Design Principles, Structure, and Development Prospects of the Software Platform of ostis-systems

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

***Abstract*—In the article, the principles of design and development of the basic *Software implementation of the ostis-platform* are described. The advantages of the ontological approach to documenting software systems of this type are shown. The structure, problems, and prospects of developing the *Software implementation of the ostis-platform* are described.**

***Keywords*—ontological design, automation tools for design and development of computer systems, knowledge base management system, universal interpreter, graph storage, ostis-platform**

I. Introduction

Modern *software computer systems* should operate not just with *data* but with *knowledge*. To understand the *meaning* of knowledge, it is necessary to represent this *knowledge* in an understandable form for any *cybernetic system*: as for any *human*, so for any *artificial system* [1]. At the same time, the form of representing this knowledge must be unified and independent of the platform on which this knowledge can be interpreted. Nevertheless, computer systems remain dependent on highly qualified specialists and experts in subject domains in which the automation of the design of these systems is carried out; therefore, the implementation of these systems requires significant resources [2]. One of the reasons for this is the need for a computer system to work on different platforms, each of which, in general, may have its own characteristics and limitations, which must be taken into account at the implementation stage.

The solution of these problems is **design and development of fundamentally new platforms**, which should provide:

- *unambiguity of interpretation* and representation of software system models provided by the unified knowledge representation language and platform design ontology used;
- *semantic compatibility* of software system models and their components [3], including interoperability between them [4];
- *platform independence* of software system models implemented and interpreted on it;
- *simplicity* and *extensibility* of their functionality;
- *functional completeness* for creating software system models due to the presence of a formal methodology for designing its implementation;
- segregation of duties between platform components.

II. Proposed approach to the design of systems for automating the design of software systems

The shortcomings of modern computer systems for design automation of other software systems, ways to solve them, as well as the approach to the solution described below were described earlier in the works [5] and [6].

Despite the vast variety of classical technologies used by mankind, there is no general solution that allows solving the problem in a complex. At the moment, the described problems can only be solved with the help of a general and universal solution — ***OSTIS Technology*** [7]. The *OSTIS Technology* is based on a unified variant of information encoding based on *semantic networks* with a basic set-theoretic interpretation, called *SC-code* [8]. The language of semantic representation of knowledge is based on two formalisms of discrete mathematics: *Set Theory* — defines the semantics of the language — and *Graph Theory* — defines the syntax of the language. Any types and models of knowledge can be described using *SC-code* [7].

One of the key principles of the *OSTIS Technology* [9] is providing ***platform independence*** of *ostis-systems* [10], i.e. strict separation of the *logical-semantic model of a cybernetic system* (*sc-model of a cybernetic system*) and the *platform for interpreting sc-models of a cybernetic system* (*ostis-platform*). The advantages of such a strict separation are quite obvious:

- transfer of the *ostis-system* from one ostis-platform to another is carried out with minimum overhead

costs (in the ideal case — comes down to simply loading the *sc-model of a cybernetic system* onto the ostis-platform);
- components of *ostis-systems* become universal, that is, can be used in any ostis-systems where their use is appropriate;
- the development of the ostis-platform and the development of sc-models of systems can be carried out in parallel and independently of each other, in the general case by separate independent development teams according to their own rules and methods [11].

logical-semantic model of a cybernetic system
:= [formal model (formal description) for the functioning of a cybernetic system, consisting of (1) a formal model of information stored in the memory of a cybernetic system and (2) a formal model of a collective of agents that process this information.]
⊃ *sc-model of a cybernetic system*
  := [logical-semantic model of a cybernetic system represented in the SC-code]
  := [logical-semantic model of an ostis-system, which, in particular, can be a functionally equivalent model of any cybernetic system that is not an ostis-system]

ostis-system
⊂ *subject*
⇒ *generalized decomposition**:
  {• *sc-model of a cybernetic system*
   • *ostis-platform*
  }

sc-model of a cybernetic system
⇒ *generalized decomposition**:
  {• *sc-memory*
   • *sc-model of the knowledge base*
   • *sc-model of the problem solver*
   • *sc-model of a cybernetic system interface*
  }

