Universal model of interpreting logical-semantic models of intelligent computer systems of a new generation

Daniil Shunkevich
Belarusian State University of Informatics and Radioelectronics
Minsk, Belarus
Email: shunkevich@bsuir.by

***Abstract*—In the article, an approach to solving the problem of the platform independence of computer systems is considered, which assumes unification of the principles for the implementation of such systems and ensuring their semantic compatibility based on the OSTIS Technology. The formalized system of concepts is given, that defines the principles of the implementation of this approach, including the principles for the implementation of the hardware platform for the implementation of systems built on the basis of the OSTIS Technology – an associative semantic computer.**

***Keywords*—OSTIS Technology, platform independence, ontology, associative semantic computer.**

I. Introduction

In general, the development of any artificial system, in particular, an *intelligent computer system*, involves the execution of two stages:

- the design stage, that is, the building of a formal model of the system, sufficient to understand the principles of its configuration and perform the subsequent stage of its implementation;
- the implementation stage, that is, the direct realization of the developed model using specific means (tools, materials, components, etc.). In the case of computer systems, the execution of this stage usually involves the selection of particular programming languages, libraries, third-party tools such as DBMS and various services, etc., as well as the programming and debugging of the system using the chosen means.

For each of these stages, distinct methods as well as automation tools for the corresponding processes may exist.

If the design stage of a computer system usually requires the participation of highly qualified specialists and experts in the subject domains in which automation is carried out, then the implementation stage, on the one hand, is usually simpler (in case of high-quality execution of the design stage) and, on the other hand, requires significant resources. One of the reasons for this is the need for a computer system to work on various platforms (devices), each of which, in general, may have its own features and limitations that need to be taken into account at the implementation stage. The solution to this problem is to ensure the platform independence (or cross-platform compatibility) of the computer systems being developed.

II. Analysis of modern approaches to ensuring the platform independence

The idea of ensuring the platform independence is widely used in modern computer systems for a long time. This problem is usually considered at two levels:

- the problem of enabling the work of the <u>software</u> system in different operation systems;
- the problem of ensuring the compatibility of the operation system with various hardware architectures. To solve this problem, different builds of the operation system kernel may exist for various hardware architectures, which is typical for Linux operation systems. At the same time, it is essential to note that in the vast majority of cases this entails not fundamentally different architectures but options for implementing the basic von Neumann architecture.

In the case when the computer system being developed is designed at a lower level than the operation system as such (for example, when programming controllers for managing various devices), the problem of ensuring the platform independence is significantly aggravated and can most often be solved only for a set of hardware of a certain class for which the access interface is standardized, that is, a set of low-level information processing commands.

Thus, it can be said that much attention in the design of modern computer systems is currently being paid to the first of the listed levels of the platform independence, that is, ensuring the operation of the software system on different operation systems. This can be achieved in different ways:

- The usage of cross-platform programming languages, which, in turn, can be divided into "fully" inter-

preted languages (Python, JavaScript and languages based on it, PHP, and others) and languages using compilation into the platform-independent low-level bytecode with its possible subsequent compilation into the machine code directly during execution (Just-in-time compilation, or JIT compilation). Languages of the second class include, for example, Java and C#. The implementation of this approach requires the installation of the appropriate programming language or bytecode on the target computer with the operation system of the interpreter.
Despite its popularity, this option has a number of limitations:
  - on average, the performance of interpreted programs is lower than compiled ones. One of the approaches to solving this problem is JIT compilation;
  - strictly speaking, cross-platform compatibility with this option is provided not for all operation systems but for a class of operation systems and the corresponding class of devices, for example, operation systems designed for personal computers. For example, an application for a personal computer written in Java cannot be directly transferred to a mobile device, because when developing mobile applications, other principles of user interaction with the system interface, the absence of multiwindowing, and much more are taken into account.
- The implementation of the system in the form of a web application, which is operated through a web browser and whose interface is thus implemented on the basis of generally accepted standards of the World Wide Web (HTML, CSS, JavaScript and languages and libraries based on it). This option provides the ability to work with the application from any device that has a web browser, including a mobile one. The disadvantages of this option include:
  - as a rule, high demands on the performance of the end device. A modern web browser is one of the most resource-intensive applications on almost any device;
  - the problem of ensuring the platform independence of the server part of the web application remains behind the scenes, which should be solved in some other way;
  - despite standardization, developers often have to take into account the specifics of particular web browsers and test the performance of applications for each of them;
  - potentially, the same web application can be used on any device, however, to ensure convenience and clarity, as a rule, it is necessary to develop separate versions of the web application adapted to different devices, having, for example, different screen sizes.
- Virtualization (containerization, emulation). The listed terms are not completely synonymous but generally denote an approach in which an isolated local environment (virtual machine, container, emulation environment) is created within the operation system, containing all the settings necessary for the work of an application and guaranteeing its work on any operation systems and devices where the corresponding virtual machine or container can be interpreted. Accordingly, the running of such environments requires the installation of an appropriate interpreter or emulator on the end device.
  This approach is rapidly developing and gaining popularity at the moment, since it allows not only solving the problem of cross-platform compatibility but also saving the consumer from installing a large number of dependencies and configuring the application on the end device.
  Among the popular tools implementing this approach, tools for virtualization (VirtualBox, DOSBox, VMware Workstation), containerization (Docker), emulation of Android applications for desktop operation systems (Genymotion, Bluestacks, Anbox), and many others can be specified.
  The disadvantages of this approach include its resource intensity and reduced performance, as well as limited usage (as a rule, the corresponding interpreters are developed only for the most popular and demanded operation systems). In addition, there is a next-level problem associated with dependence on the selected virtualization (containerization) tool.

