Principles of Automation of Development of Open Projects Based on the Ecosystem of Intelligent Computer Systems of the Next Generation

Natalia Grakova, Nikita Zotov, Maksim Orlov, Kseniya Petrochuk, Kseniya Bantsevich
department Intelligent Information Technologies
Belarusian State University of Informatics and Radioelectronics
Minsk, Belarus
grakova.nv@gmail.com, n.zotov@bsuir.by, orlovmaksimkonst@gmail.com, xenija.petrotschuk@gmail.com, ksusha.bantsevich@gmail.com

***Abstract*—The article is intended for researchers, developers, and practitioners in the fields of artificial intelligence, knowledge engineering, and intelligent system design who seek to build, extend, or integrate semantically compatible intelligent systems.**

It is particularly relevant for those interested in:

- **developing modular, reusable, and interoperable intelligent systems using open-source technologies;**
- **applying formal ontological methods and SC-code for knowledge representation and reasoning;**
- **implementing multi-agent problem solvers and integrating them with semantic knowledge bases;**
- **leveraging OSTIS Technology for educational, research, or industrial applications requiring adaptability, scalability, and semantic compatibility;**
- **research practical workflows, including installation, configuration, and testing, through real-world examples such as the ostis-example-app.**

Presented the guide assumes a basic familiarity with concepts in artificial intelligence, ontologies, and software development, but provides detailed, step-by-step instructions suitable for both newcomers and experienced professionals in the domain.

***Keywords*—OSTIS, open source project, ecosystem, basic specification, OSTIS, OSTIS Project, OSTIS Technology, OSTIS Guide, ostis-system, SC-code, knowledge base, problem solver, agent, ostis-example-app, ostis-web-platform, ostis-metasystem**

I. Introduction

Currently, a significant number of diverse projects aimed at automating the development of various software products are being developed. In particular, the creation of open-source projects is especially relevant.

Open-source projects [1] are a powerful tool that enables the development of high-quality software by combining the efforts of developers and users worldwide. However, open-source projects come with a range of advantages and disadvantages.

Advantages of open-source projects:

- large user and developer community;
- rapid identification and fixing of bugs;
- opportunity for collaborative use and code enhancement.

Disadvantages of open-source projects:

- potential vulnerabilities if the project is not properly maintained;
- unpredictability in terms of support and updates.

The development of open-source projects offers many advantages but is also associated with a number of challenges. Here are some of them [1]:

1) Coordination and management. Open-source projects often involve numerous contributors, which can lead to difficulties in coordinating work and managing the project. Without a clear strategy and role distribution, confusion may arise.
2) Diverse expertise levels. Participants may have varying levels of knowledge and experience, which can impact work quality. Training and support for newcomers are essential.
3) Lack of motivation. Volunteers may lose interest or motivation if they do not see progress or receive proper recognition for their work. This can result in delays and reduced project quality.
4) Conflicts and disputes. Differing opinions on development directions, architecture, or technology can lead to conflicts. Mechanisms for resolving disputes must be established.
5) Code quality. Maintaining uniform code standards is challenging, as contributors may follow their own practices. This complicates integration and testing.
6) Licensing and legal issues. Misunderstanding or misapplying licenses can cause legal problems. Proper documentation and adherence to licensing terms are critical.

7) Documentation challenges. Documentation may be insufficient or outdated, hindering new contributors' understanding and limiting project growth.
8) Funding. While many open-source projects rely on volunteers, funding may be necessary for infrastructure, hosting, or other resource-intensive needs.
9) Dependence on external factors. Projects may depend on changes in ecosystems (e. g., dependencies or external libraries), affecting their development and maintenance.
10) Diverse opinions. A flood of ideas and proposals can overwhelm the team and complicate decision-making. Clear processes for discussion and implementation are vital.

These challenges can be successfully addressed through effective organization, active communication, and a flexible approach to project management.

An open-source project is not merely one whose source code is made publicly available, but rather a project developed and improved by entire communities of specialists working collaboratively to evolve and refine it.

Naturally, managing, developing, and maintaining such a project—preserving its integrity and functionality—is far more complex compared to projects with centralized management.

II. Project classification

A project is a temporary endeavor aimed at creating a unique product, service, or result [2].

Let us consider the main types of projects:

1) *By dominant activity*:
   - research,
   - creative,
   - applied,
   - informational,
   - adventure, gaming, role-playing;
2) *By subject domain*:
   - *single-subject* – a project within the scope of one academic discipline,
   - *interdisciplinary* – integrating knowledge from two or more fields,
   - *transdisciplinary* – projects at the intersection of disciplines or beyond traditional subject boundaries;
3) *By coordination nature*:
   - open,
   - closed;
4) *By number of participants*:
   - individual,
   - paired,
   - group;
5) *By duration*:
   - short-term,
   - medium-term,
   - long-term;
6) *By geographical scope*:
   - district, region, territory,
   - interregional, international.

This is not an exhaustive classification. Thus, developing tools capable of adapting to diverse project types remains a relevant challenge today. Knowing the project type or at least its goal allows for determining the model.

In this work, we focus on projects classified by coordination nature.

Coordination in open projects refers to a set of measures and processes aimed at ensuring effective collaboration among all project participants. It is often characterized by open access to information and the involvement of a wide range of stakeholders. Open projects may relate to scientific research, software development, social initiatives, and other fields.

Let us examine key aspects of coordination in open projects:

1) *Communication*. Establishing clear communication channels among participants. Examples include messaging apps, email lists, forums, or collaboration platforms (e.g., GitHub, Slack).
2) *Work organization*. Creating an efficient team structure and role distribution. This may involve appointing coordinators responsible for specific project aspects and forming working groups.
3) *Documentation*. Maintaining detailed records of project progress, outcomes, and decisions. Documentation helps onboard new participants and ensures transparency.
4) *Task management*. Using project management tools (e.g., Trello, Asana, Jira) to track tasks and deadlines, enabling efficient task allocation and progress monitoring.
5) *Feedback*. Regularly gathering input from participants on workflows and results.
6) *Community engagement*. Open projects often rely on community input. Creating opportunities for stakeholder involvement and idea collection drives project development.
7) *Monitoring and evaluation*. Conducting regular assessments to identify progress and domains for improvement.

Effective coordination requires active management and attention to detail to achieve set goals.

Open-source development refers to the process of creating software with publicly accessible source code, which anyone can use, modify, and distribute. This model offers numerous advantages and unique characteristics [3].

Key aspects of open-source project development:

- *Open source code*. The program's source code is accessible to everyone, enabling users to study, modify,

and improve the software.

- *Community*. Development is often collaborative, involving contributors from around the world. The community actively participates in idea discussions, bug fixes, and providing support.
- *Licenses*. Open-source projects are governed by licenses that define how the code may be used. Popular licenses include GPL, MIT, Apache, and others, each with specific terms regulating use and distribution.
- *Flexibility and adaptability*. Users can tailor the project to their needs by adding new features or modifying existing ones.
- *Security*. Open code allows users to audit it for vulnerabilities and fix issues. However, this also means malicious actors can analyze the code to exploit weaknesses.
- *Documentation*. Successful open-source projects typically include comprehensive documentation, simplifying adoption and use. Documentation covers installation, configuration, and usage guidelines.
- *Examples of successful projects*. Many well-known projects are open-source, including the Linux operating system, Apache web server, Git version control system, Python programming language, and more.

General stages of software development:

- *Idea*. Everything begins with an idea, a problem to solve, or a feature to implement.
- *Planning*. Defining the project structure, timelines, and resource allocation.
- *Coding*. Writing code, including tests and documentation.
- *Testing*. Identifying and correcting flaws. Open-source projects often rely on external feedback to enhance quality.
- *Release*. Publishing the first version for users to adopt.
- *Feedback and evolution*. Post-release, the project evolves based on input from users and the community driving its development.

Automation of open project development is a crucial aspect that enhances code quality, accelerates development processes, and simplifies collaboration among project contributors. Below are key principles to guide automation in open projects:

- *Version control systems*. Enable efficient management of code changes and streamline collaborative work.
- *Test automation*. Rapidly identify errors, thereby improving code reliability.
- *Continuous integration and deployment (CI/CD)*. Automate building, testing, and deployment to expedite releasing changes to production.
- *Code documentation and comments*. Simplify onboarding for new contributors and ensure documentation stays current.
- *Static code analysis and formatting tools*. Enforce uniform code style and prevent common errors.
- *Dependency management and automated updates*. Mitigate risks from outdated libraries and enhance project security.
- *Containerization*. Simplify environment setup for developers and avoid dependency conflicts across projects.
- *Task management and feedback systems*. Foster community engagement and streamline collaboration for existing and new contributors.
- *Training and knowledge sharing*. Document best practices, create tutorials, and provide guides to accelerate contributor onboarding.

By adhering to these principles, development teams can significantly improve efficiency and the overall quality of open projects.

III. Unification of specifications for documentation in next-generation intelligent computer systems

To ensure both the system and developers uniformly understand the semantics of concepts used, it is necessary to specify not only the concepts themselves but also the associated fragments of knowledge bases and, consequently, entire sections of relevant subject domains [4]. Let us examine the subject domain of basic specifications.

A. Subject domain of basic specifications

The unification of concept specifications in knowledge bases is described within the *subject domain of basic specifications*. This domain must contain descriptions of all possible *classes of entities with unified basic specifications* and the *basic specifications* of various entities.

Subject domain of basic specifications
- $\in$ *subject domain*
- $\Leftarrow$ *specific subject domain**:
  - *Subject domain of semantic neighborhoods*
- $\ni$ *maximum class of research objects′*:
  - *basic specification*
- $\ni$ *class of research objects′*:
  - *class of entities with unified basic specifications*
  - *basic specification*
  - *basic specification of the basic specification concept*
- $\ni$ *research relation′*:
  - *basic specification*
  - *generalized basic specification*

Importantly, in this subject domain, *basic specification** is a binary directed non-role relation linking an

entity of the *class of entities with unified basic specifications* to its *basic specification*, indicating which part of the specification serves as the entity's foundational definition.

Meanwhile, *generalized basic specification** is a binary directed non-role relation connecting a specific *class of entities with basic specifications* to its *basic specification*, signifying that the class has a defined set of core properties for specifying its entities.

B. Class of entities with unified basic specifications

Within the subject domain of basic specifications, the concept of a *class of entities with unified basic specifications* is defined.

class of entities with unified basic specifications
:= [a class where all entities share a common set of properties essential for their basic description]
⇒ *note**:
[Some classes may include subclasses with additional basic specifications unique to those subclasses]

C. Basic specification

basic specification
⊂ *specification*
:= [a set of core properties shared by entities within a class]
:= [the minimal property set required to describe each entity in a class]
⇒ *note**:
[Every entity in a class must have its basic properties defined according to the class characteristics]
⇒ *note**:
[Basic specifications describe recommended minimum properties for class entities but are not strictly mandatory. They streamline knowledge base development and prevent redundant concepts.]