ostis-platform
:= [platform for interpreting sc-models of computer systems]
:= [interpreter for sc-models of cybernetic systems]
:= [interpreter of unified logical-semantic models of computer systems]
:= [family of platforms for interpreting sc-models of computer systems]
:= [platform for implementing sc-models of computer systems]
:= ["empty" ostis-system]
:= [sc-machine implementation]
⊂ *platform-dependent reusable ostis-systems component [11]*

sc-memory
:= [abstract sc-memory]
:= [sc-storage]
:= [semantic memory storing SC-code constructions]
:= [storage of SC-code constructions]

In general, *sc-memory* implements the following functions:
- storage of *SC-code* constructions;
- storage of information constructions (files) external to *SC-code*. In general, file storage can be implemented differently from storage of *sc-constructions*;
- access (reading, creating, deleting) to *SC-code* constructions, implemented through the corresponding *software* (*hardware*) *interface*. Such an interface is essentially a *microprogramming language* that makes it possible to implement on its basis more complex procedures for processing stored constructions, the set of which essentially determines the list of commands of such a *microprogramming language*. The *sc-memory* itself is passive in this regard and simply executes commands initiated from outside by some subjects.

Despite all the advantages of *graph databases* in comparison with *relational databases* [12], [13], *new generation computer software systems*, due to their properties, [14] should operate not simply with *data*, but *knowledge*. To understand the *meaning* of knowledge, it is necessary to represent this knowledge in an understandable form for any kind of *cybernetic system* [14]. Speaking about the unification of the representation of all *types of knowledge*, it is considered important to use *graph databases* not just as a means for storing *structured data*, but for storing *semantically coherent* and *related* knowledge among themselves. Therefore, *sc-memory* is based on a graph representation of data and knowledge.

III. Principles underlying the Software platform of ostis-systems

The specification of such a complex program object as the *ostis-platform* must be represented in some formal *knowledge representation language*, in this case, in the *SC-code*, the texts of which it stores and processes. The language that should describe the *Software implementation of the ostis-platform* should be a *sublanguage** of the *SC-code*, i.e. it should inherit all the properties of the *Syntax* and *Denotational semantics of the SC-code* [15]. This representation of *software computer systems* specifications gives certainly strong advantages over other possible representations of specifications [16]:
- The language whose texts the system stores and processes and the language that specifies how the system represents the texts of the first language in its own memory are subsets of the same language. This simplifies not only the understanding of the developer who develops a complex *software computer system*, due to the fact that the form of representation of the language processed by this system and the

language of its specification are unified, but also allows discovering new functionalities for this system in cognition of itself. Thus, this approach allows full implementation of *intelligent computer system* properties, for example, *reflexivity*.
- It is impossible to design and implement *intelligent computer systems* on a *software computer system* that is not itself one. Representing the system specification in this form allows significantly increasing the level of its *intelligence* [14].
- There is no need to create additional tools for verification and analysis of the operation of the entire system, since the representation form of the system description language is unified with the language whose texts it stores and processes. This allows not only reducing the number of tools used in the design and implementation of the *ostis-platform* but also allows unifying the information stored in the *ostis-platform* and describing the *ostis-platform* with the purpose of using this information in the evolution of *ostis-platform* components. At the same time, the *ostis-platform* specification remains *platform-independent*, so when changing one implementation of the *ostis-platform* to another, an approach to describing the *ostis-platform* remains the same.