It is also important to note that even for interpreted programming languages, there is a problem of application dependence on the set of libraries and frameworks used. So, when developing an interface of a web application, the popular AngularJS and Reactos frameworks can be used, while after selecting one of them, it is impossible to quickly transfer the application to another framework.

Thus, it can be concluded that a lot of attention is paid to the problem of ensuring the platform independence in modern computer systems, but it has not been fully solved. At the same time, there are a large number of successful private solutions, which, however, have serious limitations, primarily due to the lack of unification of modern approaches to the development of computer systems.

The problem of ensuring the platform independence becomes even more urgent in the context of the development of *intelligent computer systems*. This is conditioned by the following features of such systems:

- a much more complex structure of the represented information in comparison with traditional computer systems and, accordingly, the variety of forms of its representation, storage and processing of which on

different platforms can be organized in completely different ways;
- high performance requirements for some classes of systems, in particular, systems that use machine learning, which leads to the creation of specialized hardware architectures, such as, for example, neurocomputers [1], [2];
- a variety of problem-solving models that are generally implemented differently in various systems;
- the relevance of the development of hybrid intelligent systems [3], within which various types of knowledge and various problem-solving models are integrated. Due to the lack of a generally accepted unified foundation for their integration at the moment, such systems are created mainly with a focus on a specific platform and can hardly be transferred to other platforms.

Thus, we can say that the problem of ensuring the platform independence for intelligent systems is largely conditioned by the deficiency in the semantic compatibility of components of such systems with each other, which, in turn, creates obstacles even for the implementation of approaches to ensuring the platform independence, implemented in the development of traditional computer systems.

III. Proposed approach to ensuring the platform independence of intelligent computer systems

To solve the problem of ensuring the platform independence of intelligent systems, as it was mentioned earlier, it is necessary first to ensure the semantic compatibility of the components of such systems with each other, which, in turn, assumes:
- unifying the representation of various kinds of information stored in the knowledge bases of such systems;
- unifying the basic models of processing information stored in the knowledge bases of such systems, that is, the allocation of a universal low-level programming language that allows processing the stored information in a unified form;
- unifying the principles of implementing various problem-solving models and, as a result, the possibility of their integration within hybrid intelligent systems;
- unifying the principles of developing computer system interfaces, which would make it possible to carry out within one intelligent system the interaction with other systems and users of such systems in different external languages, including natural ones.

These principles are implemented within the Open Semantic Technology of Intelligent Systems Design (*OSTIS Technology*) [4], which is proposed to be the basis for solving the problem of ensuring the semantic compatibility of components of *intelligent computer systems* and ensuring the platform independence of such systems. In particular, within the *OSTIS Technology*, the following key principles from the point of view of ensuring the platform independence are implemented:
- the *OSTIS Technology* is based on a universal method of semantic representation (encoding) of information in the memory of intelligent computer systems, called an *SC-code*. Texts of the *SC-code* (sc-texts, sc-constructions) are unified semantic networks with a basic set-theoretic interpretation. The elements of such semantic networks are called *sc-elements* (*sc-nodes* and *sc-connectors*, which, in turn, depending on orientation, can be *sc-arcs* or *sc-edges*). The *Alphabet of the SC-code* consists of five main elements, on the basis of which SC-code constructions of any complexity are built, including more specific types of sc-elements (for example, new concepts). Universality and uniformity of the *SC-code* makes it possible to describe on its basis any *types of knowledge* and any problem-solving *methods*, which, in turn, significantly simplifies their integration within a single system;
- the basis of the knowledge base developed by the *OSTIS Technology* is a hierarchical system of semantic models of *subject domains* and *ontologies*, among which the universal *Kernel of the knowledge base semantic models* and the methodology for the development of semantic knowledge base models are allocated, which ensure the semantic compatibility of the knowledge bases being developed;
- the basis of information processing within the *OSTIS Technology* is the *SCP Language*, the program texts of which are also written in the form of SC-code constructions;
- the problem solver architecture within the *OSTIS Technology* is based on a multi-agent approach, in which agents interact with each other purely by specifying the actions they perform within a common semantic memory (such agents are called *sc-agents*). Such an approach allows ensuring the fundamental possibility of implementing any *problem-solving methods* in the form of corresponding solver components and ensuring their semantic compatibility;
- the interface of the *ostis-system* is interpreted as a specialized subsystem that is built on the same principles as any other ostis-system (that is, it has its own knowledge base and problem solver) and solves problems related to the interaction of the system with the external environment;
- all of these principles together make it possible to ensure the semantic compatibility and simplify the integration of both various components of computer systems and such systems themselves.

The listed principles allow concluding that the *OSTIS*

Technology provides a fundamental possibility of implementing the platform independence of computer systems developed on its basis (*ostis-systems*). On the other hand, thanks to its universality, the *OSTIS Technology* allows transforming any modern computer system into the *ostis-system*, which will be functionally equivalent to the original computer system but at the same time will have all the above features that create preconditions for solving the problem of the platform independence.

To solve this problem at the level of the *OSTIS Technology* it is proposed to use an ontological approach involving the building of a family of *ontologies*, providing clarification of concepts such as *ostis-system*, *ostis-platform*, their structure, typology, and the requirements imposed on them.

