Control Tools for Reusable Components of Intelligent Computer Systems of a New Generation

Maksim Orlov
Belarusian State University of Informatics and Radioelectronics
Minsk, Belarus
Email: orlovmaksimkonst@gmail.com

***Abstract*—In the article, an approach to the design of intelligent systems is considered, focused on the use of compatible reusable components, which significantly reduces the complexity of developing such systems. The key means of supporting the component design of intelligent computer systems is the manager of reusable components proposed in the work.**

***Keywords*—Component design of intelligent computer systems; reusable semantically compatible components; knowledge-driven systems; semantic networks.**

I. Introduction

The main result of artificial intelligence is not the intelligent systems themselves but powerful and effective technologies for their development. The analysis of modern technologies for designing intelligent computer systems shows that along with very impressive achievements, the following serious problems occur [1], [2], [3]:

- high requirements for the initial qualifications of users and developers. Artificial intelligence technologies are not focused on the wide range of developers and users of intelligent systems and, therefore, have not received mass distribution;
- there is no general unified solution to the problem of semantic compatibility of computer systems [4]. There are no approaches that allow integrating scientific and practical results in the field of artificial intelligence, which generates a high degree of duplication of results and a lot of non-unified formats for representation of data, models, methods, tools, and platforms;
- lack of powerful tools for designing intelligent computer systems, including intelligent training subsystems, subsystems for collective design of computer systems and their components, subsystems for verification and analysis of computer systems, subsystems for component design of computer systems;
- long terms of development of intelligent computer systems and high level of complexity of their maintenance and extension;
- the degree of dependence of artificial intelligence technologies on the platforms on which they are implemented is high, which leads to significant changes in technologies when transitioning to new platforms;
- the degree of dependence of artificial intelligence technologies on subject domains in which these technologies are used is high;
- there is a high degree of dependence of intelligent computer systems and their components on each other; the lack of their automatic synchronization. The absence of self-sufficiency of systems and components, their ability to operate separately from each other without loss of expediency of their use;
- increase in the time to solve the problem with the expansion of the functionality of the problem solver and with the expansion of the knowledge base of the system [5];
- lack of methods for designing intelligent computer systems. Updating computer systems often boils down to the development of various kinds of "patches", which eliminate not causes of the identified disadvantages of updated computer systems but only some of the consequences of these causes;
- poor adaptability of modern computers to the effective implementation of even existing knowledge representation models and models for solving problems that are difficult to formalize, which requires the development of fundamentally new computers [6];
- there is no single approach to the allocation of reusable components and the formation of libraries of such components, which leads to a high complexity of reuse and integration of previously developed components in new computer systems.

To solve these problems, it is necessary to implement a comprehensive technology for designing intelligent computer systems, which includes the following components:

- a model of an intelligent computer system [7];

- a *library of reusable components* and corresponding *tools to support component design of intelligent computer systems*;
- an intelligent integrated automation system for the collective design of intelligent computer systems, including subsystems for editing, debugging, performance evaluation, and visualization of developed components, as well as a simulation subsystem;
- methods of designing intelligent computer systems;
- an intelligent user interface;
- training subsystems for designing intelligent computer systems, including a subsystem for conducting a dialog with the developer and the user;
- a subsystem for testing and verification of intelligent computer systems, including a subsystem for testing the compatibility of the developed system with other systems;
- an information security support subsystem for the intelligent computer system.

The key component of the technology for intelligent systems design is a *library of reusable components* and the corresponding *tools for supporting component design of intelligent computer systems*. With its help, it is possible to effectively implement the typical subsystems to support the design of intelligent computer systems.

Most of the existing systems are created as self-contained software products that cannot be used as components of other systems. It is necessary to use either the whole system or nothing. A small number of systems support a component-oriented architecture capable of integrating with other systems [8], [9]. However, their integration is possible if the same technologies are used and only when designed by one development team [10].

Repeated re-development of existing technical solutions is conditioned either by the fact that known technical solutions are hardly integrated into the system being developed or by the fact that these technical solutions are difficult to be found. This problem is relevant both in general in the field of computer systems development and in the field of knowledge-based systems development, since in systems of this kind the degree of consistency of various knowledge types affects the ability of the system to solve non-trivial problems.

The development technology should allow components to be reused, integrated with other components built using both this and other technologies. It should also be open to allow using components by different development teams.

Reuse of ready-made components is widely used in many fields related to the design of various kinds of systems, since it reduces the complexity of development and its cost (by minimizing the amount of labor due to the absence of the need to develop any component), improves the quality of the systems being created, and reduces professional requirements for computer system developers [11]. Thus, there is a transition from programming components or entire systems to their design (assembly) based on ready-made components. *Component design of intelligent computer systems* involves the selection of existing components capable of solving the problem in its entirety or the decomposition of the problem into subproblems with the allocation of components for each of them (see [12]). The designed systems according to the proposed technology have a high level of flexibility, their development is carried out in stages, moving from one complete version of the system to another. At the same time, the starting version of the system can be the core of the corresponding class of systems included in the *library of reusable components*. The technology of component design of intelligent computer systems includes a set of coordinated particular technologies that ensure the comprehensive design of computer systems. It includes the technology of component design of knowledge bases, problem solvers, interfaces, and others.

The main element of the semantic technology for component design of intelligent systems is the *library of compatible reusable components*. This allows designing intelligent systems by combining existing components, selecting the right ones from the appropriate libraries. The use of ready-made components assumes that the distributed component is verified and documented, and possible errors and limitations are eliminated or specified and known. The creation of the *library of reusable components* does not mean the re-creation of all existing modern information technology products. The technology of component design of intelligent computer systems involves the use of vast experience in the development of modern computer systems, however, it is required to create a specification of each component (both newly created and existing integrated ones) to ensure its installation and compatibility with other components and systems. Nevertheless, an effective component design technology will appear only when a "critical mass" of application system developers participating in the seeding of *libraries of reusable components* of the designed systems is formed.

The problems of implementing the component approach to the design of intelligent computer systems inherit the problems of modern technologies for designing intelligent systems and also include the following ones [13]:

- incompatibility of components developed within different projects due to the lack of unification in the principles of representing different types of knowledge within the same *knowledge base*, and, as a consequence, the lack of unification in the principles of allocation and ***specification of reusable components***;
- inability to automatically integrate components into the system without manual user intervention;
- automatic updating of components leads to inconsistency of both particular modules of computer systems and the systems themselves with each other;

- lack of classification of components at different levels of detail;
- testing, verification, and analysis of the components quality are not carried out; advantages, disadvantages, limitations of components are not identified;
- development of standards that ensure the compatibility of these components is not being carried out;
- many components use the language of the developer for identification (usually English), and it is assumed that all users will use the same language. However, for many applications, this is unacceptable – identifiers that are understandable only to the developer should be hidden from end users, who should be able to choose the language for the identifiers they see;
- lack of tools to search for components that meet the specified criteria.

Component design of intelligent computer systems is possible only if the selection of components is carried out on the basis of a thorough analysis of the quality of these components. One of the most important criteria for such an analysis is the level of semantic compatibility of the analyzed components with all the components available in the current version of the library.

In addition to a powerful library of reusable semantically compatible components, an appropriate tool is needed for managing (installing into child systems, searching, updating, forming) such components. Such a tool should be built according to the same principles as intelligent computer systems of a new generation to ensure semantic compatibility of intelligent systems, their components, and their design tools.

The purpose of the work is to create a tool to support the component design of intelligent computer systems of a new generation. Such a tool is necessary to use the full potential of the infinitely extensible comprehensive *libraries of reusable components*. The fields where the technology of component design of semantically compatible intelligent systems is applied in practice have no limits.

