Comprehensive library of reusable semantically compatible components of next-generation intelligent computer systems

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

***Abstract*—The most important stage in the evolution of any technology is the transition to the component design based on a constantly augmented library of reusable components. In the article, an approach to the design of knowledge-driven systems is considered, focused on the usage of compatible reusable components, which significantly reduces the complexity of developing such systems.**

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

I. Introduction

As the analysis of modern information technologies shows, along with achievements, they have a number of serious disadvantages associated with the complexity of their development and maintenance. In particular, such disadvantages include the following ones [1], [2], [3]:

- there is no general unified solution to the problem of the semantic compatibility of computer systems, which causes a high complexity of creating complex integrated computer systems;
- there is a variety of semantically equivalent implementations of problem-solving models, duplication of knowledge base and user interface components that differ not in the essence of these components but in the form of representation of the processed information;
- the degree of dependence of computer system architectures on the platforms on which they are implemented is high, which causes the complexity of transferring computer systems to new platforms;
- modern information technologies are not oriented to a wide range of developers of applied computer systems;
- there is a lack of a unified approach to the allocation of reusable components and the formation of libraries of such components, which leads to a high complexity of reusage and integration of previously developed components in new 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 [4], [5]. However, their integration is possible if the same technologies are used and only when designed by one development team [6].

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 find [7]. 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 [8].

To solve these problems, it is proposed to implement a comprehensive library of reusable semantically compatible components of next-generation intelligent computer systems.

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.

Reusage of ready-made components is widely used in many industries related to the design of various kinds of systems, since it allows reducing the complexity of development and its cost (by minimizing the amount of work due to the absence of the need to develop any component), improving the quality of the created content, and reducing professional requirements for computer system developers [9], [10]. Thus, the transition is made from programming components or entire systems to their design based on ready-made components. The usage of ready-made components assumes that the distributed component is verified, tested, evaluated, and documented, and possible limitations are eliminated or specified and known.

The following problems exist in the implementation of the component approach to the design of next-generation intelligent computer systems [11]:

- 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;
- the inability to automatically integrate components into the system without manual user intervention;
- testing, verification, and analysis of the components quality are not carried out; advantages, disadvantages, limitations of components are not identified;
- the 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;
- the lack of tools to search for components that meet the specified criteria [12].

The purpose of the work is to create conditions for effective, meaningful, and mass design of next-generation intelligent computer systems and their components by creating an environment for the collection and sharing of components of these systems. Such conditions are realized by unlimited expansion of constantly evolving semantically compatible intelligent computer systems and their components. The spheres 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 for traditional computer systems, but such libraries do not solve the above problems.

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 and can also be located in different places, which leads to the fact that a tool is needed in the library to find components and install them [13].

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 but only install components by identifier [14]. 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. This storages are realized as some kind of repository, that is not compatible together [15]. There is no single interacting system to store, analyze and supply reusable components. 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.

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 [16]. 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 [17]. 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.

WebProtege is a multi-user web interface that allows editing and storing ontologies in the OWL format in a collaborative environment [18]. 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 reusage 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 [19]. Consequently, there is a problem of their semantic incompatibility, which, in turn, leads to the impossibility of reusage 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 [20].

The IACPaaS platform is designed to support the development, management, and remote usage of applied and instrumental multi-agent cloud services (primarily intelligent) and their components for various subject domains [21]. 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 the means for a unified representation of the components of intelligent computer systems and the means for their specification and automatic integration.

III. Proposed approach

Within this article, it is proposed to take an OSTIS Technology [22] as a basis, the principles of which make it possible to implement a library of semantically compatible components of intelligent computer systems and, accordingly, provide the ability 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 [23] 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. 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 ontologies:

Subject domain of reusable ostis-systems components
⇒ *private subject domain**:
  Subject domain and ontology of a comprehensive library of reusable semantically compatible ostis-systems components

Subject domain and ontology of a comprehensive library of reusable semantically compatible ostis-systems components
⇒ *private subject domain**:
  - *Subject domain and ontology of the library of reusable components of ostis-systems knowledge bases*
  - *Subject domain and ontology of the library of reusable components of ostis-systems problem solvers*
  - *Subject domain and ontology of the library of reusable components of ostis-systems interfaces*

In the subject domain and ontology of the library of reusable semantically compatible ostis-systems components, the concepts and principles most common to all child subject domains are described, which are valid for any library of reusable components.