Software implementation of the ostis-platform
:= [Implementation of the sc-machine]
⇒ *frequently used sc-identifier**:
  [Software platform of ostis-systems]
:= [Basic software platform for mass creation of next-generation intelligent computer systems]
:= [Our proposed software implementation of an associative semantic computer]
:= [sc-machine]
∈ *specialized ostis-platform*
∈ *web-based implementation of the ostis-platform*
  := [an option of implementing a platform for interpreting sc-models of computer systems, involving the interaction of users with the system via the Internet]
∈ *multi-user ostis-platform implementation*
∈ *reusable ostis-systems component stored as source files*
∈ *non-atomic reusable ostis-systems component*
∈ *dependent reusable ostis-systems component*
⇒ *component address**:
  [https://github.com/ostis-ai/sc-machine]
⇒ *software system decomposition**:
  {• *Implementation of memory in the ostis-platform*
   • *Implementation of the subsystem of interaction with the external environment using languages of network interaction*
   • *Implementation of the interpreter for sc-models of user interfaces*
   • *Implementation of the basic set of platform-specific sc-agents and their common components*
   • *Implementation of the manager of ostis-systems reusable components*
  }
⇒ *component dependencies**:
  {• *Implementation of memory of the ostis-platform*
   • *Implementation of the subsystem of interaction with the external environment using languages of network interaction*
   • *Implementation of the interpreter for sc-models of user interfaces*
  }

Software implementation of the ostis-platform
⇒ *underlying principles**:
- The current *Software implementation of the ostis-platform* is web-oriented, so from this point of view, each ostis-system is a web site accessible online through the usual 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 allows multiple users to work with the system in parallel.
- The implementation is *cross-platform* and can be built from source texts on various *operating systems*. At the same time, the interaction between the client and server parts is organized in such a way that a web-interface can be easily replaced with a desktop or mobile interface, both universal and specialized ones.
- The current *Software implementation of the ostis-platform* is customized, i.e. does not include the *Implementation of the SCP Language interpreter*. At the current stage of development of the *Software implementation of the ostis-platform*, all functioning *ostis-systems* are *platform-dependent*. This problem is primarily related to the shortcomings of the chosen and implemented sc-memory access control model, which does not allow fully creating distributed collectives of *sc-agents* working on sc-memory.
- The *Core of the platform* is the *Implementation of memory of the ostis-platform*, which can simultaneously interact with both *Implementation of the interpreter for sc-models of user interfaces* and with any third-party applications according to the corresponding languages of

network interaction (network protocols). From the point of view of the overall architecture, *Implementation of the interpreter for sc-models of user interfaces* acts as one of many possible external components that interact with the *Implementation of memory of the ostis-platform* over the network. From the point of view of the overall architecture, *Implementation of the interpreter for sc-models of user interfaces* acts as one of many possible external components that interact with the *Implementation of memory of the ostis-platform* over the network. The current *Implementation of the interpreter for sc-models of user interfaces of ostis-systems in the Software implementation of the ostis-platform* is *platform-dependent*, since the interpreter of the basic *SCP Language* [17] is not fully implemented.

- The current *Implementation of memory in the ostis-platform* allows storing and representing *sc-constructions* that describe any *sc-model of the ostis-system*, external *information constructions*, not belonging to the *SC-code*, as well as providing different levels of access for processing these constructions. In the context of this *Software implementation of the ostis-platform*, *Implementation of memory in the ostis-platform* consists of such components as: *Implementation of ostis-platform sc-memory*, inside which sc-constructions for *sc-models of ostis-systems* are represented, *Implementation of ostis-platform file memory*, inside which external *information constructions* are represented that do not belong to the *SC-code*, i.e. the contents of *ostis-system internal files*, but additionally describe, explain, and detail *sc-constructions* for *sc-models of ostis-systems*.
- Current *Software implementation of the ostis-platform* includes Implementation of the manager of reusable ostis-systems components. This is connected with the fact that the current *Implementation of the manager of reusable ostis-systems component* uses *Implementation of memory of the ostis-platform* to store and process the specification of installed components, regardless of their implementation language.

Principles underlying the Software implementation of the ostis-platform are only basic, all components included in the *Software implementation of the ostis-platform* have their own implementation features, as well as analogues that must be taken into account when implementing the entire *ostis-platform*.

IV. Principles of documenting the Software platform of ostis-systems

Permanent reengineering of the components of the current *Software implementation of the ostis-platform* is provided by an open team of developers, while each component being developed is documented according to generally accepted principles.

