zhou, Y. Li, X. Zhang, Q. Wu, X. Lei, and R. Q. Hu, "Cognitive semantic communication systems driven by knowledge graph," *arXiv preprint arXiv:2202.11958*, 2022.

[15] P. Hagoort, G. Baggio, and R. M. Willems, "Semantic unification," in *The cognitive neurosciences, 4th ed.* MIT press, 2009, pp. 819–836.

[16] J. H. Siekmann, "Universal unification," in *International Conference on Automated Deduction*. Springer, 1984, pp. 1–42.

[17] D. B. Lenat, R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd, "Cyc: toward programs with common sense," *Communications of the ACM*, vol. 33, no. 8, pp. 30–49, 1990.

[18] S. L. Reed, D. B. Lenat *et al.*, "Mapping ontologies into cyc," in *AAAI 2002 Conference workshop on ontologies for the semantic Web*, 2002, pp. 1–6.

[19] Zbigniew Gomolkaa and Boguslaw Twaroga and Ewa Zeslawskaa and Ewa Dudek-Dyduchb, "Knowledge base component of intelligent almm system based on the ontology approach," *Expert Systems with Applications*, vol. 199, p. 116975, 2022.

[20] V. V. Gribova, A. S. Kleschev, F. M. Moskalenko, V. A. Timchenko, L. A. Fedorishchev, E. A. Shalfeeva, "Iacpaas cloud platform for developing shells of intelligent services: state and development prospects," *Programmnyye produkty i sistemy*, 2018.

[21] Filippov A. A., Moshkin V. S., Shalaev D. O., Yarushkina N. G., "Unified ontological data mining platform," *System Analysis and Applied Informatics*, pp. 77–82, 2016.

[22] Yu. A. Zagorulko, "Semantic technology for the development of intelligent systems, focused on domain experts," *Ontologiya proyektirovaniya*, pp. 30–44, 2015.

[23] Gulyakina N. A., Golenkov V. V., "Graphic-dynamic models of parallel knowledge processing: principles of construction, implementation and design," in *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh system [Open semantic technologies for intelligent systems]*, G. V.V., Ed. BSUIR, Minsk, 2012, pp. 23–52.

[24] C. W. Holsapple and K. D. Joshi, "A collaborative approach to ontology design," *Communications of the ACM*, vol. 45, no. 2, pp. 42–47, 2002.

[25] Ford, Brian and Schiano-Phan, Rosa and Vallejo, Juan, *Component Design*, 11 2019, pp. 160–174.

[26] D. Shunkevich, D. Koronchik, "Ontological approach to the development of a software model of a semantic computer based on the traditional computer architecture," in *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh system [Open semantic technologies for intelligent systems]*. BSUIR, Minsk, 2021, pp. 75–92.

[27] C. Ballinger, "The teradata scalability story," *Technical report, Teradata Corporation*, 2009.

[28] "Cyc platform," 2022. [Online]. Available: https://cyc.com/platform/

[29] R. Gurunath and D. Samanta, "A novel approach for semantic web application in online education based on steganography," *International Journal of Web-Based Learning and Teaching Technologies (IJWLTT)*, vol. 17, no. 4, pp. 1–13, 2022.

[30] Rudikova L. V., Zhavnerko E. V., "About data modeling of subject-domains of a practice-oriented orientation for a universal system for storing and processing data," *System Analysis and Applied Informatics*, pp. 4–11, 2017.

[31] J. Bai, L. Cao, S. Mosbach, J. Akroyd, A. A. Lapkin, and M. Kraft, "From platform to knowledge graph: evolution of laboratory automation," *JACS Au*, vol. 2, no. 2, pp. 292–309, 2022.

[32] Ian Robinson, Jim Webber and Emil Eifrem, *Graph databases*. O'Reilly Media, Inc., 2015.

[33] Abramsky Mikhail Mikhailovich, Timerkhanov Timur Ildarovich, "Comparative analysis of the use of relational and graph databases in the development of digital educational systems," in *Vestnik NGU*. Russian Federation, Novosibirsk, Vestnik NSU, 2018, pp. 5–11.