The idea of a component library is not new, but the semantic potency of the ***OSTIS Library*** is significantly higher than for analogues due to the fact that the vast majority of library components are knowledge base components represented in a unified knowledge representation language (*SC-code*). Thus, the OSTIS Library provides a high level of semantic compatibility of components, which leads to a high level of semantic compatibility of ostis-systems using the library of reusable ostis-systems components.

Next, we will consider in more detail the fragments of sc-models of the specified subject domain and ontology.

IV. Concept of a library of reusable ostis-systems components

The basis for the implementation of the component approach within the *OSTIS Technology* is the OSTIS Library. The *OSTIS Metasystem* is focused on the development and practical implementation of methods and tools for component design of semantically compatible intelligent systems, which provides an opportunity to quickly create intelligent systems for various purposes. The OSTIS Metasystem includes the ***OSTIS Metasystem Library***.

library of reusable ostis-systems components
⇒ *abbreviation**:
  [library of ostis-systems components]
:= [library of reusable OSTIS components]
∋ *typical example′*:
  OSTIS Library
    := [Library of reusable ostis-systems components as part of the OSTIS Metasystem]
    := [OSTIS Metasystem Library]
⇐ *combination**:
  {• *library of typical subsystems of ostis-systems*
   • *library of templates for typical ostis-systems components*
   • *library of ostis-platforms*
   • *library of reusable knowledge base components*
   • *library of reusable problem solver components*
   • *library of reusable user interface components*
  }

The constantly expanding OSTIS Library significantly reduces the time for the development of new intelligent computer systems. The library of ostis-systems allows getting rid of duplication of semantically equivalent information components as well as from the variety of forms for the technical implementation of the problem-solving models used.

Currently, a large number of *knowledge bases* in a variety of subject domains have been developed. However, in most cases, each knowledge base is developed separately and independently of the others in the absence of a single unified formal basis for the knowledge representation, as well as common principles for the formation of concept systems for the described subject domain. In this regard, the developed bases are, as a rule, incompatible with each other and are not suitable for reusage. A component-based approach to the development of intelligent computer systems, implemented in the form of a ***library of reusable ostis-systems components***, allows solving the described problems. In the field of development of *problem solvers*, there are also a large number of specific implementations, however, problems of compatibility of different solvers when solving a single problem are hardly considered. Hypothetically, the existence of a universal problem solver combining all known problem-solving methods and ways is possible. However, the usage of such a solver for applied purposes is not advisable. Thus, the most acceptable option is to create a library of compatible components, from which a solver that meets the necessary requirements can later be compiled.

Functionality of the library of reusable ostis-systems components:

- Storing reusable ostis-systems components and their specifications. At the same time, some of the components specified within the library may be physically stored in another place due to the peculiarities of their technical implementation (for example, the source texts of the ostis-platform may be physically stored in a separate repository but be specified as a component in the corresponding library). In this case, the specification of the component within the library has to contain the description of (1) the location of the component 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.
- Storing information about which of the library components and which version are used (have been downloaded) in particular consumer ostis-systems. This is necessary at least to take into account the demand for a particular component, to assess its importance and popularity.
- Systematization of reusable ostis-systems components.
- Providing versioning of reusable ostis-systems components.
- Searching for dependencies between reusable components within the library of components.
- Ensuring automatic updating of components borrowed into the child ostis-systems. This function can be turned on and off upon request of the developers of the child ostis-system. Simultaneous updating of the same components in all systems using it should

not lead to inconsistency between these systems in any context. This requirement may be quite complicated but is essential for the operation of the OSTIS Ecosystem.

library of reusable ostis-systems components
⇒ *generalized decomposition**:
{• *knowledge base of the library of reusable ostis-systems components*
• *problem solver of the library of reusable ostis-systems components*
• *interface of the library of reusable ostis-systems components*
}

A knowledge base of the library of reusable ostis-systems components is a hierarchy of reusable ostis-systems components and their specifications, as well as a system of concepts necessary for the specification, installation, and search of components.

A problem solver of the library of reusable ostis-systems components implements the functionality of the ostis-systems library described above.

An interface of the ostis-systems library provides access to reusable components and features of the ostis-systems library for the user and other systems. The interface allows the *manager of reusable ostis-systems components*, which is part of a child ostis-system, to connect to the library of reusable ostis-systems components and use its functionality, that is, for example, to access the specification of components and install selected components in a child ostis-system, get information about the available versions of the component, its dependencies, etc.