Software implementation of the ostis-platform
$\Rightarrow$ *documentation principles**:

- Regardless of the implementation language of each *Software implementation of the ostis-platform* component, the specification of each component includes a specification directly described in the source files of the component itself, describing the programming interface of this component, as well as a specification as part of the ostis-platform knowledge base, describing in detail the implementation of this component, including the algorithms used. At the same time, duplication in the specification for the *Software implementation of the ostis-platform* components is strictly prohibited. So, for example, the specification, which is directly located in the source file with the implementation of the components themselves, describes the features of using the components from the point of view of an external or internal (that is, being part of the team) developer, and the specification, which is part of the sc-text for the knowledge base of the *Software implementation of the ostis-platform*, additionally includes features, proposed approaches to implementation, as well as the advantages and disadvantages of the components included in the composition.
- Each component of the *Software implementation of the ostis-platform* is described by means of the OSTIS Technology, that is, in the SC-code, the texts of which it processes and stores. Thus, it enables the platform to analyze its state and help maintain its life cycle without the participation of its developers. *Software implementation of the ostis-platform* acts as a full-fledged subject that is directly involved in its own development.
- *Specification of the Software implementation of the ostis-platform* is an *sc-language*, i.e. a sublanguage of the *SC-code*, for which the *Syntax* and *Denotational semantics of the SC-code* are specified. This *sc-language* can be represented as a family of more specific sc-languages that allow describing:
    - how *sc-constructions* are represented inside the *ostis-platform sc-memory*;

- how *information constructions* that do not belong to the *SC-code* are represented within the *file memory of the ostis-platform*;
- how different *ostis-platform* subsystems interact with each other;
- which methods and their corresponding agents interact with the *ostis-platform sc-memory*;
- how various interpreters for *sc-models of ostis-systems* (knowledge base, solver, interface) are represented and work;
- and so on.

This approach makes it possible to integrate descriptions of various components [18] that are part of the *Software implementation of the ostis-platform* without any particular obstacles, since the entire *Specification of the Software implementation of the ostis-platform* is its knowledge base with a clearly defined hierarchy of subject domains and ontologies (that is, *sc-languages* that describe its implementation).

- Each developer of the *Software implementation of the ostis-platform* takes care of the permanent support of not only the state of its components but also the specification of these components. A quality of *Software implementation of the ostis-platform* is guaranteed by its team of developers who are able not only to understand the implementation details of the *ostis-platform* but also to contribute to the creation of mutually beneficial cooperation to achieve the set goals.

These principles can be used to describe any other *software computer systems*, including those *software computer systems* that are not implemented on this *ostis-platform*.

V. Structure of the Software platform of ostis-systems

Implementation of memory in the ostis-platform
:= [Implementation of ostis-platform sc-memory and file memory]
:= [Our proposed software implementation of ostis-platform sc-memory and file memory]
∈ *reusable ostis-systems component stored as source files [11]*
∈ *non-atomic reusable ostis-systems component*
∈ *dependent reusable ostis-systems component*
⇒ *component address**:
  [https://github.com/ostis-ai/sc-machine/tree/main/sc-memory]
⇒ *software system decomposition**:
  {• *Implementation of ostis-platform sc-memory*
   • *Implementation of ostis-platform file memory*
  }
⇒ *component dependencies**:
  {• *GLib library of methods and data structures*
   • *C++ Standard Library of methods and data structures*
   • *Implementation of ostis-platform sc-memory*
   • *Implementation of ostis-platform file memory*
  }

Implementation of ostis-platform sc-memory
:= [Software implementation of graphodynamic associative memory in the Software ostis-systems platform]
:= [Our proposed implementation of graphodynamic associative memory for ostis-systems]
∈ *sc-memory implementation*
∈ *reusable ostis-systems component stored as source files*
∈ *atomic reusable ostis-systems component*
∈ *dependent reusable ostis-systems component*
⇐ *software model**:
  sc-memory
    ⇐ *family of subsets**:
      sc-memory segment
        := [page of sc-memory]
        ⇐ *family of subsets**:
          sc-memory element
⇒ *component dependencies**:
  {• *GLib library of methods and data structures*
   • *C++ Standard Library of methods and data structures*
  }
⇒ *programming language used**:
  • *C*
  • *C++*
⇒ *internal language**:
  • *SCin-code*