[34] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins, "A comparison of a graph database and a relational database: a data provenance perspective," in *Proceedings of the 48th annual Southeast regional conference*, 2010, pp. 1–6.

[35] Klimanskaya E.V., "Modern platforms for intelligent information processing: Graph databases," *Nauka vchera, segodnya, zavtra*, pp. 9–16, 2014.

[36] C. Chen *et al.*, "Multi-perspective evaluation of relational and graph databases," 2022.

[37] Amir Hosein Khasahmadi and Kaveh Hassani and Parsa Moradi and Leo Lee and Quaid Morris, "Memory-based graph networks," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=r1laNeBYPB

[38] O. P. Kuznecov, *Diskretnaya matematika dlya inzhenera: Uchebnik dlya vuzov [Discrete Mathematics for an Engineer: A Textbook for High Schools]*. Moscow: Lan', 2009.

[39] Reinhard Diestel, *Graph Theory*. Hamburg, Germany: Universität Hamburg, 2017.

[40] C. A. Sen, S. Parkkonen, Yu. A. Zobni, "Application of graph databases to form knowledge bases of complex systems," in *Problemy formirovaniya yedinogo prostranstva ekonomicheskogo i sotsial'nogo razvitiya stran SNG (SNG-2017)*. Tyumen: Tyumen Industrial University, 2017, pp. 165–172.

[41] A.N.Naumov, A.M.Vendrov, V.K.Ivanov, *Database and knowledge management systems*. M.: Finance and statistics, 1991.

[42] T. A. Gavrilova, V. F. Khoroshevsky, *Knowledge bases of intelligent systems*. SPb: Peter, 2000.

[43] A. A. Bashlykov, "Knowledge management systems," in *Avtomatizatsiya, telemekhanizatsiya i svyaz' v neftyanoy promyshlennosti*, 2010, pp. 33–39.

[44] ——, "Methodology for building knowledge base management systems for intelligent systems," in *Programmnyye produkty i sistemy*, 2013, pp. 131–137.

[45] V. Golenkov and N. Guliakina and I. Davydenko and A. Eremeev, "Methods and tools for ensuring compatibility of computer systems," in *Otkrytye semanticheskie tekhnologii proektirovaniya intellek-*