II. Analysis of existing approaches to solving the problem

At the moment, there is no comprehensive library of reusable semantically compatible components of computer systems in general, aside from intelligent ones. There are some attempts to create libraries of typical methods and programs for traditional computer systems, but such libraries do not solve the above problems.

The term “library of subprograms” was one of the first mentioned by M. Wilkes, D. Wheeler, and S. Gill as one of the forms for organizing calculations on a computer (considered in [14]). Based on what is stated in their book, a library is understood as a set of “short, pre-prepared programs for certain, frequently occurring (standard) computing operations”. It is worth noting that the components of libraries are not only programs but also components of interfaces and knowledge bases.

Traditional solutions include package managers of programming languages and operating systems, as well as separate systems and platforms with built-in components and tools for saving created components.

Library components can be implemented in different programming languages (which leads to the fact that for each programming language, its own libraries are developed with their own solutions to various frequently occurring situations) and can also be located in different places, which leads to the fact that a tool is needed in the library to search for components and install them.

Modern package managers such as *npm*, *pip*, *apt*, *maven*, *poetry*, and others have the advantage of being able to resolve conflicts when installing dependent components, but they do not take into account the semantics of components and only install components by ID [15]. Libraries of such components are only a storage of components, which does not take into account the purpose of components, their advantages and disadvantages, scope of application, hierarchy of components, and other information necessary for the intellectualization of component design of computer systems. Searching for components in *libraries of components* corresponding to these package managers is reduced to searching by component ID. Modern package managers are only “installers” without automatic integration of components into the system. Similarly, a significant disadvantage of the modern approach is the platform dependency of components. Modern component libraries are focused only on a specific programming language, operating system, or platform.

The ***pip*** package manager is a package management system that is used to install packages from the Python Package Index, which is some library of such packages. Pip is often installed with Python. The pip package manager is used only for the Python programming language. It has many functions for working with packages:

- installation of a package;
- installation of a package of a specialized version;
- deletion of a package;
- reinstallation of a package;
- display of installed packages;
- search for packages;
- verification of package dependencies;
- creation of a configuration file with a list of installed packages and their versions;
- installation of a set of packages from a configuration file.

The pip package manager works well with dependencies, displays unsuccessfully installed packages, and also displays information about the required package version

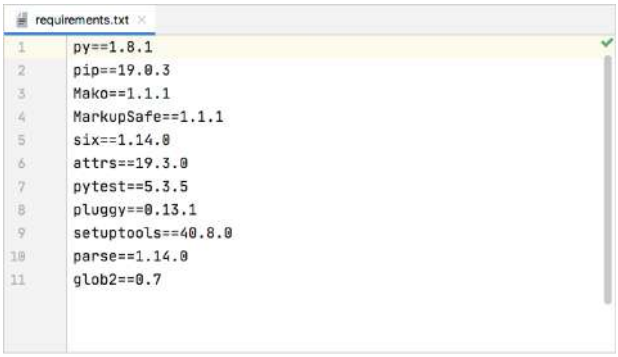

Figure 1. pip configuration file

in case of conflict with another package. An example of a pip package configuration file is shown in Figure 1.

An alternative to the pip package manager is the ***poetry*** package manager, which is also focused on the Python programming language. The advantage of poetry over pip is that it automatically works with virtual environments, is able to find and create them independently. The configuration file for poetry packages is more comprehensive than that of pip, it stores such information as the project name, project version, its description, license, list of authors, URL of the project, its documentation, and website, a list of project keywords, and a list of PyPI classifiers. Poetry allows configuring packages for Python projects more flexibly, the poetry configuration file is a more extensive project specification (see Figure 2), however, this specification does not allow for compatibility between components even within Python projects and is intended primarily for read-only by the developer.

```
[tool.poetry]
name = "first"
version = "0.1.0"
description = ""
authors = ["…"]
readme = "README.md"

[tool.poetry.dependencies]
python = "^3.10"

[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"
```

Figure 2. poetry configuration file

It is impossible to automate the design of computer systems using the poetry or pip package manager, since it requires the intervention of a developer who needs to manually combine the interfaces of the installed packages. Other package managers of programming languages and operating systems are arranged according to the same principle: there is a component storage (library), which is a set of packages of this programming language or operating system and with which the component manager interacts.

As a component approach to program design, it is possible to consider libraries of subprograms of modern programming languages, for example, ***STL Library*** (a library of standard C++ templates).

The ***STL Library*** is a set of consistent generalized algorithms, containers, means of accessing their contents, and various auxiliary functions in C++.

There are five main components of the STL Library:

- container – storage of a set of objects in memory;
- iterator – provision of access means to the contents of the container;
- algorithm – determination of the computational procedure;
- adapter – adaptation of components for providing different interface;
- functional object – privacy of a function in an object for use by other components.

The structure of the *STL Library* is shown in Figure 3.

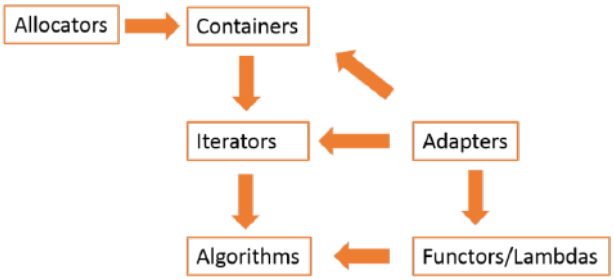

Figure 3. Structure of the STL Library

Separation allows reducing the number of components. For example, instead of writing a separate element search function for each container type, a single version is provided that works with each of them as long as the basic requirements are met.

The compatibility of components (containers) in the *STL Standard Template Library* is provided by a common interface for using these components.

The component approach to the design of computer systems can be implemented within various languages, platforms, and applications. Let us consider some of them.

The ontology implemented in ***OWL*** (Web Ontology Language) is a set of declarative statements about the entities of the dictionary of a subject domain (discussed in more detail in [16]). OWL assumes the concept of an “open world”, according to which the applicability of subject domain descriptions placed in a specific physical document is not limited only to the scope of this document – the contents of the ontology can be used and supplemented by other documents adding new facts about the same entities or describing another subject domain in

terms of this one. The "openness of the world" is achieved by adding a URI to each element of the ontology, which makes it possible to understand the ontology described in OWL as part of a universal unified knowledge.

WebProtege is a multi-user web interface that allows editing and storing ontologies in the OWL format in a collaborative environment [17]. This project allows not only creating new ontologies but also loading existing ontologies that are stored on the Stanford University server. The advantage of this project is the automatic error checking in the process of creating ontology objects. This project is an example of an attempt to solve the problem of accumulation, systematization, and reuse of existing solutions, however, the disadvantage of this solution is the isolation of the ontologies being developed. Each developed component has its own hierarchy of concepts, an approach to the allocation of classes and entities that depend on the developers of these ontologies, since within this approach, there is no universal model of knowledge representation, as well as formal specification of components represented in the form of ontologies. Consequently, there is a problem of their semantic incompatibility, which, in turn, leads to the impossibility of reuse of the developed ontologies in the knowledge bases design. This fact is confirmed by the presence on the Stanford University server of a variety of different ontologies on the same topics.

Based on the ***Modelica*** language, a large number of freely available component libraries have been developed, one of which is the Modelica_StateGraph2 library, which includes components for modeling discrete events, reactive, and hybrid systems using hierarchical state diagrams [18]. The main disadvantage of Modelica-based systems is the lack of component compatibility and sufficient documentation.

Microsoft Power Apps is a set of applications, services, and connectors, as well as a data platform that provides a development environment for efficiently creating user applications for business. The Power Apps platform provides tools for creating a library of reusable graphical interface components, as well as pre-created text recognition models (reading visiting cards or cheques) and an object detection tool that can be connected to the application being developed [19]. The Power Apps component library is a set of user-created components that can be used in any application. The advantage of the library is that components can configure default properties that can be flexibly edited in any applications that use the components. The disadvantage lies in the lack of semantic compatibility of components, the specification of components; the problem of the presence of semantically equivalent components has not been solved; there is no hierarchy of components and means of searching for these components.