In general, *sc-memory* can be implemented in different ways. So, for example, another version of *ostis-platform sc-memory* can be implemented using the software implementation of the *Neo4j Platform* [19]. The difference between this possible implementation of *sc-memory* and the current one is that the storage of *graph constructions* and the management of the flow of actions on them should be carried out to a greater extent by means provided by the *Neo4j Platform*; at the same time, the representation of *graph constructions* must be implemented in its own way, since it depends on the *Syntax of the SC-code*.

Such an sc-memory model can be easily described in the *sc-language*, that is, in the sublanguage of the *SC-code*. Such a language allows describing how texts of a language are represented inside the memory of the *ostis-platform* in the same language. At the same time, not only the unification of representing information processed by the *ostis-platform* and information describing the *ostis-platform* itself is observed, but also opportunities are given for expanding and using the language in the process of evolution of the *ostis-platform* and its components, including those in the process of evolution of *Implementation of ostis-platform sc-memory*.

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

should be distinguished*
∋ {• *SC-code*
      := [Universal language of internal semantic representation of knowledge in memory of ostis-systems]
  • *SCin-code*
      := [Metalanguage for describing the representation of the SC-code in ostis-platform sc-memory]
      ⊂ *SC-code*
  }

Software interface of Implementation of ostis-platform sc-memory
⇐ *software interface**:
  Implementation of ostis-platform sc-memory
∈ *software interface*
∈ *reusable ostis-systems component stored as source files*
∈ *atomic reusable ostis-systems component*
∈ *dependent reusable ostis-systems component*
⇒ *component dependencies**:
  {• *GLib library of methods and data structures*
   • *C++ Standard Library of methods and data structures*
  }
⇒ *method representation language used**:
  • *C*
  • *C++*
⊃ *Software interface for information-forming methods of Implementation of ostis-platform sc-memory*
  := [information-forming methods of Implementation of ostis-platform sc-memory]
  := [subsystem that is part of the implementation of ostis-platform sc-memory, which allows creating, modifying, and deleting constructions of sc-memory]
  ⇐ *software interface**:
    Implementation of the information-generating subsystem of Implementation of ostis-platform sc-memory
      ⊂ *Implementation of ostis-platform sc-memory*
⊃ *Software interface for information retrieval methods of Implementation of ostis-platform sc-memory*
  := [information retrieval methods of Implementation of ostis-platform sc-memory]
  := [subsystem that is part of Implementation of ostis-platform sc-memory that allows finding constructions in sc-memory]
  ⇐ *software interface**:
    Implementation of the information retrieval subsystem of Implementation of ostis-platform sc-memory
      ⊂ *Implementation of ostis-platform sc-memory*

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

SCfin-code
:= [Semantic Code file interior]

:= [Language for describing the representation of information constructions that do not belong to the SC-code inside the ostis-platform file memory]
:= [Metalanguage for describing the representation of information constructions that do not belong to the SC-code inside the ostis-platform file memory]
⇒ *frequently used sc-identifier**:
  [sc.fin-text]
  ∈ *common noun*
∈ *abstract language*
∈ *metalanguage*
∈ *sc-language*
⊂ *SC-code*
⊃ *ostis-platform file memory*

should be distinguished*
∋ {• *SC-code*
      := [Universal language of internal semantic representation of knowledge in memory of ostis-systems]
   • *SCfin-code*
      := [Metalanguage for describing the representation of external information constructions that do not belong to the SC-code in ostis-platform file memory]
      ⊂ *SC-code*
  }

should be distinguished*
∋ {• *SCin-code*
      := [Metalanguage for describing the representation of the SC-code in ostis-platform sc-memory]
      ⊂ *SC-code*
   • *SCfin-code*
      := [Metalanguage for describing the representation of external information constructions that do not belong to the SC-code in ostis-platform file memory]
      ⊂ *SC-code*
  }

