OSTIS Platform — a Framework for Developing Intelligent Agents Based on Semantic Networks

Nikita Zotov
Belarusian State University of Informatics and Radioelectronics
Minsk, Belarus
Email: n.zotov@bsuir.by

***Abstract*—This paper examines AI agent frameworks and introduces the *OSTIS Platform* as a solution to limitations in current approaches. It analyzes the principles of AI agents, evaluates frameworks like LangGraph, CrewAI, AutoGen, Semantic Kernel, and LlamaIndex, and details the advantages of the OSTIS Technology. These advantages include a unified semantic basis and deep knowledge representation. The paper will also discuss the implementation of *OSTIS Platform* and agent-driven models, highlighting their potential for advancing intelligent systems.**

***Keywords*—AI, Agent AI, intelligent systems, OSTIS, *OSTIS Platform*, ostis-systems, semantic networks, SC-code, knowledge base**

I. Introduction

The development of artificial intelligence has witnessed remarkable progress in recent years, particularly in the realm of intelligent agents – software entities capable of autonomous decision [1] – making and problem-solving. These agents have become fundamental components in AI systems across various domains, from virtual assistants to complex robotic systems. As AI applications grow increasingly sophisticated, the need for robust frameworks to facilitate agent development has become critical.

AI agents represent a paradigm shift in software development, moving from passive programs that merely respond to inputs toward active entities that can sense their environment, reason about it, and take actions to achieve goals. These intelligent agents work through continuous cycles of perception, reasoning, and action, adapting to changes in their environment and learning from experiences. Their applications span diverse domains including robotics, virtual assistants, smart environments, and simulation systems.

Current frameworks for developing AI agents, including *LangGraph* [2], *AutoGen* [3], *CrewAI* [4], *LlamaIndex* [5], and *Semantic Kernel* [6], have made significant contributions to this field. Each offers distinct approaches to agent orchestration, from *LangGraph*'s workflow-oriented architecture to *CrewAI*'s role-based collaborative model. However, despite their strengths, these frameworks exhibit fundamental limitations that restrict their effectiveness in addressing complex, cross-domain problems.

These limitations [7]–[10] include shallow integration of problem-solving methods, where frameworks often focus on specific problem-solving paradigms without providing a unified semantic foundation. Additionally, existing frameworks struggle with flexibility and scalability, making them less suitable for large-scale or highly dynamic AI systems. The complexity of maintaining and updating developed agents represents another significant challenge, stemming from the lack of a unified semantic basis for integrating diverse problem-solving methods.

The *OSTIS Platform* [11], [12] emerges as a comprehensive solution to these challenges, offering a framework specifically designed for developing intelligent agents based on semantic networks. Unlike traditional approaches, *OSTIS Platform* integrates diverse problem-solving methods on a common semantic basis, enabling the creation of complex, interconnected knowledge models capable of addressing nuanced problems.

This paper examines the principles and implementation of the *OSTIS Platform*, highlighting its advantages over existing frameworks and detailing its event-driven and agent-driven models. By providing a unified semantic basis and deep knowledge representation capabilities, the *OSTIS Platform* establishes a robust foundation for building intelligent agents capable of tackling complex real-world problems more effectively than existing frameworks.

II. State of the art

In recent years, the development of intelligent agents – computer programs that can make decisions and act by themselves – has become a major area in artificial intelligence (AI) [1], [13]–[15]. Many software tools, called frameworks, have been created to help build these agents. These frameworks aim to make it easier to design, build, and manage systems that can act on their own, learn, and solve different types of problems.

A. What are AI agents

AI agents are like digital workers. They can:

- sense what is happening around them (using data or sensors);

- think about what to do next (using rules or goals);
- take actions to reach their goals.

How AI Agents Work

AI agents can be defined as software programs that use AI techniques to perform tasks autonomously.

AI agents work by following a basic cycle of perception, reasoning, and action [16]:

1) The agent receives information about its environment through sensors or other data sources.
2) The agent uses this information to make decisions based on its goals, rules, and models of the environment.
3) The agent takes actions to achieve its goals, which can involve interacting with the environment or other agents.

This cycle is repeated continuously as the agent adapts to changes in its environment and learns from its experiences.

Role of AI agents in modern systems

They are used in a wide range of applications, including [1], [13], [14]:

- robots that move and interact with the world;
- virtual assistants (like Siri or Alexa);
- smart homes and cities that control lights, heating, or traffic;
- systems that simulate groups of people or organizations.

What makes an AI agent framework

AI agent frameworks provide the infrastructure and tools needed to build autonomous systems that can perceive, reason, plan, and take actions to achieve specified goals [17]. These frameworks extend the capabilities of large language models with orchestration, planning, memory, and tool-use capabilities, transforming them into systems that can interact with their environment and make decisions based on available information.

At their core, AI agent frameworks solve several challenging aspects of agent development:

- managing context and persistent memory across interactions;
- enabling structured interaction with external tools, APIs, and data sources;
- making logical decisions based on available information;
- planning multi-step processes to achieve complex goals;
- and evaluating performance and improving reliability.

B. Overview of existing frameworks

Several popular AI agent frameworks have gained traction due to their diverse capabilities [18], [19]:

- *LangGraph* [2], [20] extends the *LangChain* ecosystem with a graph-based architecture that treats agent steps as nodes in a directed graph. Developed by the creators of LangChain, it uses graph-based technology to create detailed workflows for AI agent systems. *LangGraph* provides scalable infrastructure, an opinionated API for user interfaces, and an integrated developer studio for streamlined deployment and development [21]–[23].
- *AutoGen* [24] borns out of Microsoft Research, AutoGen frames agent interactions as asynchronous conversations among specialized agents. This approach reduces blocking, making it well-suited for longer tasks or scenarios requiring real-time concurrency. AutoGen supports free-form chat among many agents and is backed by a research-driven community [3], [25].
- *CrewAI* [26] is an open-source Python framework that simplifies the development and management of multi-agent AI systems. It assigns specific roles to agents, enabling autonomous decision-making and facilitating seamless communication. CrewAI supports both sequential and hierarchical task execution modes, providing a user-friendly platform for creating and managing multi-agent systems [4], [26].
- *LlamaIndex* [27] excels in retrieval-centric applications by integrating retrieval-augmented generation (RAG) with indexing capabilities. This synergy allows for extensive data lookup and knowledge fusion, making it ideal for use-cases revolving around data retrieval [5].
- *Semantic Kernel* [6], [28] is Microsoft's .NET-first approach to orchestrating AI "skills" and combining them into full-fledged plans or workflows. It supports multiple programming languages and focuses on enterprise readiness, including security, compliance, and integration with Azure services. Semantic Kernel allows for the creation of a range of skills, some powered by AI and others by pure code, making it popular among teams integrating AI into existing business processes.

C. Comparison of frameworks

The table I summarizes the key features and limitations of each framework [7]–[10].

D. Other limitations of existing frameworks

Despite their strengths, these frameworks face other significant limitations:

1) *Shallow integration of problem-solving methods.* Most frameworks focus on specific problem-solving paradigms, such as procedural or declarative methods, without providing a unified semantic basis for integrating diverse approaches [29], [30]. For example, AutoGen excels in conversation-based workflows but lacks support for declarative knowledge representation.

Framework	Architecture	Key Features	Strengths	Limitations	Best use cases
LangGraph	Graph-based architecture, workflow-oriented	• Explicit workflow graphs • State persistence • Human-in-the-loop support	• Fine-grained control • LangChain integration	• Steep learning curve • LangChain dependency	• Complex workflows • Research • Multi-agent systems
AutoGen	Multi-agent conversational	• Asynchronous agent messaging • GUI support • Tool/human integration	• Multi-agent support • Flexible • Python/.NET	• Requires prompt engineering • Possible looping issues	• Conversational AI • Collaboration • Enterprise solutions
CrewAI	Role-based collaborative	• Pythonic annotations • UI-driven engine • 700+ integrations • Logging	• Beginner-friendly • Rapid prototyping	• Less suited for single-agent • Smaller community	• Teamwork • Project management • Healthcare
LlamaIndex	Data-centric, retrieval-focused	• Knowledge graph integration • Vector database support • Query routing	• Data integration • Knowledge-intensive tasks	• Not focused on orchestration • Needs pairing with other frameworks	• Data analysis • Research • Knowledge agents
Semantic Kernel	Plugin-based, modular	• Multi-language support • Memory management • Enterprise security • Plugins	• Enterprise-ready • Microsoft ecosystem	• Focus on C# • Steep learning curve	• Enterprise applications • Document processing

Table I
COMPARISON OF AI AGENT FRAMEWORKS

2) *Limited flexibility and scalability.* Frameworks like *LangGraph* and *AutoGen* struggle with scalability and flexibility, making them less suitable for large-scale or highly dynamic AI systems [31].
3) *Lack of support for complex tasks.* Existing frameworks often focus on specific tasks or domains, lacking the versatility needed to tackle complex, cross-domain problems [31]. For instance, CrewAI's limited orchestration strategies restrict its ability to handle complex workflows.
4) *Maintenance complexity.* The lack of a unified semantic basis for integrating diverse problem-solving methods leads to increased complexity in maintaining and updating developed agents. This complexity arises from the need to manage multiple paradigms without a common foundation, making it difficult to ensure consistency and adaptability across different components [31], [32].
5) *Lack of standardization and interoperability*. The diversity of agent architectures and communication protocols leads to integration and management challenges. The lack of standardization makes it difficult to unify practices and ensure seamless interoperability across platforms and vendors [31].

