# Implementation of Information Retrieval Subsystem in the Software Platform of ostis-systems

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

***Abstract*—The article describes the purpose and implementation variants for information retrieval subsystems of next-generation intelligent computer systems. This paper is a formal specification of how the information retrieval subsystem in the current Software implementation of the ostis-platform, as well as its software interface, are implemented, and is a continuation of a series of works on the design and implementation of the basic Software implementation of the ostis-platform [1], [2].**

***Keywords*—information retrieval, information retrieval problem, isomorphic search, graph template, graph information retrieval system, ontological design, graph storage, ostis-platform**

## I. Introduction

One of the most important tasks of *intelligent computer systems* [3] is to satisfy the information needs of users. *Intelligent computer systems* should not only find the necessary (relevant) information for the user, but also provide quality answers to the user's questions. Thus, *intelligent computer systems* based on *graph representation of knowledge* should include entire software complexes for searching information relevant to the user — *graph information retrieval subsystems* [4], [5], [6].

Existing *graph information retrieval systems* are based on the use of *graph algorithms* for searching, storing and presenting information [7], [8]. Graphs are used to model relationships between objects, such as web pages on the Internet, users on social networks, or others. In such systems, users can use search queries to find information in a graph. Queries may be similar to those used in traditional information systems, but instead of searching by keywords, the user searches for objects and the relationships between them.

*Information retrieval tasks* are of great relevance, since at present the amount of information available on the Internet is too large for a person to handle without using appropriate search engines [9]. Information flows are growing every day, and therefore a more efficient and accurate use of information is becoming increasingly important for decision-making, planning, scientific research and other activities. Moreover, the ability to conduct high-quality and accurate information searches is a key skill for people in the modern world.

## II. Existing analogs of graph information retrieval systems

Modern *graph information retrieval systems* use the *PageRank* [10] algorithm to determine the relevance of search results. *PageRank* evaluates the importance of each object in the graph based on the links it contains from other objects, and those objects that are considered more important are ranked higher. Graphs also allow the use of analytical algorithms, such as community detection algorithms, to identify subgraphs that group objects according to certain criteria. This can help users find information that might not be found in a traditional keyword search.

Examples of *graph information retrieval systems* are *Google Knowledge Graph* [11], *Facebook Graph Search*, *LinkedIn Skills Graph* [12] and *Neo4j* [13].

The use of *graph data models* in solving *information retrieval tasks* is explained as follows:

- Data processing performance is improved by one or more orders of magnitude when representing data as *graphs*, due to the properties of *graphs* themselves. Unlike *relational databases*, where query performance degrades as the dataset grows with increasing query intensity, *graph data model* performance remains constant even as the dataset grows. This is due to the fact that data processing is localized in some part of a *graph*. As a result, the execution time of each request is proportional only to the size of the *graph* part traversed to satisfy this request, and not to the size of the entire *graph* [14].
- *Graph data models* have tremendous expressive power. *Graph databases* offer an extremely flexible data model and way of representing it. *Graphs* are additive, which provides the flexibility to add new data relationships, new nodes, and new subgraphs

to an existing *graph* structure without violating its integrity and coherence.

In general, *graph information retrieval systems* allow efficient organization and retrieval of information using a graph's structure. This allows you to quickly and efficiently process large amounts of data and provide the user with the most relevant information.

## III. Implementation of the information retrieval subsystem in the current Software implementation of the OSTIS-platform

Based on the current *Software implementation of the ostis-platform* [2] for *next-generation intelligent computer systems*, implemented according to the principles of *OSTIS Technology* [15], there is a need to create an information retrieval subsystem that will allow:

- solve *information retrieval tasks* of any level of complexity [16];
- implement *information retrieval subsystems* in *platform-dependent* and *platform-independent ostis-systems* for application purposes.

In addition, to provide interaction between *information retrieval subsystem of ostis-platform* and information retrieval subsystems of platform-dependent components and ostis-systems required *programming interface*.

*SCin-code* described in [2] is enough to represent *sc-texts* inside the *sc-memory of the ostis-platform* [17]. To translate some *sc-text* into *sc-memory of the ostis-platform*, you must use the methods of the current *Implementation of sc-memory in the ostis-platform*. The sc-memory methods described below are the formal specification of the current ***Software interface of Implementation of sc-memory in the ostis-platform***, with which you can perform actions with *sc-memory*.

In the current *Implementation of sc-memory in the ostis-platform* all program methods are implemented in *method presentation languages C* and *C++*. The current *Software interface of the Implementation of sc-memory in the ostis-platform* contains the necessary functionality not only to perform actions on the *elements of sc-memory*, but also — on the *elements of file memory* [2]. This *software interface* is one of the languages of the current ***Software implementation of the ostis-platform*** for performing actions on sc-memory and can be used to solve *problems* of any information-based complexity. So, for example, this software interface is used in the current *Implementation of the Server system based on Websocket and JSON providing network access to this sc-memory* [2], *Implementation of the ostis-system reusable component manager* and *Implementation of the interpreter for logical models of problem solving* [18], as well as when implementing any *platform-dependent ostis-systems* for any purpose.

***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 to create, modify, and delete 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 to find 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*

Logically, the current *Software interface of the implementation of sc-memory in the ostis-platform* is divided into two software interfaces: ***Software interface of information-forming methods of Implementation of sc-memory in the ostis-platform*** and ***Software interface of information retrieval methods Implementation of sc-memory in ostis-platform***. First of all, this division is due to the fact that the implementation of information retrieval methods in the current *Implementation of sc-memory in the ostis-platform* is rather complicated and

requires much more clarification when describing this implementation. Also, this separation of the *software interface* allows the specification of the methods of the *Software interface of the Implementation of sc-memory in the ostis-platform* to be singled out and structured in such a way that it remains uniform, compact and simple for an external user. As such, there is no physical separation in the *Software interface of the Implementation of sc-memory in the ostis-platform*, all methods of the *Software interface of the Implementation of sc-memory in the ostis-platform* can be used in the same programmatic way and are components ***Reusable component library of the Software implementation of the ostis-platform***, that is, they can be used in the implementation of other special-purpose components.

## IV. Implementation of iterative search for constructions in the sc-memory of the ostis-platform

In tasks solved in applied ostis-systems implemented on the basis of the current *Software implementation of ostis-platform*, it is necessary to use search mechanisms for already existing elements or *constructions in sc-memory*. Such mechanisms are part of the ***Implementation of the information retrieval subsystem for the Implementation of sc-memory in the ostis-platform***, on the basis of which *information retrieval subsystems* can be implemented for *platform-dependent* and *platform-independent ostis-systems*. Despite the complexity of *information retrieval*, current *Software implementation of the ostis-platform* makes it possible to effectively use the implemented *information retrieval* methods in tasks solved by applied ostis-systems. This subsystem cannot be implemented independently of the implementation of the *ostis-platform*, that is, it cannot be made *platform-independent*, so it is necessary to separate *Implementation of the information retrieval subsystem for the Implementation of sc-memory in the ostis-platform* and *Implementation of the information retrieval subsystem of the OSTIS Metasystem* [19], which is implemented in the *SCP Language* [17], independently of the current *Software implementation of the ostis-platform*. The *scp-interpreter* itself must use information retrieval methods of the *Implementation of sc-memory in the ostis-platform*, and the *SCP language* must provide the ability to navigate through the *knowledge base* of any *ostis-systems*.

