- absence of deadlocks, races and hungry processes in sc-memory;
- fast parallel creation of sc-elements in sc-memory due to distribution of processes over sc-memory segments;
- fast non-blockable parallel search of scconstructions provided that no other operations are performed on these sc-constructions.

F. Model of sc-memory programming interface

The model of sc-memory programming interface can be defined as follows

$$PI = N^T \times C^{E\times E\times T} \times F^{N\times L}\times$$

$$\times\{E^{E\times\{E\cup T\}\times E}\cup E^{\{E\cup T\}\times T\times E}\cup E^{E\times T\times\{E\cup T\}}\}\times$$
$$\times T^E\times\ F^L\times\ L^F\times\{\top,\bot\}^E\ \times\ V^{E\times T_V\times AG}\times\{\top,\bot\}^V$$

where

- N^T is the operation of creating an sc-node with the specified type;
- $C^{E\times E\times T}$ is the operation of creating an scconnector between two given sc-elements with the specified type;
- $F^{N\times L}$ is the operation of setting the contents to an sc-node;
- $\{E^{E\times\{E\cup T\}\times E}\cup E^{\{E\cup T\}\times T\times E}\cup E^{E\times T\times\{E\cup T\}}\}$ are operations of searching for three-element sc-constructions by given first and/or second and/or third sc-elements;
- T^E is the operation of obtaining the type of the given sc-element;
- F^L is the operation of obtaining ostis-system files by their contents;
- L^F is the operation of obtaining the content from the ostis-system file;
- $\{\top,\bot\}^E$ is the operation of deleting the specified sc-element.
- $V^{E\times T_V\times AG}$ — operation of creating a subscription to an event in sc-memory;
- $\{\top,\bot\}^V$ — operation of removing a subscription to an event in sc-memory;

Let us consider some of the algorithms of the described operations. The algorithm of the operation of creating an sc-node with the specified type N^T can be described as follows:

- Step 1: Verify that the specified sc-element type is a subtype of sc-node.
  - If the specified sc-element type is not a subtype of sc-node, then terminate the algorithm with an error.
  - Otherwise, proceed to step 2.
- Step 2: Allocate a new sc-memory cell for the scnode.
  - If the sc-memory is engaged, terminate the algorithm with an error.
  - Otherwise, proceed to step 3.
- Step 3: Set the type for the cell as sc-node with the specified type, go to step 4.
- Step 4: Return the resulting sc-address of the scnode and terminate.

The algorithm for the operation of creating an scconnector between two given sc-elements with the specified type $C^{E\times E\times T}$ can be described as follows:

- Step 1: Verify that the specified sc-element type is a subtype of sc-connector.
  - If the specified sc-element type is not a subtype of sc-node, then terminate the algorithm with an error.
  - Otherwise, proceed to step 2.
- Step 2: Check that the sc-addresses of the start and end sc-elements are valid.
  - If the sc-address is not valid, then terminate the algorithm with an error.
  - Otherwise, proceed to step 3.
- Step 3: Allocate a new sc-memory cell for the scconnector
  - If the sc-memory is engaged, terminate the algorithm with an error
  - Otherwise, proceed to step 4.
- Step 4: Set the type for the cell as sc-connector with the specified type, go to step 5.
- Step 5: Add the sc-connector to the list of outgoing and incoming arcs of the start and end sc-elements, go to step 6.
- Step 6: Notify the outgoing and incoming arc addition events, go to step 7.
- Step 7: Return the resulting sc-connector address and terminate.

The algorithm of the operation of deleting a given scelement $\{\top,\bot\}^E$ can be described as follows:

- Step 1: Attempt to acquire a cell in sc-memory by the sc-address of the sc-element.
  - If the cell is not found, terminate the algorithm with an error.
  - Otherwise, proceed to step 2.
- Step 2: Initialize the stack to remove sc-elements, go to step 3.
- Step 3: Put the sc-address of the sc-element to be deleted into the deletion stack, go to step 4.
- Step 4: Place the sc-addresses of all sc-connectors for which the given sc-element is the start or end scelement and the sc-addresses of all connectors for which the found sc-connectors are the start or end sc-elements on the deletion stack, go to step 5.
- Step 5: While the deletion stack is not empty, set the cells for the sc-elements as released and release all those cells.

- For all sc-connectors to be deleted, notify outgoing and incoming sc-connector deletion events.
- For all sc-cells to be deleted, notify sc-cell deletion events.
- Go to step 6.

- Step 6: Destroy the stack for sc-element deletion, go to step 7.

- Step 7: Terminate.

The algorithm for the operation of setting the content to a given sc-node $F^{N\times L}$ can be described as follows:

- Step 1: Attempt to retrieve a cell in sc memory by the sc address of the sc node.
  - If the cell is not found, then terminate the algorithm with an error.
  - Otherwise, proceed to step 2.

- Step 2: Change the sc-node type to an ostis-system file, go to step 3.

- Step 3: Add the string to the file storage of scmemory.
  - Add the string to a free segment of the file storage.
  - Assign matches between this string and the specified ostis file.
  - Notify the event of changing the content of the ostis-system file.
  - Go to step 4.

- Step 4: Terminate.

The principles of search operations were discussed in [2].

This programming interface provides all the necessary functionality for working with sc-constructions, file constructions, events and processes in the memory.