Notably, the *basic specification* class itself is also a *class of entities with unified basic specifications*. This allows formalizing the *basic specification of the basic specification concept*.

basic specification of basic specification concept
⇒ *note**:
[This reflects properties required in every basic specification]
⇐ *generalized basic specification**:
• *basic specification*
⇒ *generalized decomposition**:
⟨• *specification of generalized basic specification*
• *specification of generalized decomposition*
• *specification example*
⟩
∋ *example'*:
basic specification of a person
⇐ *basic specification**:
• *basic specification of a person*

Each *basic specification* is decomposed into specifications of individual properties, which must themselves adhere to strict rules. Below is an example formalizing the *specification of a generalized basic specification*.

specification of generalized basic specification
⇒ *generalized decomposition**:
{• *sign of the specified object*
• *class of entities with unified basic specifications*
• *arc linking the class to the specified object*
• *arc belonging to the generalized basic specification relation*
• *generalized basic specification*
⇒ *template**:
template for describing generalized basic specifications
⇒ *illustration**:
SCg-text. Template for describing generalized basic specifications
}

The template for generalized basic specifications is shown in Figure 1.

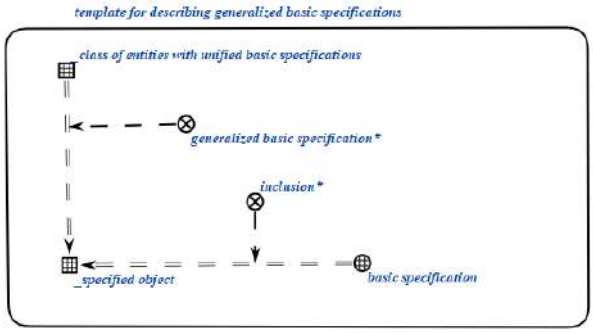

Figure 1. SCg-text. Template for describing a generalized basic specification

IV. Guide to developing and using semantically compatible intelligent systems based on OSTIS Technology

Let's look at it further a comprehensive guide to the development and application of semantically compatible intelligent systems using OSTIS Technology based on basic specifications. OSTIS Project is an open-source project designed to address the challenges of interoperability, modularity, and scalability in intelligent system

design. Step-by-step methodologies are provided for constructing ontologically structured knowledge bases and implementing agents for problem solving. The practical workflow is illustrated through the ostis-example-app, demonstrating installation, extension, and testing procedures. The guide emphasizes the advantages of OSTIS Project, including plug-and-play integration, platform independence, reflexivity, and support for parallel information processing, making it a robust foundation for next-generation intelligent systems.

V. OSTIS Project

A. About OSTIS Project

OSTIS Project [5], [6] is an open-source initiative focused on developing and promoting *Open Semantic Technology for Intelligent Systems (OSTIS)* [7]. The infrastructure of the *OSTIS Project* is hosted within the OSTIS-AI organization on GitHub [8]. It comprises multiple repositories, with the following being key components:

1) The *ostis-web-platform* repository [9] provides tools for installing and developing ostis-systems. It includes a knowledge base with top-level ontologies, semantic network interpreters, component manager, and web-oriented interface [10].
2) The *ostis-metasystem* repository [11], [12] contains comprehensive intelligent system designed to automate and manage all stages of the life cycle of ostis-systems [13].
3) The *ostis-example-app* repository [14] demonstrates the basic capabilities of ostis-systems, including knowledge base, problem solver, and web interface.

B. About OSTIS Technology

The *OSTIS Technology* enables the creation of semantically compatible intelligent systems that can seamlessly integrate various types of knowledge and problem-solving models [15], allowing them to adapt and solve new problems efficiently.

The *OSTIS Technology* has been applied in various domains, including fuzzy systems, data analysis, and knowledge management, demonstrating its versatility.

C. Why OSTIS

The *OSTIS Technology* is not just another AI project; it's an open-source technology designed to revolutionize how we build intelligent systems. The primary goal of the *OSTIS Technology* is to address the limitations of current intelligent systems, which are often monolithic, difficult to modify, and expensive to develop [16].

By providing a universal framework for representing information using *SC-code (Semantic Computer code)* [17], the *OSTIS Technology* enables the creation of modular, reusable components that can be easily combined across different systems [18].

D. Key advantages of OSTIS

The main advantages of the *OSTIS Technology* include:

1) *Plug-and-Play Integration*. Seamlessly add new problem-solving models or knowledge without complex overhead [18], [19].
2) *Universal Components*. Reusable components reduce development time and effort across different ostis-systems [19], [20].
3) *Reflexivity*. ostis-systems can analyze themselves, identify errors, and optimize performance – a hallmark of true intelligence [4], [21].
4) *Platform Independence*. ostis-systems can be implemented on various platforms, paving the way for semantic computers [10], [22].
5) *Parallel Processing*. Designed for efficient parallel information processing, especially beneficial for semantic computers [23].

E. Key features of SC-code

SC-code serves as the backbone of the *OSTIS Technology*. It is a universal language designed to represent knowledge, models, and interfaces uniformly [21]. The code is based on a minimal alphabet consisting of five elements that enable non-linear representation suitable for semantic associative computers [23].

The main features of *SC-code* include [21]:

1) *Universality*. Represents any information uniformly.
2) *Non-linearity*. Suitable for semantic associative computers.
3) *Basic Alphabet*. Comprises just five elements.
4) *Flexibility*. Can represent knowledge, models, and interfaces.

F. How to use OSTIS

To effectively use OSTIS, developers can start with the *ostis-example-app* [14]. Refer to the Getting started with OSTIS section for detailed instructions on installing and launching this example of an ostis-system.

VI. Getting started with OSTIS

A. About ostis-example-app

The *OSTIS Project* provides an example of a system that can be supplemented with new components to create a new system of any orientation. It's called *ostis-example-app*. Its sources can be found here [14].

The *ostis-example-app* serves as a practical entry point for understanding and utilizing *OSTIS Technology*. This application exemplifies the core components of an ostis-system, including its knowledge base, problem solver, and user interface.

B. Key features of ostis-example-app

The main features of *ostis-example-app* include:

1) *Knowledge base*. The knowledge base is structured using SC-code, enabling semantic representation that supports modularity and extensibility [24], [25]. It allows the integration of diverse knowledge domains without requiring significant modifications to the system architecture.
2) *Problem solver*. Utilizing a multi-agent approach, the problem solver integrates various models for addressing complex tasks [26]. This architecture ensures adaptability and scalability in solving new classes of problems.
3) *User interface*. The user interface is also described using SC-code, providing semantic compatibility with other system components [27]. This ensures seamless interaction between users and the intelligent system.

C. Step-by-step guide to using ostis-example-app

To utilize the capabilities of *ostis-example-app*, follow these steps:

1) Enshure prerequisites:
   - Install Git: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git [28].
   - Install Docker & Docker Compose: https://www.docker.com/get-started [29].

   Before proceeding with installation, ensure that system meets all prerequisites. Docker is recommended for its ability to provide a consistent environment across platforms.
2) Clone repository of ostis-example-app:

```
git clone https://github.com/ostis-apps/ostis-example-app.git
cd ostis-example-app
git checkout 0.10.0
git submodule update --init --recursive
```

   Cloning the repository gives you access to the source code and all necessary submodules required.
3) Build docker images of ostis-example-app:

```
docker compose build
```

   Building Docker images ensures that all dependencies are correctly configured within isolated containers.
4) Build knowledge base of ostis-example-app:

```
docker compose run --rm machine build
```

5) Start ostis-example-app:

```
docker compose up
# Access interface at http://localhost:8000/
# To stop system use `docker compose stop`
# Rebuild KB after changes in `.scs` or `.gwf` files
```

   Starting the system launches all services defined in the Docker Compose configuration file. Accessing the interface allows users to interact with the intelligent system.
6) To rebuild after changes in ‘.scs‘ or ‘.gwf‘, repeat step 4.

VII. Step-by-step guide to developing and using intelligent system within the OSTIS Technology

This section presents a step-by-step algorithm for developing an ostis-system, both by extending the *ostis-example-app* and by building a system from scratch.

A. Developing an ostis-system using ostis-example-app

Before proceeding to develop an ostis-system from scratch, it is often convenient to use the ready-made *ostis-example-app* as a starting point. This approach allows you to quickly become familiar with the structure of ostis-systems, experiment with knowledge base extensions, and implement new problem-solving agents without the need to design all components from the ground up. The following steps outline how to adapt and extend an ostis-system using *ostis-example-app*.

1) Clone template repository:
   - download the *ostis-example-app* repository, which provides a ready-to-use project structure with a knowledge base, problem solver, and web interface (see Getting started with OSTIS).
2) Modify knowledge base:
   - edit or extend SC-code sources in the `knowledge-base/` directory to add or adapt domain-specific concepts, relations, and others (see Step-by-step guides to developing and using knowledge base within the OSTIS Technology).
3) Extend problem solver:
   - add new agents or modify existing ones in the `problem-solver/` directory to implement the required logic (see Step-by-step guides to developing and using problem solver within the OSTIS Technology);
   - register and configure agents as needed.
4) Rebuild and launch the system:
   - rebuild the system (e.g., using Docker or CMake, as described in the *ostis-example-app* documentation Getting started with OSTIS);

- start the system and access the web interface to test and use extended ostis-system.

5) Test and iterate:
   - after any changes to the knowledge base or problem solver, rebuild the system to apply updates.

B. Developing an ostis-system from scratch

In cases where the requirements for the intelligent system are highly specific or when full control over the system architecture is needed, it may be preferable to develop an ostis-system from scratch. This approach involves designing the project structure, configuring dependencies, and developing all core components independently. The steps below describe the general workflow for building an ostis-system from the ground up.

1) Configure project:
   - organize directories for the knowledge base, problem solver, interface, and configuration files;
   - set up required dependencies (e.g., *sc-machine* [30]) using a package manager such as Conan;
   - configure the build system (e.g., CMake) for compiling agents and linking with necessary libraries (see Configuring project for new ostis-system).
2) Develop knowledge base:
   - design and formalize ontologies, concepts, and relations in SC-code, following *OSTIS Technology* methodological guidelines (see Step-by-step guides to developing and using knowledge base within the OSTIS Technology);
   - organize the knowledge base into sections and subject domains.
3) Implement problem solver:
   - develop agents that implement the logic for solving tasks relevant to required domain (see Step-by-step guides to developing and using problem solver within the OSTIS Technology);
   - ensure agents interact correctly with the knowledge base.
4) Develop user interface:
   - if needed, implement or configure the web interface for user interaction with the system.
5) Build and launch the system:
   - build the knowledge base and problem solver;
   - launch the system and verify its functionality.
6) Test and iterate:
   - test the system, add new features, and refine knowledge and agent logic as needed;

In both approaches, after every change to the knowledge base or agents, rebuild the system to apply updates [14].

