A Formal Model of Shared Semantic Memory for Next-Generation Intelligent Systems

Nikita Zotov

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

***Abstract*—This paper discusses in detail the formal model of semantic memory for intelligent systems, its structure, its elements, correspondences between them, rules and algorithms. The implementation based on this model is described, quantitative indicators of its efficiency are given.**

***Keywords*—shared memory, semantic memory, graph database, sc-memory, formal model of semantic memory, mathematical model of semantic memory, ostis-platform, intelligent system, unified knowledge representation, paral- lel information processing, semantic networks storage**

I. Introduction

Earlier in the works [1]–[3], devoted to the description of the *Software platform for intelligent systems* developed according to the principles of the *OSTIS Technology* [4] (*Software platform of ostis-systems*) — a software emulator of the future associative semantic computer [5], the software implementation of the general semantic memory (sc-memory) was considered [3], and the implementation of its programming interface was described in detail [2].

The peculiarity of previous works is that they focused not on the peculiarities of component implementation, but on approaches to describing and documenting such complex systems as the Software platform of ostis- systems [6], [7]. In this paper, the main task is to formally describe how a shared semantic memory can be realized in intelligent systems, i. e. to describe its model

Therefore, the purpose of the current work and the novelty of this paper is to describe a formal model of shared semantic memory used in ostis-systems, allowing:

- to store information of any kind in a unified semantically compatible form;
- to efficiently process this information using a specified set of operations in both single-threaded and multi-threaded modes;
- to process this information using agents that react to events in this memory, that is, allowing [8]:
- to represent knowledge in the form of semantically compatible knowledge bases of intelligent systems [9], [10];
- to create various types of interpreters of models of intellectual systems components, including interpreters for intellectual systems problem solvers;
- create libraries of reusable platform-dependent com- ponents to implement other components of ostis- platforms [11].

The relevance of the work is conditioned by the current state in the field of development of intelligent systems [12], namely:

- labor intensity of development of intellectual systems of various kinds;
- complexity of tasks solved in intelligent systems;
- complexity of integration of various components of intelligent systems;
- increase in the volume of processed information;
- growth of requirements to the speed of information processing;
- insufficient performance of modern open computing systems.

Further in this paper we will consider and describe the necessary and sufficient model of semantic memory for its implementation and use in solving the above problems [13]–[15].

II. Principles of implementation of ostis-platforms

All intelligent systems developed according to the prin- ciples of the *OSTIS Technology* are commonly referred to as *ostis-systems*. Each ostis-system consists of its own sc-model, including a knowledge base, problem solver and user interface, and an *ostis-platform* on which this sc-model is interpreted [4], [16]. Any sc-model of an ostis-system is a logical-semantic model of that system described in *SC-code*, the language of universal infor- mation encoding. By *ostis-platform* is meant a hardware-implemented computer or a software emulator of this computer for interpretation of sc-models of ostis-systems [9].

There can be a great variety of *ostis-platform* implementations on which *ostis-systems* can be interpreted, but each of such *ostis-platforms* should provide the following basic principles [1], [16], [17]:

- the development of an *ostis-system* should be reduced to the development of its sc-model (i.e. the description of the model in *SC-code* [18]) without modification of the chosen *ostis-platform* and regard- less of the means by which this ostis-platform is developed;
- transfer of the sc-model of an *ostis-system* from one ostis-platform to another is limited to loading this sc- model into the memory of the ostis-platform without loss of functionality of this ostis-system;
- addition of new information should be reduced to the "gluing" of its signs with signs of existing information with further verification of the obtained information;
- processing of information in the system should be provided by changing the configuration of links between entities in this information by means of asynchronous-parallel interaction of sc-agents reacting to the occurrence of events in the shared memory.

Therefore, all ostis-systems interpreted on ostis- platforms, unlike modern computer systems, have the following features:

- unlike modern computers, where data is represented as lines of binary code, the data stored inside the memory of ostis-systems are graph constructions written in SC-code (sc-constructions), due to which [19]:
  - any kind of knowledge is written in the same form using a minimal set of syntactically indivisible elements — the minimal alphabet of a language, which can always be augmented with new syntac- tic elements by specifying additional syntactically distinguishable classes for the elements of that language's alphabet;
  - synonymy of entity signs is forbidden, since each entity appears in the semantic network once;
  - the meaning of information is represented by explicitly specifying the relationships between entities in that information;
  - the "gluing" of information is reduced to the "gluing" of the entity signs in that information;
  - the processing of this information does not require various means of syntactic and semantic analysis.
- information processing is based on the principles of graphodynamic and agent-oriented models, so that:
  - the process of information processing is reduced not only to changing the state of elements of the semantic network, but (!) also to changing the configuration of links between them;
  - information processing is represented and stored as a semantic network;
  - it is possible to describe and solve problems of any information complexity;
  - it is possible to realize and use any existing types and models of information processing (procedu- ral, neural network, frame, logical, production, etc.);
  - information can be processed in parallelin , i.e. dif- ferent methods of problem solving can be applied simultaneously.

In other words, unlike traditional computer systems, any ostis system must be oriented towards:

- independence from the implementation of a particular ostis-platform;
- storage of information in a unified and semantically compatible form (in SC-code [18]);
- event-oriented and parallel processing of this information.