V. Place of the OSTIS Library in the architecture of the OSTIS Ecosystem

Developers of any ostis-system can include a library in its structure, which will allow them to accumulate and distribute the results of their activities among other participants of the *OSTIS Ecosystem* in the form of ***reusable components***. The decision to include the component in the library is made by the expert community of developers responsible for the quality of this library. Component verification can be automated. Within the *OSTIS Ecosystem*, there are many libraries of reusable ostis-systems components that are subsystems of the corresponding maternal ostis-systems. The main library of reusable ostis-systems components is the *OSTIS Metasystem Library*. The *OSTIS Metasystem* acts as a ***maternal system*** for all ostis-systems being developed, since it contains all the basic components (Figure *1*). The maternal system is responsible for some class of components and is a SAD for this class, contains methods for the development of such components, their classification, detailed explanations for all subclasses of components. Thus, a hierarchy of maternal ostis-systems is formed. The maternal ostis-system, in turn, can be a child ostis-system for some other ostis-system, borrowing components from the library that is part of this other ostis-system.

Publishing a component to a certified library requires additional effort from the developer to ensure the quality of the component specification and description of its relationship with other components, however, provides the following benefits of using the OSTIS Ecosystem infrastructure.

- Downloads of components registered in a certified library are captured and tracked. the quality and importance of the component is automatically proven by the number of its downloads, this is visible to all members of the community. Thus, the rating of the developer is formed, it becomes more popular and in demand. Registering a component in the library is automatically a "quality mark", showing other developers that the component has been verified and the risk of problems when using it is significantly reduced.
- Creating proprietary components that can be distributed under a paid license.

ostis-system
⊃ *maternal ostis-system*
:= [ostis-system that includes a library of reusable components]
∋ *OSTIS Metasystem*
⊃ *child ostis-system*
:= [ostis-system that contains a component borrowed from a library of reusable components]

Integration of reusable ostis-systems components is reduced to bonding key nodes by identifiers and eliminating possible duplications and contradictions based on the specification of the component and its content. Integration of any ostis-systems components occurs automatically, without the intervention of the developer. This is achieved through the usage of the *SC-code* and its advantages, the unification of the specifications of reusable components, and the thorough selection of components in libraries by the expert community responsible for this library.

VI. Concept of a reusable ostis-systems component

A reusable ostis-systems component is understood as a component of some ostis-system that can be used within another ostis-system. 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

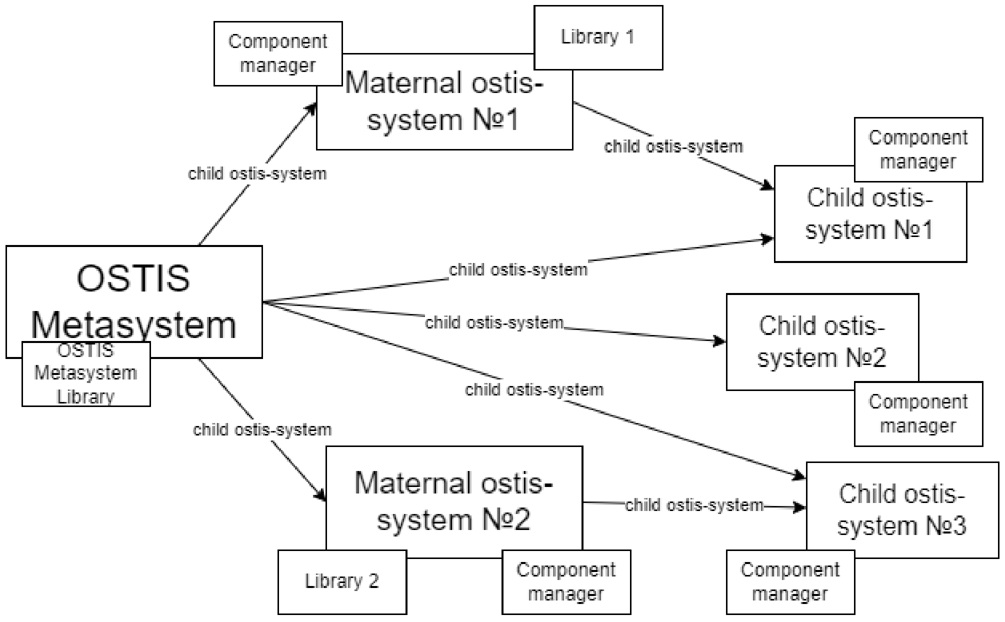

Figure 1. Architecture of the OSTIS Ecosystem

a component of some ***maternal ostis-system***, which can be used in some ***child ostis-system***. 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 expansion of the set of problems to be solved when integrating components.