The ***IACPaaS platform*** is designed to support the development, management, and remote use of applied and instrumental multi-agent cloud services (primarily intelligent ones) and their components for various subject domains [20]. The platform provides access to:

- application users (specialists in various subject domains) – to applied services;
- developers of applied and instrumental services and their components – to instrumental services;
- intelligent services managers and management services.

The IACPaaS platform supports:

- the basic technology for the development of applied and specialized instrumental (intelligent) services using the basic instrumental services of the platform that support this technology;
- a variety of specialized technologies for the development of applied and specialized instrumental (intelligent) services, using specialized platform tool services that support these technologies.

The IACPaaS platform also does not contain means for a unified representation of the components of intelligent computer systems and means for their specification and automatic integration.

Based on the analysis carried out, it can be said that at the current state of development of information technologies, there is no comprehensive library of reusable semantically compatible components of computer systems and corresponding component management tools. Thus, it is proposed to implement a library and an appropriate component management tool that will implement seamless integration of components, ensure semantic compatibility of systems and their components, and significantly simplify the design of new systems and their components.

III. Proposed approach

Within this article, it is proposed to take an *OSTIS Technology* [21] as a basis, the principles of which make it possible to implement a semantic technology for designing intelligent systems, including a library of reusable components, component design support tools, and other components of the technology. The *OSTIS Technology* makes it possible to quickly create knowledge-driven systems using ready-made compatible components.

The systems developed on the basis of the OSTIS Technology are called *ostis-systems*. The *OSTIS Technology* is based on a universal method of semantic representation (encoding) of information in the memory of intelligent computer systems, called an *SC-code*. Texts of the *SC-code* (sc-texts) are unified semantic networks with a basic set-theoretic interpretation, which allows solving the problem of compatibility of various knowledge types. The elements of such semantic networks are called *sc-elements* (*sc-nodes* and *sc-connectors*, which, in turn, depending on orientation, can be *sc-arcs* or *sc-edges*). The *Alphabet of the SC-code* consists of five main elements, on the basis

of which SC-code constructions of any complexity are built, including more specific types of sc-elements (for example, new concepts). The memory that stores SC-code constructions is called semantic memory, or *sc-memory*.

Within the technology, several universal variants of visualization of *SC-code* constructions are proposed, such as *SCg-code* (graphic variant), *SCn-code* (nonlinear hypertext variant), *SCs-code* (linear string variant).

Within this article, fragments of structured texts in the SCn code [22] will often be used, which are simultaneously fragments of the source texts of the knowledge base, understandable to both human and machine. This allows making the text more structured and formalized, while maintaining its readability. The symbol "**:=**" in such texts indicates alternative (synonymous) names of the described entity, revealing in more detail certain of its features.

The basis of the knowledge base within the *OSTIS Technology* is a hierarchical system of subject domains and ontologies.

In order to solve the problems that have arisen in the design of intelligent systems and libraries of their reusable components, it is necessary to adhere to the general principles of the technology for intelligent computer systems design, as well as meet the following requirements:

- ensuring compatibility (integrability) of components of intelligent computer systems based on the unifying representation of these components;
- clear separation of the process of developing formal descriptions of intelligent computer systems and the process of their implementation according to this description;
- clear separation of the development of a formal description for the designed intelligent system from the development of various options for the interpretation of such formal descriptions of the systems;
- availability of an ontology for component design of intelligent computer systems, including (1) a description of component design methods, (2) a model of a *library of reusable components*, (3) a model of a *specification of reusable components*, (4) a complete *classification of reusable components*, (5) a description of means for interaction of the developed intelligent computer system with *libraries of reusable components*;
- availability of *libraries of reusable components of intelligent computer systems*, including component specifications;
- availability of means for interaction of the developed intelligent computer system with libraries of reusable components for installation of any types of components and their management in the created system. The installation of a component means not only its transportation to the system (copying sc-elements and/or downloading component files) but also the subsequent execution of auxiliary actions so that the component can operate in the system being created.

Based on this, in order to solve the problems set within this article, it is proposed to develop the following system of subject domains and corresponding ontologies:

Subject domain of reusable ostis-systems components
⇒ *private subject domain**:
Subject domain of a library of reusable ostis-systems components

Subject domain of the manager of reusable ostis-systems components

The *Subject domain of reusable ostis-systems components* describes the concept of a reusable component, the classification of components, and their general specification. This subject domain allows creating new and specifying existing components to add them to the library.

As a *reusable ostis-systems component*, a component of some ostis-system that can be used within another ostis-system is understood (see [13]). This is a component of the ostis-system that can be used in other ostis-systems (***child ostis-systems***) and contains all those and only those sc-elements that are necessary for the functioning of the component in the child ostis-system. In other words, it is a component of some ***maternal ostis-system***, which can be used in some child ostis-system. To include a reusable component in some system, it must be installed in this system, that is, all the sc-elements of the component should be copied into it and, if necessary, auxiliary files, such as the source or compiled component files. *Reusable components* must have a unified specification and hierarchy to support compatibility with other components. The compatibility of *reusable components* leads the system to a new quality, to an additional extension of the set of problems to be solved when integrating components.

reusable ostis-systems component
:= [typical ostis-systems component]
:= [reused ostis-systems component]
:= [reusable OSTIS component]
:= [ostis-systems ip-component]
:= *frequently used sc-identifier**:
[reusable component]
⊂ *ostis-system component*
⊂ *sc-structure*

The requirements for *reusable ostis-systems components* inherit the common requirements for the design of software components and also include the following ones [23]:

- there is a technical possibility to embed a reusable component into a child ostis-system;

- a reusable component should perform its functions in the most general way, so that the range of possible systems in which it can be embedded is the widest;
- compatibility of a reusable component: the component should strive to increase the level of negotiability of ostis-systems in which it is embedded and be able to be automatically integrated into other systems;
- self-sufficiency of components, that is, their ability to operate separately from other components without losing the appropriateness of their use.

In the *Subject domain of the library of reusable ostis-systems components*, the most common concepts and principles are described, which are valid for any *library of reusable components*. This subject domain allows building many different libraries, each of which will be semantically compatible with any other built according to the proposed principles. Such libraries store components and their specifications for use in child ostis-systems. An example of a specification of a reusable ostis-systems component is shown in Figure 4.

library of reusable ostis-systems components
⇒ *frequently used sc-identifier**:
[library of ostis-systems components]
⇒ *frequently used sc-identifier**:
[library of components]
≔ [library of compatible reusable components]
≔ [comprehensive library of reusable semantically compatible ostis-systems components]
≔ [library of reusable and compatible components of intelligent computer systems of a new generation]
≔ [library of typical ostis-systems components]
≔ [library of reusable OSTIS components]
≔ [library of reused OSTIS components]
≔ [library of intelligent property ostis-systems components]
≔ [library of ostis-systems ip-components]
∋ *typical example′*:
OSTIS Metasystem Library
≔ [Distributed library of typical (reusable) ostis-systems components as part of the *OSTIS Metasystem*]
≔ [Library of reusable ostis-systems components as part of the *OSTIS Metasystem*]
∋ *typical example′*:
OSTIS Metasystem Library
⇒ *frequently used sc-identifier**:
[OSTIS Library]
≔ [Library of reusable and compatible components of intelligent computer systems of a new generation]
≔ [Library of typical components of intelligent computer systems of a new generation]
≔ [Distributed library of typical (reusable) ostis-systems components as part of the OSTIS Ecosystem]
≔ [Library of reusable ostis-systems components as part of the *OSTIS Ecosystem*]
⇐ *combination**:
{• *library of reusable components of ostis-systems knowledge bases*
• *library of reusable components of ostis-systems problem solvers*
• *library of reusable components of ostis-systems interfaces*
• *library of embedded ostis-systems*
• *library of ostis-platforms*
}

First versions for the full contents of the *Subject domain of reusable ostis-systems components* and the *Subject domain of the library of reusable ostis-systems components* are represented in the work [24].