***Software interface for information retrieval methods of Implementation of sc-memory in the ostis-platform***

- ∋ *Method for creating a three-element sc-memory construction search iterator*
- ∋ *Method for creating a five-element sc-memory construction search iterator*
- ∋ *Method for finding sc-memory constructions isomorphic to the specified graph template*
- ∋ *Method for creating sc-memory constructions isomorphic to the specified graph template*
- ∋ *Method for creating an object of the graph template*

***Method for creating a three-element sc-memory construction search iterator***

- ∈ *method*
- ⇒ *input argument classes of a method**:
  - ⟨•
    - *parameter of the Method for creating an sc-memory construction search iterator*
    - *parameter of the Method for creating an sc-memory construction search iterator*
    - *parameter of the Method for creating an sc-memory construction search iterator*
  - ⟩
- ⇒ *method result class**:
  - *three-element sc-memory construction search iterator*
- ⇒ *class of exceptions**:
  - *element with the specified sc-address does not exist in sc-memory*

***Method for creating a five-element sc-memory construction search iterator***

- ⇒ *method input argument classes**:
  - ⟨•
    - *parameter of the Method for creating an sc-memory construction search iterator*
    - *parameter of the Method for creating an sc-memory construction search iterator*
    - *parameter of the Method for creating an sc-memory construction search iterator*
    - *parameter of the Method for creating an sc-memory construction search iterator*
    - *parameter of the Method for creating an sc-memory construction search iterator*
  - ⟩
- ⇒ *method result class**:
  - *five-element sc-memory construction search iterator*
- ⇒ *class of exceptions**:
  - *element with the specified sc-address does not exist in sc-memory*

According to the rules of *SC-code syntax*, *sc-constructions*, i.e. constructions consisting of *sc-elements*, can consist of three *sc-elements* (Figure. 1), five *sc-elements* (Figure. 2), seven *sc-elements*, and so on [20]. In the *sc-memory of the ostis-platform*, the equivalent of an *sc-construction* is a construction consisting of *sc-memory elements* (***sc-memory construction***). ***Method for creating a three-element sc-memory construction search iterator*** and ***Method for creating a five-element sc-memory***

***construction search iterator*** allow you to create iterators for searching for three- and five-element *constructions in sc-memory of the ostis-platform*, respectively. Using the parameters of these methods, you can create iterators of any necessary configuration to search for *three-* and *five-element sc-constructions*. So, for example, if it is necessary to find all *sc-memory elements corresponding to sc-elements* that belong to some *sc-set* to which a given *sc-memory element* corresponds, then it is necessary to use *Method for creating a three-element sc-memory construction search iterator* to create a search iterator, specifying as the first argument the *sc-memory element* corresponding to the specified *sc-set*, and as the second and third arguments — *class of sc-memory elements corresponding to base sc-arc^* and *class of sc-memory elements corresponding to sc-elements of unspecified class^*, respectively. To search *sc-memory* for more complex structures consisting of seven or more elements, you need to combine the search iterators for three- and five-element *constructions in sc-memory*, or use the ***Method for searching for sc-memory constructions isomorphic to the specified graph template***.


Figure 1. SC.g-text. Example of three-element sc-construction

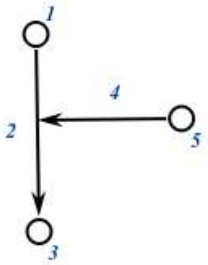


Figure 2. SC.g-text. Example of five-element sc-construction

The *software interface* of these *methods* is constrained by the *method representation language C++*. For example, using these *methods* you cannot create *iterators* by specifying only *sc-memory element classes^* as arguments, by which you need to find all *corresponding constructions in sc-memory* , where *classes of their sc-storage^* are the classes passed as arguments to the specified methods, nor is it possible to specify as arguments anything other than an *sc-address* or an *sc-memory element class^*. An attempt to perform any of the above will result in an error of "gluing" the interface of one of the specified *methods* specified in the *C++* header file with the implementation with the specified parameters specified in the *C++* source file because the *C++* compiler cannot find an implementation for the specified *software interface*. Thus, due to the aforementioned problem, the parameters for *Method for creating a three-element sc-memory construction search iterator* and *Method for creating a five-element sc-memory construction search iterator* can be *sc-address* and/or *sc-memory element class^*.

The *iterators* created using the specified *methods* also have a *software interface*. The results of both *methods* are different *iterators*, that share, however, the same *software interface*. Such *iterators* allow you to work with *sc-memory constructions* at the same moment they are found. Using the *Method of moving to the next sc-memory construction "suitable" for the specified iterator*, the *iterator* updates its internal state each time a new *sc-memory construction* is found. By *next sc-memory construction "suitable" for the specified iterator* we mean an *sc-memory construction* whose elements match the configuration of the created *sc-memory construction search iterator* and which was not found earlier. The result of the latter *method* is the boolean *True* value if the next "suitable" construction for the specified *iterator* exists in *sc-memory*. If there are no "suitable" constructions for this *iterator* in *sc-memory*, then the *Method of moving to the next sc-memory construction "suitable" for the specified iterator* results in the boolean value *False*. To get the *sc-address* of some of the elements of the found *sc-memory construction*, you must use the *Method of accessing the sc-address of the specified sc-memory construction element by the position number of this element in the specified sc-memory construction*, specifying as an argument an integer value in the form of the position index of the searched element in this construction. In this case, the indexing of the positions of elements in the *sc-memory construction* in the current *Implementation of sc-memory in the ostis-platform* starts from zero, not from one, and the indexing order is determined by the order of arguments that have been used when creating the *iterator*. If you try to specify for this method an index for which there is no position in this construction, this method will result in an invalid element position in the specified *sc-memory construction exception*. Thus, the range of indices for *three-element constructions* is limited from zero to two, and for five-element constructions, from zero to four.

***Software interface for information retrieval methods of Implementation of sc-memory in the ostis-platform***

⊃=

{

⊃ *Extension of Software interface for information*

*retrieval methods of Implementation of sc-memory in the ostis-platform*

***three- and five-element sc-memory construction search iterator***
⊂ *software object*
:= [ScIterator]
∈ *C++*

***Software interface for three- and five-element sc-memory construction search iterator***
⊃=
{
⇐ *software interface**:
*three- and five-element sc-memory construction search iterator*

***Method of moving to the next sc-memory construction "suitable" for the specified iterator***
∈ *method*
⇒ *method header in method representation language**:
[bool Next() const]
∈ *C++*
⇒ *method result class**:
• *boolean*

***Method of accessing the sc-address of an element of the specified sc-memory construction by the position index of this element in the specified construction***
∈ *method*
⇒ *method header in method representation language**:
[ScAddr Get(size_t idx) const]
∈ *C++*
⇒ *method input argument classes**:
⟨• *32-bit integer*
⟩
⇒ *method result class**:
• *sc-address of sc-memory element*
⇒ *class of exceptions**:
• *invalid element position in the specified sc-memory construction*
}
}

For *Method for creating a three-element sc-memory construction search iterator*, as well as for *Method for creating a five-element sc-memory construction search iterator*, various combinations of parameters can be used, except for combinations where all parameters are classes of *sc-memory elements*. For simplicity and compactness of the terms used at the level of implementation of methods for creating iterators for searching for structures in sc-memory, additional notations are introduced: the symbol "f" (from the English word “fixed”) denotes the fact that the parameter of a given method for creating an *sc-memory construction* search iterator is the *sc-address* of some *sc-memory element*, and the symbol "a" (from the English word “assign”) denotes *sc-memory element class^*. For *Method for creating a three-element sc-memory construction search iterator*, the correct designation of the desired constructions will be a three character long combination of characters "f" and "a", and for *Method for creating a five-element sc-memory construction search iterator* — a five-character combination of "f" and "a". In the *SCP* Language, to indicate whether a variable has the specified value, the corresponding role relations are used: for variables of class "f" — *scp-operand with the specified value'*, for variables of class "a" — *scp-operand with free value'* [17].