reusable ostis-systems component
:= [reusable OSTIS component]
:= *frequently used sc-identifier**:
[reusable component]
$\subset$ *ostis-system component*

The ostis-system component is an integral part of the ostis-system, which contains all those (and only those) sc-elements that are necessary for its functioning in the ostis-system. The difference between a reusable ostis-systems component and an ostis-system component is that a reusable component has a specification sufficient to install this component in a child ostis-system. The specification is part in the knowledge base of the ***library of reusable components*** for the corresponding maternal ostis-system.

Necessary requirements for reusable ostis-systems components:

- there is a technical possibility to embed a reusable component into a child ostis-system;
- completeness of a reusable component: the component has to fully perform its functions, correspond to its purpose;
- coherence of a reusable component: a component should logically perform only one task from the subject domain for which it is intended. A reusable component has to 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 the ostis-systems in which it is embedded and have an ability to be integrated automatically into other systems;
- self-sufficiency of the components (or groups of components) of the technology, i.e. their ability to function separately from other components without losing the reasonableness of their usage.

VII. Classification of reusable ostis-systems components

Let us consider the classification of reusable ostis-systems components. The class of a reusable ostis-systems component is an important part of the component specification, which allows better understanding of the purpose and application scope of this component, as well as the class of a reusable component is the most important criterion for searching for components in the ostis-systems library. The main feature of the classification of reusable components is the attribute of the subject domain to which the component belongs. Here the structure can be quite extensive in accordance with the hierarchy of areas of human activity. The following list is not full, it is a short example.

reusable ostis-systems component

⊃ *reusable component of subject domain of medicine*
⊃ *reusable component of subject domain of mathematics*
⊃ *reusable component of subject domain of economics*
⊃ *reusable component of subject domain of art*
⊃ *reusable component of subject domain of computer systems*
⊃ *reusable component of subject domain of plants*

reusable ostis-systems component
⇒ *subdividing**:
{• *reusable knowledge base component*
  ⊃ *semantic neighborhood*
  ⊃ *subject domain and ontology*
  ⊃ *knowledge base*
  ⊃ *template of a typical ostis-systems component*
    ∋ *Template for the subject domain description*
    ∋ *Template for the relation description*
• *reusable problem solver component*
  ⊃ *atomic knowledge processing agent*
  ⊃ *knowledge processing program*
• *reusable interface component*
  ⊃ *reusable user interface component for display*
  ⊃ *interactive reusable user interface component*
}

For knowledge base components, the most important feature of the classification of reusable components is the type of knowledge used. For the components of the problem solver, there is a problem-solving model, for interface components – the type of interface in accordance with the classification of user interface components.

reusable ostis-systems component
⇒ *subdividing**:
{• *atomic reusable ostis-systems component*
  ∋ *semantic neighborhood of set*
  ∋ *sc-agent of set power calculating*
• *non-atomic reusable ostis-systems component*
  ∋ *abstract sc-agent of logical inference*
  ∋ *knowledge base of geometry*
}

The typology of the 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. The belonging of a reusable ostis-systems component to a class of atomic components depends on its specification and on the currently existing components in the library. 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 parts. Without any part of the non-atomic component, its purpose restricts. In general, an atomic component can become non-atomic if it is decided to allocate some of its parts as a separate component. All of the above, however, does not mean that even in the case of its platform independence, the atomic component is always stored in sc-memory as a formed sc-structure.

For example, a platform-independent implementation of the sc-agent will always be represented by a set of scp programs but will be an atomic reusable OSTIS component if none of these programs will be of interest as an independent component. In general, a non-atomic component can become atomic if it is decided for some reason to exclude all its parts from consideration as separate components. It should be noted that a non-atomic component does not necessarily have to be composed entirely of other components – it may also include parts that are not independent components. For example, an agent implemented in the SCP language, which is a non-atomic reusable component, may include both scp programs that may (or may not) be reusable components, as well as an agent scp program that does not make sense as a reusable component.

reusable ostis-systems component
⇒ *subdividing**:
{• *dependent reusable ostis-systems component*
  ∋ *chemistry visualizer*
  ∋ *subject domain of artificial neural networks*
• *independent reusable ostis-systems component*
  ∋ *semantic neighborhood of set*
  ∋ *interface button 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.