G. Conclusions

The proposed model of the shared semantic memory includes a formal description of the following (!):

- how to represent, store, and process graph and string constructions, events, and processes in the memory;

- how to ensure efficient execution of operations in this shared memory;

- how to coordinate multiple processes running at the same time on the same memory location,

- how to efficiently utilize the available computing power, etc.,

and allows to (!):

- efficiently organize joint storage of graph constructions and string content of external information constructions not represented as a graph, using graph-dynamic and event-driven models;

- efficiently manage the address space, i.e. distribute information about these constructions the in memory in the most effective way;

- efficiently allocate processes to work with these constructions in single-threaded and multi-threaded environments;

- provide coordinated (synchronized) execution of several processes in one memory;

- ensure consistency of operations at the level of representation and processing of data in the memory.

This model has many merits, but the following issues remain unresolved:

- how to ensure the security of information storage and processing, that is:
  - how to ensure access rights for constructions stored in the memory;
  - how to efficiently process and assign these access rights to processes;
  - and more;

- how to ensure consistency of operations at the knowledge representation and processing level, i.e.:
  - how to implement transactions for graphs;
  - how to ensure the integrity and atomicity of some group of operations on a subgraph;
  - how to ensure error-free execution of these transactions;
  - and more;

- how to ensure the storage and processing of information in teams of intelligent systems [28], that is:
  - how to organize storage and processing of information in distributed memory, i.e. in memory not on one device, but on multiple devices;
  - how to efficiently organize data transfer over a network between several devices;
  - and more;

- how to organize the configuration of memory components from the memory itself.
- and so on.

Nevertheless, the results of this paper are very significant for future work. These questions will be discussed in the following papers. Let us consider some obtained quantitative characteristics of the implementation of the proposed model.

IV. The software implementation of sc-memory for next-generation intelligent systems

A. Description of the sc-memory implementation

The current version of sc-machine is implemented on the Linux operating system (Ubuntu-22.04) [29] and is available on GitHub [30]. When developing sc-machine according to the described model, we used modern development environments (CLion, VSCode), containerization tools (Docker), programming languages (C, C++,

CMake), as well as standard libraries and frameworks supplied together with compilers of the programming languages used. The development was based on the models and tools described in the previous section, as well as the *OSTIS Technology Standard* described in the current version of the *e OSTIS-2023 monographs* [9].

The current Software implementation of sc-memory has the following features:

- The memory allocation and destruction mechanisms of the GLib library are used to manage dynamic memory.
- Prefix trees [31] and linked lists are used as data structures to store *information constructions* that do not belong to *SC-code.* The reasons for that are as follows [32]:
  - prefix structures are fairly easy to understand and minimal in their syntax;
  - prefix structures are convenient enough to store and handle "key-value" relations;
  - access to a value by a key occurs in the worst case for the length of this key [33].

The Implementation of file memory allows storing and searching any kind of information constructs (including binary files).

- To synchronize processes in sc-memory, monitors are implemented and used [34], [27]. They provide:
  - locking mechanisms to prevent multiple processes from simultaneously accessing shared resources, eliminating the possibility of mutual exceptions, race conditions, and data access conflicts;
  - high-precision time synchronization between processes that allows them to work in a coordinated mode, which eliminates the possibility of some processes being hungry.

The implementation of monitors uses mutexes, condition variables, and queues.

- The current *Programming interface of the Software implementation of sc-memory* allows:
  - implement platform-dependent components to a necessary and sufficient extent, almost independently of sc-memory implementation.
  - implement basic tools for designing platformindependent ostis-systems.

- The current Implementation of sc-memory is fully consistent with the current Implementation of scpinterpreter.

In general, *sc-memory* can be implemented in different ways. For example, another variant of *ostis-platform scmemory* can be realized by a program implementation of *Neo4j DBMS.* The difference between such a possible *scmemory* implementation and the current one is that the storage of *graph constructions* and the control of the flow of actions over them should be realized more by means provided by *Neo4j DBMS,* while the representation of *graph constructions* s should be implemented in its own way, because it depends on the *SC-code syntax.* [18].

B. Efficiency of sc-memory operations

The current Software implementation of sc-memory in the Software platform for ostis-systems allows to store and represent *sc-constructions,* external *information constructions* not belonging to *SC-code,* as well as to control and coordinate processes in it.

The results of sc-memory operations testing, which includes the implementation of the process control model, showed that parallel execution of sc-memory operations is efficient when the number of operations is large enough (e.g., 1,000,000 operations) (Table 1).

Table I
Efficiency of using 4 physical threads to perform 1,000,000 sc-memory operations compared to 1 physical thread

Number of physical threads	1 thread	4 thread	
	Response time, ms	Response time, ms	Speedup times
Operations of addition (modification)			
Operation of scn-ode creation	958,025	369,680	2.591
Operation of sc-connector creation	1,299.740	787.001	1.652
Operation of adding content to ostis-system file	29,885.500	9,555.450	3.128
Operations of search			
Operation of searching sc-connectors outgoing from a given scelement	642.378	203.005	3.164
Operation of searching an ostis-system file by its contents	1,608.650	928.555	1.732
Operations of deletion			
Operation of deleting an sc-element	1,850.950	1,746.270	1,060
Operation of deleting sc-connectors outgoing from a given scelement	1,704.620	2,115.500	0.806

Testing and evaluation of the effectiveness of the ostissystems software platform were conducted on one of its latest versions — 0.9.0. This version of the platform solved the problem of controlling processes in the shared semantic memory. During the testing we calculated the main efficiency (performance) indicators of operations over sc-memory in single-threaded and multithreaded environments: response time and throughput, and also calculated the speedup [35] obtained by using parallelism when performing a group of operations of the same class over sc-memory [36]. The computer used was an *HP ProBook Hewlett Packard* laptop with a *Intel(R)*