***Software interface for information retrieval methods in the Implementation of sc-memory in the ostis-platform***
⊃=
{

***Method for creating a three-element sc-memory construction search iterator***
⊃ *Method for creating an fff-construction search iterator*
∈ *method*
⇒ *method input argument classes**:
⟨• *sc-address of sc-memory element*
• *sc-address of sc-memory element*
• *sc-address of sc-memory element*
⟩
⊃ *Method for creating an faa-construction search iterator*
∈ *method*
⇒ *method input argument classes**:
⟨• *sc-address of sc-memory element*
• *sc-memory element class^*
• *sc-memory element class^*
⟩
⊃ *Method for creating an aaf-construction search iterator*
∈ *method*
⇒ *method input argument classes**:
⟨• *sc-memory element class^*
• *sc-memory element class^*
• *sc-address of sc-memory element*
⟩
⊃ *Method for creating an faf-construction search iterator*
∈ *method*
⇒ *method input argument classes**:
⟨• *sc-address of sc-memory element*

- *sc-memory element class*^
- *sc-address of sc-memory element*

⟩

⊃ *Method for creating an afa-construction search iterator*

∈ *method*

⇒ *method input argument classes**:

⟨• *sc-memory element class*^
- *sc-address of sc-memory element*
- *sc-memory element class*^

⟩

}

These variants of the implementation of the *Method for creating a three-element sc-memory construction search iterator* are sufficient for solving any search and navigation tasks. ***Method for creating an ffa-construction search iterator*** and ***Method for creating an aff-construction search iterator*** are possible, but in practice there is no need to look for a third sc-memory element by the known element corresponding to the sc-connector and the element corresponding to the sc-element from which this sc-connector exits or into which this sc-connector enters. Such a problem can be solved using ***Method for creating an afa-construction search iterator***. However, the implementation of *Method for creating a five-element sc-memory construction search iterator* is not at all necessary, since all tasks solved using this method can also be solved using *Method for creating a three-element sc-memory construction search iterator* , however, the implementation of *Method for creating a five-element sc-memory construction search iterator* allows you to make the text of the method more compact compared to the method that would use *Method for creating a three-element sc-memory construction search iterator*.

The following can be specified as all three arguments for the *Method for creating a three-element sc-memory construction search iterator*:

- *sc-addresses of sc-memory elements* (for example, when solving the problem of checking the incidence of all three specified sc-memory elements),
- *sc-storage element address*, *class of sc-storage elements corresponding to sc-connectors*^ that come out of the *sc-storage element* passed as the first argument, and *sc-address of the sc-memory element* corresponding to some *sc-element* that contains the required *sc-connectors* (for example, when solving the problem of finding all *sc-memory elements corresponding to sc-connectors between sc-elements* for which the specified *sc-memory elements* correspond),
- *sc-storage element address*, *class of sc-storage elements corresponding to sc-connectors*^ that come out of the *sc-storage element* passed as the first argument, and *class of sc-memory elements corresponding to some sc-elements*^, which include the required *sc-connectors* (for example, when solving the problem of finding all *sc-memory elements* that *correspond to sc-connectors* coming from the *sc-element* that matches the *sc-memory element* passed as the first argument),
- *class of sc-memory sc-elements*^, *class of sc-memory elements corresponding to sc-connectors*^ that come out of the *sc-memory elements specified as the first argument*, and *sc-address of the sc-memory element corresponding to some sc-element*, which contains the required *sc-connectors* (for example, when solving the problem of finding all *sc-memory elements corresponding to sc-connectors*, contained in the *sc-element* that matches the *sc-memory element* specified as the third argument),
- *sc-memory element class*^, *sc-address of the sc-memory element corresponding to the sc-connector* that comes out of the *sc-memory element* passed as the first argument, and *class of sc-memory elements corresponding to some sc-element*^, which contains the required *sc-connector* (for example, the task of finding *sc-memory elements corresponding to sc-elements*, one of which is the *sc-element* from which the *sc-connector* emerges, for which the specified *sc-memory element* matches, and the second of which is the *sc-element* that includes this *sc-connector* for which the specified *sc-memory element* matches)
- and so on.

As all five arguments for the ***Method for creating a five-element sc-memory construction search iterator***, other combinations can be specified that are not specified in the presented classification. However, this is not necessary, since all tasks solved using such *iterators* can be solved by already existing *five-element sc-memory construction search iterators*.

## V. Implementation of isomorphic search for sc-memory constructions of the OSTIS-platform according to the specified graph template

*Method for creating a three-element sc-memory construction search iterator* and *Method for creating a five-element sc-memory construction search iterator*, as well as *Software interface for three- and five-element sc-memory construction search iterator* are quite powerful tools for solving any *information retrieval problems* in applied *ostis-systems*. For example, in *inference* [18] problems, it is considered convenient to solve *problems* when *search for structures* of any necessary *configuration in sc-memory* reduces to ***isomorphic search*** of these constructions according to the specified ***graph template*** (Figure. 3). Such *graph templates* can be any *atomic logical formulas* included in any other *non-atomic formula* [18].

*Isomorphic search* is one way to solve the *problem* of finding a subgraph in a *graph* (see [21]). The problem consists of finding all occurrences of the specified *graph*

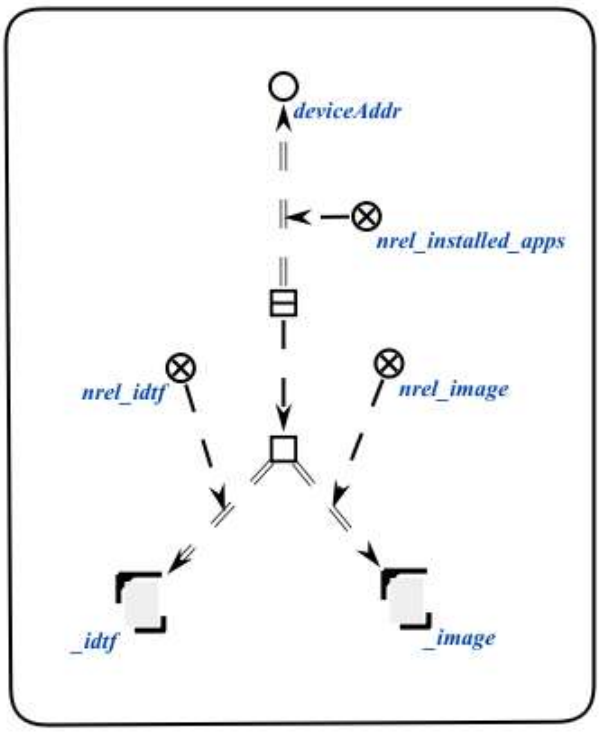


Figure 3. SC.g-text. Graph template example

*template* in the source graph. The *isomorphic search* process can be implemented using various algorithms. One of them is *Ullman's Algorithm* [22], which is based on using an adjacency matrix to determine the correspondence between graph vertices. Another algorithm is the *VF2 Algorithm*, which uses a comparison function to check if the corresponding nodes and edges in two graphs match (see [23]). In modern computer science, there are algorithms that allow solving the *problem* of *isomorphic search* in subexponential time (see [24]).