The principles of ostis-systems, first of all, should be provided by a concrete implementation of the ostis- platform. Within each *ostis-platform* there must be im- plemented:

- the shared semantic memory that allows [20]:
  - to store information constructions belonging to *SC-code* (sc-texts);
  - to store information constructions that do not belong to *SC-code* (images, text files, audio and video files, etc.);
  - to store subscriptions to occurrences of events in memory;
  - to initiate agents after these events appear in memory;
  - to use a programming interface to work with *SC- code* and *non-SC-code* information constructions, including:
    * operations to create, search, modify, and delete these constructions in the memory;
    * operations for subscribing to the occurrence of events in the memory;
    * operations for controlling and synchronizing processes in the memory;
    * programming interface for creating platform- dependent agents;
- the interpreter of the *SCP* asynchronous-parallel programming language, which is a platform- independent programming interface that implements platform-independent operations on the shared semantic memory.

The shared semantic memory that allows for fulfillment of all of the above mentioned tasks is called sc-memory, and the interpreter of the basic language of asynchronous-parallel programming *SCP — scp- interpreter*. In the context of this paper only the sc- memory model is considered. The *scp-interpreter* model and its implementation were considered in [21]

All listed principles of ostis-systems are provided in the first (basic) Software variant of the implementation of the ostis-platform — *Software platform of ostis-systems*. Drawing an analogy with modern developments in the field, the Software platform of ostis-systems can be considered as a graph database management system, for example, *Neo4J* [22]. However, unlike existing database management systems, the Software platform of ostis- systems acts as an interpreter of sc-models of seman- tically compatible ostis-systems. Therefore, the Software platform ostis-systems should also be considered as a software alternative for modern von Neumann computers. In general, the above mentioned features of the imple- mented Software platform of ostis-systems are also true for all ostis-platforms regardless of the means by which they are implemented.

An efficient implementation of sc-memory must fulfill the following requirements:

- high performance — minimizing the time spent on operations of adding, searching, modifying and deleting stored information
- minimum memory and disk space usage for storing sc-texts;
- scalability — the ability to add computing power as the load increases without difficulty.

The following sections of this paper will discuss a possible sc-memory model and its implementation.

III. Proposed sc-memory model for next-generation intelligent systems

So, as mentioned above, in general *sc-memory* performs the following tasks:

- the task of storing sc-constructs and information constructs external to the *SC-code* (ostis-system files),
- the task of managing events and processes working on these constructs,
- the task of managing access to sc-constructions, including tasks of:
  - creating and deleting sc-elements (sc-nodes, sc- connectors, etc.);
  - searching sc-constructions by specified sc- elements;
  - getting sc-element characteristics (type, incident sc-elements);
  - adding content to sc nodes;
  - retrieving ostis-system files by known contents
  - retrieving content from ostis-system files.

In this regard, all entities in sc-memory are:

- elements of sc-constructions:
  - sc-nodes, ostis-system files,
  - or sc-connectors (sc-arcs, sc-ribs) between sc-nodes, ostis-system files;
- elements of information constructions that are exter- nal to the SC-code
  - string content;
  - or binary content;
- subscriptions to events in this sc-memory, that is, subscriptions to the occurrence of items in it;
- active or inactive processes in this sc-memory,
- the synchronization objects of the processes in this sc-memory,
- operations performed by processes in this sc- memory.

Thus, *the model of sc-memory* can be defined quintuple

$$SM = \langle SS, FS, RS, PS, PI \rangle,$$

where

- SS — the sc-element storage model, which is a structure of information about sc-elements,
- FS — the external information construction storage model (file memory), which is a structure of information about external information constructions,
- RS — the event subscription storage model, which is a structure of information about event subscriptions in sc-memory,
- PS — the process storage model that represents the structure of process information in sc-memory,
- PI — the set of operations over sc-memory, i.e., the programming interface of sc-memory.

A. Model of storage of sc-elements in sc-memory

The model of sc-element storage in sc-memory can be represented as

$$SS = \langle S, M_e, n_{s_{le}}, n_{s_{max}}, n_{s_{lv}}, n_{s_{lr}}, m_s, m^n_{s_{lv}}, m^n_{s_{lr}}, SSPI \rangle,$$

where

- $S = \langle s_1, s_2, ..., s_i, ..., s_n \rangle, \overline{i = 1, n}$ — sequence of allocated cell segments in sc-memory of fixed size n;
- $s_i = \{\langle e_i 1, e_i 2, ..., e_i j, ..., e_i m \rangle, n_{e_{le}}, n_{e_{lr}}, m_e\}, j = \overline{1, m},$ — i-th segment of fixed size m, consisting of cells (elements) of sc-memory e_{ij} of fixed size k,
- $n_{e_{le}} \in (\overline{\mathbb{N}} \cup \{0\})$ — the index of the last engaged cell in the segment s_i ,
- $n_{e_{lr}} \in (\overline{\mathbb{N}} \cup \{0\})$ — the index of the last released cell in the segment s_i ,
- $m_e \in M$ — the object that synchronizes access to $n_{e_{le}}$ and n_{e_r}
- $M_e \subseteq E \times M$ — a dynamic oriented set of pairs of sc-elements and corresponding synchronization objects;
- $n_{s_{le}} \in (\overline{\mathbb{N}} \cup \{0\})$ — the index of the last engaged segment in sc-memory ($n_{s_{le}} = n$),
- $n_{s_{max}} \in (\overline{\mathbb{N}} \cup \{0\})$ — the maximum number of segments in sc-memory;