The ***manager of reusable ostis-systems components*** is the main means of supporting component design of intelligent computer systems built by the *OSTIS Technology*. It allows installing reusable components in ostis-systems and controlling them. The *Subject domain of the manager of reusable ostis-systems components* contains the full specification for the manager of ostis-systems components, the requirements for the component manager, its functionality, the specification of the implementation option for the manager of ostis-systems components, including the sc-model of the knowledge base, the problem solver, and the interface.

Before considering the model of the *manager of reusable ostis-systems components*, let us consider the general model of any *library of reusable ostis-systems components*, with which the component manager interacts, and the most important classes of reusable components. Next, we will consider in more detail the fragments for sc-models of the *Subject domain of the manager of reusable ostis-systems components*.

IV. Library of reusable ostis-systems components

The basis for the implementation of the component approach within the *OSTIS Technology* is the ***OSTIS Metasystem Library***. The *OSTIS Metasystem* is focused on the development and practical implementation of methods and tools for component design of semantically compatible intelligent computer systems, which provides an opportunity to quickly create intelligent systems for various purposes. The *OSTIS Metasystem* includes the *OSTIS Metasystem Library*. The scope of practical application for the technology of component design of semantically compatible intelligent systems does not have any limits.

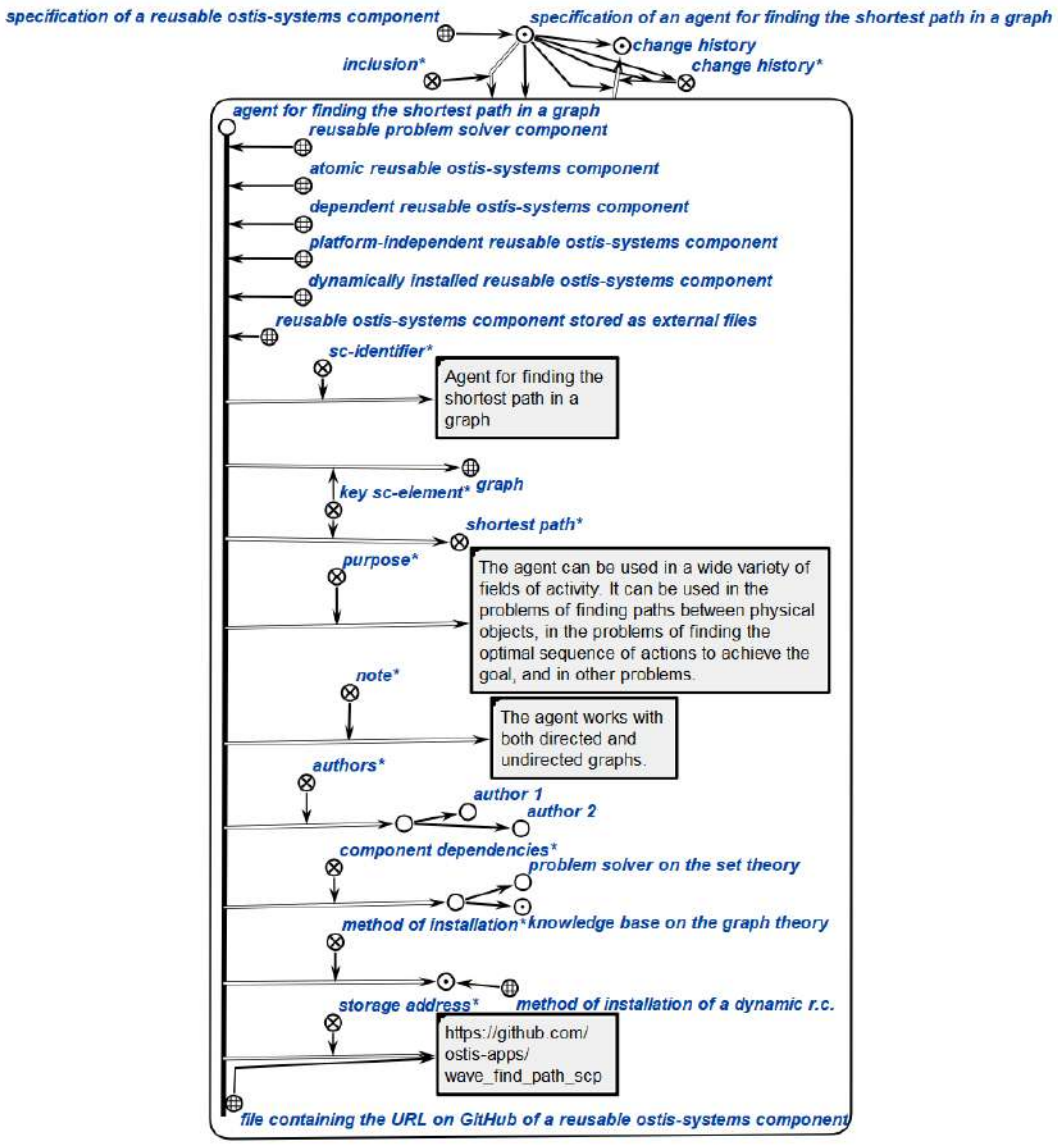

Figure 4. An example of a specification of a reusable ostis-systems component

The *OSTIS Metasystem* acts as the ***maternal system*** for all ostis-systems being developed, since it contains all the basic components for their development (Figure 5).

Functionality of any *library of reusable ostis-systems components* (see [24]):

- storage of reusable ostis-systems components and their specifications. At the same time, some of the components specified within the library may be physically stored elsewhere due to the peculiarities of their technical implementation (for example, the source texts of the ostis-platform may be physically stored in some separate storage, but they will be specified as a component of the corresponding library). In this case, the specification of the component within the library should also include a description of (1) the where the component is located and (2) the scenario of its automatic installation in a child ostis-system;
- viewing available components and their specifications, as well as searching for components by fragments of their specification;
- storage of information about components use statistics. For example, in which *child ostis-systems* which of the library components and which version are used (are downloaded). This is necessary to take into account the demand for a particular component, to assess its importance and popularity;
- systematization of *reusable ostis-systems components*;
- provision of versioning of *reusable ostis-systems*

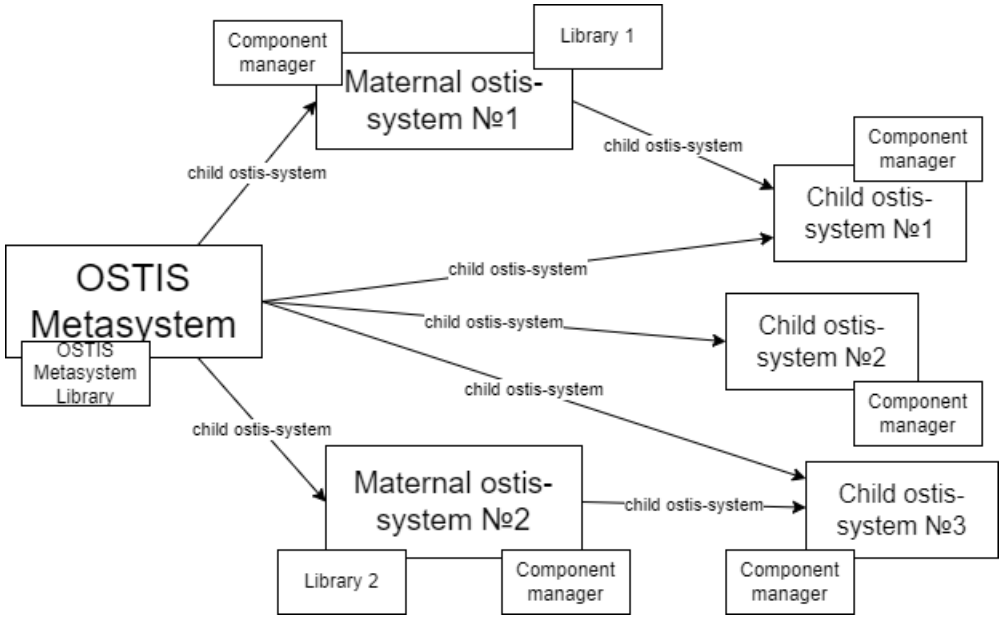

Figure 5. Architecture of the OSTIS Ecosystem in terms of libraries of reusable components

components;