The fundamental principle of the *OSTIS Technology* that is currently under development is the principle of adopting the best existing technologies for the development of *ostis-systems* [25]. However, due to various circumstances, for example, connected with the specific features of the *Implementation of sc-memory in the ostis-platform*, as well as the requirements imposed on sc-agents involved in logical inference, it is necessary to apply and test new solutions. Within the framework of the current*Implementation of sc-memory in the ostis-platform*, a concept of *isomorphic search* has been developed, which allows to find graphs isomorphic to fragments of a given *graph template in optimal time*.

In general there is no need to implement *isomorphic search* in a generic way. This is explained by the following:

- *isomorphic search* is an NP-complete problem, which means that the cost of solving it grows exponentially with the size of the input data, and there is no efficient algorithm for solving the *isomorphic search* problem yet;
- as a result of determining the isomorphism of two given graphs, several nodes can be found that correspond to each other, but are not actually isomorphic;
- due to the exponential growth in the number of possible isomorphism variants with increasing graph size, even small errors in the calculation of isomorphism can lead to severe distortion of the results;
- for large graphs, the time spent on enumeration of all possible isomorphisms can be very high. This can reduce search efficiency and limit the use of *isomorphic search* in real-world problems (see [26]);
- search complexity increases with the number of loops in the original *graph*, as it results in more iterations;
- existing algorithms are either slow or waste memory, resulting in *isomorphic search* being slow.

Some isomorphic search algorithms even have *O(n!)* complexity and cannot be used for large graphs (see [21]). Despite all the problems associated with *isomorphic search*, for the convenience of solving logical problems, the current *Software implementation of the ostis-platform* implements the "most appropriate" *isomorphic search* algorithm. The current version of *isomorphic search* is implemented in the *Method for finding sc-memory constructions isomorphic to the specified graph template*. This method allows you to find *sc-memory constructions* that are isomorphic not just to some *graph template* that is represented in *sc-memory*, but to the *program object* of this *graph template*, i.e. *graph template*, which is presented in a program format convenient for quick processing.

∋ *Method for finding sc-memory constructions isomorphic to the specified graph template*
∈ *method*
⇒ *method header in method representation language**:
**[**ScTemplate::Result HelperSearchTemplate(ScTemplate const & templ, ScTemplateSearchResult & result)**]**
∈ *C++*
⇒ *input argument classes of a method**:
⟨• *graph template program object*
• *tuple of program objects of sc-memory constructions isomorphic to the specified graph template*
⟩
⇒ *method result class**:
• *error code of the result of creating a program object by the specified element corresponding to the graph template*
⇒ *class of exceptions**:
• *syntactically incorrect graph template*

*program object*
- *semantically incorrect graph template program object*

∋ *Method for creating a construction in sc-memory that is isomorphic to the specified graph template*
∈ *method*
⇒ *method header in method representation language**:
[ScTemplate::Result HelperGenTemplate(ScTemplate const & templ, ScTemplateGenResult & result)]
∈ *C++*
⇒ *input argument classes of a method**:
⟨• *graph template program object*
• *program object of the sc-memory construction isomorphic to the specified graph template*
⟩
⇒ *method result class**:
- *error code of the result of creating a program object by the specified element corresponding to the graph template*

⇒ *class of exceptions**:
- *syntactically incorrect graph template program object*
- *semantically incorrect graph template program object*

***Method for creating a graph template program object***
∈ *method*
⇒ *method header in method representation language**:
[ScTemplate::Result HelperBuildTemplate(ScTemplate & templ, ScAddr const & templAddr)]
∈ *C++*
⇒ *input argument classes of a method**:
⟨• *graph template program object*
• *sc-address of sc-memory element*
⟩
⇒ *method result class**:
- *error code of the result of creating a program object by the specified element corresponding to the graph template*

⇒ *class of exceptions**:
- *syntactically incorrect graph template program object*
- *semantically incorrect graph template program object*

***Graph template program object*** can be generated using the *Method for creating graph template program object* by passing as arguments *graph template program object* as an output parameter and *sc-address of the sc-memory element corresponding to the sc-structure* of an *atomic logical formula* (sc-template). Like *three- and five-element sc-memory construction search iterator*, *graph template program object* has a specialized *software interface*. Initially, *graph template program object* is set to empty. Using ***Method of adding a three-element construction for the specified graph template*** and ***Method of adding a five-element construction for the specified graph template***, you can extend the specified *graph template program object*. The *graph template program object* is expanded with the addition of three-element constructions to the ***tuple of program objects of three-element constructions in the specified graph template***, while the order of the program objects of constructions in this set is specified by the execution sequence *Method for adding a three-element construction for the specified graph template* and *Method for adding a five-element construction for the specified graph template*.

***Software interface for information retrieval methods of Implementation of sc-memory in the ostis-platform***
⊃=
{

***graph-template program object***
:= [atomic logical formula program object]
:= [sc-template program object]
⊂ *program object*
⇒ *concept specifying the specified entity**:
{• *tuple of program objects of three-element constructions in the specified graph template*
• *local identifier of the sc-memory element and position indices of this element in the specified graph template**
• *set of sets of indices of program objects of three-element construction in the specified graph template, ordered by the search priority*
• *local identifier of the sc-memory element and sc-address of this element in the specified graph template**
• *local identifier of the sc-memory element and the class of this element in the specified graph template**
}

***program object of a three-element construction of the specified graph template***
:= [ScTemplateTriple]
∈ *C++*
∈ *program object*
⇒ *concept specifying the specified entity**:
{• *index of the program object of the three-element structure in the specified graph template*

:= [size_t m_index]

- *tuple of three elements of a program object of a three-element construction*
  := [std::array<ScTemplateItem, 3> m_values]

}

***element of program object of three-element construction***

:= [ScTemplateItem]

∈ *C++*

∈ *program object*

⇒ *concept specifying the specified entity**:

{•
- *sc-address of the program object element of the three-element structure*
  := [ScAddr m_addrValue]
  ⊂ *sc-address of sc-memory element*
- *element class of a program object of a three-element construction*
  := [ScType m_typeValue]
  ⊂ *sc-memory element class^*
- *local identifier of the program object of the three-element construction*
  := [std::string m_name]

}

***Software interface of graph template program object***

⊃=

{

⇐ *software interface**:
*graph template program object*

***Method for adding three-element construction to the specified graph template***

∈ *method*

⇒ *method header in method representation language**:
[ScTemplate & Triple(ScTemplateItemValue const & param1, ScTemplateItemValue const & param2, ScTemplateItemValue const & param3)]

∈ *C++*

⇒ *input argument classes of a method**:

⟨•
- *parameter of the Method for adding construction to the specified graph template*
- *parameter of the Method for adding construction to the specified graph template*
- *parameter of the Method for adding construction to the specified graph template*

⟩

⇒ *method result class**:
- *graph template program object*

⇒ *class of exceptions**:
- *incorrect parameter of the Method for adding construction to the specified graph template*
  ⊃ *local identifier is not previously bound to the sc-address of the program object element of the three-element construction in the specified graph template program object*
  ⊃ *local identifier has already been used for another sc-address of the three-element construction program object element in the specified graph template program object*
  ⊃ *the same local identifier is simultaneously specified for the second and first (third) element of the specified program object of the three-element construction*
  ⊃ *element class of the created program object of the three-element construction in the specified program object of the graph template is the class of elements in the sc-memory corresponding to the sc-constants*
  ⊃ *element with the sc-address of the element of the created program object of the three-element construction in the specified program object of the graph template does not exist in sc-memory*