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

Implementation of the network interaction subsystem with sc-memory based on JSON in the ostis-platform
:= [Subsystem for interacting with sc-memory based on the JSON format]
:= [Network software interface of Implementation of ostis-platform sc-memory]
:= [Our proposed option of implementing the mechanism for accessing the ostis-platform sc-memory in a distributed collective of ostis-systems]
∈ *reusable ostis-systems component stored as source files*
∈ *non-atomic reusable ostis-systems component*
∈ *dependent reusable ostis-systems component*
∈ *client-server system*
⇒ *method representation language used**:
  • *C*
  • *C++*
  • *Python*
  • *TypeScript*
  • *C#*
  • *Java*
⇒ *language used**:
  • *SC-JSON-code*
⇒ *software system decomposition**:
  {• *Implementation of the Server System based on Websocket and JSON, providing network access to memory of the ostis-platform*
   {}
   = {• *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*
     }
  }

SC-JSON-code
:= [Semantic JSON-code]
:= [Semantic JavaScript Object Notation code]
:= [Metalanguage for describing the representation of messages between subsystems of the ostis-platform]
⇒ *frequently used sc-id**:
  [sc-json-text]
:= [The language we propose for interaction in a distributed collective of ostis-systems]
  ∈ *common noun*

∈ *abstract language*
⊂ *SC-code*
⊂ *JSON*

Implementation of the Server System based on Websocket and JSON, providing network access to memory of the ostis-platform
:= [Implementation of a 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 sc-identifier**:
[sc-server]
:= [sc-server]
∈ *reusable ostis-systems component stored as source files*
∈ *atomic reusable ostis-systems component*
∈ *dependent reusable ostis-systems component*
⇒ *method representation language used**:
- *C*
- *C++*

⇒ *language used**:
- *SC-JSON-code*

⇒ *component address**:
[https://github.com/ostis-ai/sc-machine/sc-tools/sc-server]
⇒ *component dependencies**:
{
- *Library of software components for processing json texts*
- *Library of cross-platform software components for implementing server applications based on Websocket*
- *Software component for setting up software components of ostis-systems*
- *Implementation of sc-memory*

}

Implementation of the interpreter for sc-models of user interfaces
:= [Our proposed interpreter for interpreting sc-models of ostis-systems user interfaces]
∈ *reusable ostis-systems component stored as source files*
∈ *non-atomic reusable ostis-systems component*
∈ *dependent reusable ostis-systems component*
⇒ *method representation language used**:
- *JavaScript*
- *TypeScript*
- *Python*
- *HTML*
- *CSS*

⇒ *component address**:
[https://github.com/ostis-ai/sc-web]
⇒ *component dependencies**:
{
- *Library of standard interface components in the JavaScript programming language*
- *Library for implementing server applications in the Python Tornado programming language*
- *Implementation of the client system in the TypeScript programming language*
- *Implementation of the client system in the Python programming language*

}

VI. Prospects for developing the Software platform of ostis-systems

Software implementation of the ostis-platform
⇒ *prospects for development**:
- Despite the fact that the *Implementation of ostis-platform sc-memory* is functionally complete for the development of *semantically compatible interoperable ostis-systems* and is *multi-user*, i.e. it can execute *actions* of different users in parallel, significant restrictions are imposed on the *actions* of these users. First of all, these restrictions are connected not so much with the memory model underlying the implementation but with the model of asynchronous access to it. The implemented model of asynchronous memory access requires blocking access to a group of related *sc-elements* and not to a particular one of these *sc-elements*. For example, to create an *outgoing sc-arc* from a given *sc-element*, it is necessary to lock not only the cell in memory in which this sc-element is stored but also the initial *incoming* and *outgoing sc-connectors* from the list of *incoming* and *outgoing sc-connectors* of this sc-element, respectively. In the process of parallel operation of sc-memory, blunders can often occur: deadlocking of processes performing actions on the same sc-elements, resource races on the same sc-elements, etc. To eliminate these problems, a transition to a new model of asynchronous access to sc-memory is required or a transition to a new implementation of sc-memory without changing the existing programming interface for the implementation of sc-memory.
- The current *Software implementation of the ostis-platform* is *customized* and does not include the *Implementation of the SCP Language interpreter* (that is, when the *ostis-platform* is running, the *SCP Language interpreter* is not used), which hinders the development of *platform-independent ostis-systems*. This is in no way related to the complexity of developing