As for the above-mentioned problem of the dependence of computer systems on specific frameworks, a similar problem may arise with the further development of the *OSTIS Technology*, in a situation where the corresponding libraries will contain a sufficiently large number of functionally equivalent components. However, thanks to the principles underlying the *OSTIS Technology*, in particular, the semantic representation of information and semantic compatibility of components, this problem will be much less acute, since:

- the number of functionally equivalent components will be significantly lower than in traditional information technologies; it is not necessary to create syntactically different components: the differences will be only at the semantic level;
- independently, the components will be more universal, that is, they can be used in a much larger number of systems;
- there is an opportunity to automatically identify close components, their similarities, differences, potential conflicts, and dependencies of components;
- it is possible to build fairly simple (compared to traditional technologies) procedures for the transition from one framework to another, since all components and frameworks have a common formal semantic basis of a level that is higher than in traditional technologies.

Within the *OSTIS Technology*, several universal variants of visualization of *SC-code* constructions are proposed, such as *SCg-code* (graphic variant), *SCn-code* (nonlinear hypertext variant), *SCs-code* (linear string variant). Within this article, fragments of structured texts in the SCn code [4] will often be used, which are simultaneously fragments of the source texts of the knowledge base, understandable to both human and machine. This allows making the text more structured and formalized, while maintaining its readability. The symbol ":=" in such texts indicates alternative (synonymous) names of the described entity, revealing in more detail certain of its features.

IV. Architecture and principles for the ostis-systems implementation

Let us consider the proposed approach to organizing the implementation of *ostis-systems*. One of the key principles of the *OSTIS Technology* is to ensure the platform independence of *ostis-systems*, that is, a strict separation of the logical-semantic model of the cybernetic system (*sc-models of the cybernetic system*) and the interpretation platform of the sc-models of the cybernetic system (*ostis-platform*). The advantages of such a strict separation are quite obvious:

- the transfer of the *ostis-system* from one platform to another (for example, a newer and more efficient or focused on a certain class of devices) is performed with minimum overhead costs (in the ideal case, it generally comes down to loading the *sc-model of a cybernetic system* onto the platform);
- the components of *ostis-systems* become universal, that is, they can be used in any ostis-systems where their usage is appropriate;
- the development of the platform and the development of sc-models of systems can be carried out in parallel and independently of each other, in general, by separate independent teams of developers according to their own rules and methods.

Let us consider in more detail the concept of the *logical-semantic model of a cybernetic system*.

logical-semantic model of a cybernetic system
:= [formal model (formal description) of 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 group of agents processing the specified information]
⊃ *sc-model of a cybernetic system*
  := [logical-semantic model of a cybernetic system, represented in an SC-code]
  := [logical-semantic model of an ostis-system, which, in particular, can be a functionally equivalent model of some cybernetic system that is not an ostis-system]

cybernetic system
⊃ *computer system*
  := [artificial cybernetic system]
  ⊃ *ostis-system*
    := [computer system built using the OSTIS Technology based on the interpretation of the designed logical-semantic sc-model of this 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 the cybernetic system interface*
}

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

The ***sc-memory*** is, on the one hand, a common environment for storing the *knowledge base* and, on the other hand, an environment for interaction of *sc-agents*. At the same time, each *sc-agent* relies on some known *sc-elements* stored in the *sc-memory* (*key sc-elements* of this *sc-agent*).

In general, the *sc-memory* implements the following functions:

- storage of *SC-code* constructions;
- storage of information constructions (files) external to the *SC-code*. In general, file storage can be implemented in a way different from storing sc-constructions;
- access to *SC-code* constructions (reading, creating, deleting), implemented through the appropriate software or hardware interface. Such an interface is essentially a microprogramming language that allows implementing more complex procedures for processing stored constructions based on it, including the operators of the *SCP Language*, the set of which determines the list of commands of such a microprogramming language. The *sc-memory* itself is passive in this regard and only executes commands initiated from the outside by any subjects.

Note that the separation of the storage and access functions is rather conditional, since it seems impractical to implement the function of storing constructions without the possibility of accessing them at least at the lowest level, because it will be impossible to use such storage.

The terms "*sc-memory*" and "*abstract sc-memory*" are synonyms in the way that mentioning the *sc-memory* we mean some abstraction for which its maximum volume is not specified (the maximum number of *sc-elements*, that can be stored in this memory simultaneously), a particular method of storing *sc-elements*, means for ensuring the storage reliability, etc. All these features are specified at the level of the *sc-memory implementation* in a hardware version or a software model based on some other architecture.

The explicit allocation of the *sc-model of the knowledge base*, *sc-model of the problem solver*, and *sc-model of the cybernetic system interface* within the *sc-model of the cybernetic system* is to a certain extent conditional, since to ensure the platform independence, *sc-models of a cybernetic system*, the *problem solver*, and the *system interface* are described by means of the *SC-code* and, thus, are also part of the *knowledge base*. Such an explicit allocation of these components is conditioned by the convenience of designing and maintaining the system.

Thus, on condition of strict separation of the *sc-model of the cybernetic system* and *ostis-platform*, as well as ensuring the universality of the *ostis-platform*, that is, the ability to interpret <u>any</u> *sc-model of the cybernetic system* on any variant of the *ostis-platform*, the <u>implementation</u> stage of the *ostis-system* actually comes down to loading the *sc-model of a cybernetic system* on the selected variant of the *ostis-platform*.

It is important to note that the universality of a particular implementation option of the *ostis-platform* is obviously limited by the physical (hardware) part of this implementation. For example, if the hardware of the selected platform option is a conventional personal computer, then without the addition of extra hardware components, the system will not be able to solve problems related to the physical movement of itself and other objects in space, even if the software part of the system is able to perform the necessary calculations. In other words, any *ostis-platform* will always be limited in solving *behavioral problems* of any classes, no matter how powerful physical resources it possesses. Thus, it is more correct to talk about the <u>universality of the *ostis-platform* in the context of solving *information problems*</u>, that is, the ability to interpret any *sc-models of cybernetic systems* regardless of what kind of *information problems* these systems solve.