E. Advantages of OSTIS Platform

The *OSTIS Platform* addresses these limitations by providing a comprehensive framework for developing intelligent agents based on semantic networks. The *OSTIS Technology* [33] offers several key advantages:

- *Unified semantic basis.* The *OSTIS Technology* integrates diverse problem-solving methods (both declarative and procedural) on a common semantic basis, allowing for the creation of complex, interconnected models that can handle diverse and nuanced information [34]–[37].
- *Deep knowledge representation.* The *OSTIS Technology* utilizes semantic networks to represent knowledge, enabling rich, interconnected models that can address complex tasks more effectively than existing frameworks [12], [34].
- *Flexibility and scalability.* The modular design of *OSTIS Technology* ensures that systems can scale efficiently without compromising performance, making it suitable for large-scale AI applications [38], [39].
- *Adaptability and learning.* ostis-systems are designed to adapt to changing conditions and learn from interactions, enhancing their effectiveness in dynamic

environments [40].

The key factor hindering the development of systems where various problem-solving models could be freely integrated is the lack of compatibility among different problem-solving approaches. This stems from the absence of a common formal framework that would enable the implementation of models in such a way that they can be easily integrated into a single system and supplemented with new models as needed [32], [41].

Having established the advantages of the *OSTIS Platform*, the following section will delve into the principles of its implementation.

III. Principles of implementation of ostis-platforms

All intelligent systems developed according to the principles of the *OSTIS Technology* are commonly referred to as *ostis-systems*. Each *ostis-system* consists of an sc-model, including a knowledge base, problem solver, a user interface, and an *ostis-platform* on which the sc-model is interpreted [30], [38]. An sc-model of an ostis-system constitutes a logical-semantic model of that system described in *SC-code*, the language of universal information encoding. An *ostis-platform* represents a hardware-implemented computer or a software emulator for interpretation of sc-models of ostis-systems [42].

Implementations of *ostis-platform* may vary, but each should adhere to basic principles described in [12].

In contrast to traditional computer systems, ostis systems orient towards:

- independence from the implementation of a particular ostis-platform;
- storage of information in a unified and semantically compatible form (in *SC-code* [43]);
- event-oriented and parallel processing of this information.

The principles of ostis-systems are provided by a concrete implementation of the ostis-platform. Within each *ostis-platform*, there exists:

- a shared semantic memory that allows [44]:
  - storage of information constructions belonging to *SC-code* (sc-texts);
  - storage of information constructions not belonging to *SC-code* (images, text files, audio and video files, etc.);
  - storage of subscriptions to occurrences of events in memory;
  - initiation of agents after events appear in memory;
  - the use of a programming interface to work with *SC-code* and *non-SC-code* information constructions, including:
    * operations to create, search, modify, and delete constructions in the memory;
    * operations for subscribing to the occurrence of events in the memory;
    * operations for controlling and synchronizing processes in the memory;
    * programming interface for creating platform-dependent agents;
- an interpreter of the *SCP* asynchronous-parallel programming language, which is a platform-independent programming interface that implements platform-independent operations on the shared semantic memory.

IV. About sc-machine

sc-machine [45] is the core of the *OSTIS Platform* [11], 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*.

Key features of *sc-machine* are:

1) *Unified knowledge representation. sc-machine* uses *SC-code*, a universal knowledge representation language, to encode both declarative (facts, data structures, documentation) and procedural (agents, algorithms, workflows) knowledge. This approach ensures semantic compatibility and interoperability across different intelligent systems.
2) *Agent-based processing.* The system leverages an agent-based architecture, where agents are autonomous components that process knowledge graphs, execute tasks, and solve problems. Agents can be implemented in various languages, including C++ and SCP Language, and interact with sc-memory through well-defined APIs.
3) *Event-driven workflow. sc-machine* includes an event manager that supports asynchronous, event-based processing. Agents are triggered by events in the knowledge base, enabling dynamic and parallel task execution.
4) *Extensible APIs.* The platform provides native C++ APIs, as well as network APIs via the sc-server (WebSocket/JSON), allowing integration with external applications and services.
5) *Tools. sc-machine* includes tools such as sc-builder, which loads *SCs-code* files into storage, and sc-server, which exposes the knowledge base over the network for remote access and manipulation

V. Event-driven model within the OSTIS Platform

The `sc-machine` uses event-driven model to manage processing sc-constructions. The *sc-memory* stores *SC-code* constructions, which are graph structures, then any kind of events, occurring in *sc-memory*, is related to changes in these graph constructions [46].

These are methods that generate events:

- GenerateConnector,
- EraseElement,
- SetLinkContent.

They publish events to an event queue without needing to know which consumers will receive them. These components filter and distribute events to appropriate consumers. They manage the flow of events and ensure that they reach the correct destinations. Event consumers are the components that listen for and process events. Event consumers can be modules, agents or something else.

Within the *OSTIS Technology*, events are considered only situations in which relationships have changed or new relationships have been generated, or link content have been changed, or some sc-element have been erased.

The *sc-machine* provides functionality for subscribing to the following elementary types of sc-events:

- ScElementaryEvent is base class for all sc-events, it can be used to handle all sc-events for specified sc-element;
- ScEventAfterGenerateConnector, emits each time, when sc-connector from or to specified sc-element is generated;
- ScEventAfterGenerateOutgoingArc, emits each time, when outgoing sc-arc from specified sc-element is generated;
- ScEventAfterGenerateIncomingArc, emits each time, when incoming sc-arc to specified sc-element is generated;
- ScEventAfterGenerateEdge, emits each time, when sc-edge from or to specified sc-element is generated;
- ScEventBeforeEraseConnector, emits each time, when sc-connector from or to specified sc-element is erasing;
- ScEventBeforeEraseOutgoingArc, emits each time, when outgoing sc-arc from specified sc-element is erasing;
- ScEventBeforeEraseIncomingArc, emits each time, when incoming sc-arc to specified sc-element is erasing;
- ScEventBeforeEraseEdge, emits each time, when sc-edge from or to specified sc-element is erasing;
- ScEventBeforeEraseElement, emits, when specified sc-element is erasing;
- ScEventBeforeChangeLinkContent, emits each time, when content of specified sc-link is changing.

All these sc-events classes are inherited from ScElementaryEvent class. ScElementaryEvent class is inherited from ScEvent class that is an abstract class.

The ScElementaryEventSubscription class serves as the base class for all sc-event subscriptions. It is utilized to capture all sc-events for a specified sc-element.

Each sc-event subscription constructor, excluding the ScElementaryEventSubscription constructor, requires three parameters:

- context is an object of ScMemoryContext used to interact with sc-events.
- subscriptionElementAddr is an object of ScAddr representing the sc-element that needs to be monitored for a specific sc-event.
- delegateFunc is a delegate to a callback function that will be invoked upon each event emission. The callback function signature is void delegateFunc(TScEvent const &), where TScEvent corresponds to the respective sc-event class.

The constructor for the *ScElementaryEventSubscription* class takes four parameters:

- context is An object of ScMemoryContext used for sc-event handling.
- eventClassAddr is an object of ScAddr representing the sc-event class.
- subscriptionElementAddr is an object of ScAddr for the sc-element to be monitored.
- delegateFunc is a delegate to a callback function invoked on each event emission, with the signature void delegateFunc(ScElementaryEvent const &).

These constructors are private and cannot be called directly.