tual'nykh system [Open semantic technologies for intelligent systems], V. Golenkov, Ed. BSUIR, Minsk, 2019, pp. 25–52.
[46] Golenkov V.V., Gulyakina N.A., Davydenko I.T., Shunkevich D.V., Eremeev A.P., "Ontological design of hybrid semantically compatible intelligent systems based on the semantic representation of knowledge," in *Ontologiya proyektirovaniya*, G. V.V., Ed. Russian Federation, Samara: Samara National Research University named after Academician S.P. Korolev, 2019, pp. 132–148.
[47] M. J. Jacobs, "A software development project ontology," Master's thesis, University of Twente, 2022.
[48] V.V. Gribova and A.S. Kleschev and D.A. Krylov and F.M. Moscalenko, "The basic technology development of intelligent services on cloud platform iacpaas. part 1. the development of a knowledge base and a solver of problems," *Software engineering*, no. 12, pp. 3–11, 2015.
[49] Yurii I. Molorodov, "Development of information system based on ontological design patterns," in *5th International Conference Information Technologies in Earth Sciences and Applications for Geology, Mining and Economy,*. Institute of Computational Technologies, Siberian Branch of the Russian Academy of Sciences, 2019.
[50] C. M. Zapata Jaramillo, G. L. Giraldo, and G. A. Urrego Giraldo, "Ontologies in software engineering: approaching two great knowledge areas," *Revista Ingenierías Universidad de Medellín*, vol. 9, no. 16, pp. 91–99, 2010.
[51] D. N. Koronchik, "Unificirovannye semanticheskie modeli pol'zovatel'skih interfejsov intellektual'nyh sistem i tekhnologiya ih komponentnogo proektirovaniya [Unified semantic models of user interface for intelligent systems and technology for their develop]," in *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh system [Open semantic technologies for intelligent systems]*, V. Golenkov, Ed. BSUIR, Minsk, 2013, pp. 403–406.
[52] V. P. Ivashenko and N. L. Verenik and A. I. Girel' and E. N. Sejtkulov and M. M. Tatur, "Predstavlenie semanticheskih setej i algoritmy ih organizacii i semanticheskoj obrabotki na vychislitel'nyh sistemah s massovym parallelizmom [Semantic networks representation and algorithms for their organization and semantic processing on massively parallel computers]," in *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh system [Open semantic technologies for intelligent systems]*, V. Golenkov, Ed. BSUIR, Minsk, 2015, pp. 133–140.
[53] E. Iotti, "An agent-oriented programming language for jade multi-agent systems," 2018.
[54] D. Shunkevich, "Agentno-orientirovannye reshateli zadach intellektual'nyh sistem [Agent-oriented models, method and tools of compatible problem solvers development for intelligent systems]," in *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh system [Open semantic technologies for intelligent systems]*, V. Golenkov, Ed. BSUIR, Minsk, 2018, pp. 119–132.
[55] (2022, Nov) Implementation of the sc-memory. [Online]. Available: https://github.com/ostis-ai/sc-machine
[56] R. Bayer and K. Unterauer, "Prefix b-trees," *ACM Transactions on Database Systems (TODS)*, vol. 2, no. 1, pp. 11–26, 1977.
[57] K. Tsuruta, D. Köppl, S. Kanda, Y. Nakashima, S. Inenaga, H. Bannai, and M. Takeda, "c-trie++: A dynamic trie tailored for fast prefix searches," *Information and Computation*, vol. 285, p. 104794, 2022.
[58] D. Belazzougui, P. Boldi, R. Pagh, and S. Vigna, "Fast prefix search in little space, with applications," in *European Symposium on Algorithms*. Springer, 2010, pp. 427–438.
[59] Bhumij Gupta1, Dr. M.P. Vani, "An overview of web sockets: The future of real-time communication," in *International Research Journal of Engineering and Technology (IRJET)*, 2018.
[60] A. A. Naik and M. R. Khare, "Study of "websocket protocol for real-time data transfer"," *International Reasearch Journal of Engineering and Technology*, 2020.
[61] M. Tomasetti, "An analysis of the performance of websockets in various programming languages and libraries," *Available at SSRN 3778525*, 2021.
[62] Q. Liu and X. Sun, "Research of web real-time communication based on web socket," 2012.
[63] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, "Foundations of json schema," in *Proceedings of the 25th International Conference on World Wide Web*, 2016, pp. 263–273.
[64] T. Marrs, *JSON at work: practical data integration for the web*. " O'Reilly Media, Inc.", 2017.
[65] Myers B.A., Rosson M.B., "Survey on user interface programming," in *Proceedings SIGCHI'92: Human Factors in Computing Systems*. Monterrey, CA, 1992, pp. 195–202.
[66] (2022, Niv) Implementation of the interpreter of user interface sc-models. [Online]. Available: https://github.com/ostis-ai/sc-web
[67] M. Sadouski, "Semantic-based design of an adaptive user interface," in *International Conference on Open Semantic Technologies for Intelligent Systems*. Springer, 2022, pp. 165–191.
[68] J. Kong, W. Zhang, N. Yu, and X. Xia, "Design of human-centric adaptive multimodal interfaces," *Int. J. Hum.-Comput. Stud.*, vol. 69, pp. 854–869, 12 2011.
[69] (2022, Nov) OSTIS Metasystem. [Online]. Available: https://ims.ostis.net

Программная платформа для интеллектуальных компьютерных систем нового поколения

Зотов Н.В.

Данная работа посвящена проблемам обеспечения проектирования семантически совместимых компьютерных систем и их независимости от реализации платформ проектирования таких систем. Работа показывает высокий уровень значимости проектирования и реализации платформ нового поколения, а также определяет решение задачи в виде проектирования и разработки универсальных интерпретаторов логико-семантических моделей систем по принципам, лежащим в основе Технологии OSTIS. Вторая часть работы отражает текущее состояние реализуемой платформы, приводит достоинства и недостатки реализуемых в ней компонентов, предлагает пути совершенствования платформы.

Received 28.10.2022