Based on this, it is possible to formulate a key requirement for the *sc-model of a cybernetic system* – at none of the stages of solving any *information problem* in this system the features of the platform on which the specified *sc-model* will be interpreted in the future should be taken into account. Similarly, the key requirement for the *ostis-platform* is to provide an interface for accessing (searching and converting) information stored in the *sc-memory* in some universal way, independent from the specifics of the implementation of a particular platform. Thus, the most important problem to ensure the platform independence of *ostis-systems* is a clear specification of the requirements for each implementation of the *ostis-platform*, that is, <u>standardization</u> of *ostis-platforms*. It is important to note that such standardization should not depend on the form in which the *ostis-platform* is implemented and, accordingly, be suitable for the hardware version of the implementation.

To clarify the requirements for the *ostis-platform*, we introduce the concept of an *sc-machine*, which is an analogue of such models as the Post Machine and the Turing Machine [5], the von Neumann Machine [6].

sc-machine
:= [abstract sc-machine]
:= [generalization of various implementations of ostis-platforms, for which general functional requirements are set]
:= [generalized model describing the functioning of any ostis-platform, regardless of the way it is implemented]
:= [generalized model that defines the general patterns of any ostis-platform, regardless of the way it is implemented]
:= [generalized information image of the ostis-platform]
⇐ *generalized model**:
   ostis-platform
⇒ *generalized decomposition**:
   {• *sc-memory*
      ⇐ *generalized model**:
         implementation of the sc-memory
   • *abstract machine of knowledge processing*
      ⊂ *abstract sc-agent*
   }
⊃ *scp-machine*
   ⇐ *generalized model**:
      scp-interpreter
   := [sc-machine that provides interpretation of the ostis-systems basic programming language]
   := [generalized model of the interpreter of the ostis-systems basic programming language]
   := [generalized model defining the general principles for the interpretation of the ostis-systems basic programming language]
   := [generalized model of operational semantics of the ostis-systems basic programming language]

Potentially, we can talk about several possible functionally equivalent variants of the *scp-machine*, which will correspond to different variants of the basic programming language. Within the current version of the *OSTIS Technology*, both the denotational semantics of the *SCP Language* and its operational semantics, implemented in the form of an *abstract scp-machine*, are fixed [4].

It is important to emphasize that despite the advantages of the platform-independent implementation of *ostis-systems*, it sometimes turns out to be advisable to implement some components of *ostis-systems* (for example, specific *sc-agents* or user interface components) at the level of the *ostis-platform*. In the case of such an implementation of the *sc-agents* programs, an analogy can be drawn with the implementation of any subprograms at the level of microprogramming languages for modern computers. Most often, the reasonableness of such a solution is conditioned by an increase in the performance of such components and the system as a whole, since the implementation of the component, taking into account the features of the platform, is generally more productive. At the same time, let us note that the latter statement is not always true, since when implementing a component at the level of a logical-semantic model, for example, parallel information processing models can be implemented, which are not always easily and clearly implemented at the platform level.

Thus, when designing each specific *ostis-system*, the developer needs to make a decision about the implementation of certain components at a platform or platform-independent level. At the same time, it is obvious that from the point of view of technology development and the accumulation of project experience, the implementation of *ostis-systems* components at a platform-independent level is a higher priority.

Based on the above, we can assume the existence of *ostis-systems* in which all *sc-agents* are implemented at the platform level, which in this case is essentially "cut out" for a specific *ostis-system* and can be considered as an analogue of a specialized computer focused on solving problems of only a certain limited class. Let us call such an option for the implementation of *ostis-systems* the *minimum ostis-system configuration*. In order for *minimum ostis-system configuration* to be considered an *ostis-system* at all, that is, a system that is built in accordance with the principles of the *OSTIS Technology*, it must meet the following minimum set of requirements:

- the usage of the *SC-code* as a basic language for encoding information in the knowledge base and, accordingly, the presence of memory storing *SC-code* constructions;
- the presence of the *knowledge base* defining the denotational semantics of the concepts used by the system;
- the presence of at least one *internal sc-agent* performing knowledge processing in the memory of the *ostis-system*. This *sc-agent* can be implemented at the platform level, accordingly, the *knowledge base* of such a system may not contain procedural knowledge (methods).

Such a variant of *minimum ostis-system configuration* has only an *internal sc-agent* and, accordingly, has no ability to communicate with the external world (we can say that such an *ostis-system* does not have "sense organs"). In order for the system to be able to communicate with the external world, it is necessary to add at least one *receptor sc-agent* and at least one *effector sc-agent* to the *minimum ostis-system configuration*.

It is important to note that, as can be seen from the description of the *minimum ostis-system configuration*, in general, the *ostis-system* does not have to be an *intelligent system* by default. Usage of the *OSTIS Technology* for the development of computer systems does not automatically make them intelligent – it allows ensuring the possibility of subsequent unlimited intellectualization of such systems with minimum overhead costs, provided that all the principles of the *OSTIS Technology* are satisfied during their development.