this kind of interpreter. On the contrary, the problem lies in the model of asynchronous access to sc-memory, which prevents the collection of sc-agents that are part of the *Implementation of the SCP Language interpreter* to work smoothly. To implement a full-fledged collective of *ostis-systems* interacting with each other, it is necessary to transfer the *Software implementation of the ostis-platform* from the *specialized ostis-platforms* class to the *basic ostis-platforms* class. Thus, it is necessary to switch to a new version of the *ostis-platform* (not a modification (!)), which will contain the current *Implementation of the SCP Language interpreter*.

- The current *Implementation of ostis-platform sc-memory* is efficient for storing large amounts of *knowledge* in *ostis-systems knowledge bases*. However, in information retrieval problems, rather complex tools and subsystems are required to ensure the most effective solution of these problems. So, for example, to find all pairs of a given relation whose first component is a given sc-element, it is necessary to check the entire list of outgoing sc-connectors of a given sc-element, including those sc-connectors that do not have the specified syntactic or semantic sc-element class. The solution to this problem is possible by modifying the existing *Implementation of ostis-platform sc-memory*, namely, the implementation of a new sc-memory model (for example, on the file system of the modern Linux operating system).
- Implementation of the *OSTIS Ecosystem* [20], [21] requires strong development of the *Implementation of the subsystem for interacting with the external environment using languages of network interaction*, with the help of which ostis-systems, which are developed on the current *Software implementation of the ostis-platform*, will be able to fully communicate with each other. The transition of *Software implementation of the ostis-platform* from the class of server platforms to the class of client-server platforms is required.

VII. Conclusion

Let us briefly list the main provisions of this work:

- The current *Software implementation of the ostis-platform* is cross-platform, which allows:
  - developing and maintaining the state of its components, regardless of the implementation of the platforms on which the tools for their design and development are used;
  - using it to solve problems on any available devices.
- The current *Software implementation of the ostis-platform* is multi-user, that is, it allows processing several actions at the same time.
- The current *Implementation of memory in the ostis-platform* is complete enough to:
  - one-to-one interpret *sc-models of ostis-systems*, including external information constructions that do not belong to the SC-code;
  - develop platform-specific components that require access to sc-memory (for example, the *Software interface of Implementation of ostis-platform sc-memory*).
- The current *Software implementation of the ostis-platform* is specialized, that is, it allows creating only platform-dependent ostis-systems.
- On the basis of the current *Software implementation of the ostis-platform*, the *interpreter of sc-models of ostis-systems user interfaces*, *interpreter of logical models for solving problems in ostis-systems*, as well as the *manager of reusable ostis-systems components* are used.

In this article, the principles underlying the Software implementation of the ostis-platform, the principles of documenting its components, as well as the prospects for further development are described.

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] V. Golenkov, N. Guliakina, V. Golovko, V. Krasnoproshin, “Methodological problems of the current state of works in the field of artificial intelligence,” *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual’nykh system [Open semantic technologies for intelligent systems]*, pp. 17–24, 2021.

[2] A. Sokolov, A. Golubev, “Sistema avtomatizirovannogo proektirovaniya kompozicionnyh materialov. CHast’ 3. Grafoorientirovannaya metodologiya razrabotki sredstv vzaimodejstviya pol’zovatel’-sistema [System of computer-aided design of composite materials. Part 3. Graph-oriented methodology for the development of user-system interaction tools],” *Izvestiya SPBGETU LETI*, pp. 43–57, 2021.