C. Configuring project for new ostis-system

Step 1. Define project structure

Create a basic directory structure for an ostis-system project (listing 1).

```
ostis-system-example/
|-- knowledge-base    # SC-code sources (.scs, .gwf) defining ontologies, rules, and facts
|-- problem-solver    # C++ agents implementing problem-solving logic
|-- CMakeLists.txt    # Build configuration for agents and dependencies
|-- conanfile.txt     # Conan package manager specification for agent dependencies
```

Listing 1. Project directory structure

Step 2. Configure dependencies with Conan

Specify Conan package manager configuration [31] for required components (listing 2).

```
[requires]
sc-machine/0.10.0   # Specifies that the project requires the sc-machine library version 0.10.0
gtest/1.14.0        # Specifies that the project requires the gtest library version 1.14.0

[generators]
CMakeDeps           # Generator for creating dependency information files for CMake
CMakeToolchain      # Generator for creating a toolchain file for CMake

[layout]
cmake_layout        # Uses the standard project layout
```

Listing 2. Conan dependencies

Step 3. Define CMake build system

Define main CMake configuration file [32] for build setup (listing 3).

```
# Minimum required CMake version
cmake_minimum_required(VERSION 3.24
)
# Set C++17 standard
set(CMAKE_CXX_STANDARD 17)
# Project definition
project(my-ostis-system VERSION 0.10.0 LANGUAGES C CXX)
```

```
# Version policy configuration
cmake_policy(SET CMP0048 NEW)

# Output directories configuration
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY
    ${CMAKE_BINARY_DIR}/bin
)
set(CMAKE_LIBRARY_OUTPUT_DIRECTORY
    ${CMAKE_BINARY_DIR}/lib
)
set(SC_EXTENSIONS_DIRECTORY
    ${CMAKE_LIBRARY_OUTPUT_DIRECTORY}/extensions
)

# Looking for sc-machine package for implementing agents
find_package(sc-machine
    REQUIRED
)

# Looking for GTest package for testing agents
include(GoogleTest)
find_package(GTest
    REQUIRED
)

# Include problem solver subsystem
add_subdirectory(
    ${CMAKE_CURRENT_SOURCE_DIR}/problem-solver
)
```

Listing 3. CMake configuration

Step 5. Install basic tools for development environment

- ```
  # Ubuntu/Debian (GCC)
  sudo apt update

  sudo apt install --yes --no-install-recommends \
      curl \
      ccache \
      python3 \
      python3-pip \
      build-essential \
      ninja-build
  ```

```
# macOS
brew update && brew upgrade
brew install \
    curl \
    ccache \
    cmake \
    ninja
```

For Linux distributions that doe support apt, ensure following packages are installed:

- *curl*: A tool for transferring data with URLs;
- *ccache*: A compiler cache to speed up compilation processes;
- *python3* and *python3-pip*: Python 3 interpreter and package installer;
- *build-essential*: Includes a C++ compiler, necessary for building C++ components;
- *ninja-build*: An alternative build system designed to be faster than traditional ones.

Step 6. Install Conan package manager

- Install pipx following the instructions based on operating system: https://pipx.pypa.io/stable/installation/ [33].
- Check CMake version and if it less than 3.24 update it:

```
pipx install cmake
pipx ensurepath
```

- Install Conan and restart the terminal:

```
pipx install conan
pipx ensurepath
exec $SHELL
```

Step 7. Install sc-machine libraries

sc-machine [30] is the core software platform of the OSTIS Technology, designed to emulate semantic computer behavior by storing and processing knowledge in the form of semantic networks. At its foundation, *sc-machine* functions as a graph database management system that enables efficient storage, retrieval, and manipulation of knowledge graphs within a shared memory structure called *sc-memory*.

- Add the *ostis-ai* repository [34] for Conan packages:

```
conan remote add ostis-ai
    https://conan.ostis.net/
    artifactory/api/conan/
    ostis-ai-library
conan profile detect
```

- Install *sc-machine* libraries from ostis-ai repository using Conan:

```
# Fetch sc-machine libraries and builds missing dependencies
conan install . --build=missing
```
```

sc-machine libraries – the core components of the OSTIS Platform, used to develop C++ agents.

Step 8. Install sc-machine binaries and scp-machine [35] extensions

- Install *sc-machine* binaries and extensions. The installation process differs slightly between Linux and macOS:

```
# Fetch release archive with sc-machine binaries and extensions for Linux
curl -LO https://github.com/ostis-ai/sc-machine/releases/download/0.10.0/sc-machine-0.10.0-Linux.tar.gz
# Unpack fetched release archive
mkdir sc-machine && tar -xvzf sc-machine-0.10.0-Linux.tar.gz -C sc-machine --strip-components 1
# Remove unneccessary sources
rm -rf sc-machine-0.10.0-Linux.tar.gz && rm -rf sc-machine/include
```

```
# Fetch release archive with sc-machine binaries and extensions for macOS
curl -LO https://github.com/ostis-ai/sc-machine/releases/download/0.10.0/sc-machine-0.10.0-Darwin.tar.gz
# Unpack fetched release archive
mkdir sc-machine && tar -xvzf sc-machine-0.10.0-Darwin.tar.gz -C sc-machine --strip-components 1
# Remove unneccessary sources
rm -rf sc-machine-0.10.0-Darwin.tar.gz && rm -rf sc-machine/include
```

sc-machine binaries are pre-compiled executables that provide the runtime environment for the ostis-system: build knowledge base source and launch the ostis-system.

- Install *scp-machine* extensions. The installation process differs slightly between Linux and macOS:

```
# Fetch release archive with scp-machine extensions for Linux
curl -LO https://github.com/ostis-ai/scp-machine/releases/download/0.1.0/scp-machine-0.1.0-Linux.tar.gz
# Unpack fetched release archive
mkdir scp-machine && tar -xvzf scp-machine-0.1.0-Linux.tar.gz -C scp-machine --strip-components 1
# Remove unneccessary sources
rm -rf scp-machine-0.1.0-Linux.tar.gz && rm -rf scp-machine/include
```

```
# Fetch release archive with scp-machine extensions for macOS
curl -LO https://github.com/ostis-ai/scp-machine/releases/download/0.1.0/scp-machine-0.1.0-Darwin.tar.gz
# Unpack fetched release archive
mkdir scp-machine && tar -xvzf scp-machine-0.1.0-Darwin.tar.gz -C scp-machine --strip-components 1
# Remove unneccessary sources
rm -rf scp-machine-0.1.0-Darwin.tar.gz && rm -rf scp-machine/include
```

The following sections (Step-by-step guides to developing and using knowledge base within the OSTIS Technology and Step-by-step guides to developing and using problem solver within the OSTIS Technology) provide methodologies outlined for developing *knowledge bases* and *problem solvers* within *OSTIS Technology*.

VIII. Step-by-step guides to developing and using knowledge base within the OSTIS Technology

A. About knowledge base of ostis-systems

Developing and utilizing knowledge bases within the *OSTIS Technology* requires a structured, ontology-driven approach [15] to ensure semantic compatibility, modularity, and interoperability across ostis-systems. This section provides methodological guidelines for constructing and applying knowledge bases that adhere to the *OSTIS Standard* [4].

Key principles underpinning this process include:

- *Semantic formalization*. Knowledge is represented using SC-code, enabling machine-readable and human-understandable structures [18].
- *Hierarchical organization*. Knowledge bases are divided into modular sections, each corresponding to a specific subject domain, with inheritance mechanisms ensuring consistency across layers [21].
- *Ontological integrity*. Each section must comprehensively define its concepts, relations, and rules, avoiding redundancy and ensuring logical coherence [36].

By following the next steps, developers can ensure their systems align with the *OSTIS Standard's* requirements for scalability, maintainability, and interoperability [4].

B. Guide to formalizing section of knowledge base using SCs-code

Step 1. Specify didactic structure of section

A *section of a knowledge base* is a modular, ontologically complete, and semantically formalized component that represents all knowledge relevant to a particular subject domain, supporting hierarchical organization and integration within the overall knowledge base [25].

Create a directory for the section within the `knowledge-base/` hierarchy (listing 4). Inside this directory, create .scs source (e.g., `section_subject_domain_of_sets.scs`) and specify didactic structure of the section: identifiers, definitions, explanations, notes, and quotes related to the section (listing 5).

```
knowledge-base/
+|-- section_subject_domain_of_sets/
+|   |-- section_subject_domain_of_sets.scs
```

Listing 4. Section directory

`section_subject_domain_of_sets.scs`

```
section_subject_domain_of_sets
=> nrel_main_idtf:
    [Section. Subject domain of sets]
    (* <- lang_en;; *);
=> nrel_idtf:
        [Section of knowledge base, describing sets]
        (* <- lang_en;; *);
=> nrel_idtf:
    [Formalized description of set-theoretic concepts]
    (* <- lang_en;; *);
=> nrel_idtf:
    [Mathematical foundation for collections and their operations]
    (* <- lang_en;; *);
<- sc_node_structure;
<- section;
=> nrel_definition: [
    Set theory serves as the cornerstone of modern mathematics, providing a rigorous framework for defining collections of distinct objects. Fundamental operations such as union, intersection, and complement enable systematic manipulation of these collections. This formalism underpins advanced mathematical disciplines and finds practical utility in computer science (data structures), logic, and data analysis.
];
=> nrel_explanation: [
    Set theory is fundamental in mathematics as it provides a way to describe collections of objects. It includes operations like union, intersection, and complement, which allow us to combine and manipulate sets. Understanding sets is crucial for advanced mathematical concepts and has practical applications in fields like computer science and data analysis.
];
=> nrel_note: [
    An important aspect of sets is that they do not consider the order of elements or duplicates. This means {a, b} is the same as {b, a} or {a, b, b}. Additionally, sets
```

```
        have been a subject of
        historical development,
        notably through Georg
        Cantor's work in the late
         19th century.
];
=> nrel_quote: [
    The essence of mathematics
        lies in its freedom (
        Georg Cantor, 1883)
];
```

Listing 5. Section. Subject domain of sets

Step 2. Specify decomposition of section

Identify the sub-sections into which the described section can be decomposed: add a decomposition statement in `section_subject_domain_of_sets.scs` (listing 6), and create directories, .scs sources for each sub-section (listing 7), and specify basic information for sub-sections (listings 8 and 9).

section_subject_domain_of_sets.scs

```
]
section_subject_domain_of_sets
// information, specified above
    ...
+ => nrel_section_decomposition:
     <
+    section_subject_domain_of
    _basic_concepts_of_sets,
+    section_subject_domain_of
    _set_operations
+ >;;
```

Listing 6. Decomposition of section of subject domain of sets

```
 knowledge-base/
 |-- section_subject_domain_of
     _sets/
+|    |--
    section_subject_domain_of
    _basic_concepts_of_sets/
+|    |    |--
    section_subject_domain_of
    _basic_concepts_of_sets.scs
+|    |--
    section_subject_domain_of
    _set_operations/
+|    |    |--
    section_subject_domain_of
    _set_operations.scs
 |    |--
      section_subject_domain_of
      _sets.scs
```

Listing 7. Subsection directories

`section_subject_domain_of_basic_concepts_of_sets.scs`

```
section_subject_domain_of_basic
    _concepts_of_sets
=> nrel_main_idtf:
     [Section. Subject domain of
        basic concepts of sets]
     (* <- lang_en;; *);
=> nrel_idtf:
     [Basic concepts of set
        theory]
     (* <- lang_en;; *);
<- sc_node_structure;
<- section;;
```

Listing 8. Section. Subject domain of basic concepts of sets

`section_subject_domain_of_set_operations.scs`

```
section_subject_domain_of_set
    _operations
=> nrel_main_idtf:
     [Section. Subject domain of
        set operation]
     (* <- lang_en;; *);
=> nrel_idtf:
     [Set operations]
     (* <- lang_en;; *);
<- sc_node_structure;
<- section;;
```

Listing 9. Section. Subject domain of set operations

Step 3. Specify subject domain for given section

A *subject domain* is a comprehensive structure that organizes all relevant objects, their classifications, the relations between them, and any additional components necessary to fully describe the domain of investigation []. *Subject domain* is a structure that includes:

- the main research (described) objects;
- classes of research objects;
- links, the components of which are the research objects;
- classes of the above-mentioned links (i.e. relations);
- classes of objects that are neither the research objects nor the above-mentioned links, but are components of these links.

Define the subject domain describing all investigated entities within the section: create a file named `subject_domain_of_sets.scs` in the section directory (listing 10) and specify didactic structure of subject domain (listing 11).