All sc-event classes are located in core keynodes:

- ScKeynodes::sc_event_after_generate_connector;
- ScKeynodes::sc_event_after_generate_outgoing_arc;
- ScKeynodes::sc_event_after_generate_incoming_arc;
- ScKeynodes::sc_event_after_generate_edge;
- ScKeynodes::sc_event_before_erase_connector;
- ScKeynodes::sc_event_before_erase_outgoing_arc;
- ScKeynodes::sc_event_before_erase_incoming_arc;
- ScKeynodes::sc_event_before_erase_edge;
- ScKeynodes::sc_event_before_erase_element;
- ScKeynodes::sc_event_before_change_link_content.

They can be used as eventClassAddr for CreateElementaryEventSubscription.

The table II describes the parameters of the callback function, named in the figures. If no parameter name is provided in the figure, it defaults to an empty value. Here, `context` is a pointer to an object of the `ScAgentContext` class.

VI. Agent-driven model within the OSTIS Platform

The `sc-machine` employs an agent-driven model for knowledge processing. This model facilitates message exchange between agents through shared memory. Agents can be added or removed without affecting others, promoting decentralized and independent initiation. The *sc-machine* `API in C++` provides tools for creating, managing, and integrating agents within the `sc-machine` [47].

Within the *OSTIS Technology*, agents are classified as either platform-independent or platform-dependent [47]. Platform-independent agents are implemented using *SC-code*, interpreted by the scp-machine [48]. Platform-dependent agents are implemented using the *sc-machine* `API in C++`.

Agents react to events (sc-events) in *sc-memory*. An agent is triggered when a subscribed sc-event occurs. The primary initiation condition defines the sc-event that awakens the agent. Upon awakening, the agent checks its full initiation condition. If successful, it initiates and executes an action using an agent program. After execution, the agent checks for a result [47].

Since the *OSTIS Platform* 0.10.0 [45], the API for agents has been significantly modified—transitioning from code generation to template-based programming. New classes and methods have been introduced for working with agents:

- Two base classes for all types of agents [47]:
  - The `ScAgent` class for implementing agent classes that respond to any elementary events in sc-memory.
  - The `ScActionInitiatedAgent` class for implementing agents that respond to events of initiated actions in sc-memory.
- The `ScAgentContext` class for working with `ScEvent` events, `ScEventSubscription` subscriptions, and `ScWait` and `ScEventWaiter` waiters [49].
- The `ScAction` class for handling actions in sc-memory.
- The `ScAgentBuilder` class for managing dynamic agent specifications [50].
- The `ScKeynodes` and `ScModule` classes have been simplified for use [51].

`ScAgent` can be compared to a person who reacts to any sc-events in their environment. For instance, if someone shouts "Fire!", this person immediately responds and starts acting according to a plan to help in the situation. In terms of `ScAgent`, this means the agent reacts to any elementary sc-events in sc-memory.

`ScActionInitiatedAgent` is similar to a person who waits for a specific signal to start an action. For example, if someone says "Begin the rescue operation!", this person knows exactly what to do and starts acting. In the case of `ScActionInitiatedAgent`, the agent reacts to events related to the initiation of specific actions in sc-memory.

The key scientific distinction between these agent architectures lies in their event-processing mechanisms and behavioral complexity. While `ScAgent` exhibits stimulus-response patterns characteristic of purely reactive architectures, `ScActionInitiatedAgent` demonstrates targeted responsiveness with higher-level goal orientation.

VII. Agent specification

A. Agent specification relations

The agent specification within the *OSTIS Technology* is a formalized approach to defining and managing agents, which are entities responsible for performing transformations in sc-memory of ostis-systems. This approach includes [52]:

1) Agents are described using a set of ontologies that define their concept, roles, and relationships. These ontologies also provide formal tools to synchronize the actions performed by agents in sc-memory.
2) The specification ensures compatibility and synchronization of sc-agent actions within the semantic network, contributing to the seamless operation of intelligent systems developed under the *OSTIS Technology*.

The agent's specification includes:

- its primary initiation condition,
- action class it performs,
- initiation condition,
- result condition,
- key sc-elements used during action execution,
- and other details.

Storing agent specifications in a knowledge base provides several benefits [29], [52]:

1) Agent specifications allow for easy modification or extension of agent behavior without needing to rewrite code. This makes the system more flexible and adaptable to new conditions.
2) By storing specifications in a knowledge base, it becomes easier to manage agent behavior, add new agents, or remove existing ones without affecting other parts of the system.
3) Agent specifications help understand how and why agents make decisions, which is crucial for debugging and optimizing the system.
4) Agent specifications provide a unified representation of their behavior, facilitating integration with other system components and understanding their interactions.

<table>
<tr><th>Class</th><th>Description</th></tr>
<tr><td><code>ScElementaryEventSubscription</code></td><td><pre><code>auto subscription = context->CreateElementaryEventSubscription(
  eventClassAddr,
  subscriptionElementAddr,
  [](ScElementaryEvent const & event) -> void
{
  // Handle sc-event.
});</code></pre></td></tr>
<tr><td><code>ScEventAfterGenerateConnector</code> (similarly <code>ScEventAfterGenerateOutgoingArc</code>, <code>ScEventAfterGenerateIncomingArc</code> and <code>ScEventAfterGenerateEdge</code>)</td><td><pre><code>auto subscription = context->CreateElementaryEventSubscription<
  ScEventAfterGenerateConnector<ScType::ConstPermPosArc>>(
  subscriptionElementAddr,
  [](ScEventAfterGenerateConnector<ScType::ConstPermPosArc> const & event) -> void
{
  // Handle sc-event.
});</code></pre></td></tr>
<tr><td><code>ScEventBeforeEraseConnector</code> (similarly <code>ScEventBeforeEraseOutgoingArc</code>, <code>ScEventBeforeEraseIncomingArc</code> and <code>ScEventBeforeEraseEdge</code>)</td><td><pre><code>auto subscription = context->CreateElementaryEventSubscription<
  ScEventBeforeEraseConnector<ScType::ConstPermPosArc>>(
  subscriptionElementAddr,
  [](ScEventBeforeEraseConnector<ScType::ConstPermPosArc> const & event) -> void
{
  // Handle sc-event.
});</code></pre></td></tr>
<tr><td><code>ScEventBeforeEraseElement</code></td><td><pre><code>auto subscription = context->CreateElementaryEventSubscription<
  ScEventBeforeEraseElement>(
  subscriptionElementAddr,
  [](ScEventBeforeEraseElement const & event) -> void
{
  // Handle sc-event.
});</code></pre></td></tr>
<tr><td><code>ScEventBeforeChangeLinkContent</code></td><td><pre><code>auto subscription = context->CreateElementaryEventSubscription<
  ScEventBeforeChangeLinkContent>(
  subscriptionElementAddr,
  [](ScEventBeforeChangeLinkContent const & event) -> void
{
  // Handle sc-event.
});</code></pre></td></tr>
</table>

Table II

TYPES OF SC-EVENT SUBSCRIPTION IN SC-MEMORY

Thus, storing agent specifications in a knowledge base is a key aspect of supporting complex agent-based systems and allows for the creation of more effective and adaptive solutions.

Key to this API are the relations that define the agent's specification, connecting the agent to events, actions, conditions, key elements, and its program. The denotational semantics of these relations define their meaning in specifying agent behavior, while the operational semantics describe how the *sc-machine* uses these relations during agent execution (table III).

This specification can be represented in a *knowledge base* using *SC-code* [53] or programmatically using the `sc-machine` API in C++.

Consider an abstract sc-agent for calculating the power of a set. The following *SCs-code* (listing 1) and *SCg-code* (figure 1) illustrates its specification:

The agent specification is directly involved in its invocation process.

B. Agent call process

Below is a detailed enumeration of the steps in this process, followed by a sequence diagram (figure 2) illustrating the flow of operations.

1) *Event occurrence.* When a specific sc-event occurs in sc-memory, the system checks for any agents subscribed to that event type.
2) *Checking primary initiation condition.* The primary initiation condition defines the sc-event that will trigger the agent. This condition acts as a preliminary filter.
3) *Checking full initiation condition.* Upon an event, the agent checks its full initiation condition. This is a more detailed check to ensure the agent should execute.
4) *Action initiation.* If the full initiation condition is met, the agent initiates an action of a specified class.
5) *Agent program execution.* The agent executes its program (defined in the DoProgram method). This program performs the agent's task, processing input and generating output.
6) *Checking result condition.* After executing its program, the agent can check if a result condition is met, which might involve verifying the outcome of the action.