[3] V. Golenkov, N. Guliakina, I. Davydenko, A. Eremeev, “Methods and tools for ensuring compatibility of computer systems,” *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual’nykh system [Open semantic technologies for intelligent systems]*, pp. 25–52, 2019.

[4] A. M. Ouksel and A. Sheth, “Semantic interoperability in global information systems,” *SIGMOD Rec.*, vol. 28, no. 1, p. 5–12, Mar. 1999.

[5] N. Zotov, “Software platform for next-generation intelligent computer systems,” *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual’nykh system [Open semantic technologies for intelligent systems]*, pp. 297—-326, 2022.

[6] D. Shunkevich, D. Koronchik, “Ontological approach to the development of a software model of a semantic computer based on the traditional computer architecture,” *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual’nykh system [Open semantic technologies for intelligent systems]*, pp. 75–92, 2021.
[7] V. Golenkov, N. Gulyakina, and D. Shunkevich, *Otkrytaya tekhnologiya ontologicheskogo proektirovaniya, proizvodstva i ekspluatatsii semanticheski sovmestimykh gibridnykh intellektual’nykh komp’yuternykh sistem [Open technology of ontological design, production and operation of semantically compatible hybrid intelligent computer systems]*, V. Golenkov, Ed. Minsk: Bestprint, 2021.
[8] V. Ivashenko, “General-purpose semantic representation language and semantic space,” *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual’nykh system [Open semantic technologies for intelligent systems]*, pp. 41–64, 2022.
[9] V. Golenkov, N. Guliakina, G. V., and E. V., “The standardization of intelligent computer systems as a key challenge of the current stage of development of artificial intelligence technologies,” *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual’nykh system [Open semantic technologies for intelligent systems]*, pp. 73–88, 2018.
[10] D. Shunkevich, “Universal model of interpreting logical-semantic models of intelligent computer systems of a new generation,” *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual’nykh system [Open semantic technologies for intelligent systems]*, p. 285–296, 2022.
[11] M. Orlov, “Comprehensive library of reusable semantically compatible components of next-generation intelligent computer systems,” *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual’nykh system [Open semantic technologies for intelligent systems]*, pp. 261–272, 2022.
[12] 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.
[13] C. Chen *et al.*, “Multi-perspective evaluation of relational and graph databases,” 2022.
[14] A. Zagorskiy, “Factors that determine the level of intelligence of cybernetic systems,” *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual’nykh system [Open semantic technologies for intelligent systems]*, p. 13–26, 2022.
[15] V. V. Golenkov, “Ontology-based design of intelligent systems,” *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual’nykh system [Open semantic technologies for intelligent systems]*, pp. 37–56, 2017.
[16] T. Dillon, E. Chang, and P. Wongthongtham, “Ontology-based software engineering-software engineering 2.0,” 04 2008, pp. 13–23.
[17] D. Shunkevich, “Ontology-based design of hybrid problem solvers,” *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual’nykh system [Open semantic technologies for intelligent systems]*, pp. 101–131, 2022.
[18] N. Zotov, “Semantic theory of programs in next-generation intelligent computer systems,” *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual’nykh system [Open semantic technologies for intelligent systems]*, pp. 297—-326, 2022.
[19] Ian Robinson, Jim Webber and Emil Eifrem, *Graph databases*. O’Reilly Media, Inc., 2015.
[20] A. Zagorskiy, “Principles for implementing the ecosystem of next-generation intelligent computer systems,” *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual’nykh system [Open semantic technologies for intelligent systems]*, p. 347–356, 2022.
[21] T. Dillon, C. wu, and E. Chang, “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.

Принципы проектирования, структура и перспективы развития Программной платформы ostis-систем

Зотов Н. В.

Данная работа является краткой спецификацией текущего Программного варианта реализации ostis-платформы. Работа показывает принципы, структуру и перспективы развития программной платформы для логико-семантических моделей систем, построенных по принципам Технологии OSTIS.

Received 13.03.2023