***Method for adding five-element construction to the specified graph template***

∈ *method*

⇒ *method header in method representation language**:
[ScTemplate & Fiver(ScTemplateItemValue const & param1, ScTemplateItemValue const & param2, ScTemplateItemValue const & param3, ScTemplateItemValue const & param4, ScTemplateItemValue const & param5)]

∈ *C++*

⇒ *input argument classes of a method**:

⟨•
- *parameter of the Method for adding construction to the specified graph template*
- *parameter of the Method for adding construction to the specified graph template*
- *parameter of the Method for adding*

*construction to the specified graph template*
- *parameter of the Method for adding construction to the specified graph template*
- *parameter of the Method for adding construction to the specified graph template*

⟩
⇒ *method result class**:
- *graph template program object*

⇒ *class of exceptions**:
- *incorrect parameter of the Method for adding construction to the specified graph template*
  - ⊃ *local identifier is not previously bound to the sc-address of the program object element of the three-element construct in the specified graph template program object*
  - ⊃ *local identifier has already been used for another sc-address of the three-element construction program object element in the specified graph template program object*
  - ⊃ *the same local identifier is simultaneously specified for the second and first (third) elements of the specified program object of the three-element construction*
  - ⊃ *element class of the created program object of the three-element construction in the specified program object of the graph template is the class of elements in the sc-memory corresponding to the sc-constants*
  - ⊃ *element with the sc-address of the element of the created program object of the three-element construction in the specified program object of the graph template does not exist in sc-memory*

}
}

To form the necessary *graph template program object*, do the following:

- If *graph template program object* has not been created before, then it must be created.
- For the created *graph template program object* apply several times *Method of adding a three-element structure for the specified graph template* (*Method of adding a five-element structure for the specified graph template*), specifying as three (five) input parameters the ***parameters of the Method for adding a construction for the specified graph template***, depending on the desired configuration of the *three-element (five-element) construction program object* to be added.

At the same time, *parameter of the Method for adding a construction for the specified graph template* differs significantly from the *parameter of the Method for creating sc-memory construction search interator*. In addition to sc-addresses and classes of sc-memory elements, local identifiers of these addresses or classes can be specified in the created graph template. This greatly simplifies the process of creating the *graph template program object*, when it is necessary to specify *sc-address* or *sc-memory element class^* in the added construction, which was already specified earlier in another construction the specified *graph template*. Thus, using such a local identifier, it is possible to refer to the parameter of an already previously added construction in the specified graph template. In addition, this method allows you to get elements from the structures found by the specified *graph template* in the sc-memory using such local identifiers.

Regardless of what methods were applied to the created *graph template program object*, in the structure of the *graph template program object* itself, for the convenience of representing and processing data, only ***software objects of three-element constructions*** are created. Each element in *program object of a three-element construction*, except for the sc-address, class and local identifier, has its own position index within this construction, set in the range from zero to two, as well as the position index within the entire graph template, calculated as the sum of the product of the number of *the three-element construction in the specified graph template* and *number three* and *position index of this element within the graph template*. Adding *program object of a three-element construction in the specified graph template program object* to *graph template program object* is done as follows:

- If the argument specified as the second parameter has a local identifier in the specified graph template, and this local identifier is also specified for the first or second argument, then terminate the *Method for adding a three-element construct for the specified graph template* with the following exception: *the same local identifier is simultaneously specified for the second and first (third) elements of the specified program object of the three-element construction in the specified program object of the graph template*.
- If any of the parameters is specified as the class of the sc-memory element corresponding to the sc-constant, then terminate the *Method of adding a three-element construction for the specified graph template* with the following exception: *the element class of the three-*

*element construct program object being created in the specified graph template program object is the class of the elements in the sc-memory corresponding to the sc-constants*.

- If any of the parameters is given as the sc-address of a non-existent element in sc-memory, then terminate the *Method of adding a three-element construction for the specified graph template* with the following exception: *the element with the sc-address of the element of the created program object of the three-element construction in the specified program object of the graph template does not exist in the sc-memory*.
- For all parameters that have local identifiers in the specified graph template and sc-addresses of elements in sc-memory, add all pairs with these local identifiers and the corresponding sc-addresses of elements in sc-memory to the *r*elation local identifier of the sc-storage element and the sc-address of this element in the specified graph template*, otherwise, if the sc-addresses of the elements in the sc-storage for these local identifiers are known in relation to *local identifier of the sc-storage element and sc- the address of this element in the specified graph template**, specify the known sc-addresses of the elements in the sc-memory for the specified parameters.
- For all parameters that have local identifiers in the specified graph template and element classes in sc-memory, add all pairs with these local identifiers and the corresponding element classes in sc-memory to the relation *local element identifier sc-memory and the class of this element in the specified reference graph**.
- For all parameters for which only local identifiers are specified in the specified reference graph, as well as for *the local identifier of the sc-memory element and the position numbers of this element in the specified reference graph**, if position indices are unknown by these local identifiers of the corresponding elements in the specified *graph template*, then add to this relation all pairs with local identifiers and position indices of the corresponding elements in the specified template graph.
- For the obtained *program object of a three-element construct in the specified graph template program object*, calculate the priority number required when executing the *Method for finding sc-memory constructions isomorphic to the specified graph template*:
  - If *program object of a three-element construction* contains *sc-addresses of sc-memory elements* for all elements, then the priority number of the specified construct is considered equal to zero (that is, it is considered the highest priority).
  - If the *software object of a three-element construction* contains *sc-address of the sc-memory element corresponding to the sc-connector* for the second element, then the priority number of the specified construct is considered equal to one (i.e. it is the second by priority).
  - If in the *program object of a three-element construction* for the first and third elements *sc-addresses of the sc-memory element* are specified, then the priority number of the specified construction is considered equal to two (that is, it is considered the third in priority).
  - If the *program object of a three-element construction* contains *sc-address of the sc-memory element* only for the third element, then the priority number of the specified construct is considered equal to three.
  - If in the *program object of a three-element construction* the first element is *sc-address of the sc-memory element*, and the third element is *the class of the sc-memory element corresponding to sc-node^*, then the priority number of the specified structure is considered equal to four.
  - If in the *program object of a three-element construction* the first element is *sc-address of the sc-memory element*, and the third element is *the class of the sc-memory element corresponding to sc-connector^*, then the priority number of the specified structure is considered equal to five.
  - If there are no elements in the *software object of a three-element construction* for which *sc-addresses of sc-memory elements* are specified, then the priority number of the specified construct is considered equal to six (that is, it is considered to be of the lowest priority).

  After determining the priority number of the specified *program object of the three-element construct in the specified graph template program object*, add this object to the set with the position equal to the calculated priority number of the *set of sets of indices of program objects of three-element construction in the specified graph template, ordered by the search priority*.
- The obtained *program object of a three-element construction of the specified graph template* is added to *tuple of program objects of three-element constructions in the specified graph template*.

Adding a *five-element construction program object* to *graph template program object* amounts to adding two *three-element construction program object* to this *graph template program object*. At the same time, in the second *three-element construction program object* only the local identifier of the second element of the first added *three-element construct program object in the specified graph template program object* is specified for the specified

*graph template program object*.

To find all *sc-memory constructions* isomorphic to a given *graph template program object*, do the following:

- For the generated *graph template program object* apply the *Method for finding sc-memory constructions isomorphic to the specified graph template*.
- The result of this method will be a *tuple of program objects of all constructions in sc-memory isomorphic to the specified graph template*, which, like the search iterator for three- and five-element constructions in sc-memory, has its own *program interface*.