C. Ways of providing agent's specification

The `sc-machine` API provides two methods for implementing agents in C++:

- when the agent's specification is represented in the knowledge base;
- when the agent's specification is represented directly in C++ code.

Agent specifications can be static, dynamic, or semi-dynamic.

1) *Static agent specification* is provided externally in the agent's class (via overriding public getters). It is not stored in the knowledge base (see Static agent specification).
2) *Dynamic agent specification* is provided in the knowledge base or initially in the code but is automatically saved into the knowledge base. Use the API of `ScModule` and `ScAgentBuilder` classes (see Dynamic agent specification).
3) *Semi-dynamic agent specification* is provided in the knowledge base or initially in the code and appended externally (via overriding public getters, see Semi-dynamic agent specification).

D. Static agent specification

This section discusses implementing an agent with a static specification. For dynamic agent specifications, see Dynamic agent specification.

Two main classes are used for implementing agents: `ScAgent` and `ScActionInitiatedAgent`.

`ScAgent`

It is a base class for agents in C++. This class provides implemented methods to retrieve elements of the agent's specification from the knowledge base. All these methods can be overridden in agent class [47].

A distinction should be made between an abstract sc-agent as a class of functionally equivalent sc-agents described in the knowledge base and `ScAgent` as a C++ class that implements an API to work with abstract sc-agents in the knowledge base.

This class can be used for all types of platform-dependent agents. Agents of this class react for events in the knowledge base, check the full initiation condition. If the check is successful, generate, initiate and perform the action. After that, they check full result condition. The example using this class is represented in listing 2.

```
// File my_agent.hpp
#pragma once

#include <sc-memory/sc_agent.hpp>

// The agent class should inherit from
    the ScAgent class and specify the
    template argument as the sc-event
    class. Here,
    ScEventAfterGenerateIncomingArc<
    ScType::ConstPermPosArc> is the type
     of event to which the given agent
    reacts.
class MyAgent : public ScAgent<
  ScEventAfterGenerateIncomingArc<ScType
      ::ConstPermPosArc>>
{
public:
  // Here, the class of actions that the
       given agent performs should be
      specified.
  // Here 'GetActionClass' overrides '
      GetActionClass' in 'ScAgent' class
      . This overriding is required.
  ScAddr GetActionClass() const override
      ;
```

Relation identifier	Denotational semantics	Operational semantics
`nrel_primary_initiation_condition`	Specifies the initial event in sc-memory that triggers the agent. Indicates which event will cause the agent to "awaken."	The system checks for agents subscribed to an event type. The relation is used to determine if an agent is subscribed to that event.
`nrel_sc_agent_action_class`	Specifies the type or class of actions that the agent is designed to perform. Defines the agent's role or the kind of actions it's capable of executing.	If the full initiation condition is met, the agent initiates an action of the class specified by this relation. Determines what "action" will be created when the agent starts to perform a task.
`nrel_initiation_condition_and_result`	Encapsulates a pair of conditions: the initiation condition (a detailed check after the primary condition) and the result condition (a check after the action's execution).	Upon awakening, the agent checks its full initiation condition. After executing its program, the agent checks if the result condition is met. These conditions allow the agent to verify its context and the outcome of its actions.
`nrel_sc_agent_key_sc_elements`	Defines the set of key knowledge elements that the agent needs to access and manipulate during its operation. These are important concepts or data structures the agent relies on.	During the agent's program execution, this relation identifies the specific SC-elements needed by the agent, allowing the agent to quickly locate and use them.
`nrel_sc_agent_program`	Specifies the actual code or program that the agent executes. This is the implementation of the agent's logic, defining how the agent processes input and generates output.	The agent executes the program specified by this relation. This program processes input, interacts with the knowledge base, and generates the desired output based on the agent's purpose.
`nrel_inclusion`	This relation connects an abstract sc-agent to a concrete implementation of that agent. It specifies implementations of an abstract agent, which can be implemented in C++ or SC-code. It is important for linking the general specification of an agent to its specific implementation details.	–

Table III
DENOTATIONAL AND OPERATIONAL SEMANTICS OF RELATIONS IN AGENT SPECIFICATION

```
  // Here, the program of the given
      agent should be implemented. This
      overriding is required.
  ScResult DoProgram(
    ScEventAfterGenerateIncomingArc<
      ScType::ConstPermPosArc> const &
          event,
    ScAction & action) override;

  // Other user-defined methods.
};
```

Listing 2. Definition of an agent inheriting ScAgent class

It is possible to override `DoProgram` without sc-event argument (listing 3).

```
// File my_agent.hpp
#pragma once

#include <sc-memory/sc_agent.hpp>

class MyAgent : public ScAgent<
  ScEventAfterGenerateIncomingArc<ScType
      ::ConstPermPosArc>>
{
public:
  ScAddr GetActionClass() const override
      ;
  ScResult DoProgram(ScAction & action)
      override;

  // Other user-defined methods.
};
```

Listing 3. Definition of an agent inheriting ScAgent class with one-argument DoProgram

Any existing event types can be specified as a template argument to the `ScAgent` class. For example, an agent can be created that will be triggered by an sc-event involving the removal of an sc-element (see listing 4).

```
// File my_agent.hpp
#pragma once

#include <sc-memory/sc_agent.hpp>

class MyAgent : public ScAgent<
    ScEventBeforeEraseElement>
{
public:
  ScAddr GetActionClass() const override
      ;
  ScResult DoProgram(
    ScEventBeforeEraseElement const &
        event, ScAction & action)
        override;

  // Other user-defined methods.
};
```

Listing 4. Example of an agent triggered by removing sc-element

`ScActionInitiatedAgent`

`ScActionInitiatedAgent` facilitates the implementation of agents that execute actions initiated by other agents. It requires passing the action class node rather than manually checking the initiation condition.

This class is only applicable for agents triggered by generating an outgoing sc-arc from the `action_initiated` class node (listing 5).

```
// Abstract sc-agent
agent_calculate_set_power
<- abstract_sc_agent;
=> nrel_primary_initiation_condition:
    // Class of sc-event and listen (subscription) sc-element
    (sc_event_after_generate_outgoing_arc => action_initiated);
=> nrel_sc_agent_action_class:
    // Class of actions to be performed by agent
    action_calculate_set_power;
=> nrel_initiation_condition_and_result:
    (..agent_calculate_set_power_initiation_condition
        => ..agent_calculate_set_power_result_condition);
<= nrel_sc_agent_key_sc_elements:
// Set of key sc-elements used by this agent
{
    action_initiated;
    action_calculate_set_power;
    concept_set;
    nrel_set_power
};
=> nrel_inclusion:
    // Instance of abstract sc-agent; concrete implementation of agent in C++
    agent_calculate_set_power_implementation
    (*
        <- platform_dependent_abstract_sc_agent;;
        // Set of links with paths to sources of agent programs
        <= nrel_sc_agent_program:
        {
            [github.com/path/to/agent/sources]
            (* => nrel_format: format_github_source_link;; *)
        };;
    *);;

// Full initiation condition of agent
..agent_calculate_set_power_initiation_condition
= [*
    action_calculate_set_power _-> .._action;;
    action_initiated _-> .._action;;
    .._action _-> rrel_1:: .._set;;
    concept_set _-> .._set;;
*];;
// Agent should check by this template that initiated action is instance of
// class 'action_calculate_set_power' and that it has argument.

// Full result condition of agent
..agent_calculate_set_power_result_condition
= [*
    .._set _=> nrel_set_power:: _[];;
*];;
// Agent should check by this template that action result contains
// sc-construction generated after performing action.
```

Listing 1. An example of abstract sc-agent spefication represented in SCs-code

```
// File my_agent.hpp
#pragma once

#include <sc-memory/sc_agent.hpp>

// The agent class should inherit from the ScActionInitiatedAgent class.
class MyAgent : public ScActionInitiatedAgent
{
public:
  // Here, the class of actions that the given agent performs should be specified.
  // This overriding is required.
  ScAddr GetActionClass() const override;
  // Here, the program of the given agent should be implemented.
  // This overriding is required.
  ScResult DoProgram(
    ScActionInitiatedEvent const & event, ScAction & action) override;
  // Here 'ScActionInitiatedEvent' is type of event to which the given agent reacts.

  // Other user-defined methods.
};
```