V. Clarification of the ostis-platform concept

ostis-platform
:= [platform for interpreting sc-models of computer systems]
:= [interpreter of 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]
:= [embedded empty ostis-system]
:= [sc-machine implementation]
⊂ *embedded ostis-system*
⊂ *platform-dependent reusable component of ostis-systems*

The implementation of the *ostis-platform* (interpreter of sc-models of cybernetic systems) can have a large number of variants – both software and hardware implemented. If necessary, any components of problem solvers or knowledge bases can be included in the ***ostis-platform*** in advance at the platform-dependent level, for example, in order to simplify the creation of the first version of an *applied ostis-system*. The implementation of the *ostis-platform* can be carried out on the basis of an undefined set of existing technologies, including the hardware implementation of any of its parts. From the point of view of the component approach, any ***ostis-platform*** is a ***platform-dependent reusable component of ostis-systems***.

ostis-platform
⇒ *subdividing**:
{• *basic ostis-platform*
:= [basic interpreter of logical-semantic models of ostis-systems]
:= [minimum universal ostis-platform that provides interpretation of the sc-model of any *ostis-system* and includes an interpreter of the *ostis-systems* basic programming language (SCP Language)]
:= [universal interpreter of sc-models of ostis-systems]
:= [universal basic ostis-system that provides an imitation of any *ostis-system* by interpreting the sc-model of the imitated ostis-system]
• *extended ostis-platform*
:= [ostis-platform containing additional components implemented at the platform level]
:= [basic ostis-platform and many components implemented at the platform level]
• *specialized ostis-platform*
:= [ostis-platform that does not contain an implementation of the SCP language interpreter]
:= [non-universal ostis-platform]
}

The concept of a *basic ostis-platform* is key from the point of view of ensuring the platform independence of *ostis-systems*. The universality of the *basic ostis-platform* implies the possibility of interpreting any *sc-model of a cybernetic system* based on it. This is accomplished by the presence of means within the *OSTIS Technology*, that allow describing the *knowledge base*, *problem solver*, and *cybernetic system interface* at the level of the sc-model, as well as by the availability of a Basic universal programming language for *ostis-systems* (*SCP Language*). In this case, the *SCP Language* acts as a basic low-level standard (assembler) for processing *SC-code* constructions, guaranteeing completeness from the point of view of processing, that is, providing the ability to perform any transformation of any fragment of the *SC-code* on condition that the syntactic correctness of this fragment is maintained. It should be noted that in general there may be several such functionally equivalent assemblers (and, as a consequence, corresponding *scp-machines*), but to ensure compatibility within the *OSTIS Technology* one of these options is selected as a standard and described in the corresponding section of the *OSTIS Standard* [4].

Thus, the main and only requirement imposed on all *basic ostis-platforms* to ensure their universality is the need to provide interpretation of the *SCP Language* standardized within the *OSTIS Technology*. It is important to note that all *basic ostis-platforms* must be functionally equivalent, since they interpret the same standard of the *SCP Language*.

Each *basic ostis-platform* contains:

- implementation of the means for storing *SC-code* constructions (sc-memory), including the implementation of file memory;
- implementation of tools for processing *SC-code* constructions – an *scp-interpreter*;
- implementation of a basic set of *receptor sc-agents* and *effector sc-agents*, providing the minimum necessary information exchange between the *ostis-system* and the external environment. The specific list of such agents requires clarification, however, we can say that in general they can be implemented as part of the *scp-interpreter* (in this case, they will correspond to certain classes of *scp-operators*) or separately from it as part of the platform;
- implementation of a set of sc-agents that provide the basic functions of the *ostis-system*, related to ensuring its operation, which in principle cannot be implemented at a platform-independent level. Such functions include, for example, starting the system, loading the knowledge base into the system memory, starting the *scp-interpreter*, etc.

More formally, the model of the *basic ostis-platform* can be written as follows:

basic ostis-platform
⇒ *generalized decomposition**:
  {• *sc-memory implementation*
    ⇒ *generalized part**:
      implementation of the sc-machine file memory
  • *scp-interpreter*
  • *basic subsystem for interaction of the ostis-system with the external environment*
  • *subsystem for ensuring the operation of the ostis-system*
  }

An *extended ostis-platform* is a *basic ostis-platform* supplemented with any set of components (at least one) implemented at the platform level, provided that all the features of the *basic ostis-platform* are maintained. Thus, an *extended ostis-platform* is essentially a *basic ostis-platform* adapted to more efficiently solve problems of certain classes within a specific class of *ostis-systems*. A component implemented at the platform level becomes part of this platform and thus transforms the *basic ostis-platform* into the *extended ostis-platform*.

Introduction of the concept of the *extended ostis-platform* allows formulating a number of additional principles for the implementation of *ostis-systems*:

- there may be an undefined number of ostis-systems, each of which will have its own unique *extended ostis-platform*, but they will all be based on the same variant of the *basic ostis-platform*;
- for each variant of the *basic ostis-platform*, there may be its own *library of reusable ostis-platform components* compatible with this variant of the *basic ostis-platform* and that allows composing various variants of the *extended ostis-platform* based on the *basic ostis-platform.*

A ***specialized ostis-platform*** is a bounded implementation of the *ostis-platform* that does not contain an *scp-interpreter*. Thus, <u>all</u> *sc-agents*, within the *ostis-system* based on the *specialized ostis-platform*, must be implemented at the platform-dependent level. Such a *specialized ostis-platform* is an analogue of a specialized computer implemented for a specific computer system. Thus, in general, each *ostis-system* implemented on the *specialized ostis-platform* will have its <u>unique</u> *specialized ostis-platform.*

The ***specialized ostis-platform*** can be obtained from the *basic ostis-platform* by excluding the implementation of the *scp-interpreter* from it and implementing all necessary *sc-agents* at the platform level (or borrowing all or part of the agents from a *library of reusable ostis-platform components*, that corresponds to the given variant of the *basic ostis-platform*).