***Software interface for information retrieval methods of Implementation of sc-memory in the ostis-platform***
⊃=
{

***tuple of program objects of sc-memory constructions isomorphic to the specified graph template***
:= [ScTemplateSearchResult]
∈ *C++*
⊂ *program object*

***Software interface of the tuple of program objects of sc-memory constructions isomorphic to the specified graph template***
⊃=
{
⇐ *software interface**:
*tuple of program objects of sc-memory constructions isomorphic to the specified graph template*

***Method of obtaining a program object of an sc-memory construction isomorphic to the specified graph template by its index in a tuple***
∈ *method*
⇒ *method header in method representation language**:
[ScTemplateSearchResultItem operator[](size_t index) const noexcept(false)]
∈ *C++*
⇒ *input argument classes of a method**:
⟨• *32-bit integer*
⟩
⇒ *method result class**:
• *program object of an sc-memory construction isomorphic to the specified graph template*
⇒ *class of exceptions**:
• *the element with the specified index does not exist in the tuple*

***Method of obtaining a program object of an sc-memory construction isomorphic to the specified graph template by its index in a tuple with a preliminary check for the specified index***
∈ *method*
∈ *method without exceptions*
⇒ *method header in method representation language**:
[bool Get(size_t index, ScTemplateSearchResultItem & outItem) const noexcept]
∈ *C++*
⇒ *input argument classes of a method**:
⟨• *32-bit integer*
• *program object of an sc-memory construction isomorphic to the specified graph template*
⟩
⇒ *method result class**:
∈ *boolean*
}

***program object of an sc-memory construction isomorphic to the specified graph template***
:= [ScTemplateSearchResultItem]
∈ *C++*
⊂ *program object*

***Software interface of the program object of an sc-memory construction isomorphic to the specified graph template***
⊃=
{
⇐ *programming interface**:
*program object of an sc-memory construction isomorphic to the specified graph template*

***Method of obtaining the sc-address of an element of a program object of an sc-memory construction isomorphic to the specified graph template by its index in this program object***
∈ *method*
⇒ *method header in method representation language**:
[ScAddr const & operator[](size_t index) const noexcept(false)]
∈ *C++*
⇒ *input argument classes of a method**:
⟨• *32-bit integer*
⟩
⇒ *method result class**:
• *sc-address of sc-memory element*
⇒ *class of exceptions**:
• *there is no element at the specified index*

*in the program object of the sc-memory construction isomorphic to the specified graph template*

***Method of obtaining the sc-address of an element of a program object of an sc-memory construction isomorphic to the specified graph template by its index in this program object with a preliminary check of the specified index***

∈ *method*
∈ *method without exceptions*
⇒ *method header in method representation language**:
[bool Get(size_t index, ScAddr & outAddr) const noexcept]
∈ *C++*
⇒ *input argument classes of a method**:
⟨• *32-bit integer*
• *sc-address of sc-memory element*
⟩
⇒ *method result class**:
• *boolean*

***Method of obtaining the sc-address of an element of a program object of an sc-memory construction isomorphic to the specified graph template by the local element identifier of the corresponding program object of the three-element construction in the specified program object of the graph template***

∈ *method*
⇒ *method header in method representation language**:
[ScAddr const & operator[](std::string const & name) const noexcept(false)]
∈ *C++*
⇒ *input argument classes of a method**:
⟨• *local identifier of an element of a three-element construction in the specified graph template*
⟩
⇒ *method result class**:
• *sc-address of sc-memory element*
⇒ *class of exceptions**:
• *an element with the specified local identifier does not exist in the program object of an sc-memory construction isomorphic to the specified graph template*

***Method of obtaining the sc-address of an element of a program object of an sc-memory construction isomorphic to the specified graph template by the local identifier of the element of the corresponding program object of a three-element construction of the specified graph template with a preliminary check of the specified local identifier***

∈ *method*
∈ *method without exceptions*
⇒ *method header in method representation language**:
[bool Get(std::string const & name, ScAddr & outAddr) const noexcept]
∈ *C++*
⇒ *input argument classes of a method**:
⟨• *local identifier of an element of a three-element construction in the specified graph template*
• *sc-address of sc-memory element*
⟩
⇒ *method result class**:
• *boolean*
}
}

The current ***Method for finding sc-memory constructions isomorphic to the specified graph template*** consists of two stages: (1) *Stage of preprocessing of the graph template*, (2) *Stage of searching for sc-memory constructions isomorphic to the specified graph template*. At the same time, inside the *Method for finding sc-memory constructions isomorphic to the specified graph template*, a software *iterator for finding sc-memory constructions isomorphic to the specified graph template* is created, which performs the entire isomorphic search algorithm.

***Software interface for information retrieval methods of Implementation of sc-memory in the ostis-platform***
⊃=
{

***Software interface of graph template program object***
⊃=
{

***iterator for finding sc-memory constructions isomorphic to the specified graph template***
:= [ScTemplateSearch]
∈ *C++*
⊂ *program object*
⇒ *concept specifying the specified entity**:
{• *local identifier of some element in some program object of a three-element construction of the specified graph template and the set of all indices of program objects of three-element constructions in the specified graph template with this element**
• *tuple of sets of indices of program objects*

*of three-element constructions of connectivity components in the specified graph template*

- *set of indices of the highest-priority three-element constructions for the search for program objects of connectivity components in the specified graph template*
- *tuple of sets of sc-addresses of sc-memory elements corresponding to sc-connectors, such three-element constructions that are not isomorphic to the corresponding three-element constructions of the specified graph template, whose indices are equal to the position indices of sets in this oriented set*
- *tuple of sets of sc-addresses of sc-memory elements corresponding to sc-connectors of such three-element constructions that are isomorphic to the corresponding three-element constructions of the specified graph template, whose indices are equal to the position indices of the sets in this oriented set*
- *tuple of sets of sc-addresses of sc-memory elements corresponding to sc-connectors that are the second elements of the corresponding three-element constructions of found sc-memory constructions isomorphic to the specified graph template, whose indices are equal to the position indices of sets in this oriented set*
- *tuple of sets of indices of program objects of three-element constructions of the specified graph template for constructions isomorphic to it found from in sc-memory, whose indices are equal to the position indices of sets in this oriented set*
- *index of the last found sc-memory construction according to the specified graph template*
- *set of indices of all found and isomorphic constructions in sc-memory according to the specified graph template*

}
}
}

***The graph template preprocessing step*** consists of the following intermediate processing steps:

- Addition to the relation *local identifier of some element in some program object of a three-element construction of the specified graph template and the set of all indices of program objects of three-element constructions in the specified graph template with this element** of all pairs with local element identifiers in some program object of a three-element construction of the specified graph template and sets of indices of corresponding program objects of three-element constructions in the specified graph template. The sets of such program objects do not include those *program objects of three-element constructions* in the specified graph template, whose indices are included in the local identifier of the element itself. This element's local identifier is not the element's local identifier within the entire graph template. Such a local identifier is formed by the system itself, and not by the user of the *Method of adding a three-element construction for the specified graph template* (*Method of adding a five-element construction for the specified reference graph*) and consists of a local element identifier within the entire graph template and the number of the corresponding program object of the three-element structure in the specified graph template. Knowing such a local identifier of an element of some *program object of a three-element construction* of the graph template, one can quickly access other *program objects of three-element constructions* that contain this element.
- Removal from the sets of indices of *program objects of three-element constructions*, which are the second components of pairs of the relation *local identifier of some element in some program object of a three-element construction of the specified graph template and the set of all indices of program objects of three-element constructions in the specified graph template with this element** and in which there are elements that are the first components of these pairs, all such indices of *program objects of three-element constructions*, passing through which in the process of searching for constructions isomorphic according to the specified graph template can lead to a looping of the search algorithm. This pre-processing stage of the formed *graph template program object* makes it possible to eliminate transitions along such *program objects of three-element constructions* of the specified graph template as much as possible, which significantly complicating the process of isomorphic search for constructions according to the specified graph template. Since SC-code itself allows one to represent constructions of any possible configuration, it is impossible to say exactly which configurations of constructions can lead to cyclic situations when the isomorphic search algorithm for these structures is executed according to the specified graph template. A more universal algorithm for eliminating loops in the graph template can lead to significant additional time costs, since it may require a deeper syntactic analysis in the original graph template, so it is recommended to implement the conditions by which you can determine the program objects of three-

element constructions, the transition to which can lead to cyclic situations in the processing of the graph template. The elimination of loops in the graph template allows the algorithm of isomorphic search for structures on the specified graph template to more efficiently perform all the required operations on graphs, therefore it is a key step in preprocessing the original *graph template program object*. Also, this step cannot be performed together with the previous step, since in order to eliminate all loops in the graph template, it is necessary to know completely all possible transitions along this graph template.