Listing 5. Definition of an agent inheriting ScActionInitiatedAgent class

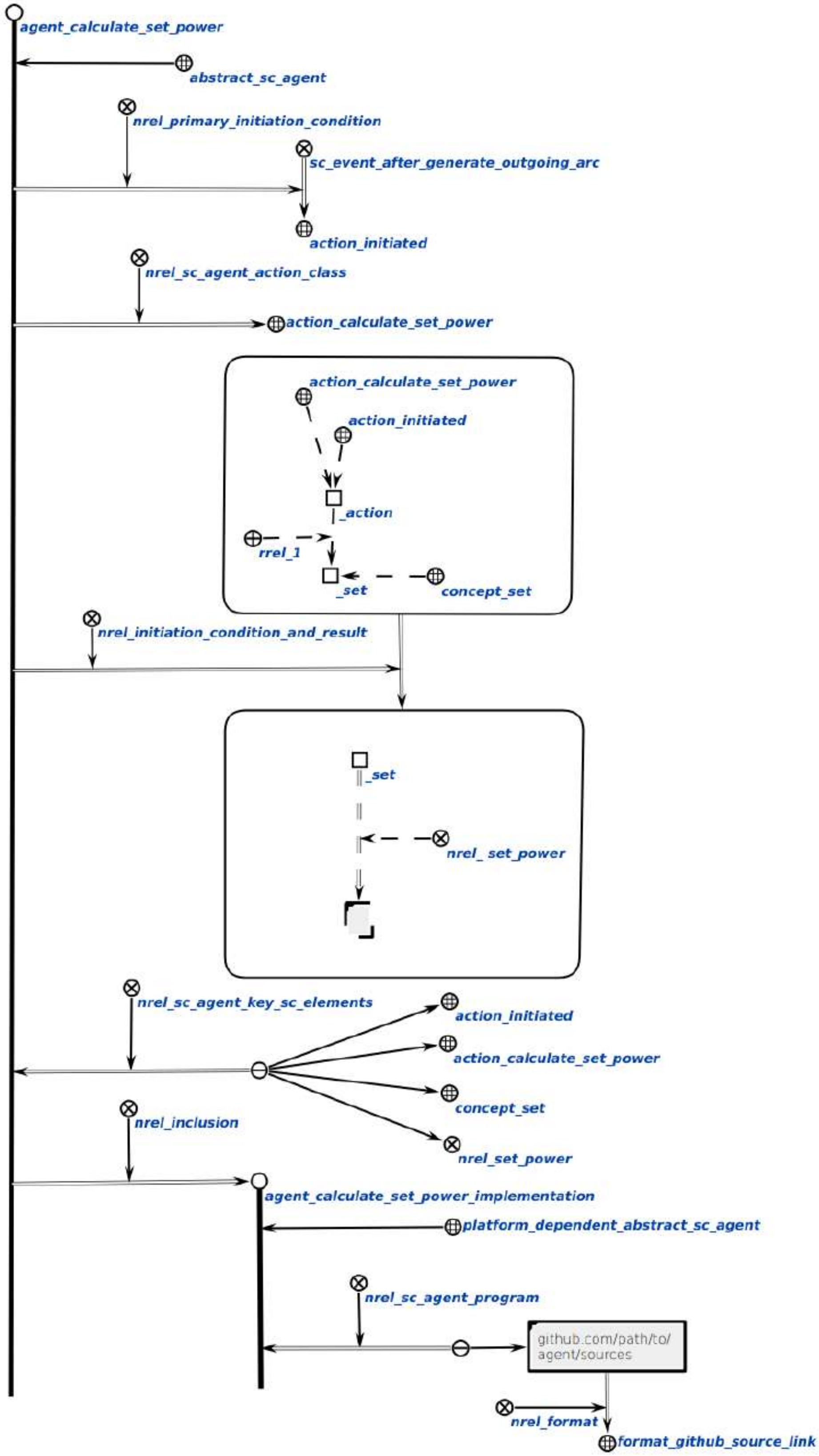

Figure 1. An example of abstract sc-agent spefication represented in SCg-code

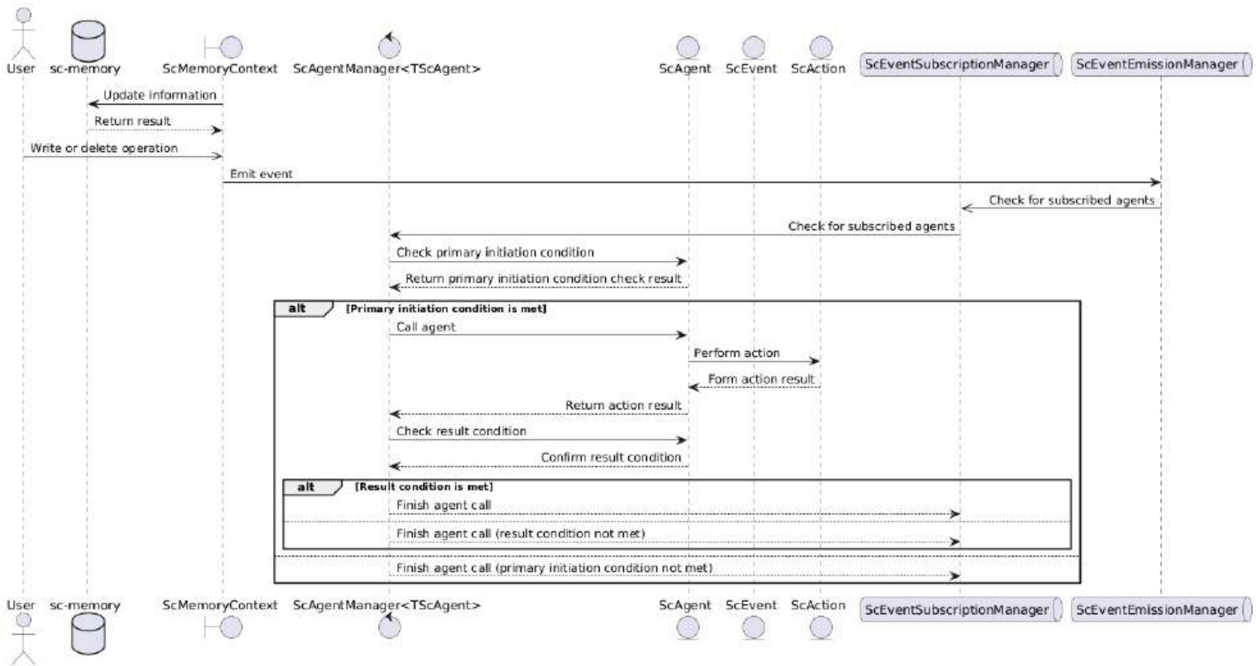

Figure 2. Sequence diagram of agent call

`ScActionInitiatedAgent` has default `GetInitiationConditionTemplate` that returns template that can be used to check that initiated action is action with class of specified agent.

`ScActionInitiatedEvent` is alias for `ScEventAfterGenerateOutgoingArc` with subscription sc-element `action_initiated`.

`Required Methods`

`GetActionClass`

This method retrieves the action class performed by the agent. If the abstract sc-agent for this agent class lacks an action class, the method throws `utils::ExceptionItemNotFound`. See listing 6.

```
// File my_agent.cpp
#include "my_agent.hpp"
#include "keynodes/my_keynodes.hpp"

ScAddr MyAgent::GetActionClass() const
{
  // A valid sc-address of the action
      class must be specified, and the
      action class must be one of the
      following types: 'receptor_action
      ', 'effector_action', '
      behavioral_action', or '
      information_action'. Otherwise,
      the given sc-agent cannot be
      subscribed to the sc-event.
  return MyKeynodes::my_action;
}
```

Listing 6. Implementation of GetActionClass method

`DoProgram`

This method is executed when the agent successfully checks the initiation condition. The agent processes an input construction and generates an output construction (listing 7).

```
ScResult MyAgent::DoProgram(
    ScActionInitiatedEvent const & event
    , ScAction & action)
{
  // Class 'ScAction' encapsulates
      information about sc-action. The
      provided action is action that the
       given agent performs right now.
      It belongs to 'MyKeynodes::
      my_action' class. If  agent
      inherits 'ScActionInitiatedAgent'
      class then this agent performs
      action initiated externally. If
      agent inherits 'ScAgent' then this
       agent generates action, initiates
       and performs new action, not
      provided externally. Actions are
      copyable and movable. 'ScAction'
      is inherited from 'ScAddr'.

  // 'ScActionInitiatedEvent' class is
      event type on which the given
      agent is triggered. It is
      encapsulate information about sc-
      event. The provided event is event
       on which the agent is triggered
      right now. It has methods to get
      information about initiated sc-
      event: 'GetUser', 'GetArc', '
      GetSubscriptionElement', '
      GetArcSourceElement'.

  // Main logic of agent...