specialized ostis-platform
⇒ *generalized decomposition**:
  {• *sc-memory implementation*
    ⇒ *generalized part**:
      implementation of the sc-machine file memory
  • *basic subsystem for interaction of the ostis-system with the external environment*
  • *subsystem for ensuring the operation of the ostis-system*
  • *specialized platform-dependent knowledge processing machine*
    := [sc-agent, as a rule, a non-atomic one, providing the performance of all the functions of some specialized ostis-platform related to knowledge processing]
    ⊂ *platform-dependent sc-agent*
  }

The concept of the *minimum ostis-system configuration* introduced earlier can be clarified taking into account the concept of the *specialized ostis-platform.*

minimum ostis-system configuration
⇒ *generalized decomposition**:
  {• *sc-model of the knowledge base*
  • *specialized ostis-platform*
  }

The usage of *specialized ostis-platforms* may be reasonable at the initial stage of the development of the *OSTIS Technology*, as well as in order to improve the performance of particular *ostis-systems* that are most highly loaded, however, the active development of such *specialized ostis-platforms* and their components from the point of view of the *OSTIS Technology* is impractical, since:

- if any component is designed with a focus on a specific platform, then there are no guarantees that it can be reused in other *ostis-platform* implementations options (at least, components developed for the *ostis-platform software implementation* will not be able to be used within the *associative semantic computer*);
- the availability of a large number of platform-dependent components requires the development and maintenance of a separate library infrastructure for storing and reusing such components. The greater the number of platform-dependent components and the more variants of *ostis-platforms* exist, the more complex and lengthy such an infrastructure will be. At a minimum, it will be necessary to monitor the compatibility of components with different versions of various *ostis-platforms* implementation options;
- changes in the *specialized ostis-platform*, for example, related to the transition to a newer and more efficient version of the *basic ostis-platform*, on the basis of which this *specialized ostis-platform* is built, in general, may lead to the need in making changes to components that depend on this *ostis-platform* implementation option. The more such platform-dependent components exist, the more potential changes may be required and, accordingly, the more difficult the evolution of the platform will be, provided that the operability of the *ostis-systems* in which it is used is preserved.

The above theses are also true for *extended ostis-platforms*, however, in the case of *extended ostis-platform*, problems associated with the transition to a newer version of the platform and changes in the corresponding components can always be solved by temporarily replacing platform-dependent components with their platform-independent versions with a corresponding decreased performance but maintaining the functional integrity of the system.

ostis-platform
⇒ *subdividing**:
{• *single-user ostis-platform*
:= [option for implementing a platform for interpreting sc-models of computer systems, designed for the case when only one user (owner) interacts with a particular *ostis-system*]
• *multi-user ostis-platform*
:= [option for implementing the platform for interpreting sc-models of computer systems, designed for the case when different users can interact with a particular *ostis-system* at the same time or at different times, generally having different rights, areas of responsibility, level of experience and having their own confidential part of the information stored in the knowledge base]
}

With a single-user platform implementation, it turns out to be impossible to implement some important principles of the *OSTIS Technology*, for example, the collective coordinated development of the knowledge base of the system during its operation. At the same time, various third-party tools can be used, for example, for developing a knowledge base at the level of source texts.

ostis-platform
⇒ *subdividing**:
{• *software version of the ostis-platform*
:= [platform for interpreting sc-models of ostis-systems, implemented as a software system based on traditional computer architecture]
:= [software platform for interpreting sc-models of ostis-systems]
:= [software interpreter of sc-models of ostis-systems]
• *associative semantic computer*
:= [hardware platform for interpreting sc-models of ostis-systems]
:= [hardware implemented basic interpreter of sc-models of ostis-systems]
}

It is important to note that in any *ostis-platform* implementation option, both software and hardware are always present. So, any *software version of the ostis-platform* assumes its subsequent interpretation on some hardware basis, for example, on a personal computer with a traditional architecture. At the same time, the development of the *ostis-platform* in the form of an *associative semantic computer* involves the development of a set of micro-programs implementing basic operations of searching and converting sc-constructions stored in the *sc-memory*.

Thus, the separation of the set of possible *ostis-platform* implementations into software and hardware variants rather reflects the variant of the hardware architecture on which one or another variant of the platform implementation is ultimately oriented – either the traditional von Neumann architecture or the specialized architecture of the *associative semantic computer* with structurally reconfigurable (graphodynamic) memory. In fact, the *software version of the ostis-platform* is a model (virtual machine) of the *associative semantic computer*, built on the basis of the traditional von Neumann architecture, and the *SCP Language* acts as an assembler for the *associative semantic computer* and can also be interpreted both within the hardware implementation of such a computer and within its software model.

The appropriateness of developing *ostis-platform software options* at the moment is conditioned by the obvious prevalence of the von Neumann architecture and, accordingly, the need to implement *ostis-systems* on modern computers of various types. At the same time, it is obvious that the development of specialized *associative semantic computers* will significantly increase the efficiency of *ostis-systems*, and a clear separation of the *sc-model of a cybernetic system* and its interpretation platform will allow the translation of already working *ostis-systems* from traditional architectures on *associative semantic computers* with minimum overhead costs.

Each specific *ostis-system* uniquely corresponds to a particular *ostis-platform*, which can relate to a different set of classes of *ostis-platforms*. At the same time, it is obvious that at the stage of platform development, some variant of the *ostis-platform* is designed and implemented, which is then replicated into different *ostis-systems*. Subsequently, changes can be made to this variant of the *ostis-platform* in each *ostis-system*, but in general, in a large number of *ostis-systems*, fully equivalent *ostis-platforms* can be used. Thus, it is advisable to talk about *typical ostis-platforms*, which:

- are the object of development for developers of *ostis-platforms*;
- *are a reusable component of ostis-systems* and are specified within the appropriate libraries;
- are a sample for replication (copying) when creating new *ostis-systems*.