- Search for all connectivity components in the specified graph template, that is, unrelated subgraphs in this graph, and add all *program objects of three-element constructions* corresponding to these connected components to the *tuples of sets of numbers of program objects of three-element constructions connectivity components in the specified graph template*. Thus, this makes it possible to find even such connected components that could be obtained after performing the second step of the algorithm for preprocessing the specified graph template, that is, eliminating cycles in the specified graph template. Dividing a graph template into connectivity components could be one of the solutions to the problem of eliminating a cycle in the specified graph template, however, the algorithm for isomorphic search for structures on the specified graph template is more advanced and allows one to find all three-element constructions for topics of three-element graph template constructions that have the same first or third element, so splitting the graph template into connected components is not used in the previous step.
- The last step of the *graph template preprocessing stage* is to select the connectivity components found at the previous *Pre-processing stage of graph template* of the highest-priority *program objects of three-element constructions* in the specified graph template. The highest priority program object of the three-element design is the object with a priority number equal to zero, the most non-priority object is the object with a priority number equal to six. In this case, if there are several *program objects of three-element constructions* that have the same priority number, then the object with the first (third) element, which is the sc-address of the sc-memory element, is considered to have the highest priority. has the least number of elements corresponding to outgoing (incoming) sc-connectors. As a result, *a set of numbers of the highest-priority three-element constructions for the search for program objects of the connectivity components in the specified graph template* is formed.

Thus, *graph template preprocessing stage* makes it possible to significantly simplify the processing of a graph template at the stage of searching for constructions isomorphic to it. The next ***stage of searching for sc-memory constructions isomorphic to the specified graph template*** includes the following steps:

- If the specified *tuple of program objects of sc-memory constructions isomorphic to the specified graph template* is not empty, then delete all program objects of constructions in sc-memory from it.
- If *the set of numbers of the highest-priority three-element constructions for searching program objects of connectivity components in the specified graph template* is empty, then this means that the specified *graph template* is empty. In this case, the result of the search is an empty *tuple of program objects of sc-memory constructions isomorphic to the specified graph template* and *The stage of searching for sc-memory constructions isomorphic to the specified graph template* ends with a successful result.
- Initialize *number of the last found sc-memory construction according to the specified graph template* with a value equal to zero. Set *Number of the current found sc-memory construction according to the specified graph template* equal to *number of the last found sc-memory construction according to the specified graph template*. Set *set of numbers "equivalent" program objects of constructions* to be equal to set of numbers of the highest priority for searching program objects of three-element constructions of connectivity components in the specified graph template. Set *set of numbers of current program objects of constructions* equal to *set of numbers of "equivalent" program objects of constructions*.
- Select the next number from *set of numbers "equivalent" program objects of constructions*. According to the received number from *tuple of program objects of three-element constructions in the specified graph template* take the corresponding *program object of three-element construction of the specified graph template* in this graph template.
- For the *selected program object of a three-element construction* in the specified graph template, find all such *program objects of three-element constructions*, (1) whose elements have the specified classes and sc-addresses the same as the classes and sc-addresses of the *elements of the selected program object of the three-element construction*, respectively, while either the first or third of their elements have the same local identifiers in the specified graph template or do not have them at all, (2) for which the corresponding replacements were not found, that is, the set located in the *tuple of sets of numbers of program objects of three-element constructions*

*of the specified graph template for isomorphic constructions found using it in sc-memory, whose numbers are equal to the position numbers of sets in this oriented set* by *the number of the current found sc-memory construction according to the specified graph template*, the numbers of the found *program objects of three-element constructions* do not belong, and also the numbers of which do not belong to the *set of numbers of the current program objects of the constructions*.

- If the received *set of numbers of "equivalent" program objects of constructions* is empty, then terminate this iteration of the algorithm.
- If the received *set of numbers "equivalent" program objects of constructions* is not empty, then choose a random number from the *set of numbers "equivalent" program objects of constructions*. According to the received number from *tuple of program objects of three-element constructions in the specified graph template* take the corresponding *program object of three-element construction of the specified graph template* in this graph template.
- Based on the obtained *program object of a three-element construction*, create a *an iterator for searching for three-element sc-memory construction*. Creation of *iterator for searching three-element sc-memory construction* is done using the *Method of creating iterator for searching three-element sc-memory construction*. The parameters of the method are assigned to the elements of the specified *program object of the three-element construction*, while instead of *classes of sc-memory elements corresponding to sc-variables* (*classes of sc-memory elements corresponding to sc-metavariables*), the corresponding them *classes of sc-memory elements corresponding to sc-constants* (*classes of sc-memory elements corresponding to sc-variables*). That is, for example, if some *program object of a three-element construction* element has a *class of sc-memory elements corresponding to variable sc-nodes*, then instead of it for the *Method of creating a three-element sc-memory construction search iterator*, the corresponding *class of sc-memory elements corresponding to constant sc-nodes* is used. Thus, when passing from *program object of a three-element construction* to *program three-element sc-memory construction search iterator*, the degree of variability of the elements of *program object of a three-element construction* decreases: *classes of sc-memory elements corresponding to sc-metavariables* are converted to *classes of sc-memory elements corresponding to sc-variables*, and *classes of sc-memory elements corresponding to sc-variables* — to *classes of sc-memory elements corresponding to sc-constants*. If *class of element of program object of three-element construction* is not a non-strict subset of *class of sc-memory elements corresponding to sc-metavariables* or *class of sc-memory elements corresponding to sc-variables*, then the degree of variability is not reduced .
- Using the *Method of moving to the next sc-memory construction "suitable" for the specified iterator* go to the *sc-memory construction* isomorphic to the specified construction in the graph template.

Despite the wide range of tasks that can be performed using the current implementation of *isomorphic search*, there are a number of reasons why this and other implementations of *isomorphic search* should not be used:

- *isomorphic search* allows you to solve a wide range of problems if all knowledge isomorphic to the specified *graph template* is in the *knowledge base* or is missing. That is, the quality level of *isomorphic search* directly depends on the state of *knowledge base*. The more diverse *knowledge base fragments* are, the worse the performance of *isomorphic search* is;
- The cost of searching for large *graph templates* may be at odds with the desires of the developer or user. The larger the *graph template*, the more situations in which the *isomorphic search* algorithm can behave abnormally.
- Most of the problems solved with *isomorphic search* can and should be solved with *three- and five-element search iterators*. The simpler the method of solving problems, the fewer errors and emergency situations you can get.