```
 knowledge-base/
 |-- section_subject_domain_of
     _sets/
```

```
|   |--
    section_subject_domain_of
    _basic_concepts_of_sets/
|   |   |--
    section_subject_domain_of
    _basic_concepts_of_sets.scs
|   |--
    section_subject_domain_of
    _set_operations/
|   |   |--
    section_subject_domain_of
    _set_operations.scs
|   |--
    section_subject_domain_of
    _sets.scs
+|  |-- subject_domain_of_sets.
    scs
```

Listing 10. Location of subject domain

subject_domain_of_sets.scs

```
section_subject_domain_of_sets =
[*
subject_domain_of_sets
=> nrel_main_idtf:
    [Subject domain of sets]
    (* <- lang_en;; *);
=> nrel_idtf:
    [Subject domain of set
       theory]
    (* <- lang_en;; *);
    [Subject domain, in which
       research objects are sets
       ]
    (* <- lang_en;; *);
<- sc_node_structure;
<- subject_domain;
*];;
```

Listing 11. Subject domain of sets

Step 4. Specify child subject domains for given subject domain

List child subject domains that are hierarchically subordinate to the current domain. Add child subject domains in subject_domain_of_sets.scs (listing 12).

subject_domain_of_sets.scs

```
section_subject_domain_of_sets =
[*
subject_domain_of_sets
// information, specified above
    ...
<= nrel_child_subject_domain:
    subject_domain_of_discrete
       _mathematics;
=> nrel_child_subject_domain:
    subject_domain_of_basic
       _concepts_of_sets;
    subject_domain_of_set
       _operations;
*];;
```

Listing 12. Child subject domains

Step 5. Identify classes of objects for given subject domain

List the classes of objects researched within the corresponding subject domain (listing 13).

Distinguish between maximal and non-maximal classes of objects:

- *maximal class* is a class for which there is no other class in the subject domain that is a superset.
- *non-maximal class* is a class for which there exists another class in the subject domain that is a superset.

subject_domain_of_sets.scs

```
section_subject_domain_of_sets =
[*
subject_domain_of_sets
// information, specified above
    ...
+ -> rrel_maximal_research
    _object_class:
+    concept_set;
+ -> rrel_not_maximal_research
    _object_class:
+    concept_finite_set;
+    concept_infinite_set;
+    concept_countable_set;
+    concept_uncountable_set;
+    concept_cantor_set;
+    concept_multiset;
+    concept_fuzzy_set;
+    concept_crisp_set;
+    concept_set_of_sets;
+    concept_non_reflexive_set;
+    concept_reflexive_set;
+    concept_formed_set;
+    concept_unformed_set;
+    concept_empty_set;
+    concept_singletone;
+    concept_pair_set;
+    concept_cantor_pair;
+    concept_triple_set;
+    concept_oriented_set;
+    concept_cartesian_product;
+    concept_boolean;
*];;
```

Listing 13. Classes of objects in subject domain of sets

Step 6. Identify research relations for given subject domain

Determine the relations being explored in the section (listing 14).

subject_domain_of_sets.scs

```
section_subject_domain_of_sets =
[*
subject_domain_of_sets
// information, specified above
    ...
-> rrel_not_maximal_research
    _object_class:
     // other research classes of
         objects
         concept_boolean;;
+ -> rrel_research_relation:
+    nrel_membership;
+    rrel_example;
+    nrel_inclusion;
+    nrel_strict_inclusion;
+    nrel_set_power;
+    nrel_equinumerous;;
*];;
```

Listing 14. Research relations in subject domain of sets

Step 6. Integrate into ostis-system

Knowledge base of ostis-system needs to be built before launching the system or after making changes. To load the formalization result into the ostis-system, run the command shown in listing 15 in terminal.

```
./sc-machine/bin/sc-builder -i
    knowledge-base -o kb.bin --
    clear
```

Listing 15. Build knowledge base

This command builds the knowledge base from the .scs and .gwf sources in the `knowledge-base` directory, creating the `kb.bin` file. The `-clear` flag clears the knowledge base before building.

C. Guide to formalizing concepts using SCs-code

In the context of the *OSTIS Standard*, an object is usually defined as either a concept, i.e. an abstract entity that combines other *abstract* or *concrete entities*, or an instance of a concept, i.e. a concrete entity.

Concepts can be absolute or relative. *Absolute concepts* denote the same attributes of some group of concepts or entities, *relative concepts* — connections and relations between other concepts or entities.

Formalizing absolute concepts (classes)

To formalize absolute concepts, which are classes or sets of objects with common properties.

Step 1. Specify didactic structure of class

Create .scs source (e.g., `concept_set.scs`) (listing 16) and specify sc-identifiers for class in various external languages to ensure cross-referencing and integration with other systems (listing 17).

```
knowledge-base/
|-- section_subject_domain_of
    _sets/
|   |--
    section_subject_domain_of
    _basic_concepts_of_sets/
|   |   |--
    section_subject_domain_of
    _basic_concepts_of_sets.scs
|   |--
    section_subject_domain_of
    _set_operations/
|   |   |--
    section_subject_domain_of
    _set_operations.scs
|   |-- concepts/
+|  |   |-- concept_set.scs
|   |--
    section_subject_domain_of
    _sets.scs
|   |-- subject_domain_of_sets.
    scs
```

Listing 16. Location of concept_set source

concept_set.scs

```
concept_set
=> nrel_main_idtf:
    [set]
    (* <- lang_en;; *);
=> nrel_idtf:
    [set of signs]
    (* <- lang_en;; *);
=> nrel_definition:
    [A collection of distinct
    objects where:
    ∀x, y ∈ S : x ≠ y ⇒ count(x) = 1
    and order is irrelevant]
    (* <- lang_en;; *);
    [A = {x | P(x)} where P is a
    membership property]
    (* <- lang_en;; *);
=> nrel_example:
    [ℕ = {1, 2, 3, ...} - set of
    natural numbers]
    (* <- lang_en;; *);
=> nrel_key_properties:
    [Unordered: {a, b} = {b, a}]
    (* <- lang_en;; *);
    [Unique elements:
    {a, a, b} ≡ {a, b}]
    (* <- lang_en;; *);
    [Extensionality:
    A = B ⟺ ∀x(x ∈ A ↔ x ∈ B)]
    (* <- lang_en;; *);
```

```
<- sc_node_class;
<- concept;
```

Listing 17. Didactic structure of concept_set

Step 2. Specify theoretical set relations

Describe connections with other sc-elements to establish relationships between concepts (listing 18).

concept_set.scs

```
concept_set
// information, specified above
    ...
+ => nrel_subdivision: {
+    concept_finite_set;
+    concept_infinite_set
+ };
+ => nrel_subdivision: {
+    concept_cantor_set;
+    concept_multiset
+ };
+ => nrel_subdivision: {
+    concept_crisp_set;
+    concept_fuzzy_set
+ };
+ => nrel_subdivision: {
+    concept_reflexive_set;
+    concept_non_reflexive_set
+ };
+ <= nrel_subdividing: {
+    concept_oriented_set;
+    concept_non_oriented_set
+ };
+ => nrel_strict_inclusion:
+    concept_empty_set;
+    concept_singletone;
+    concept_pair_set;
+    concept_triple_set;
```

Listing 18. Theoretical set relations of concept_set

Step 3. Specify instances

List instances of the described concept to provide concrete examples (listing 19).

concept_set.scs

```
concept_set
// information, specified above
    ...
+ -> rrel_example:
+    set_of_natural_numbers;
+    set_of_atoms_in_the_universe
    ;
+    set_of_all_continuous
    _functions;
```

Listing 19. Examples of instances of concept_set

Formalizing relative concepts (relations)

To formalize relative concepts, which are relations between objects.

Step 1. Specify didactic structure of relation

Create .scs source (e.g., nrel_inclusion.scs) (listing 20) and specify sc-identifiers in various external languages to ensure cross-referencing and integration with other systems (listing 21).

```
knowledge-base/
|-- section_subject_domain_of
    _sets/
|    |--
    section_subject_domain_of
    _basic_concepts_of_sets/
|    |    |--
    section_subject_domain_of
    _basic_concepts_of_sets.scs
|    |--
    section_subject_domain_of
    _set_operations/
|    |    |--
    section_subject_domain_of
    _set_operations.scs
|    |-- concepts/
|    |    |-- concept_set.scs
|    |-- relations/
+|   |    |-- nrel_inclusion.scs
|    |--
    section_subject_domain_of
    _sets.scs
|    |-- subject_domain_of_sets.
    scs
```

Listing 20. Location of nrel_inclusion source

nrel_inclusion.scs

```
nrel_inclusion
=> nrel_main_idtf:
    [subset]
    (* <- lang_en;; *);
=> nrel_idtf:
    [subset relation]
    (* <- lang_en;; *);
=> nrel_definition:
    [Binary relation where
    A ⊆ B ⟺ ∀x(x ∈ A → x ∈ B)]
    (* <- lang_en;; *);
=> nrel_example:
    [{1,2} ⊆ {1,2,3}]
    (* <- lang_en;; *);
<- concept_binary_relation;
<- concept_non_role_relation;
```

Listing 21. Didactic structure of nrel_inclusion

Step 2. Specify theoretical set relations

Describe connections with other sc-elements to establish relationships between relations (listing 22).

nrel_inclusion.scs

```
nrel_inclusion
// information, specified above
    ...
+ => nrel_inclusion:
+    nrel_strict_inclusion (*
+      => nrel_comment:
+         [Hierarchy: A ⊂ B ⇒ A ⊆ B
+         but A ⊆ B ⇏ A ⊂ B];
+         [Strict inclusion
    requires: ∃x ∈ B : x ∉ A];;
+ *);
```

Listing 22. Theoretical set relations of nrel_inclusion

Step 3. Specify relation domains

Specify the domains of the relation to define the sets of objects involved (listing 23).

nrel_inclusion.scs