- search for dependencies between reusable components within the library of components;
- provision of automatic updating of components borrowed into child ostis-systems. This function can be turned on and off at the request of the developers of the child ostis-system. Simultaneous updating of the same components in all systems using it should not in any context lead to inconsistency between these systems. This requirement may turn out to be quite complicated, but without it the work of the *OSTIS Ecosystem* is impossible.

The ***library of reusable ostis-systems components*** is an embedded ostis-system. It has its own knowledge base, its own problem solver, and its own interface. However, not every ostis-system is required to have a library of components.

Let us consider the most important classes of reusable ostis-systems components from the point of view of the *manager of reusable components*.

reusable ostis-systems component
⇒ *subdividing**:
{• *atomic reusable ostis-systems component*
• *non-atomic reusable ostis-systems component*
}

The typology of ostis-systems components by atomicity. An atomic reusable ostis-systems component is a component that in the current state of the ostis-systems library is considered as indivisible, that is, does not contain other components in its structure. A non-atomic reusable component in the current state of the ostis-systems library contains other atomic or non-atomic components in its structure and does not depend on its own parts. Without any part of the non-atomic component, its purpose restricts. The *manager of reusable ostis-systems components* allows installing both a whole non-atomic component and a selected subset of its constituent components. At the same time, there can be no inconsistency of such a combined component due to semantic compatibility of *reusable ostis-systems components*.

reusable ostis-systems component
⇒ *subdividing**:
{• *dependent reusable ostis-systems component*
• *independent reusable ostis-systems component*
}

The typology of ostis-systems components by dependency. A dependent reusable ostis-systems component depends on at least one other component of the ostis-systems library, i.e. it cannot be embedded in a child ostis-system without the components on which it depends. The independent component does not depend on any other component of the ostis-systems library. When installing dependent components, the component manager installs all its dependencies, otherwise the component cannot operate in a child ostis-system. If any component, which is a dependency of another, is not installed and it is not possible to install any equivalent component, then the

installation of the dependent component in the current state of the ostis-system and the component libraries used by it is impossible.

reusable ostis-systems component
⇒ *subdividing**:
{• *reusable ostis-systems component stored as external files*
• *reusable ostis-systems component stored as an sc-structure*
}

reusable ostis-systems component stored as external files
⇒ *subdividing**:
{• *reusable ostis-systems component stored as source files*
• *reusable ostis-systems component stored as compiled files*
}

The typology of ostis-systems components by their storage method. A *reusable component stored as an sc-structure* is integrated into child systems in the simplest and most convenient way. The installation of such components takes place by copying the sc-elements of the structure from one ostis-system to another. When storing reusable components as external files, not all components can be dynamically installed. At this stage of development of the *OSTIS Technology*, it is more convenient to store components in the form of source texts.

reusable ostis-systems component
⇒ *subdividing**:
{• *platform-dependent reusable ostis-systems component*
• *platform-independent reusable ostis-systems component*
}

The typology of ostis-systems components depending on the ostis-platform. A platform-dependent reusable ostis-systems component is a component partially or fully implemented with the help of any third-party means from the point of view of the *OSTIS Technology*. The disadvantage of such components is that the integration of such components into intelligent systems may be accompanied by additional difficulties depending on the specific means of implementing the component. As a potential advantage of platform-dependent reusable ostis-systems components, it is possible to allocate their, as a rule, higher performance due to their implementation at a level closer to the platform one. In general, a platform-dependent reusable ostis-systems component can be supplied either as a set of source codes or compiled. The process of integrating a platform-dependent reusable ostis-systems component into a child system developed using the OSTIS Technology strongly depends on the implementation technologies of this component and in each case may consist of various stages. Each platform-dependent reusable ostis-systems component must have the appropriate detailed, correct, and understandable instructions for its installation and implementation in the child system using the component manager. A platform-independent reusable ostis-systems component is a component that is entirely represented in the *SC-code*. The process of integrating a platform-dependent reusable ostis-systems component into a child system developed using the OSTIS Technology is significantly simplified by using a common unified formal basis for knowledge representation and processing.

The most valuable are platform-independent reusable ostis-systems components.

reusable ostis-systems component
⇒ *subdividing**:
{• *dynamically installed reusable ostis-systems component*
:= [reusable component, the installation of which does not require a restart of the system]
• *reusable component, the installation of which requires a restart of the system*
}

dynamically installed reusable ostis-systems component
⇒ *decomposition**:
{• *reusable component stored as compiled files*
• *reusable knowledge base component*
}

The typology of ostis-systems components according to the dynamics of their installation. The process of integrating components of different types at different stages of the ostis-systems life cycle can be different. The most valuable components are those that can be integrated into a working system without stopping its functioning. Some systems, especially control ones, cannot be stopped, but components need to be installed and updated.

An ***embedded ostis-system*** is a non-atomic reusable component that consists of a knowledge base, a problem solver, and an interface.

embedded ostis-system
⊂ *ostis-system*
⊂ *non-atomic reusable ostis-systems component*
⇒ *decomposition**:
{

- *reusable component of ostis-systems knowledge bases*
- *reusable component of ostis-systems problem solvers*
- *reusable component of ostis-systems interfaces*

}

As such systems, for example, an intelligent interface (including a natural language interface), an environment for collective design of knowledge bases, a manager of reusable ostis-systems components, a training system, a system for testing and verification of intelligent systems, a visual web-oriented editor of sc.g-texts, and others can act.

The peculiarity of *embedded ostis-systems* is that the integration of entire intelligent systems involves the integration of the knowledge bases of these systems, the integration of their problem solvers, and the integration of their intelligent interfaces. When integrating embedded ostis-systems, the knowledge base of the embedded system becomes part of the knowledge base of the system into which it is embedded. The problem solver of the embedded ostis-system becomes part of the problem solver of the system into which it is embedded. And the interface of the embedded ostis-system becomes part of the interface of the system into which it is embedded. From the point of view of the component manager, this is equivalent to installing a non-atomic reusable ostis-systems component, that is, a separate installation of all its components. At the same time, the embedded system is integral and can function separately from other ostis-systems, unlike other reusable components.

Embedded ostis-systems are often subject-independent reusable components. Thus, for example, an embedded ostis-system in the form of a knowledge base design environment can be integrated both into a system from the subject domain of geometry and into an agricultural facilities management system.

The *embedded ostis-system*, like any other reusable ostis-systems component, should support semantic compatibility of ostis-systems. Both the embedded ostis-system itself and all its components must be specified and coordinated. Components of embedded ostis-systems can be replaced with others having the same purpose, for example, a natural language interface can have different versions of the knowledge base depending on the natural language supported by the system, different interface options, depending on the requirements and convenience of users, and also various options for implementing a problem solver for natural language processing, which can use different models but solve the same problem. The *manager of reusable components* allows flexibly selecting certain components of *embedded ostis-systems*, while maintaining their integrity and overall functionality. The embedded ostis-system connects with the system in which it is embedded using the ***embedded ostis-system**** relation.