## VI. Advantages and disadvantages of the implementation of the information retrieval subsystem in the ostis-platform

The current *Software interface of Implementation of sc-memory in the ostis-platform* allows:

- Implementing platform-specific subsystems of the current *software implementation of the ostis-platform* to the extent necessary and sufficient, practically independently of the *Implementation of sc-memory in the ostis-platform*. That is, the current *Software interface of Implementation of sc-memory in the ostis-platform* is a way to unify access to the software *Implementation of sc-memory in the ostis-platform* and allows easily to replace various implementations of sc-memory with *method representation language C++*, while the *Software interface of Implementation of sc-memory in the ostis-platform* itself practically does not change or does not change at all.
- Implementing basic tools for designing platform-independent ostis systems, e.g. *Implementation of scp-interpreter*.
- Generating and expanding the *Library of reusable components of Software implementation of the ostis-*

*platform* with components that use the methods of *Implementation of the sc-memory in the ostis-platform* and are part of various plug-ins of the current *Software interface of Implementation of sc-memory in the ostis-platform*.
- Providing different levels of access to *Implementation of sc-memory in the ostis-platform*, including the levels of access for different users of the *Software implementation of the ostis-platform*.

It is worth noting that *Software interface of Implementation of sc-memory in ostis-platform* cannot exist separately from the current *Implementation of sc-memory in the ostis-platform*. In addition, it is part of the *Implementation of sc-memory in the ostis-platform*, that is, it is designed and developed in accordance with the implementation of the sc-memory itself. However, if necessary, it can be used for various modifications or versions of the current *Implementation of sc-memory in the ostis-platform*.

Despite the wide range of functionality of the current *Software interface of Implementation of sc-memory in the ostis-platform*, its disadvantages include the following:

- At the level of the *Software interface of Implementation of sc-memory in the ostis-platform*, there is no limit to the range of classes of sc-elements in sc-memory that can be set as arguments, for example, to the *Method of creating an sc-memory element of a given class, corresponding to an sc-node* and *Method of creating an sc-memory element of a given class, corresponding to some sc-connector*.
- Due to shortcomings in the current implementation of the agent architecture in the *software implementation of the ostis-platform* it is impossible for the *Software interface of Implementation of sc-memory in the ostis-platform* to use *Implementation of sc-memory in ostis-platform* stored as a compiled file. First of all, this is due to the fact that platform-specific agents are implemented by means that utilize creation of source files when building the entire platform. Thus compiled files remain dependent on the device where they were built.

In general, isomorphic search can be a useful tool in theoretical studies and some specialized applications, but in most cases there are better ways to work with graphs.

## VII. Conslusion

Let us list the main ideas of this work:

- to solve information retrieval tasks in ostis-systems, the *Implementation of the information retrieval subsystem* of the current *Software implementation of the ostis-platform* is used;
- *Implementation of the information retrieval subsystem in the Software implementation of the ostis-platform* has a software interface that can be used in any platform-dependent component (subsystem);
- *Implementation of the information retrieval subsystem in the Software implementation of the ostis-platform* includes iterative methods for searching for sc-memory constructions and methods for searching for sc-memory constructions according to the specified graph template;
- to solve most information retrieval problems, it is sufficient to use iterative methods for searching for sc-memory constructions;
- the current implementation of isomorphic search is not universal and is limited to a certain set of graph templates, and also strongly depends on the state of the knowledge base.

When designing graph templates in one of the languages of the external representation of SC-code [20], one should:

- Minimize the number of cycles by splitting, for example, key constant sets into subsets that are not interconnected in this graph template. If the cycle in the graph cannot be eliminated, then leave it as it is, or reconsider the original problem for the possibility of simplifying its solution.
- Select among all those sc-constructions that can be selected by the search procedure as the first sc-construction, only the one that simplifies the work of the search procedure as much as possible for the specified one in the subject domain.
- Minimize the number of sc-constructions, the removal of which does not change the meaning of the found constructions and/or can be specified/checked later (for example, when the entity is already found and the class membership can be checked later) and/or the removal of which simplifies the choice of path search in a graph isomorphic to the specified graph template.

## 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] 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.

[2] 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.

[3] 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.

[4] R. Reinanda, E. Meij, M. de Rijke *et al.*, "Knowledge graphs: An information retrieval perspective," *Foundations and Trends® in Information Retrieval*, vol. 14, no. 4, pp. 289–444, 2020.

[5] Y. WHAN KIM and J. H. Kim, "A model of knowledge based information retrieval with hierarchical concept graph," *Journal of Documentation*, vol. 46, no. 2, pp. 113–136, 1990.

[6] A. Parameswaran, A. D. Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom, "Human-assisted graph search: it's okay to ask questions," *arXiv preprint arXiv:1103.3102*, 2011.

[7] S. Ma, J. Li, C. Hu, X. Lin, and J. Huai, "Big graph search: challenges and techniques," *Frontiers of Computer Science*, vol. 10, pp. 387–398, 2016.

[8] J.-A. Fernández-Madrigal and J. González, "Multihierarchical graph search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 1, pp. 103–113, 2002.

[9] G. J. Kowalski, *Information retrieval systems: theory and implementation*. springer, 2007, vol. 1.

[10] W.-C. Yeh, W. Zhu, C.-L. Huang, T.-Y. Hsu, Z. Liu, and S.-Y. Tan, "A new bat and pagerank algorithm for propagation probability in social networks," *Applied Sciences*, vol. 12, no. 14, p. 6858, 2022.

[11] H. Paulheim, "Knowledge graph refinement: A survey of approaches and evaluation methods," *Semantic web*, vol. 8, no. 3, pp. 489–508, 2017.

[12] I. Kivimäki, A. Panchenko, A. Dessy, D. Verdegem, P. Francq, H. Bersini, and M. Saerens, "A graph-based approach to skill extraction from text," in *Proceedings of TextGraphs-8 graph-based methods for natural language processing*, 2013, pp. 79–87.

[13] 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.

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

[15] 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.

[16] 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.

[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] M. K. Orlov and A. P. Vasilevskaya, "Non-procedural problem-solving models in next-generation intelligent computer systems," *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh system [Open semantic technologies for intelligent systems]*, pp. 161–172, 2022.

[19] K. Bantsevich, "Metasystem of the ostis technology and the standard of the ostis technology," *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh system [Open semantic technologies for intelligent systems]*, pp. 357–368, 2022.

[20] 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.

[21] S. Fortin, "The graph isomorphism problem," 1996.

[22] J. R. Ullmann, "An algorithm for subgraph isomorphism," *Journal of the ACM (JACM)*, vol. 23, no. 1, pp. 31–42, 1976.

[23] P. Foggia, C. Sansone, and M. Vento, "A performance comparison of five algorithms for graph isomorphism," in *Proceedings of the 3rd IAPR TC-15 Workshop on Graph-based Representations in Pattern Recognition*. Citeseer, 2001, pp. 188–199.

[24] B. D. McKay, "Nauty user's guide (version 2.4)," *Computer Science Dept., Australian National University*, pp. 225–239, 2007.

[25] 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.

[26] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004.

## Реализация информационно-поисковой подсистемы в Программной платформе ostis-систем

Зотов Н. В.

Описываются назначение и варианты реализации информационно-поисковых подсистем интеллектуальных компьютерных систем нового поколения. Данная работа является формальной спецификацией Реализации информационно-поисковой подсистемы в текущем Программном варианте реализации ostis-платформы, а также её программного интерфейса, и является продолжением серии работ по проектированию и реализации базового Программного варианта реализации ostis-платформы [1], [2]

Received 13.03.2023