```
nrel_inclusion
// information, specified above
    ...
+ => nrel_first_domain:
    concept_set (*
+      => nrel_comment:
+         [Left operand must be a
    set: ∀A ∈ Set];;
+ *);
+ => nrel_second_domain:
    concept_set (*
+      => nrel_comment:
+         [Right operand must be a
     set: ∀B ∈ Set];;
+ *);
```

Listing 23. Domains of nrel_inclusion

Step 4. Specify domain of definition

Define the domain of definition of the relation to specify where it applies (listing 24).

nrel_inclusion.scs

```
nrel_inclusion
// information, specified above
    ...
+ => nrel_definitional_domain:
    concept_set (*
+      => nrel_comment:
+         [It is a union of the
    first and second domains.];;
+ *);
```

Listing 24. Definitional domain of nrel_inclusion

Step 5. Specify properties of relation

Describe the properties of the relation to clarify its nature (listing 25).

nrel_inclusion.scs

```
nrel_inclusion
// information, specified above
    ...
+ <- concept_reflexive_relation
    (*
+    => nrel_comment:
+    [Inherits reflexivity:
    ∀S(S ⊆ S)];;
+ *);
+ <- concept_antisymmetric
    _relation (*
+    => nrel_comment:
+    [Inherits antisymmetry:
    (A ⊆ B ∧ B ⊆ A) ⇒ A = B];;
+ *);
+ <- concept_transitive_relation
     (*
+    => nrel_comment:
+    [Inherits transitivity:
    (A ⊆ B ∧ B ⊆ C) ⇒ A ⊆ C
    enabling hierarchy chains];;
+ *);
```

Listing 25. Properties of nrel_inclusion

Step 6. Specify examples of connectivity of relation

Describe a examples of a connectivity of the specified relation to illustrate its use (listing 26).

nrel_inclusion.scs

```
nrel_inclusion
// information, specified above
    ...
+ -> (concept_countable_set =>
    concept_finite_set) (*
+      => nrel_comment:
+         [Hierarchy:
    FinSet ⊆ CountableSet ⊆ Set];;
+ *);
+ -> (concept_set => concept_set
    );
+ -> (concept_finite_set =>
    concept_empty_set) (*
+      => nrel_comment:
+         [∅ ⊆ S ∀S ∈ Set];;
+ *);;
```

Listing 26. Examples of connectivities of nrel_inclusion

IX. Step-by-step guides to developing and using problem solver within the OSTIS Technology

A. About problem solver of ostis-system

While the knowledge base provides the semantic foundation through SC-code ontologies, the problem solver implements actionable logic via sc-agents that interact dynamically with this knowledge.

B. Guide to developing and using platform-dependent agent of problem solver in C++ within the OSTIS Technology

General algorithm

All agents in C++ represent some classes in C++. To implement an agent in C++, the following common steps are performed:

1) Write input (initial) construction and output (result) construction of the future agent in SC-code.
2) Create folder with source and header files for sc-agent implementation.
3) Write `CMakeLists.txt` file. CMake is used to build projects in C++.
4) In the header file, define a class in C++ for the agent and specify at least the class of actions that the agent performs and its program. In such class, primary initiation condition, initiation condition, and result condition can also be specified.
5) In the source file, implement all declared methods of the agent's class. Additional methods can also be implemented and used in an agent program. All C++ and OOP tools can be used as much as possible.
6) Create file and implement class for keynodes used by the implemented agent.
7) Implement class for module for subscribing the implemented agent.
8) Write tests for the implemented agent.

When research set theory applications, one common task involves computing the power or cardinality of a set. Consider an example of implementing an agent to count power of a given set. This agent would be designed to determine the number of elements within the set, providing insights into its size and structure.

Step 1. Specify inputs and outputs of agent

The initial construction of the agent might look like this (listing 27 and figure 2):

```
..action
<- action_calculate_set_power;
<- action_initiated;
-> rrel_1: ..some_set;;

..some_set
-> ..element_1;
-> ..element_2;
-> ..element_3;;
```

Listing 27. Initial sc-construction for agent in SCs-code

The result sc-construction of the agent might look like this (listing 28 and figure 3):

```
..some_action
=> nrel_result: [*
  ..some_set => nrel_set_power:
      [3];;
*];;
```

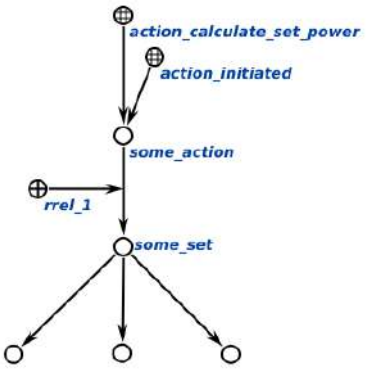

Figure 2. Initial sc-construction for agent in SCg-code

Listing 28. Result sc-construction for agent in SCs-code

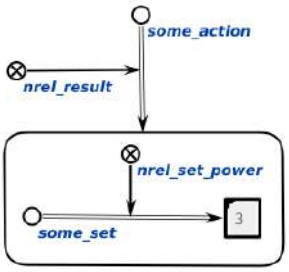

Figure 3. Result sc-construction for agent in SCg-code

In addition to agents that initiate actions themselves and then perform these actions, there is a need to implement agents that perform actions initiated by other agents. For this class of agents, it is much easier to generate an initial initiation construction in the knowledge base.

Step 2. Create folder with source and header files for agent

Create folder for agent implementation (listings 29, 30 and 31).

```
problem-solver/
|-- set-agents-module/
|-- CMakeLists.txt
```

Listing 29. Problem solver directory structure

```
set-agents-module/
|-- agent/
|   |-- sc_agent_calculate_set
    _power.hpp
|   |-- sc_agent_calculate_set
    _power.cpp
|-- CMakeLists.txt
```

Listing 30. Agent module directory structure

CMakeLists.txt

```
add_subdirectory(${
    CMAKE_CURRENT_SOURCE_DIR}/set
    -agents-module)
```

Listing 31. CMake build configuration for problem solver

Step 3. Write CMakeLists.txt file

The `CMakeLists.txt` file should describe a process of building the agent code (listing 32). It should contain various instructions and parameters needed to compile and link agent source code to *sc-machine* libraries.

CMakeLists.txt

```
# Find all files with the
    extensions .cpp and .hpp in
    the current directory and the
     agent subdirectory.
file(GLOB SOURCES
    CONFIGURE_DEPENDS
     "*.cpp" "*.hpp"
     "agent/*.cpp" "agent/*.hpp"
)

# Create a dynamic library
    called set-agents.
add_library(set-agents SHARED ${
    SOURCES})
# Add a public dependency on the
     sc-memory library from the
    sc-machine package.
target_link_libraries(set-agents
     LINK_PUBLIC sc-machine::sc-
    memory)
# Add the current source
    directory to the list of
    paths for header file lookup.
target_include_directories(set-
    agents PRIVATE ${
    CMAKE_CURRENT_SOURCE_DIR})
# Set properties for the set-
    agents target.
set_target_properties(set-agents
     # Specify the output
        directory for the
        compiled library.
    PROPERTIES
         LIBRARY_OUTPUT_DIRECTORY
    ${SC_EXTENSIONS_DIRECTORY}
)
```

Listing 32. CMake build configuration for agent module

`LIBRARY_OUTPUT_DIRECTORY` property should be set only for libraries that represent modules with agents.

`CMAKE_OUTPUT_LIBRARY_DIRECTORY` can be set instead of setting `LIBRARY_OUTPUT_DIRECTORY` property for each extension.

`SC_EXTENSIONS_DIRECTORY` variable should have path to the directory with extensions for the sc-machine. After building the module with the agent, this directory path should be specified via `-extensions` when starting the *sc-machine* to load the implemented module with the agent.

Step 4. Define an agent class in C++

Define a class in C++ for the agent and specifies the class of actions that the agent performs and its program (listing 33).

sc_agent_calculate_set_power.hpp

```
#pragma once

#include <sc-memory/sc_agent.hpp
    >

class ScAgentCalculateSetPower :
     public
    ScActionInitiatedAgent
{
public:
  ScAddr GetActionClass() const
      override;

  ScResult DoProgram(ScAction &
      action) override;
};
```

Listing 33. Agent class header definition

An agent's class to be implemented must comply with the following requirements:

- It must inherit one of the common classes for implementing agents:
  - `template <class TScEvent> class ScAgent`,
  - or `class ScActionInitiatedAgent`.

  The base class `ScAgent` contains API to implement agents that react to any sc-events. The base class `ScActionInitiatedAgent` inherits base class `ScAgent` and provides API to implement agents that react to sc-events of initiating sc-action.
- It must override at least methods `ScAddr GetAction() const` and `ScResult DoProgram(ScActionInitiatedEvent const & event, ScAction & action)`.
- Override methods must be public. Otherwise, the code cannot be built because the *sc-machine* won't be able to call methods on the agent class.
- Other methods can be implemented in the agent's class.

To learn more about opportunities and restrictions for implementing agents, see [37].

Step 5. Implement all declared methods

Implement all declared methods of the agent's class (listing 34).

sc_agent_calculate_set_power.cpp

```
#include "sc_agent_calculate_set_power.hpp"

#include <sc-memory/sc_memory_headers.hpp>

ScAddr ScAgentCalculateSetPower::GetActionClass() const
{
  return m_context.SearchElementBySystemIdentifier("action_calculate_set_power");
  // Must make sure that this class is in the knowledge base.
}
// Must specify valid action class. In other case, the agent cannot be
// subscribed to sc-event.

ScResult ScAgentCalculateSetPower::DoProgram(ScAction & action)
{
  // 'ScAction' class encapsulates information about sc-action.
  // The provided action is action that the given agent performs right now.
  // It belongs to class action_calculate_set_power'.
  // Actions are copyable and movable. ScAction is inherited from ScAddr.

  auto const & [setAddr] = action.GetArguments<1>();
  // This method finds construction 'action -> rrel_1: setAddr'.
  // Here the 1 is number of arguments which action must have. In step 1, an action should have a set as its the first and only one argument was specified. But the one who calls this agent may not specify argument for the action. So need to check that the action has argument.
  if (!m_context.IsElement(setAddr))
  {
    m_logger.Error("Action does not have argument.");
    // output: "ScAgentCalculateSetPower: Action does not have argument."
    return action.FinishWithError();
  }
  // There may be a situation where someone is trying to specify a number of arguments more than needed. This can also be checked by specifying, for example, number 2 instead of number 1. But it's not always necessary to do this.