VI. Associative semantic computers for ostis-systems

The usage of modern hardware and software platforms focused on address access to data stored in memory for the development of *ostis-systems* is not always efficient, since when developing intelligent systems, it is actually necessary to model nonlinear memory based on the linear one. Improving the efficiency of problem solving by intelligent systems requires the development of specialized platforms, including hardware ones, focused on unified semantic models of information representation and processing. Thus, the main purpose of creating *associative semantic computers* is to increase the performance of ostis-systems.

Let us consider in more detail the features for the logical organization of the traditional architecture of computer systems, which significantly complicate the effective implementation of *ostis-systems* based on it:

- low level of memory access, i.e. complexity and lengthiness of performing the procedure of associative search for the necessary knowledge fragment;
- linear memory organization and an extremely simple view of constructive objects directly stored in memory. This leads to the fact that in intelligent systems built on the basis of modern computers, the manipulation of knowledge is carried out with great difficulty. Firstly, it is necessary to operate not with the structures themselves but with their lengthy linear representations (lists, adjacency matrices, incidence matrices); secondly, the linearization of complex structures destroys the locality of their transformations;
- the information representation in the memory of modern computers has a level that is very far from the semantic one, which makes the processing of knowledge rather lengthy, requiring consideration of a large number of details concerning not the meaning of the processed information but the way it is represented in memory;
- in modern computers, there is a low level of hardware-implemented operations on non-numeric data and there is no hardware support for logical operations on knowledge fragments with a complex structure, which makes manipulating such fragments complicated.

The listed features, in fact, are not eliminated either in the approaches to build non-traditional high-performance computers (for example, computers designed for training and interpretation of artificial neural networks [1], [2]) currently being developed, because, basically, all these approaches (even if they deviate far enough from the basic principles of the organization of computing machines, proposed by von Neumann) implicitly preserve the point of view of the computer as a large arithmometer and thereby preserve its orientation to numerical tasks.

There are a number of articles [7]–[14] and patents [15]–[17] aimed at developing hardware architectures designed to process information represented in more complex forms than in traditional architectures, but they have not gained widespread distribution and application, due, firstly, to the particular solutions offered and, secondly, due to the lack of a common universal and unified coding language for any information, in the role of which, within the OSTIS Technology, the SC-code acts.

The *SC-code*, which is the formal basis of the *OSTIS Technology* was originally developed as a language for encoding information in memory of *associative semantic computers*, so it originally laid down such principles as universality (the ability to represent knowledge of any kind) and unification (uniformity) of representation, as well as minimization of the *Alphabet of the SC-code*, which, in turn, makes it easier to create a hardware platform that allows storing and processing texts of the *SC-code*.

associative semantic computer
:= [hardware implemented interpreter of semantic models (sc-models) of computer systems]
:= [semantic associative computer controlled by knowledge]
:= [computer with a nonlinear structurally configurable (graphodynamic) associative memory, the processing of information in which is reduced not to a change in the state of memory elements but to a change in the configuration of the connections between them]
:= [sc-computer]
:= [scp-computer]
:= [new generation universal computer specially designed for the implementation of semantically compatible hybrid intelligent computer systems]
:= [new generation universal computer focused on hardware interpretation of logical-semantic models of intelligent computer systems]
:= [new generation universal computer focused on hardware interpretation of ostis-systems]
:= [ostis-computer]
:= [computer for the implementation of ostis-systems]
:= [computer controlled by the knowledge represented in the SC-code]
:= [computer focused on SC-code text processing]

Let us consider the principles underlying the implementation of *associative semantic computers*:

- nonlinear memory – each elementary fragment of a text stored in memory can be incident to an unlimited number of other elementary fragments of this text. Thus, memory cells, unlike ordinary memory, are connected not by fixed conditional connections that specify a fixed sequence (order) of cells in memory but by actually (physically) conducted connections of undefined configuration. These connections correspond to arcs, edges, hyperedges of the graph (sc-text) recorded in memory;
- structurally tunable (reconfigurable) memory – the procedure of processing information stored in memory is reduced not only to changing the state of elements but also to reconfiguring the connections between them. That is, during the processing of information in structurally-tunable memory, the changes concern not only and not even so much the states of memory cells, as in ordinary memory, as the configuration of the connections between these cells. I.e., in structurally-tunable memory, during the processing of information, not only the labels on the vertices of the graph recorded in memory are redistributed, but the structure of this graph itself is also changing;
- as an internal way of encoding knowledge stored in the memory of the *associative semantic computer*, a universal (!) method of nonlinear (graph-like) semantic representation of knowledge – SC-code – is used;
- information processing is carried out by a group of agents working on shared memory. Each of them reacts to a corresponding situation or event in memory (a computer controlled by stored knowledge);
- there are software-implemented agents whose behavior is described by agent-oriented programs stored in memory, which are interpreted by the corresponding groups of agents;
- there are basic agents that cannot be software implemented (in particular, these are agents of interpretation of agent programs, basic receptor agents-sensors, basic effector agents);
- all agents work on shared memory at the same time. Moreover, if several conditions of its usage arise for an agent at some point in time in different parts of memory, different information processes corresponding to the specified agent in different parts of memory can be performed simultaneously;
- in order for the agents' information processes running in parallel in shared memory not to "interfere" with each other, the current state is recorded and constantly updated in memory for each information process. That is, each information process informs others about its intentions and wishes, which other information processes should not interfere with. The implementation of this approach can be carried out, for example, on the basis of the mechanism of locking elements of semantic memory [4];
- the processor and memory of the *associative semantic computer* are deeply integrated and form a single processor-memory. The processor of the *associative semantic computer* is evenly "distributed" over its memory so that the processor elements are simultaneously computer memory elements. That is, each cell is supplemented by a functional (processor) element, and the tunable connections between the cells become switched communication channels between the functional elements. At the same time, each functional element has its own special internal register memory, reflecting aspects of the current state of performing elementary operations of the internal language, that are important for this

functional element.