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. 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*
⊃ *ostis-platform*
⊃ *abstract sc-agent that is not implemented in the SCP Language*
● *platform-independent reusable ostis-systems component*
⊃ *reusable knowledge base component*
⊃ *SCP agent*
⊃ *SCP program*
}

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. 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. A platform-independent reusable ostis-systems component is a component that is entirely represented in the *SC-code*. In the case of a non-atomic reusable component, this means that all the simpler components that are part of it must also be platform-independent reusable ostis-systems components. 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.

reusable ostis-systems component
⊃ *typical subsystem of ostis-systems*
∋ *Environment for the collective development of ostis-systems knowledge bases*
∋ *Visual web-oriented editor of sc.g-texts*

For storing reusable ostis-systems components, some storage is required. Such storage can be either an ostis-system or a third-party storage, for example, a cloud service. In addition to the external files of the component, its specification must be located in the storage.

VIII. Specification of reusable ostis-systems components

Each *reusable ostis-systems component* must be specified within the library. The specification includes basic knowledge about the component, which allows ensuring the building of a complete hierarchy of components and their dependencies, and also provides unrestricted integration of components into child ostis-systems. Both relations and component classes are used for component specification.

In order for a reusable component to be accepted into the library, it is required to specify it using a *relation, necessary for installation, that specifies a reusable ostis-systems component*. At the same time, a *relation, optional for installation, that specifies a reusable ostis-systems component* helps to better understand the essence of the component, simplifies the search, but is not necessary for the installation of the component in the ostis-system.

relation specifying a reusable ostis-systems component^
:= [relation that is used in the specification of a reusable ostis-systems component]
⇒ *subdividing**:
{• *relation, necessary for installation, that specifies the reusable ostis-systems component*
∋ *installation method**
∋ *storage address**
∋ *component dependencies**
• *relation, optional for installation, that specifies a reusable ostis-systems component*
∋ *related component**
∋ *change history**
∋ *authors**
∋ *note**
∋ *explanation**
∋ *identifier**
∋ *key sc-element**
∋ *purpose**
}

The installation method allows the user to install the component manually and the ***component manager*** – automatically. Two main methods of installing reusable components are the method of installing a dynamically installed reusable ostis-systems component and the method of installing a reusable component, when installing which the system requires a restart. With a dynamic installation, it is only necessary to download the component – and it immediately works in the system.

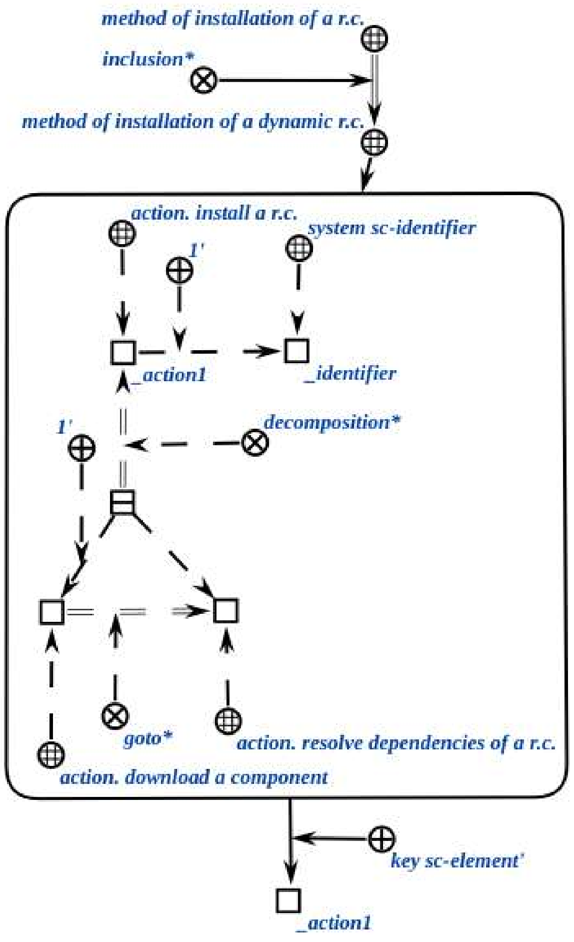
method of installation of a r.c.
inclusion*
method of installation of a dynamic r.c.
action. install a r.c.
system sc-identifier
1'
_action1
_identifier
1'
decomposition*
goto*
action. resolve dependencies of a r.c.
action. download a component
key sc-element'
_action1

Figure 2. The method of installation of a dynamically installed reusable ostis-systems component

When installing a component, during the installation of which the system requires a restart, it is necessary to translate it into the system memory in addition to downloading the component.