  // To calculate power of the set, all accessory constant positive permanents arcs from the set can be traversed and count number of these arcs. But, in any problem, the presence of NON-factors must be considered, but this is omitted here.
  uint32_t setPower = 0;
  ScIterator3Ptr const it3 = m_context.CreateIterator3(
    setAddr,
    ScType::ConstPermPosArc,
    ScType::ConstNode
  );
  while (it3->Next())
    ++setPower;

  ScAddr const & setPowerAddr = m_context.GenerateLink(ScType::ConstNodeLink);
  m_context.SetLinkContent(setPowerAddr, setPower);
  ScAddr const & arcCommonAddr
    = m_context.GenerateConnector(ScType::ConstCommonArc, setAddr, setPowerAddr);
  ScAddr const & nrelSetPowerAddr
```

```
    = m_context.
        SearchElementBySystem
        Identifier("nrel_set_power
        ");
  // Must make sure that this non
     -role relation is in the
     knowledge base.
  ScAddr const &
     membershipArcAddr =
     m_context.GenerateConnector(
    ScType::ConstPermPosArc,
        nrelSetPowerAddr,
        arcCommonAddr);

  action.FormResult(
    setAddr, arcCommonAddr,
        setPowerAddr,
        membershipArcAddr,
        nrelSetPowerAddr);
  m_logger.Debug("Set power was
     counted: ", setPower, ".");
  // At the end of the agent's
     program, one of three
     methods ('FinishSuccessfully
     ', 'FinishUnsuccessfully', '
     FinishWithError') must be
     called to indicate that the
     agent's performing of action
      is complete:
  // - Method 'FinishSuccessfully
     ' indicates that action was
     performed by agent
     successfully (sets class '
     action_finished_successfully
     '). It means that the agent
     solved specified problem.
  // - Method '
     FinishUnsuccessfully'
     indicates that action was
     performed by agent
     unsuccessfully (sets class '
     action_finished
     _unsuccessfully'). It means
     that the agent didn't solve
     specified problem.
  // - Method 'FinishWithError'
     indicates that action was
     performed by agent with
     error (sets class '
     action_finished_with_error')
     . It means that some
     incorrect situation was
     occurred in knowledge base.
  // All these methods return
     objects of 'ScResult' class.
      An object of ScResult
     cannot be generated via
     constructor, because it is
     private.
    return action.
       FinishSuccessfully();
}
```

Listing 34. Agent class implementation

Step 6. Define keynodes

For each agent, key sc-elements that the agent uses during the execution of its program can be specified. These key sc-elements are sc-elements that the agent does not generate, but uses in the process of searching for or generating connections between entities in knowledge base. Key sc-elements are named keynodes. These keynodes can be found by its system identifiers (method `SearchElementBySystemIdentifier`) if they have such identifiers.

`ScKeynode` class can be used to define keynodes as static objects and use them in agents. `ScKeynodes` class is base class for all classes with keynodes. It contains core keynodes, that can be used in each agent. See [38] to learn more about keynodes.

Define keynodes class for implemented agent (listings 35, 36 and 37).

```
 set-agents-module/
 |-- agent/
 |   |-- sc_agent_calculate_set
    _power.hpp
 |   |-- sc_agent_calculate_set
    _power.cpp
+|-- keynodes/
+|   |-- sc_set_keynodes.hpp
 |-- CMakeLists.txt
```

Listing 35. Keynodes source location

CMakeLists.txt

```
file(GLOB SOURCES
   CONFIGURE_DEPENDS
    "*.cpp" "*.hpp"
    "agent/*.cpp" "agent/*.hpp"
+   "keynodes/*.hpp"
)
)
```

Listing 36. CMake build configuration update

sc_set_keynodes.hpp

```
#include <sc-memory/sc_keynodes.
   hpp>

// This class unites keynodes
   that used by agents of one
   module (with one sense).
```

```
4  class ScSetKeynodes : public
       ScKeynodes
5  {
6  public:
7    static inline ScKeynode const
         action_calculate_set_power{
8      "action_calculate_set_power"
           , ScType::ConstNodeClass
           };
9    static inline ScKeynode const
         nrel_set_power{
10     "nrel_set_power", ScType::
           ConstNodeNonRole};
11   // Here the first argument in
         constructor is system
         identifier of sc-keynode
         and the second argument is
         sc-type of this sc-keynode.
          If there is no sc-keynode
         with such system identifier
          in  knowledge base, then
         the one with specified sc-
         type will be generated.
12   // Type of sc-keynode may not
         be specified here, be
         default it is 'ScType::
         ConstNode'. But ensure that
          the code will use this
         keynode with type 'ScType::
         ConstNode' correctly.
13 };
```

Listing 37. Keynodes class definition

sc-keynode with empty system identifier cannot be specified. It can be invalid.

All keynodes must be static objects. Keynodes can be defined as static objects everywhere (not only in classes).

Inject using keynodes in agent implementation (listing 38).

sc_agent_calculate_set_power.cpp

```
1  #include "
       sc_agent_calculate_set_power.
       hpp"
2
3  #include <sc-memory/
       sc_memory_headers.hpp>
4
5  + #include "keynodes/
       sc_set_keynodes.hpp"
6
7  ScAddr ScAgentCalculateSetPower
       ::GetActionClass() const
8  {
9  -  return m_context.
       SearchElementBySystem
       Identifier("
       action_calculate_set_power");
10 +  return ScKeynodes::
       action_calculate_set_power;
11 }
12
13 ScResult
       ScAgentCalculateSetPower::
       DoProgram(ScAction & action)
14 {
15 - ScAddr const &
       nrelSetPowerAddr
16 -    = m_context.
       SearchElementBySystem
       Identifier("nrel_set_power");
17 - ScAddr const &
       membershipArcAddr = m_context
       .GenerateConnector(
18     ScType::ConstPermPosArc,
           nrelSetPowerAddr,
           arcCommonAddr);
19 + ScAddr const &
       membershipArcAddr = m_context
       .GenerateConnector(
20     ScType::ConstPermPosArc,
           ScKeynodes::
           nrel_set_power,
           arcCommonAddr);
21
22 - action.FormResult(
23 -    setAddr, arcCommonAddr,
       setPowerAddr,
       membershipArcAddr,
       nrelSetPowerAddr);
24 + action.FormResult(
25 +    setAddr, arcCommonAddr,
       setPowerAddr,
       membershipArcAddr, ScKeynodes
       ::nrel_set_power);
26   m_logger.Debug("Set␣power␣was␣
         counted:␣", setPower, ".");
27   return action.
         FinishSuccessfully();
28 }
```

Listing 38. Agent class implementation

Step 7. Implement module class for agent

Someone should subscribe the agent to event. It can be other agent, or any code at all. A class that allows subscribing agents can be implemented. This class is named sc-module. Each sc-module should subscribe agents with common sense.

Implement module and subscribe implemented agent to event (listings 39, 40 and 41).

```
1  set-agents-module/
```

```
 |-- agent/
 |   |-- sc_agent_calculate_set_power.hpp
 |   |-- sc_agent_calculate_set_power.cpp
 |-- keynodes/
 |   |-- sc_set_keynodes.hpp
+|-- sc_set_module.hpp
+|-- sc_set_module.cpp
 |-- CMakeLists.txt
```

Listing 39. Module sources location

sc_set_module.hpp

```cpp
#pragma once

#include <sc-memory/sc_module.hpp>

class ScSetModule : public ScModule
{
  // Here class is empty. No need to implement any methods.
  // 'ScModule' class contains all necessary API to subscribe agents as separate sc-module.
};
```

Listing 40. Module class definition

sc_set_module.cpp

```cpp
#include "sc_set_module.hpp"

#include "agent/sc_agent_calculate_set_power.hpp"

SC_MODULE_REGISTER(ScSetModule)
  ->Agent<ScAgentCalculateSetPower>();
  // This method pointers to module that agent class 'ScAgentCalculateSetPower' should be subscribed to sc-event of adding outgoing sc-arc from sc-element 'action_initiated'. It is default parameter in this method if an agent class inherited from 'ScActionInitiatedAgent' is subscribed.

// This way of subscribing agents makes it easier to write code.
// There is no need to think about unsubscribing agents after the system shutdown - the module will do it all by itself.
```

Listing 41. Module class implementation

If something else in the module besides agents needs to be initialized, methods `Initialize(ScMemoryContext * context) override;` and `Shutdown(ScMemoryContext * context) override;` can be overriden.

All modules functionality can be found in the [39].

Step 8. Write tests

To make sure how the agent works, it is best to generate tests and cover in them all possible cases that the agent has to handle. For this, create a separate file with test cases and implement them. A good code is a code covered by tests.

Write tests for implemented agent (listing 42, 43 and 44).

```
 set-agents-module/
 |-- agent/
 |   |-- sc_agent_calculate_set_power.hpp
 |   |-- sc_agent_calculate_set_power.cpp
 |-- keynodes/
 |   |-- sc_set_keynodes.hpp
+|-- tests/
+|   |-- test_sc_agent_calculate_set_power.cpp
 |-- sc_set_module.hpp
 |-- sc_set_module.cpp
 |-- CMakeLists.txt
```

Listing 42. Agent tests location

CMakeLists.txt

```
# code, specified above...

# Collect all files with the .cpp extension from the tests directory.
file(GLOB TEST_SOURCES CONFIGURE_DEPENDS
    "tests/*.cpp"
)

# Create an executable file for tests.
```

```
add_executable(set-agents-tests
    ${TEST_SOURCES})
# Link the tests with the agent
    module library.
target_link_libraries(set-agents
    -tests
    LINK_PRIVATE GTest::
        gtest_main
    LINK_PRIVATE set-agents
)
# Add the source directory to
    the list of paths for header
    file lookup.
target_include_directories(set-
    agents-tests
    PRIVATE ${
        CMAKE_CURRENT_SOURCE_DIR}
)

# Add tests to the project.
# WORKING_DIRECTORY sets the
    working directory for running
     tests.
gtest_discover_tests(set-agents-
    tests
    WORKING_DIRECTORY ${
        CMAKE_CURRENT_SOURCE_DIR
        }/tests
)
```

Listing 43. Tests build configuration

test_sc_agent_calculate_set_power.cpp

```
// Include the header file for
    testing agents
#include <sc-memory/test/sc_test
    .hpp>

#include <sc-memory/
    sc_memory_headers.hpp>

#include "agent/
    sc_agent_calculate_set_power.
    hpp"
#include "keynodes/
    sc_set_keynodes.hpp"

using AgentTest = ScMemoryTest;

TEST_F(AgentTest,
    AgentCalculateSetPower
    FinishedSuccessfully)
{
  // Register the agent in sc-
      memory.
  m_ctx->SubscribeAgent<
      ScAgentCalculateSetPower>()
      ;

  // Create an action with a
      class for the agent to
      execute.
  ScAction action
    = m_ctx->GenerateAction(
        ScSetKeynodes::
        action_calculate_set_power
        );

  // Create a set with two sc-
      elements.
  ScSet set = m_ctx->GenerateSet
      ();
  ScAddr nodeAddr1 = m_ctx->
      GenerateNode(ScType::
      ConstNode);
  ScAddr nodeAddr2 = m_ctx->
      GenerateNode(ScType::
      ConstNode);
  set << nodeAddr1 << nodeAddr2;

  // Set the created set as an
      argument for the action.
  action.SetArgument(1, set);

  // Initiate and wait for the
      action to complete.
  action.InitiateAndWait();

  // Verify that the action was
      completed successfully.
  EXPECT_TRUE(action.
      IsFinishedSuccessfully());

  // Get the result structure of
       the action.
  ScStructure structure = action
      .GetResult();
  // Verify that it contains sc-
      elements.
  EXPECT_FALSE(structure.IsEmpty
      ());