Information processing in the *associative semantic computer* is reduced to reconfiguration of communication channels between processor elements, therefore the memory of such a computer is nothing but a switchboard (!) of these communication channels. Thus, the current state of the configuration of these communication channels is the current state of the information being processed. This principle provides a significant acceleration of information processing by eliminating the stages of transferring information from memory to the processor and back, but it is paid for at the cost of a large redundancy of functional (processor) means evenly distributed over memory.

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.

The work was carried out with the partial financial support of the BRFFR (BRFFR-RFFR No. F21RM-139).

References

[1] L. G. Komarcova and A. V. Maksimov, *Neurocomputers: A Textbook for Universities. - 2nd ed., revised and enlarged*, ser. Informatics at the Technical University. Moscow: Bauman Moscow State Technical University, 2004, (In Russ).

[2] (2021, Jun) USB Accelerator | Coral. [Online]. Available: https://coral.ai/products/accelerator/

[3] A. Kolesnikov, *Gibridnye intellektual'nye sistemy: Teoriya i tekhnologiya razrabotki [Hybrid intelligent systems: theory and technology of development]*, A. M. Yashin, Ed. SPb.: SPbGTU, 2001.

[4] V. Golenkov, N. Guliakina, and D. Shunkevich, *Otkrytaja tehnologija ontologicheskogo proektirovanija, proizvodstva i jekspluatacii semanticheski sovmestimyh gibridnyh intellektual'nyh komp'juternyh sistem [Open technology of ontological design, production and operation of semantically compatible hybrid intelligent computer systems]*, V. Golenkov, Ed. Minsk: Bestprint [Bestprint], 2021.

[5] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to automata theory, languages, and computation*, 2nd ed. Upper Saddle River, NJ: Pearson, Nov. 2000.

[6] M. Godfrey and D. Hendry, "The computer as von Neumann planned it," *IEEE Annals of the History of Computing*, vol. 15, no. 1, pp. 11–21, 1993. [Online]. Available: https://doi.org/10.1109/85.194088

[7] H.-N. Tran and E. Cambria, "A survey of graph processing on graphics processing units," *The Journal of Supercomputing*, vol. 74, no. 5, pp. 2086–2115, Jan. 2018. [Online]. Available: https://doi.org/10.1007/s11227-017-2225-1

[8] X. Shi, Z. Zheng, Y. Zhou, H. Jin, L. He, B. Liu, and Q.-S. Hua, "Graph processing on GPUs," *ACM Computing Surveys*, vol. 50, no. 6, pp. 1–35, Nov. 2018. [Online]. Available: https://doi.org/10.1145/3128571

[9] Y. Lü, H. Guo, L. Huang, Q. Yu, L. Shen, N. Xiao, and Z. Wang, "GraphPEG," *ACM Transactions on Architecture and Code Optimization*, vol. 18, no. 3, pp. 1–24, Sep. 2021. [Online]. Available: https://doi.org/10.1145/3450440

[10] I. V. Afanasyev, V. V. Voevodin, K. Komatsu, and H. Kobayashi, "VGL: a high-performance graph processing framework for the NEC SX-aurora TSUBASA vector architecture," *The Journal of Supercomputing*, vol. 77, no. 8, pp. 8694–8715, Jan. 2021. [Online]. Available: https://doi.org/10.1007/s11227-020-03564-9

[11] J. Zhang, S. Khoram, and J. J. Li, "Boosting the Performance of FPGA-based Graph Processor using Hybrid Memory Cube: A Case for Breadth First Search," *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017.

[12] Y. Hu, Y. Du, E. Ustun, and Z. Zhang, "GraphLily: Accelerating Graph Linear Algebra on HBM-Equipped FPGAs," *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, 2021.

[13] L. Minati, V. Movsisyan, M. Mccormick, K. Gyozalyan, T. Papazyan, H. Makaryan, S. Aldrigo, T. Harutyunyan, H. T. Ghaltaghchyan, C. Mccormick, and M. L. Fandrich, "iFLEX: A Fully Open-Source, High-Density Field-Programmable Gate Array (FPGA)-Based Hardware Co-Processor for Vector Similarity Searching," *IEEE Access*, vol. 7, pp. 112 269–112 283, 2019.

[14] W. S. Song, V. Gleyzer, A. Lomakin, and J. Kepner, "Novel graph processor architecture, prototype system, and results," in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, Sep. 2016. [Online]. Available: https://doi.org/10.1109/hpec.2016.7761635

[15] S. Somsubhra, "Reconfigurable semantic processor," Oct 2006.

[16] J. D. Allen, J. Philip, and L. Butler, "Parallel machine architecture for production rule systems," Jun 1989.

[17] M. Moussa, A. Savich, and S. Areibi, "Architecture, system and method for artificial neural network implementation," Jun 2013.

Универсальная модель интерпретации логико-семантических моделей интеллектуальных компьютерных систем нового поколения

Шункевич Д.В.

В работе рассматривается подход к решению проблемы платформенной независимости компьютерных систем, предполагающий унификацию принципов реализации таких систем и обеспечения их семантической совместимости на основе Технологии OSTIS. Приводится формализованная система понятий, определяющая принципы реализации данного подхода, включая прицнипы реализации аппаратной платформы для реализации систем, построенных на основе Технологии OSTIS, – ассоциативного семантического компьютера.

Received 30.10.2022