```

```
  // The action state must be specified
      at all ends of the agent program.
      'FinishSuccessfully' sets the
      action as '
      action_finished_successfully'. The
       'ScResult' object cannot be
      generated via a constructor
      because it is private.
  return action.FinishSuccessfully();
}
```

Listing 7. Implementation of DoProgram method

The `ScAgent` class has a field `m_context`, an object of the `ScAgentContext` class, which can be used to complete operations in `sc-memory`. The `ScAgent` class also has a field `m_logger`, an object of the `ScLogger` class, for logging code.

If sc-exceptions are not caught in `DoProgram`, then `sc-machine` will catch them, finish the action with an error, and issue a warning about it.

Handling action arguments

Various methods are available for retrieving action arguments to simplify code (listings 8 and 9).

```
ScResult MyAgent::DoProgram(
  ScActionInitiatedEvent const & event,
      ScAction & action)
{
  auto [argAddr1, argAddr] = action.
      GetArguments();

  // Some logic...

  return action.FinishSuccessfully();
}
```

Listing 8. Retrieving action arguments via GetArguments method

```
ScResult MyAgent::DoProgram(
  ScActionInitiatedEvent const & event,
      ScAction & action)
{
  ScAddr const & argAddr1 = action.
      GetArgument(ScKeynodes::rrel_1);
  // Parameter has ScAddr type.

  // Some logic...

  return action.FinishSuccessfully();
}
```

Listing 9. Retrieving action arguments via GetArgument method

Retrieving action arguments

```
ScResult MyAgent::DoProgram(
  ScActionInitiatedEvent const & event,
      ScAction & action)
{
  ScAddr const & argAddr1 = action.GetArgument
      (1); // size_t
  // This would be the same if ScKeynodes::
      rrel_1 were passed instead of 1.

  // Some logic...

  return action.FinishSuccessfully();
}
```

Listing 10. Retrieving action argument by position

```
ScResult MyAgent::DoProgram(
  ScActionInitiatedEvent const & event,
      ScAction & action)
{
  ScAddr const & argAddr1
    = action.GetArgument(1, MyKeynodes::
        default_text_link);
  // If the action does not have the first
      argument, MyKeynodes::default_text_link
      will be returned.

  // Some logic...

  return action.FinishSuccessfully();
}
```

Listing 11. Retrieving action argument with a default value

Using ScAction as ScAddr

```
ScResult MyAgent::DoProgram(
  ScActionInitiatedEvent const & event,
      ScAction & action)
{
  // The ScAction object can be used as ScAddr.
  ScIterator3Ptr const it3 = m_context.
      CreateIterator3(action, ..., ...);

  // Some logic...

  return action.FinishSuccessfully();
}
```

Listing 12. Using ScAction as ScAddr

Handling action result

```
ScResult MyAgent::DoProgram(
  ScActionInitiatedEvent const & event,
      ScAction & action)
{
  // Some logic...

  action.FormResult(foundAddr1, generatedAddr1,
       ...);
  // Or the 'UpdateResult' method can be used
      for this.
  return action.FinishSuccessfully();
}
```

Listing 13. Forming an action result

```
ScResult MyAgent::DoProgram(
  ScActionInitiatedEvent const & event,
      ScAction & action)
{
  // Some logic...

  action.SetResult(structureAddr);
  return action.FinishSuccessfully();
}
```

Listing 14. Setting an action result

Handling action finish state

```
ScResult MyAgent::DoProgram(
  ScActionInitiatedEvent const & event,
      ScAction & action)
{
  // Some logic...

  if (/* case 1 */)
    return action.FinishSuccessfully();
```

```
  else if (/* case 2 */)
    return action.FinishUnsuccessfully();
  else
    return action.FinishWithError();
}
```

Listing 15. Finishing an action with different statuses

```
ScResult MyAgent::DoProgram(
  ScActionInitiatedEvent const & event,
      ScAction & action)
{
  action.IsInitiated(); // result: true
  action.IsFinished(); // result: false
  action.IsFinishedSuccessfully(); // result:
      false

  // Some logic...

  return action.FinishSuccessfully();
}
```

Listing 16. Checking action status

Optional methods

GetAbstractAgent

This method searches for an abstract agent for an agent of the specified class. If the agent implementation for this agent class is not included in any abstract sc-agent, `GetAbstractAgent` will throw a `utils::ExceptionItemNotFound`.

```
ScAddr MyAgent::GetAbstractAgent() const
{
  // A valid sc-address of the abstract agent
      must be specified here. Otherwise, the
      given sc-agent cannot be subscribed to an
       sc-event.
  return MyKeynodes::my_abstract_agent;
}
```

Listing 17. Overriding the GetAbstractAgent method

Remember, if only this method and the required methods are overridden, other getters will return elements of the specification for the specified abstract agent. All non-overridden getters call `GetAbstractAgent`.

GetEventClass

This method searches for the sc-event class to which the agent class is subscribed. It will throw a `utils::ExceptionItemNotFound` if the abstract sc-agent for this agent class does not have a primary initiation condition.

```
ScAddr MyAgent::GetEventClass() const
{
  // It is necessary to specify a valid sc-
      address of the event class. Otherwise,
      the given sc-agent cannot be subscribed
      to an sc-event.
  return ScKeynodes::
      sc_event_after_generate_outgoing_arc;
}
```

Listing 18. Overriding the GetEventClass method

GetEventSubscriptionElement

This method searches for the sc-event subscription sc-element that initiates the sc-event. It will throw a `utils::ExceptionItemNotFound` if the abstract sc-agent for this agent class does not have a primary initiation condition.

```
ScAddr MyAgent::GetEventSubscriptionElement()
    const
{
  // It is necessary to specify a valid sc-
      address of the sc-event subscription sc-
      element. Otherwise, the given sc-agent
      cannot be subscribed to an sc-event.
  return ScKeynodes::action_initiated;
}
```

Listing 19. Overriding the GetEventSubscriptionElement method

Do not override `GetEventClass` and `GetEventSubscriptionElement` for agents with statically specified sc-event types. Such code cannot be compiled. Override them if agent class inherits from `ScAgent<ScElementaryEvent>` (`ScElementaryEventAgent`).

`ScModule`

This class is a base class for subscribing/unsubscribing agents to/from sc-events. It's like a complex component that contains connected agents.

To subscribe agents to sc-events, implement module class (listing 20) and call `Agent` methods to subscribe agents (listing 21).

```
// File my_module.hpp
#pragma once

#include <sc-memory/sc_module.hpp>

class MyModule : public ScModule
{
};
```

Listing 20. Definition of module class inheriting ScModule class

```
// File my_module.cpp:
#include "my-module/my_module.hpp"

#include "my-module/keynodes/my_keynodes
    .hpp"
#include "my-module/agent/my_agent.hpp"

SC_MODULE_REGISTER(MyModule)
  // It initializes static object of '
      MyModule' class that can be used
      to call methods for subscribing
      agents to sc-events.
  ->Agent<MyAgent>();
  // This method subscribes the agent
      and returns an object of MyModule.
       MyAgent is inherited from
      ScActionInitiatedAgent. It points
      to the module to which the agent
      class MyAgent should be subscribed
       to the sc-event of adding an
      outgoing sc-arc from the sc-
      element action_initiated. This is
      a default parameter in this method
       for subscribing agent classes
      inherited from
      ScActionInitiatedAgent.
```

Listing 21. Subscribing agent via module class

The `Agent` method should be called without arguments for agent classes that inherit from `ScActionInitiatedAgent`. However, for agent classes that inherit from `ScAgent`, the `Agent` method should be called while providing an sc-event subscription sc-element.

A module subscribes agents when the sc-memory initializes and it unsubscribes them when the sc-memory shutdowns.