  // Check sc-constructions in
      the result structure.
  // Verify the first three-
      element construction.
  ScIterator3Ptr it3 = m_ctx->
      CreateIterator3(
    structure, ScType::
        ConstPermPosArc, ScType::
```

```
        ConstCommonArc);
    EXPECT_TRUE(it3->Next());
    ScAddr arcAddr = it3->Get(2);

    auto [beginAddr, linkAddr] =
        m_ctx->
        GetConnectorIncidentElements
        (arcAddr);
    EXPECT_EQ(beginAddr, set);
    EXPECT_TRUE(m_ctx->
        GetElementType(linkAddr).
        IsLink());

    // Verify that the content of
        the link equals 2.
    uint32_t setPower;
    EXPECT_TRUE(m_ctx->
        GetLinkContent(linkAddr,
        setPower));
    EXPECT_EQ(setPower, 2u);

    // Verify the second three-
        element construction.
    it3 = m_ctx->CreateIterator3(
      structure, ScType::
          ConstPermPosArc, ScType::
          ConstPermPosArc);
    EXPECT_TRUE(it3->Next());
    ScAddr arcAddr2 = it3->Get(2);

    auto [relationAddr,
        targetArcAddr] = m_ctx->
        GetConnectorIncidentElements
        (arcAddr2);
    EXPECT_EQ(relationAddr,
        ScSetKeynodes::
        nrel_set_power);
    EXPECT_EQ(targetArcAddr,
        arcAddr);

    // Unregister the agent from
        sc-memory.
    m_ctx->UnsubscribeAgent<
        ScAgentCalculateSetPower>()
        ;
}

// Provide tests for
    unsuccessful and error
    situations.
// ...
```

Listing 44. Agent tests implementation

`ScMemoryTest` class includes `m_ctx` that is object of `ScAgentContext` class. You can use it to work with sc-memory. See [40] and [41] to learn more about available methods for working with sc-memory.

Step 9. Build problem solver

Build it using CMake (listing 45).

```
cmake --preset release-conan
cmake --build --preset release
```

Listing 45. Build problem solver

These commands use CMake to build the C++ problem solver in Release mode. The `-preset` option specifies a pre-configured build setup.

Step 10. Run sc-machine

```
./sc-machine/bin/sc-machine -s
    kb.bin -e "sc-machine/lib/
    extensions;scp-machine/lib/
    extensions;build/Release/lib/
    extensions"
```

Listing 46. Run sc-machine

It starts the *sc-machine*, loading the knowledge base (`kb.bin`) and specifying the paths to the extensions.

To stop the running server, press `Ctrl+C` in the terminal where *sc-machine* is running.

X. Step-by-step guide to using web user interface within the OSTIS Technology

A. About sc-web

sc-web [42] is an intelligent user interface that serves as the primary web component of the *OSTIS Technology*. It provides a universal rendering mechanism for semantic interfaces defined within knowledge bases and is included as part of the *OSTIS Platform*.

B. Quick start with Docker

To quickly deploy *sc-web*, use the official Docker image (listing 47). It allows to connect the user interface to a running *sc-machine* server either locally or remotely.

```
# Connect to a remote sc-machine
    server
docker run --rm -it -p 8000:8000
    ostis/sc-web:latest --
    server_host=<server-ip>

# Connect to a local sc-machine
    instance
docker run --rm -it --network=
    host ostis/sc-web:latest
```

Listing 47. Run sc-web with Docker

After starting, the web interface will be available at `http://localhost:8000`.

C. Manual installation

Step 1. Clone repository

Clone *sc-web* repository (listing 48).

```
git clone https://github.com/
    ostis-ai/sc-web --recursive
cd sc-web
```

Listing 48. Clone sc-web repository

Step 2. Install dependencies

For Ubuntu/macOS, install Node.js dependencies (listing 49) and Python dependencies (listing 50).

```
./scripts/install_dependencies.
    sh
nvm use 16
npm install
```

Listing 49. Install Node.js dependencies

```
python3 -m venv .venv
source .venv/bin/activate
pip3 install -r requirements.txt
```

Listing 50. Install Python dependencies

Otherwise, ensure the following are installed:

- `python3`,
- `pip`,
- `nodejs`,
- `npm`,
- `grunt-cli`, and Python modules (`tornado`, `sqlalchemy`, `numpy`, `configparser`, `py-sc-client`).

Step 3. Build frontend

Build frontend of *sc-web* (listing 51).

```
npm run build
```

Listing 51. Build frontend

Step 4. Run backend server

Ensure *sc-machine* is running. Then start the backend (listing 52).