V. Manager of reusable ostis-systems components

The *manager of reusable ostis-systems components* is a subsystem of the ostis-system, through which interaction with the *library of reusable ostis-systems components* takes place.

manager of reusable ostis-systems components
⊂ *embedded ostis-system*
⊂ *platform-dependent reusable ostis-systems component*
:= *frequently used sc-identifier**:
[manager of reusable components]
:= *frequently used sc-identifier**:
[component manager]
⇒ *generalized decomposition**:
{• *knowledge base of the manager of reusable ostis-systems components*
• *problem solver of the manager of reusable ostis-systems components*
• *interface of the manager of reusable ostis-systems components*
}
∋ *Implementation of the manager of reusable ostis-systems components*
⇒ *component address**:
[https://github.com/ostis-ai/sc-component-manager]

The ***knowledge base of the component manager*** contains all the knowledge that is necessary to install reusable components in the *child ostis-system*. Such knowledge includes knowledge about the specification of reusable components, methods of installing components, knowledge about the libraries of ostis-systems with which interaction occurs, classification of components, and others.

The ***problem solver of the manager of reusable ostis-systems components*** interacts with the *library of reusable ostis-systems components* and allows installing and integrating reusable components into a child ostis-system, as well as searching, updating, publishing, deleting components, and other operations with them. At a minimum, the component manager should provide the following functionality:

- **Search for reusable ostis-systems components.** The set of possible search criteria corresponds to the *specification of reusable components*. As such criteria, the component classes, its authors, identifiers, a fragment of a note, purpose, belonging to a subject domain, the type of knowledge of the component, and others can serve.

- **Installation of a reusable ostis-systems component.** The installation of a reusable component takes place regardless of the type, installation method, and location of the component. A necessary condition for the possibility of installing a reusable component is the availability of the ***specification of a reusable ostis-systems component***. Before installing a reusable component into a child system, all dependent components must be installed. Also, for platform-dependent components, it may be necessary to perform additional steps for component installation, depending on the specific implementation of the component. After the component has been successfully installed, an information construction is generated in the knowledge base of the child system, indicating the fact that the component has been installed into the system using the *installed components** relation.
- **Addition and removal of library components controlled by the manager.** The component manager contains information about a variety of sources for installing components, the list of which can be supplemented. By default, the component manager monitors the *OSTIS Metasystem Library*, however, it is possible to create and add optional ostis-systems libraries.

Based on the specified minimum functionality, the *problem solver of the manager of reusable ostis-systems components* represents the following hierarchy of abstract sc-agents:

problem solver of the manager of reusable ostis-systems components
⇒ *decomposition of an abstract sc-agent**:
{•
Abstract sc-agent for searching for reusable ostis-systems components
•
Abstract sc-agent for installing reusable ostis-systems components
•
Abstract sc-agent for managing library components controlled by the manager
}

Abstract sc-agent for managing library components controlled by the manager
⇒ *decomposition of an abstract sc-agent**:
{•
Abstract sc-agent for adding a library controlled by the component manager
•
Abstract sc-agent for deleting a library controlled by the component manager
}

Using minimal functionality, the component manager can install components that will extend its functionality. Components that implement the extended functionality of the component manager are part of the *OSTIS Metasystem Library*. The extended functionality includes:

- **Specification** of a reusable ostis-systems component. The component manager allows specifying the components that are part of the ostis-systems library, as well as specifying new components that are being created, which will be published to the ostis-systems library. In this case, the specification can occur both automatically and manually. For example, the component manager can update the specification of the component used in accordance with which new ostis-systems have installed it, update the specification of the authorship of the component when editing it in the ostis-systems library or the specification of errors detected during the operation of the component, etc.
- **Creation** of a reusable ostis-systems component according to a template with specified parameters. When installing a template for a reusable ostis-systems component, the component manager allows creating a specific component based on it. To do this, the user is asked to determine the values of all sc-variables in the template to form a specific component from a certain subject domain. For example, to form a reusable component of knowledge bases, which is a semantic neighborhood of some relation (see Figure 6), it is necessary to determine the values of all variables, except for the variable that is the key sc-element of this structure.
- **Publication** of a reusable ostis-systems component to the ostis-systems library. When a component is published to the ostis-systems library, verification takes place based on the component specification. The publication of a component can be accompanied by the assembly of a non-atomic component from existing atomic ones. It is also possible to update the version of the published component by the team of its developers.
- **Update** of an installed reusable ostis-systems component.
- **Deletion** of an installed reusable component. As in the case of installation, after deleting a reusable component from the ostis-system, the fact of deleting the component is established in the knowledge base of the system. This information is an important part of the operational history of the ostis-system.
- **Edition** of a reusable component in the ostis-systems library.
- **Comparison** of reusable ostis-systems components.

The ***interface of the manager of reusable ostis-systems components*** provides convenient use of the component manager for the user and other systems. The minimal interface of the component manager is console-based and allows accessing the functionality of the component manager using commands. The minimal interface is available for both users and other ostis-systems. The

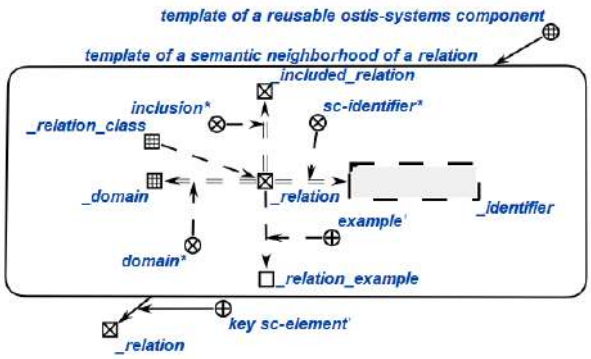

Figure 6. Example of a template for a semantic neighborhood of a relation

extended interface is graphical and is part of the *OSTIS Metasystem Library*. It can be installed using the reusable component manager. Thus, the component manager allows developing not only the systems in which it is embedded but also itself.

In order to create a new ostis-system "from scratch" using the *ostis-platform*, it is necessary to install some *Software implementation option of the ostis-platform* using third-party tools. As such tools, (1) platform source code storages, for example, cloud storages, such as a GitHub repository, with an appropriate set of platform installation instructions, or (2) installation tools for a pre-compiled software implementation of the platform, for example, an apt software installation tool, can serve. Next, the installation of reusable components in the ostis-system (regardless of the type of components) is carried out using the component manager. When installing platform-dependent components, the component manager must manage the appropriate means of assembling such components (CMake, Ninja, npm, grunt, and others).

In order to store reusable ostis-systems components, some storage is needed. In addition to the external files of the component, its specification must be located in the storage. As such a storage, (1) a *library of reusable ostis-systems components* or (2) files in some cloud storage can act. In the case when a component is stored in a library, to install it, the component manager copies all the sc-elements, which represent the component, to a child ostis-system. In the case when the component is stored as files in the cloud storage, the component manager downloads the component files and installs them according to the specification. The addresses of component specification storages should be stored in the knowledge base of the component manager in order to have access to component specifications for their subsequent use (search, installation, etc.). In Figure 7, there is an example for a fragment of the knowledge base of the component manager, which describes where the specifications of the components available for installation are stored. Such a storage is a set consisting of two sets: (1) a set of addresses of component specifications and (2) a set of addresses of specifications for other storages. Thus, a tree-like structure is formed in accordance with the hierarchy of the maternal ostis-systems and their corresponding libraries.

When specifying the address for the root of the specification address storage tree, the component manager gets access to all the specifications of the child storages. When processing such a specification tree, the component manager immerses the specifications of the components available for installation into sc-memory but not these components themselves.