Additionally, a module can be used to subscribe a set of agents (see listing 22).

```
// File my_module.cpp:
#include "my-module/my_module.hpp"

#include "my-module/agent/my_agent1.hpp"
#include "my-module/agent/my_agent2.hpp"
#include "my-module/agent/my_agent3.hpp"
#include "my-module/agent/my_agent4.hpp"
#include "my-module/agent/my_agent5.hpp"

SC_MODULE_REGISTER(MyModule)
  ->Agent<MyAgent1>()
  ->Agent<MyAgent2>()
  ->Agent<MyAgent3>()
  ->Agent<MyAgent4>()
  ->Agent<MyAgent5>()
  // ...
  ;
```

Listing 22. Subscribing several agents via module class

If initialization of non-agent objects in a module is required, the `Initialize` and `Shutdown` methods can be overridden in the module class (see listings 23 and 24).

```
// File my_module.hpp:
class MyModule : public ScModule
{
+ void Initialize(ScMemoryContext * context) override;
+ void Shutdown(ScMemoryContext * context) override;
};
```

Listing 23. Definition of module class with overriding Initialize and Shutdown methods

```
// File my_module.cpp:
#include "my-module/my_module.hpp"

#include "my-module/agent/my_agent.hpp"

SC_MODULE_REGISTER(MyModule)
  ->Agent<MyAgent>();

+ // This method will be called once.
+ void MyModule::Initialize(
    ScMemoryContext * context)
+ {
+   // Implement initialize of non-agent objects here.
+ }
+ // This method will be called once.
+ void MyModule::Shutdown(
    ScMemoryContext * context)
+ {
+   // Implement shutdown of non-agent objects here.
+ }
```

Listing 24. Implementation of module class with overriding Initialize and Shutdown methods

E. Dynamic agent specification

Modules allow to subscribe agents with dynamic specification provided in knowledge base or in code. Dynamic specification can be changed by other agents.

The `ScModule` class includes the `AgentBuilder` method. This method can be called with an agent class, providing the keynode of the agent implementation specified in the knowledge base, or by calling methods after this to set the specification elements for the given agent [50].

`ScAgentBuilder`

The `AgentBuilder` method creates object of `ScAgentBuilder` class that is needed to initialize agent specification from code or from knowledge base.

Loading initial agent specification in C++.

An initial specification for an agent class can be defined in code using the `ScAgentBuilder` (see listing 25).

```
// File my_module.cpp:
#include "my-module/my_module.hpp"

#include "my-module/agent/my_agent.hpp"

SC_MODULE_REGISTER(MyModule)
  ->AgentBuilder<MyAgent>()
    // Abstract agent must belong to 'abstract_sc_agent'.
    ->SetAbstractAgent(MyKeynodes::my_abstract_agent)
    ->SetPrimaryInitiationCondition({
        // Event class must belong to 'sc_event'.
        ScKeynodes::sc_event_after_generate_outgoing_arc,
        ScKeynodes::action_initiated
    })
    // The action class should be one of the following types:
    // 'receptor_action', 'effector_action', 'behavioral_action' or
    // 'information_action'.
    ->SetActionClass(MyKeynodes::my_action_class)
    ->SetInitiationConditionAndResult({
        MyKeynodes::my_agent_initiation_condition_template,
        MyKeynodes::my_agent_result_condition_template
    })
    ->FinishBuild();
```

Listing 25. Definition of initial agent specification

So, the initial specification for an agent can be loaded into the knowledge base from the code. It can be modified or left unchanged, depending on the specific problem.

If a specification for an agent already exists in the knowledge base, no new connections will be generated, i.e., there will be no duplicates. All provided arguments must be valid; otherwise, the module will not be subscribed, as errors will occur. If the specification for an agent is not already in the knowledge base, all the methods listed after the `AgentBuilder` call must be invoked.

At the end of the list following the `AgentBuilder` call, the `FinishBuild` method must be called; otherwise, the code cannot be compiled.

Loading agent specification from knowledge base.

If a specification for an agent exists in the knowledge base, written in *SCs-code* or *SCg-code*, then the implementation of the agent can be specified.

Write the scs-specification (see listings 26) (or scg-specification (figure 3)) for the agent and use it to subscribe the agent within a module (listing 27).

```
// File my_module.cpp:
#include "my-module/my_module.hpp"

#include "my-module/agent/my_agent.hpp"

SC_MODULE_REGISTER(MyModule)
  ->AgentBuilder<MyAgent>(ScKeynodes::
      my_agent_implementation)
    ->FinishBuild();
```

Listing 27. Subscribing agent with dynamic specification within ScModule class

If the specification of an agent is not complete in the knowledge base, the module will not be subscribed, as errors will occur. Other correctly specified agents will be subscribed without errors.

F. Semi-dynamic agent specification

Semi-dynamic agent specification is a hybrid approach that combines the advantages of both static and dynamic agent specifications within the *OSTIS Platform*. In this approach, part of the agent's specification is stored in the knowledge base and can be modified at runtime, while another part is defined directly in the agent's source code by overriding public getter methods.

Key features of semi-dynamic specification are:

1) *Partial storage in the knowledge base.* Some specification elements (such as initiation conditions or key sc-elements) are defined in the knowledge base, allowing them to be analyzed and modified by other agents during system operation.
2) *Partial implementation in code.* Other specification elements (such as the action class or the agent's program) are implemented directly in the agent's code, providing fast access and execution.

When to use semi-dynamic specification:

1) When certain aspects of agent behavior require high performance, while others need to be adaptable at runtime.
2) In systems where some specification elements are frequently accessed and should be retrieved quickly, while others may change.
3) For incremental migration from static to dynamic specification.

Implementation example

```
// File my_agent.hpp
#pragma once

#include <sc-memory/sc_agent.hpp>

class MyAgent : public ScAgent<
    ScEventAfterGenerateIncomingArc<
    ScType::ConstPermPosArc>>
{
public:
  // Static part: action class is
      defined in code
  ScAddr GetActionClass() const override
  {
    return MyKeynodes::my_action;
  }

  // Static part: agent program
      implemented in code
  ScResult DoProgram(ScAction & action)
      override
  {
    // Agent logic implementation
    return action.FinishSuccessfully();
  }

  // Dynamic part: initiation condition
      retrieved from knowledge base
  ScAddr GetInitiationCondition() const
      override
  {
    ScAddr abstractAgent =
        GetAbstractAgent();
    if (!abstractAgent.IsValid())
      return ScAddr::Empty;
    ScAddr result =
        FindInitiationConditionInKB(
        abstractAgent);
    return result.IsValid() ? result :
        ScAgent::GetInitiationCondition
        ();
  }

  // Other methods can be similarly
      implemented
};
```

Listing 28. Example of a semi-dynamic agent specification

Subscribing agent with semi-dynamic specification

```
// File my_module.cpp
#include "my-module/my_module.hpp"
#include "my-module/agent/my_agent.hpp"

SC_MODULE_REGISTER(MyModule)
  ->AgentBuilder<MyAgent>(MyKeynodes::
      my_agent_implementation)
    // Only dynamic parts of the
        specification are set here
    ->SetInitiationConditionAndResult({
        MyKeynodes::my_agent_initiation
            _condition_template,
        MyKeynodes::my_agent_result
            _condition_template
    })
    ->FinishBuild();
```

Listing 29. Agent subscribing with semi-dynamic specification

```
// Specification of agent in knowledge base.
my_abstract_agent
<- abstract_sc_agent;
=> nrel_primary_initiation_condition:
    (sc_event_after_generate_outgoing_arc => action_initiated);
=> nrel_sc_agent_action_class:
    my_action_class;
=> nrel_initiation_condition_and_result:
    (my_agent_initiation_condition_template
        => my_agent_result_condition_template);
<= nrel_sc_agent_key_sc_elements:
{
    action_initiated;
    my_action_class;
    my_class
};
=> nrel_inclusion:
    my_agent_implementation
    (*
        <- platform_dependent_abstract_sc_agent;;
        <= nrel_sc_agent_program:
        {
            [github.com/path/to/agent/sources]
            (* => nrel_format: format_github_source_link;; *)
        };;
    *);;

my_agent_initiation_condition_template
= [*
    my_action_class _-> .._action;;
    action_initiated _-> .._action;;
    .._action _-> rrel_1:: .._parameter;;
*];;

my_agent_result_condition_template
= [*
    my_class _-> .._my_node;;
*];;
```

Listing 26. An example of dynamic agent specification represented in SCs-code

G. Comparative analysis of types of agent specifications

Agent specification in the *OSTIS Platform* can be implemented in three principal ways: static, dynamic, and semi-dynamic. Each approach offers distinct advantages and trade-offs, making them suitable for different development scenarios.