The connectives of the *storage address** relation link a reusable component stored as external files and a file containing the URL of a reusable ostis-systems component. Such a file can be a file containing the URL on GitHub of a reusable ostis-systems component, a file containing the URL on Google Drive of a reusable ostis-systems component, a file containing the URL on Docker Hub of a reusable ostis-systems component, and others.

The connectives of the *component dependencies** relation link a reusable component and a set of components, without which the installed component cannot be embedded in a child ostis-system. This components must be successfully installed before the installation of the dependent component.

In some cases, it may turn out that in order to use one reusable OSTIS component, it is advisable or even necessary to additionally use several other reusable OSTIS

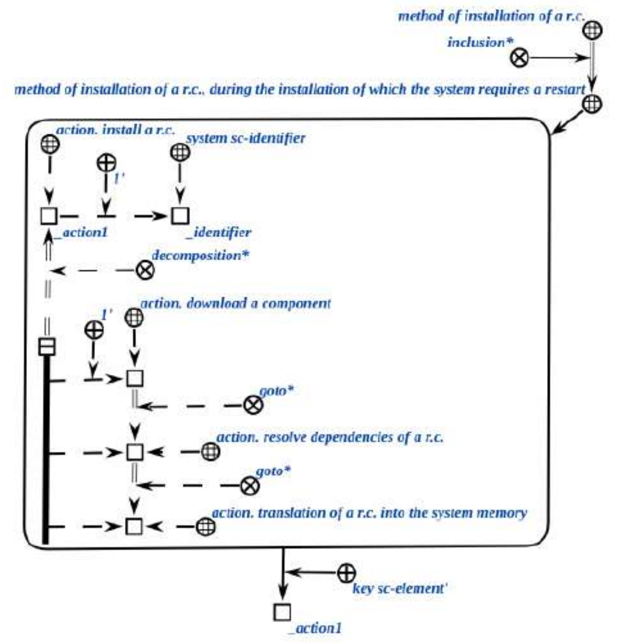
method of installation of a r.c.
inclusion*
method of installation of a r.c., during the installation of which the system requires a restart
action. install a r.c.
system sc-identifier
1'
_action1
_identifier
decomposition*
action. download a component
1'
goto*
action. resolve dependencies of a r.c.
goto*
action. translation of a r.c. into the system memory
key sc-element'
_action1

Figure 3. The method of installing a dynamically installed reusable component, during the installation of which the system requires a restart

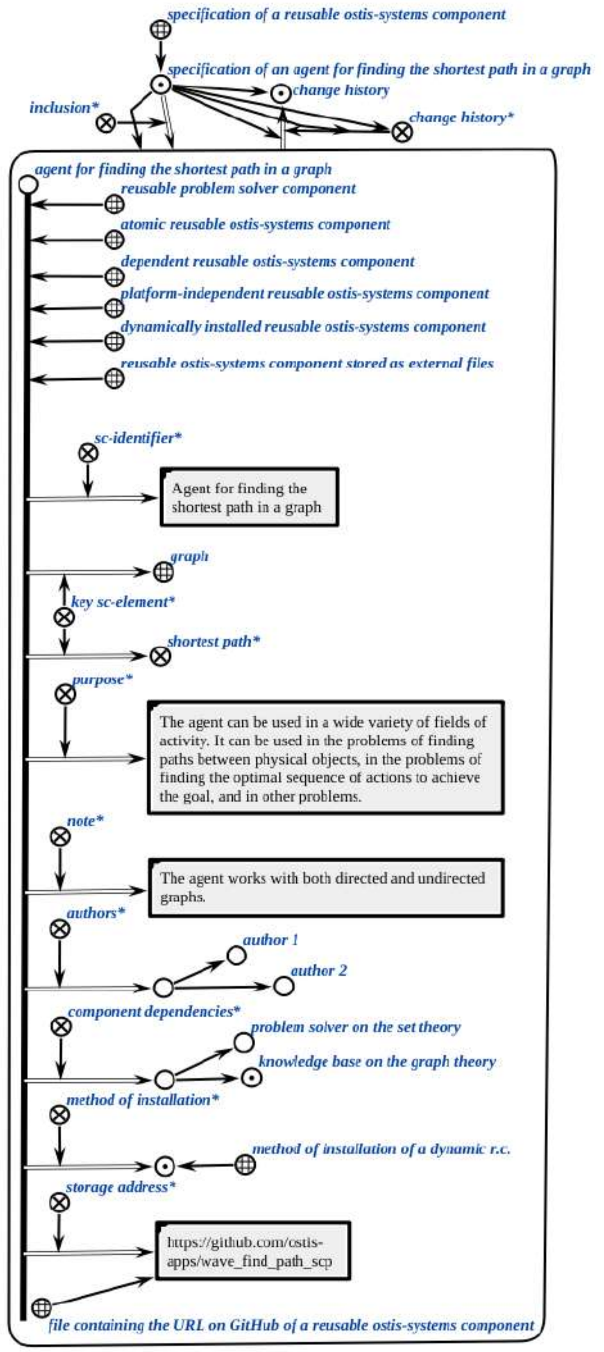
specification of a reusable ostis-systems component
specification of an agent for finding the shortest path in a graph
change history
inclusion*
change history*
agent for finding the shortest path in a graph
reusable problem solver component
atomic reusable ostis-systems component
dependent reusable ostis-systems component
platform-independent reusable ostis-systems component
dynamically installed reusable ostis-systems component
reusable ostis-systems component stored as external files
sc-identifier*
Agent for finding the shortest path in a graph
graph
key sc-element*
shortest path*
purpose*
The agent can be used in a wide variety of fields of activity. It can be used in the problems of finding paths between physical objects, in the problems of finding the optimal sequence of actions to achieve the goal, and in other problems.
note*
The agent works with both directed and undirected graphs.
authors*
author 1
author 2
component dependencies*
problem solver on the set theory
knowledge base on the graph theory
method of installation*
method of installation of a dynamic r.c.
storage address*
https://github.com/ostis-apps/wave_find_path_scp
file containing the URL on GitHub of a reusable ostis-systems component