The *manager of reusable ostis-systems components* is an optional subsystem of the ostis-platform. However, a system with a component manager can install components not only into itself but also into other systems if it has an access. Thus, one system can replace the ostis-platform of another system, while leaving the *sc-model of a cybernetic system*. In the same way, some ostis-system can generate other ostis-systems using a component approach.

Inclusion of the component in the *child ostis-system* generally consists of the following steps:

- search for a suitable component in a set of available libraries;
- allocation of a component in a form convenient for transportation to a child ostis-system with an indication of the version and modification, if necessary (for example, selection of the available component storage, selection of the optimal implementation option for the component taking into account the composition of the child system);
- installation of a reusable component and its dependencies (indicating the version and modification, if necessary);
- integration of a component into a child system;
- search and elimination of errors and contradictions in the child system.

From the point of view of the user, this process does not depend on the type of the component and the specifics of its implementation.

VI. Integration of reusable ostis-systems components

Let us consider in more detail the process of integrating reusable ostis-systems components as a process of immersing sc-constructions into the knowledge base. The most valuable components are those that can be integrated into a working system without stopping its functioning. Such components include, for example, knowledge base components.

The problem of integration of reusable ostis-systems components is solved by the interaction of components through a common knowledge base. Components can only use common key nodes (concepts) in the knowledge base (see [25] and [26]). Integration of reusable ostis-systems components is reduced to matching the key nodes and

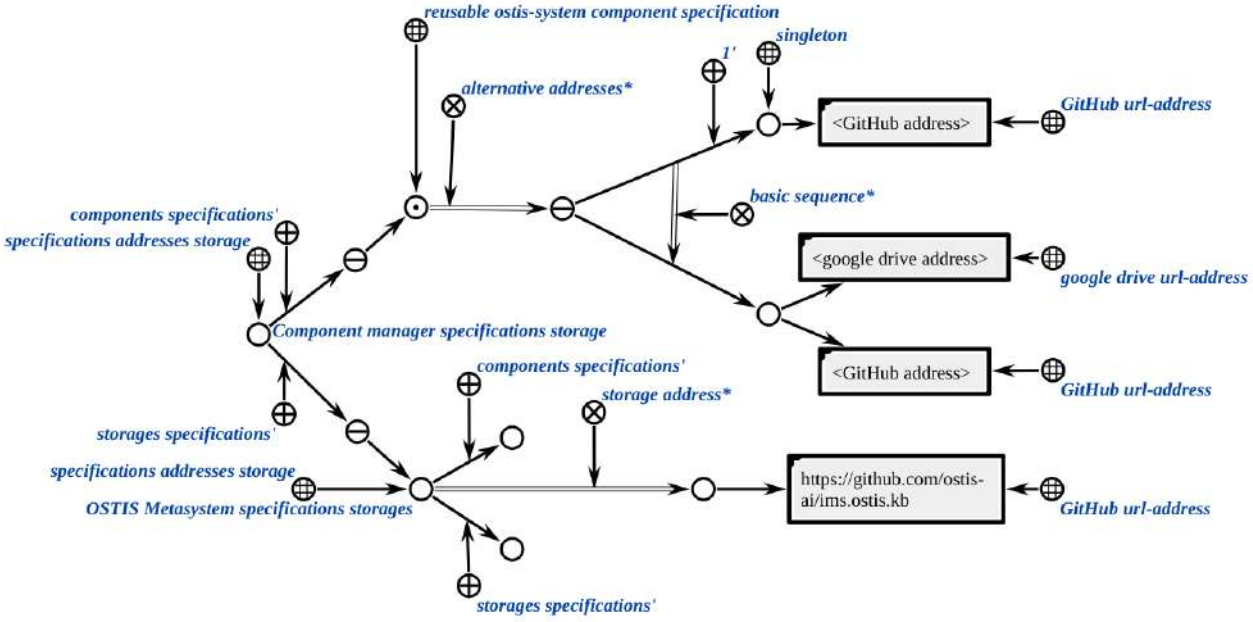

Figure 7. Example for the structure of the address storage of component specifications

eliminating possible duplications and contradictions based on the specification of the component and its contents. This way of integrating components allows them to be developed in parallel and independently of each other, which significantly reduces the time for design.

The process of integrating a reusable component into an ostis-system is a system of the following interacting processes:

- reducing a component to a unified form, that is, representing it in the SC-code;
- coordination of the key nodes and ontologies of the component and the knowledge base of the system;
- allocation of such sc-elements in the integrated component that have global (unique) identifiers;
- allocation of sc-elements in the integrated component that have local identifiers together with the scope of each such identifier. The scope of the local identifier is such a fragment of the knowledge base within which different sc-elements having this local identifier are considered synonymous;
- gluing together sc-elements having the same global identifiers;
- gluing together sc-elements having the same local identifiers if each of these sc-elements belongs to the scope of its local identifier and the scope of the local identifier of another sc-element;
- gluing together sc-elements based on the unambiguity of the algebraic operations used;
- gluing together sc-elements based on logical statements about the existence of uniqueness;
- gluing together multiple connectives belonging to relations in which multiple connectives are absent either always or under certain clearly specified conditions.

Thus, the integration of components, that is, the process of immersion (understanding) of one semantic network into another, is a non-trivial process of reasoning aimed at identifying pairs of synonymous elements of a semantic network based on certain knowledge available in the knowledge base of an intelligent system. Based on these stages, it is important to clearly structure and fill the knowledge base of the ostis-system into which the component is integrated, as well as qualitatively specify components and the contents of these components.

Let us consider the simplest example of integrating a reusable component of problem solvers, which is an sc-agent for finding the shortest path in a graph, into a logistics process management system. Let us assume that the logistics process management system is able to solve problems related to cost optimization in the process of creating and warehousing of goods. An example for a fragment of a logistics system model is shown in Figure 8.

At the same time, the knowledge base of the system describes what problems the system should solve and the corresponding actions. For example, “action. find the shortest path in the graph”, which is currently difficult for the system to perform.

The graph theory problem-solving system has a rich knowledge base and a problem solver, including an sc-agent for finding the shortest path in a graph, which is specified as a reusable component and can be used in a logistics system (see Figure 9).

As a result of installing a *reusable component* in the

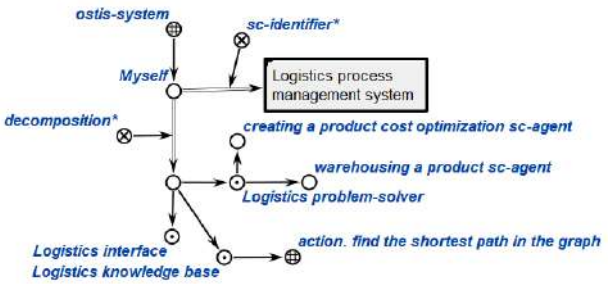

Figure 8. Structure of the logistics process management system

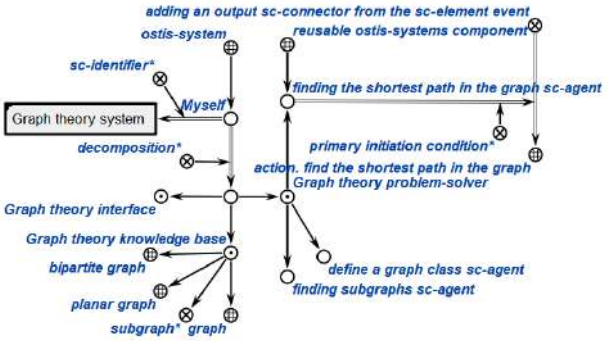

Figure 9. Structure of the graph theory problem-solving system

form of an sc-agent for finding the shortest path in the graph, its entire sc-model is immersed in a logistics problem-solving system. When integrating a reusable component, which is an sc-agent for finding the shortest path in a graph, into a logistics system, the key node of the logistics system *"action. find the shortest path in the graph"* is matched with the same node from the installed component from the graph theory system. Thus, when solving logistical problems, the system will be able to interpret the action of finding the shortest paths using an integrated component.