```
source .venv/bin/activate
python3 server/app.py
```

Listing 52. Run backend server

The user interface will be accessible at `http://localhost:8000`.

To stop the running server, press `Ctrl+C` in the terminal where *sc-web* is running.

XI. Contributing and engaging with OSTIS Community

Active participation in the *OSTIS Community* accelerates knowledge sharing, troubleshooting, and collaborative advancement of the technology. Developers are encouraged to contribute, seek support, and stay informed about ongoing updates and events.

Join *OSTIS Community* by research these opportunities:

1) Join Element Chat – OSTIS Tech Support [43]. Engage in real-time discussions, seek technical support, and collaborate with peers through the dedicated OSTIS Tech Support room.
2) Explore GitHub Repositories [8]. Contribute to the ongoing development of *OSTIS Technology* by accessing source code, submitting issues, and proposing enhancements via the official GitHub repositories. Use GitHub's issue tracker if you encounter issues or have suggestions.
3) Attend OSTIS Conference [44]. Participate in the annual *OSTIS Conference*, which brings together academics, industry professionals, and students to present research, share experiences, and initiate collaborative projects. The conference is open to all interested parties.

The *OSTIS Community* actively promotes community-engaged research practices, emphasizing the importance of trust-building, transparent role definition, and shared vision among stakeholders. The annual *OSTIS Conference* exemplifies this approach by providing a venue for presenting peer-reviewed research, networking, and forming new partnerships across academia and industry

XII. Development directions of the OSTIS Project

The *OSTIS Project* is evolving along several strategic directions aimed at advancing the theory, technology, and practical application of intelligent systems. The main development domains are as follows:

- formalization and continuous refinement of the *OSTIS Standard*;
- development of the *OSTIS Metasystem* for intelligent project management and knowledge support;
- development of applied intelligent systems for automation and decision support;
- development of platform and tools for scalable intelligent system implementation;
- formation of a global *OSTIS Ecosystem* for knowledge exchange and collaboration;
- development of educational programs and training materials;
- maintenance of up-to-date documentation and a centralized knowledge portal.

These directions collectively aim to ensure the sustainable evolution of *OSTIS Technology* as a comprehensive, open, and interoperable technology for intelligent systems, supporting both foundational research and practical applications in diverse fields.

XIII. Conclusion

The *OSTIS Technology* provides a practical and modular framework for building semantically compatible intelligent systems. The approach centers on clear ontological

structuring, use of *SC-code* for knowledge representation, and a multi-agent architecture for problem solving.

The step-by-step methodology – defining project structure, configuring dependencies, formalizing knowledge bases, and implementing agents – enables fast prototyping and easy extension of intelligent systems. OSTIS's plug-and-play integration, platform independence, and reflexivity make it suitable for scalable and maintainable solutions in research and industry.

The provided workflows ensure that developers can quickly assemble, verify, and expand their systems, focusing on real tasks rather than boilerplate or infrastructure issues.

Acknowledgment

The authors would like to thank the scientific team of the Department of Intelligent Information Technologies at the Belarusian State University of Informatics and Radioelectronics for their assistance and valuable comments.

This work was carried out with financial support from the Belarusian Republican Foundation for Fundamental Research (contract with BRFFR № F24MV-011 from 15.04.2024).

References

[1] S. A. Titov and N. V. Titova, "Otkrytyj proekt kak osobyj tip proektov [open source project as a special type of project]," *Fundamental'nye issledovanija [Fundamental research]*, no. 9-2, pp. 384–388, 2015, in Russian. [Online]. Available: https://fundamental-research.ru/ru/article/view?id=39112

[2] *A Guide to the Project Management Body of Knowledge (PMBOK Guide) – Seventh Edition and The Standard for Project Management*. Project Management Institute, 2021.

[3] *Putevoditel' po osnovnym ponjatijam i shemam metodologii Organizacii, Rukovodstva i Upravlenija: Hrestomatija po rabotam G.P. Shhedrovickogo [Guide to the Basic Concepts and Schemes of the Methodology of Organization, Management and Control: Anthology of the Works of G.P. Shchedrovitsky]*. Moskow: Delo, 2004, in Russian. [Online]. Available: https://pqm-online.com/assets/files/lib/books/zinchenko.pdf

[4] V. V. Golenkov, N. A. Gulyakina, and D. V. Shunkevich, *Open Technology of Ontological Design, Production and Operation of Semantically Compatible Hybrid Intelligent Computer Systems*. Minsk: Bestprint, 2021.

[5] V. V. Golenkov and N. A. Gulyakina, "Open project aimed at creating a technology for component-based design of intelligent systems," in *Open Semantic Technologies for Intelligent Systems (OSTIS-2013): Proceedings of the 3rd International Scientific and Technical Conference, Minsk, February 21-23, 2013*, V. V. Golenkov, Ed. Minsk: BSUIR, 2013, pp. 55–78. [Online]. Available: https://libeldoc.bsuir.by/bitstream/123456789/4150/1/Golenkov_Otkrytiy.PDF

[6] OSTIS-AI, "OSTIS-AI: Open Semantic Technology for Intelligent Systems," 2025, accessed: 15.03.2025. [Online]. Available: https://ostis-ai.github.io

[7] V. V. Golenkov, N. A. Gulyakina, V. P. Ivashenko, and D. V. Shunkevich, "Intelligent computer systems of new generation and complex technology of their development, application and modernization," *Doklady BSUIR*, vol. 22, no. 2, pp. 70–79, 2024, in Russian. [Online]. Available: https://doklady.bsuir.by/jour/article/view/3906/1994

[8] OSTIS-AI, "OSTIS Project: Open Semantic Technology for Intelligent Systems," 2025, accessed: 15.03.2025. [Online]. Available: https://github.com/ostis-ai

[9] ——, "OSTIS Platform for Intelligent Systems," 2025, accessed: 15.03.2025. [Online]. Available: https://github.com/ostis-ai/ostis-web-platform

[10] N. Zotov, "Design principles, structure, and development prospects of the software platform of ostis-systems," in *Open Semantic Technologies for Intelligent Systems: Research Papers Collection*, vol. 7. Minsk: BSUIR, 2023, pp. 67–76. [Online]. Available: https://libeldoc.bsuir.by/bitstream/123456789/51247/1/Zotov_Design.pdf

[11] OSTIS-AI, "OSTIS Metasystem," 2025, accessed: 15.03.2025. [Online]. Available: https://github.com/ostis-ai/ostis-metasystem

[12] ——, "OSTIS Metasystem Documentation," 2025, accessed: 15.03.2025. [Online]. Available: https://ostis-ai.github.io/ostis-metasystem/

[13] K. Bantsevich, "Metasystem of the ostis technology and the standard of the ostis technology," in *Open Semantic Technologies for Intelligent Systems: Research Papers Collection*, vol. 6. Minsk: BSUIR, 2022, pp. 357–368. [Online]. Available: https://libeldoc.bsuir.by/bitstream/123456789/49330/1/Bantsevich_Metasystem.pdf

[14] OSTIS-APPS, "Example Application for OSTIS Technology," 2025, accessed: 15.03.2025. [Online]. Available: https://github.com/ostis-apps/ostis-example-app

[15] V. V. Golenkov, N. A. Gulyakina, I. T. Davydenko, D. V. Shunkevich, and A. P. Eremeev, "Ontological design of hybrid semantically compatible intelligent systems based on semantic representation of knowledge," *Ontology of Designing*, vol. 9, no. 1(31), pp. 132–151, 2019. [Online]. Available: https://libeldoc.bsuir.by/bitstream/123456789/37841/1/Golenkov_Ontologicheskoye.pdf

[16] V. V. Golenkov, N. A. Guliakina, and D. V. Shunkevich, "Methodological problems and strategic goals of the work on creation of the theory and technology of new generation intelligent computer systems," *Digital Transformation*, vol. 30, no. 1, pp. 40–51, 2024. [Online]. Available: https://dt.bsuir.by/jour/article/view/819/308

[17] V. Ivashenko, "Towards the theory of semantic space," in *Open Semantic Technologies for Intelligent Systems: Research Papers Collection*, vol. 8. Minsk: BSUIR, 2024, pp. 29–42. [Online]. Available: https://libeldoc.bsuir.by/bitstream/123456789/55571/1/Towards_the_Theory.pdf

[18] V. V. Golenkov, N. A. Gulyakina, I. T. Davydenko, and A. P. Eremeev, "Methods and tools for ensuring compatibility of computer systems," in *Open Semantic Technologies for Intelligent Systems (OSTIS-2019): Proceedings of the International Scientific and Technical Conference, Minsk, February 21-23, 2019*. BSUIR, 2019, pp. 25–52. [Online]. Available: https://libeldoc.bsuir.by/bitstream/123456789/34574/1/Golenkov_Methods.PDF

[19] M. Orlov, "Control tools for reusable components of intelligent computer systems of a new generation," in *Open Semantic Technologies for Intelligent Systems: Collection of Scientific Papers*, vol. 7. Minsk: BSUIR, 2023, pp. 191–206. [Online]. Available: https://proc.ostis.net/proc/Proceedings%20OSTIS-2023.pdf#page=191

[20] ——, "Comprehensive library of reusable semantically compatible components of next-generation intelligent computer systems," in *Open Semantic Technologies for Intelligent Systems: Research Papers Collection*, vol. 6. Minsk: BSUIR, 2022, pp. 261–272. [Online]. Available: https://libeldoc.bsuir.by/bitstream/123456789/49369/1/Orlov_Comprehensive.pdf

[21] V. Ivashenko, "General-purpose semantic representation language and semantic space," in *Open Semantic Technologies for Intelligent Systems (OSTIS-2022): Collection of Scientific Papers*, vol. 6. Minsk: BSUIR, 2022, pp. 41–64. [Online]. Available: https://libeldoc.bsuir.by/bitstream/123456789/49363/1/Ivashenko_General-purpose.pdf

[22] N. Zotov, "Software platform for next-generation intelligent computer systems," in *Open Semantic Technologies for Intelligent Systems: Research Papers Collection*, vol. 6. Minsk: BSUIR, 2022, pp. 297–326. [Online]. Available: https://libeldoc.bsuir.by/bitstream/123456789/49395/1/Zotov_Software.pdf

[23] V. V. Golenkov, D. V. Shunkevich, N. A. Gulyakina, V. P. Ivashenko, and V. A. Zahariev, "Associative semantic computers

for intelligent computer systems of a new generation," in *Open Semantic Technologies for Intelligent Systems: Research Papers Collection*, vol. 7. Minsk: BSUIR, 2023, pp. 39–60. [Online]. Available: https://proc.ostis.net/proc/Proceedings%20OSTIS-2023.pdf#page=39

[24] K. Bantsevich, "Structure of knowledge bases of next-generation intelligent computer systems: A hierarchical system of subject domains and their corresponding ontologies," in *Open Semantic Technologies for Intelligent Systems: Research Papers Collection*, vol. 6. Minsk: BSUIR, 2022, pp. 87–98. [Online]. Available: https://libeldoc.bsuir.by/bitstream/123456789/49331/1/Bantsevich_Structure.pdf

[25] N. Zotov, T. Khodosov, M. Ostrov, A. Poznyak, I. Romanchuk, K. Rublevskaya, B. Semchenko, D. Sergievich, A. Titov, and F. Sharou, "OSTIS Glossary — the Tool to Ensure Consistent and Compatible Activity for the Development of the New Generation Intelligent Systems," in *Open Semantic Technologies for Intelligent Systems: Research Papers Collection*, vol. 8. Minsk: BSUIR, 2024, pp. 127–148. [Online]. Available: https://libeldoc.bsuir.by/bitstream/123456789/55565/1/OSTIS_Glossary.pdf

[26] D. Shunkevich, "Principles of problem solving in distributed teams of intelligent computer systems of a new generation," in *Open Semantic Technologies for Intelligent Systems (OSTIS): Collection of Scientific Papers*, V. V. Golenkov *et al.*, Eds., vol. 7. Minsk: BSUIR, 2023, pp. 115–120. [Online]. Available: https://libeldoc.bsuir.by/bitstream/123456789/51252/1/Shunkevich_Principles.pdf

[27] M. Sadouski, P. Nasevich, M. Orlov, and A. Zhmyrko, "Adaptive user interfaces for intelligent systems: Unlocking the potential of human-system interaction," in *Open Semantic Technologies for Intelligent Systems: Research Papers Collection*, vol. 8. Minsk: BSUIR, 2024, pp. 79–86. [Online]. Available: https://libeldoc.bsuir.by/bitstream/123456789/55532/1/Adaptive_User.pdf

[28] S. Chacon and B. Straub, "Pro Git Book: 1.5 Getting Started - Installing Git," 2025, accessed: 15.03.2025. [Online]. Available: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

[29] Docker, Inc., "Get Started with Docker," 2025, accessed: 15.03.2025. [Online]. Available: https://www.docker.com/get-started/

[30] OSTIS-AI, "sc-machine: Software implementation of semantic memory and its APIs," 2025, accessed: 15.03.2025. [Online]. Available: https://github.com/ostis-ai/sc-machine

[31] Conan contributors, "Conan: C and C++ Open Source Package Manager," 2025, accessed: 15.03.2025. [Online]. Available: https://conan.io

[32] Kitware, Inc., "CMake: Cross-Platform Make," 2025, accessed: 15.03.2025. [Online]. Available: https://cmake.org

[33] pipx contributors, "pipx Installation Guide," 2025, accessed: 15.03.2025. [Online]. Available: https://pipx.pypa.io/stable/installation/

[34] OSTIS-AI, "OSTIS AI Library Conan Repository," 2025, accessed: 15.03.2025. [Online]. Available: https://conan.ostis.net/artifactory/api/conan/ostis-ai-library

[35] ——, "scp-machine: Software implementation of semantic network program interpreter," 2025, accessed: 15.03.2025. [Online]. Available: https://github.com/ostis-ai/scp-machine

[36] N. Zotov, "An ontology-based approach as foundation for multidisciplinary synthesis in modern science," in *Topical Issues of Economics and Information Technologies: Proceedings of the 60th Anniversary Scientific Conference of Postgraduates, Master's Degree Students and Students of BSUIR, Minsk, April 22–26, 2024*. Minsk: BSUIR, 2024, pp. 745–747.

[37] OSTIS-AI, "C++ Agents API for sc-machine," https://ostis-ai.github.io/sc-machine/sc-memory/api/cpp/extended/agents/agents/, 2025, accessed: 15.03.2025.

[38] ——, "C++ Keynodes API for sc-machine," 2025, accessed: 15.03.2025. [Online]. Available: https://ostis-ai.github.io/sc-machine/sc-memory/api/cpp/extended/agents/keynodes/

[39] ——, "C++ Modules API for Agent Management in sc-machine," 2025, accessed: 15.03.2025. [Online]. Available: https://ostis-ai.github.io/sc-machine/sc-memory/api/cpp/extended/agents/modules/

[40] ——, "C++ Core API for sc-memory in sc-machine," 2025, accessed: 15.03.2025. [Online]. Available: https://ostis-ai.github.io/sc-machine/sc-memory/api/cpp/core/api/

[41] ——, "C++ Agent Context API for sc-machine," 2025, accessed: 15.03.2025. [Online]. Available: https://ostis-ai.github.io/sc-machine/sc-memory/api/cpp/extended/agents/agent_context/

[42] ——, "sc-web: Web oriented sc-models interpreter," 2025, accessed: 15.03.2025. [Online]. Available: https://github.com/ostis-ai/sc-web

[43] OSTIS Project, "OSTIS Technology Support Chat (Matrix)," https://app.element.io/#/room/#ostis_tech_support:matrix.org, 2025, official technical support chat for OSTIS Technology on Matrix/Element.

[44] OSTIS Conference Organizing Committee, "International Scientific and Technical Conference "Open Semantic Technologies for Intelligent Systems" (OSTIS)," http://conf.ostis.net/en/ostis-conference/, 2025, Annual international conference on open semantic technologies for intelligent systems.

ПРИНЦИПЫ АВТОМАТИЗАЦИИ РАЗРАБОТКИ ОТКРЫТЫХ ПРОЕКТОВ НА ОСНОВЕ ЭКОСИСТЕМЫ ИНТЕЛЛЕКТУАЛЬНЫХ КОМПЬЮТЕРНЫХ СИСТЕМ НОВОГО ПОКОЛЕНИЯ

Гракова Н. В., Зотов Н. В.,
Орлов М. К., Петрочук К. Д., Банцевич К. А.

Статья предназначена для исследователей, разработчиков и практиков в области искусственного интеллекта, инженерии знаний и проектирования интеллектуальных систем, которые стремятся создавать, расширять или интегрировать семантически совместимые интеллектуальные системы.

Данная статья особенно актуальна для тех, кто интересуется:

- разработкой модульных, повторно используемых и совместимых интеллектуальных систем с использованием технологий с открытым исходным кодом;
- применением формальных онтологических методов и SC-кода для представления знаний и рассуждений;
- реализацией многоагентных решателей проблем и их интеграцией с семантическими базами знаний;
- использованием технологии OSTIS для образовательных, исследовательских или промышленных приложений, требующих адаптивности, масштабируемости и семантической совместимости;
- изучением практических рабочих процессов, включая установку, настройку и тестирование, с помощью реальных примеров, таких как ostis-example-app.

Представленное в статье руководство предполагает базовое знакомство с понятиями искусственного интеллекта, онтологий и разработки программного обеспечения, но предоставляет подробные пошаговые инструкции, подходящие как для новичков, так и для опытных специалистов в этой области.

Received 13.03.2025