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

components. For example, it may be advisable to use the appropriate interface command combined with any sc-agent of information search, which is represented by a separate component and will allow the user to ask a question for the specified sc-agent through the system interface. In such cases, the related component* relation is used to link components. The presence of such links makes it possible to eliminate possible problems of incomplete knowledge and skills in the child system, due to which any of the components may not perform their functions. The connectives of the related component* relation link reusable ostis-systems components that it is advisable to use in a child system together. Each such connective can additionally be provided with a sc-comment or sc-explanation reflecting the essence of the specified dependency.

IX. 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 ostis-systems components takes place. The manager of reusable ostis-systems components should be implemented using as few dependencies (ostis-platform components dependencies as well as external dependencies) as possible to provide the maximum level of configurability of developed ostis-sistems.

manager of reusable ostis-systems components
:= *frequently used sc-identifier**:

[manager of reusable components]
:= *frequently used sc-identifier**:
[manager of components]
⇒ *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*
}

The knowledge base of the manager of components contains all the knowledge that is necessary to install a reusable component in a child ostis-system. Such knowledge includes knowledge about the specification of reusable components, methods of installing components, knowledge about the ostis-systems libraries with which interaction takes place. The problem solver of the manager of components interacts with the ostis-systems library and allows installing and integrating reusable components into a child ostis-system, as well as searching, updating, publishing, and deleting components. The interface of the manager of reusable components provides convenient usage of the manager of components for the user and other systems.

The functionality of the manager of ostis-systems components is as follows:

- **Search for reusable ostis-systems components**. The set of possible search criteria corresponds to the specification of reusable components. Such criteria can be component classes, its authors, identifier, fragment of a note, purpose, belonging to a subject domain, type of component knowledge, and others.
- **Installation of a reusable ostis-systems component**. The installation of a reusable component takes place regardless of the typology, 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 in a child system, it is necessary to resolve all dependencies by installing dependent components. After successful installation of the component, an information construction is generated in the knowledge base of the child system, indicating the fact of installing the component into the system using the *installed components** relation. After installing the component in the ostis-system, contradictions may arise in the knowledge base, which are eliminated by means of detecting and analyzing errors and contradictions in the ostis-system knowledge base.
- **Publishing 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. It is also possible to update the version of the published component by the community of its developers.
- **Updating an installed reusable ostis-systems component**.
- **Deleting 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.
- **Adding and deleting libraries tracked by the ostis-system**. The manager of components contains information about a variety of sources for installing components, the list of which can be supplemented manually. By default, the manager of components tracks the OSTIS Metasystem Library, however, it is possible to create and add extra ostis-systems libraries.