Integration of any ostis-systems components occurs automatically, without developer intervention. This is achieved through the use of the *SC-code* and its advantages, the unification of the specifications of reusable components, and the careful selection of components in libraries by the expert community responsible for this library.

VII. Conclusion

The component approach is key in the technology of designing intelligent computer systems. At the same time, the technology of component design is closely related to the other components of the technology of designing intelligent computer systems and ensures their compatibility, producing a powerful synergetic effect when using the entire complex of private technologies for designing intelligent systems. The most important principle in the implementation of the component approach is the semantic compatibility of reusable components, which minimizes the participation of programmers in the creation of new computer systems and the improvement of existing ones.

To implement the component approach, in the article, a library of reusable compatible components of intelligent computer systems based on the *OSTIS Technology* is proposed, classification and specification of reusable ostis-systems components is introduced, a component manager model is proposed that allows ostis-systems to interact with libraries of reusable components and manage components in the system, the architecture of the ecosystem of intelligent computer systems is considered from the point of view of using a library of reusable components.

The results obtained will improve the design efficiency of intelligent systems and automation tools for the development of such systems, as well as provide an opportunity not only for the developer but also for the intelligent system to automatically supplement the system with new knowledge and skills.

References

[1] K. Yaghoobirafi and A. Farahani, "An approach for semantic interoperability in autonomic distributed intelligent systems," *Journal of Software: Evolution and Process*, vol. 34, p. 18, 02 2022.

[2] N. N. Skeeter, N. V. Ketko, A. B. Simonov, A. G. Gagarin, and I. Tislenkova, "Artificial intelligence: Problems and prospects of development," *Artificial Intelligence: Anthropogenic Nature vs. Social Origin*, p. 12, 2020.

[3] O. Yara, A. Brazheyev, L. Golovko, L. Golovko, and V. Bashkatova, "Legal regulation of the use of artificial intelligence: Problems and development prospects," *European Journal of Sustainable Development*, p. 281, 2021.

[4] J. Waters, B. J. Powers, and M. G. Ceruti, "Global interoperability using semantics, standards, science and technology (gis3t)," *Computer Standards & Interfaces*, vol. 31, no. 6, pp. 1158–1166, 2009.

[5] M. Wooldridge, *An introduction to multiagent systems*, 2nd ed. Chichester : J. Wiley, 2009.

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

[7] P. Lopes de Lopes de Souza, W. Lopes de Lopes de Souza, and R. R. Ciferri, "Semantic interoperability in the internet of things: A systematic literature review," in *ITNG 2022 19th International Conference on Information Technology-New Generations*, S. Latifi, Ed. Cham: Springer International Publishing, 2022, pp. 333–340.

[8] I. Ashvin, *Component Design for Relational Databases*, 12 2021, pp. 143–156.

[9] B. Ford, R. Schiano-Phan, and J. Vallejo, *Component Design*, 11 2019, pp. 160–174.

[10] A. Donatis, *OOP in Component Design*, 01 2006, pp. 3–31.

[11] Ryndin, Nikita and Sapegin, Sergey, "Component design of the complex software systems, based on solutions' multivariant synthesis," *International Journal of Engineering Trends and Technology*, vol. 69, pp. 280–286, 12 2021.

[12] D. Shunkevich, "Agent-oriented models, method and tools of compatible problem solvers development for intelligent systems," in *Open semantic technologies for intelligent systems*, ser. 2, V. Golenkov, Ed. BSUIR, Minsk, 2018, pp. 119–132.

[13] D. V. Shunkevich, I. T. Davydenko, D. N. Koronchik, I. I. Zukov, and A. V. Parkalov, "Sredstva podderzki komponentnogo projectirovanija sistem ypravlyaenih znaniyami [support tools knowledge-based systems component design]," *Open semantic technologies for intelligent systems*, pp. 79–88, 2015. [Online]. Available: http://proc.ostis.net/proc/Proceedings%20OSTIS-2015.pdf

[14] M. Wilkes, *The Preparation of Programs for an Electronic Digital Computer: With Special Reference to the EDSAC and the Use of a Library of Subroutines*, ser. Addison-Wesley mathematics series. Addison-Wesley Press, 1951. [Online]. Available: https://books.google.by/books?id=PzVVAAAAMAAJ

[15] J. Blähser, T. Göller, and M. Böhmer, "Thine — approach for a fault tolerant distributed packet manager based on hypercore protocol," in *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2021, pp. 1778–1782.

[16] V. Torres da Silva, J. S. dos Santos, R. Thiago, E. Soares, and L. Guerreiro Azevedo, "OWL ontology evolution: understanding and unifying the complex changes," *The Knowledge Engineering Review*, vol. 37, p. e10, 2022.

[17] A. Memduhoglu and M. Basaraner, "Possible contributions of spatial semantic methods and technologies to multi-representation spatial database paradigm," *International Journal of Engineering and Geosciences*, pp. 108 – 118, Oct. 2018. [Online]. Available: https://doi.org/10.26833/ijeg.413473

[18] P. Fritzson, *Modelica Library Overview*, 2015, pp. 909–975.

[19] S. Prakash Pradhan, *Working with Microsoft Power Apps*. Berkeley, CA: Apress, 2022, pp. 79–131. [Online]. Available: https://doi.org/10.1007/978-1-4842-8600-5_3

[20] V. Gribova, L. Fedorischev, P. Moskalenko, and V. Timchenko, "Interaction of cloud services with external software and its implementation on the IACPaaS platform," pp. 1–11, 2021.

[21] V. Golenkov, N. Guliakina, and D. Shunkevich, *Otkrytaya tekhnologiya ontologicheskogo proektirovaniya, proizvodstva i ekspluatatsii semanticheski sovmestimykh gibridnykh intellektual'nykh komp'yuternykh sistem [Open technology of ontological design, production and operation of semantically compatible hybrid intelligent computer systems]*. Minsk: Bestprint [Bestprint], 2021, (In Russ.).

[22] (2023, September) IMS.ostis Metasystem. [Online]. Available: https://ims.ostis.net

[23] G. Sellitto, E. Iannone, Z. Codabux, V. Lenarduzzi, A. De Lucia, F. Palomba, and F. Ferrucci, "Toward understanding the impact of refactoring on program comprehension," in *29th International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2022, pp. 1–12.

[24] M. K. Orlov, "Comprehensive library of reusable semantically compatible components of next-generation intelligent computer systems," in *Open semantic technologies for intelligent systems*, ser. Iss. 6. Minsk : BSUIR, 2022, pp. 261–272.

[25] C. Acharya, D. Ojha, R. Gokhale, and P. C. Patel, "Managing information for innovation using knowledge integration capability: The role of boundary spanning objects," *International Journal of Information Management*, vol. 62, p. 102438, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0268401221001316

[26] V. Ivashenko, "Application of an integration platform for ontological model-based problem solving using an unified semantic knowledge representation," in *Open semantic technologies for intelligent systems*, ser. Iss. 5. Minsk : BSUIR, 2022, pp. 179–186.

Средства управления многократно используемыми компонентами интеллектуальных компьютерных систем нового поколения

Орлов М. К.

Важнейшим этапом эволюции любой технологии является переход к компонентному проектированию на основе постоянно пополняемый библиотеки многократно используемых компонентов. В работе рассматривается подход к проектированию систем, управляемых знаниями, ориентированный на использование совместимых многократно используемых компонентов, что существенно сокращает трудоемкость разработки таких систем.

Received 13.03.2023