X. Conclusion

In the article, the implementation of a library of reusable compatible components of intelligent computer systems based on the OSTIS Technology is proposed, which makes it possible to use a component-based approach to the design of intelligent systems and reduce the time and complexity of system development, as well as increase the level of compatibility of systems using reusable ostis-systems components.

The classification and specification of reusable ostis-systems components are clarified, the concepts of a components library and manager are considered.

An example for the specification of a component that can be found in the library of compatible ostis-systems components is given. 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] Natalia N. Skeeter, Natalia V. Ketko, Aleksey B. Simonov, Aleksey G. Gagarin, Irina Tislenkova, "Artificial intelligence: Problems and prospects of development," *Artificial Intelligence: Anthropogenic Nature vs. Social Origin*, 2020.

[2] Olena Yara, Anatoliy Brazheyev, Liudmyla Golovko, Liudmyla Golovko, Viktoriia Bashkatova, "Legal regulation of the use of artificial intelligence: Problems and development prospects," *European Journal of Sustainable Development*, 2021.

[3] Golenkov, V. V., "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.

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

[5] Ford, Brian and Schiano-Phan, Rosa and Vallejo, Juan, *Component Design*, 11 2019, pp. 160–174.

[6] Donatis, Antonio, *OOP in Component Design*, 01 2006.

[7] Nazia Bibi, Tauseef Rana , Ayesha Maqbool, Tamim Alkhalifah, Wazir Zada Khan, Ali Kashif Bashir, Yousaf Bin Zikria, "Reusable component retrieval: A semantic search approach for low resource languages," *ACM Transactions on Asian and Low-Resource Language Information Processing*, 2022.

[8] Bukhari, S. A. C., Krauthammer, M. & Baker, C. J. O., "An architecture for biomedical image discovery, interoperability and reusability based on semantic enrichment," *SWAT4LS(Citeseer, 2014)*, 2014.

[9] 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.

[10] L. Cafaro, R. Francese, C. Palumbo, M. Risi, and G. Tortora, "An agile process supporting software reuse: An industrial experience," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, 2018, pp. 1544–1551.

[11] Shunkevich D.V., Davydenko I.T., Koronchik D.N., Zukov I.I., Parkalov A.V., "Support tools knowledge-based systems component design," *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh system [Open semantic technologies for intelligent systems]*, pp. 79–88, 2015. [Online]. Available: http://proc.ostis.net/proc/Proceedings%20OSTIS-2015.pdf

[12] X. Qu, X. Feng, Y. Zhang, S. Wang, L. Sun, P. Hua, and Y. Wang, "Research on component retrieval and matching methods," in *2022 International Seminar on Computer Science and Engineering Technology (SCSET)*, 2022, pp. 358–362.

[13] T. Diamantopoulos and A. L. Symeonidis, "Mining source code for component reuse," in *Mining Software Engineering Data for Software Reuse*. Springer, 2020, pp. 133–174.

[14] Blähser, Jannik and Göller, Tim and Böhmer, Matthias, "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.

[15] V. A. Buregio, E. S. Almeida, D. Lucredio, and S. L. Meira, "Specification, design and implementation of a reuse repository," in *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, vol. 1, 2007, pp. 579–582.

[16] Fritzson, Peter, *Modelica Library Overview*, 2015, pp. 909–975.

[17] Prakash Pradhan, Sanjaya, *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

[18] Memduhoğlu, Abdulkadir and Basaraner, Melih, "Possible contributions of spatial semantic methods and technologies to multi-representation spatial database paradigm," *International Journal of Engineering and Geosciences*, vol. 3, pp. 108–118, 10 2018.

[19] M. Atzeni and M. Atzori, "Codeontology: Rdf-ization of source code," in *International Semantic Web Conference*. Springer, 2017, pp. 20–28.

[20] B. Antunes, P. Gomes, and N. Seco, "Srs: A software reuse system based on the semantic web," in *3rd International Workshop on Semantic Web Enabled Software Engineering (SWESE)*, 2007.

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

[22] Vladimir Golenkov and Natalia Guliakina and Daniil Shunkevich, *Open technology of ontological design, production and operation of semantically compatible hybrid intelligent computer systems*, V. Golenkov, Ed. Minsk: Bestprint [Bestprint], 2021.

[23] (2022, September) IMS.ostis Metasystem. [Online]. Available: https://ims.ostis.net

Комплексная библиотека многократно используемых семантически совместимых компонентов 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 30.10.2022