Static agent specification is defined entirely in the agent's source code by overriding public getter methods of the `ScAgent` or `ScActionInitiatedAgent` classes. This method ensures maximum performance and predictability, as the specification is hardcoded and not affected by changes in the knowledge base. It is ideal for agents whose behavior is fixed and does not require runtime adaptation. However, modifying the agent's behavior requires changes to the source code and recompilation, which can limit flexibility in dynamic environments.

Dynamic agent specification is stored in the knowledge base, either defined directly in SCs/SCg-code or loaded via the `ScAgentBuilder` API. This approach allows agent specifications to be modified at runtime by other agents or system components, supporting greater flexibility and adaptability. It is particularly useful in systems where agent behavior must evolve in response to changing data or requirements. The trade-off is a potential performance overhead due to the need to query the knowledge base for specification details, and the increased complexity of ensuring that all specification elements are correctly defined and synchronized.

Semi-dynamic agent specification combines elements of both static and dynamic approaches. Some parts of the specification – typically those that are performance-critical or rarely changed – are defined in code, while others are stored in the knowledge base and can be modified at runtime. This hybrid method offers a balance between performance and flexibility, enabling developers to optimize access to critical specification elements while still allowing for dynamic adaptation where needed. However, it introduces additional complexity, as developers must carefully manage the division between static and dynamic components to avoid inconsistencies.

The implementation of agent specifications in the *OSTIS Platform* establishes a clear mapping between semantic relations in the knowledge base and program interfaces. Table IV demonstrates this correspondence.

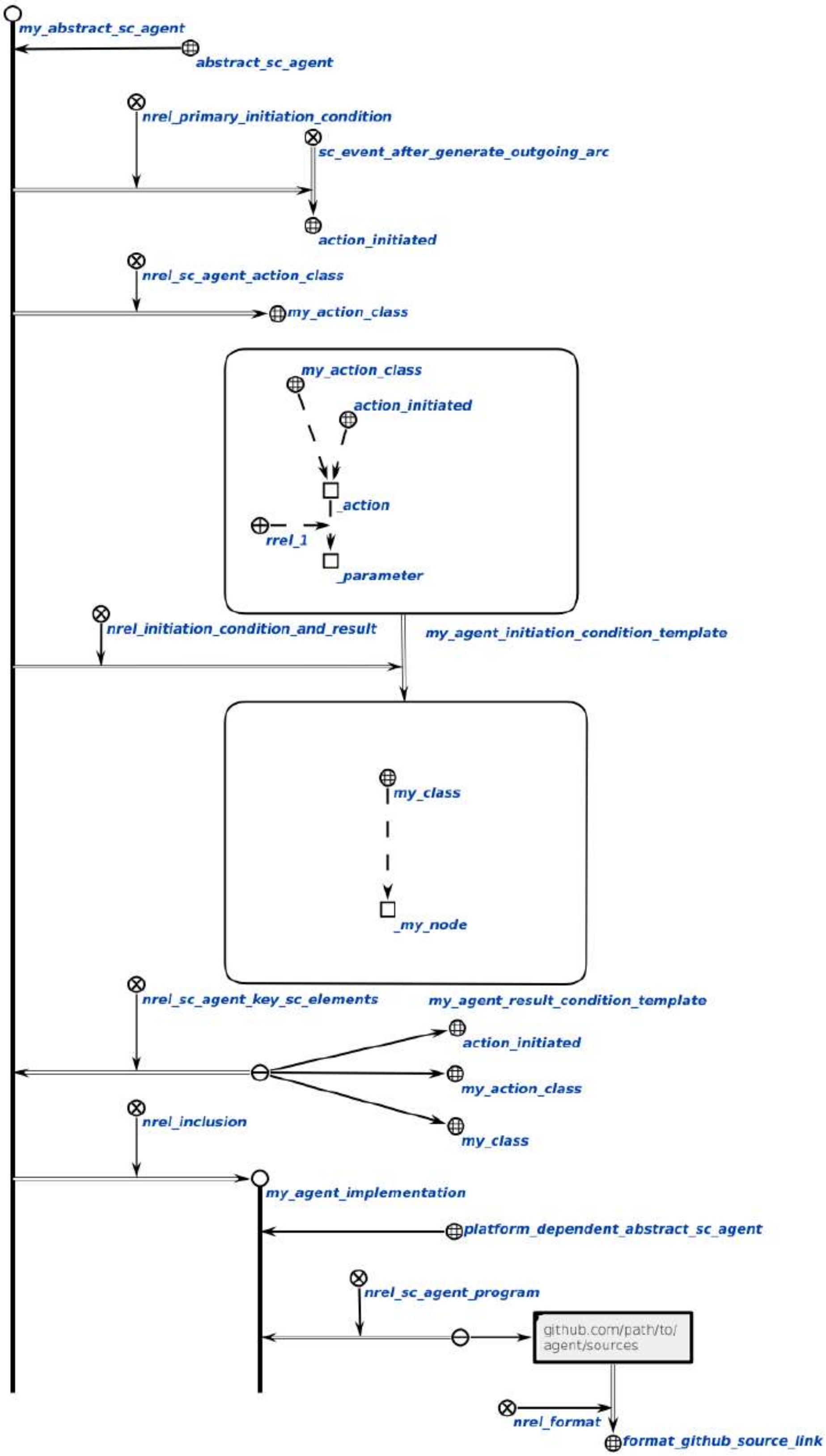

Figure 3. An example of dynamic agent specification represented in SCg-code

Table V summarizes the main characteristics of each specification type.

VIII. Conclusion

The *OSTIS Platform* marks a major step forward in intelligent agent frameworks by introducing an architecture grounded in semantic networks, effectively overcoming key limitations of current approaches.

By establishing a unified semantic foundation, *OSTIS Platform* enables integration of diverse problem-solving methods, supporting the development of advanced AI systems that can tackle complex, cross-domain challenges beyond the reach of conventional frameworks.

The sc-machine, as the central component of the *OSTIS Technology*, delivers critical features including unified knowledge representation, agent-based processing, event-driven workflow, and extensible APIs. This infrastructure supports the development of intelligent agents that can operate autonomously within a shared semantic environment, reacting to events and executing tasks based on their specifications.

The event-driven model in *OSTIS Platform* allows agents to process semantic constructions by responding to specific events in sc-memory, supporting decentralized and independent knowledge processing. Agents can be added or removed without disrupting others, ensuring system robustness and flexibility.

The agent-driven model enhances adaptability and autonomy by enabling agents to exchange messages via shared memory, fostering collaborative decision-making and dynamic response to new situations.

The flexible specification system for agents – offering static, dynamic, and semi-dynamic approaches – provides developers with multiple options for implementing agents according to their specific requirements. This flexibility, combined with the platform's robust API for creating, managing, and integrating agents, makes *OSTIS Platform* a versatile framework suitable for diverse AI applications.

As the field of artificial intelligence continues to evolve, the *OSTIS Platform* stands as a promising foundation for developing next-generation intelligent systems.

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] A. Kapoor *et al.*, "AI Agents: Evolution, Architecture, and Real-World Applications," *arXiv preprint arXiv:2503.12687*, 2024, comprehensive review of AI agent architectures, knowledge representation, evaluation, and real-world applications. [Online]. Available: https://arxiv.org/abs/2503.12687

[2] Zep AI, "LangGraph Tutorial: Building LLM Agents with LangChain's Agent Framework," 2024, tutorial and conceptual overview of LangGraph's architecture and features. [Online]. Available: https://www.getzep.com/ai-agents/langgraph-tutorial

[3] Qingyun Wu and others, "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation," *arXiv preprint arXiv:2308.08155*, 2023, foundational paper on AutoGen's multi-agent conversational framework for LLM applications. [Online]. Available: https://arxiv.org/abs/2308.08155

[4] WorkOS, "Top AI Agent Frameworks and Platforms in 2025," *WorkOS Blog*, 2025, feature analysis and evaluation of CrewAI among leading frameworks. [Online]. Available: https://workos.com/blog/top-ai-agent-frameworks-and-platforms-in-2025

[5] AIM Research, "LlamaIndex is Building AI Agents That Actually Understand Your Data," *AIM Research*, 2025, in-depth look at LlamaIndex's architecture, data integration, and enterprise applications. [Online]. Available: https://aimresearch.co/ai-startups/llamaindex-is-building-ai-agents-that-actually-understand-your-data

[6] Microsoft Developer Blogs, "The Future of AI: Customizing AI Agents with the Semantic Kernel Agent Framework," 2025, explains agent orchestration and multi-agent system construction with Semantic Kernel. [Online]. Available: https://devblogs.microsoft.com/semantic-kernel/the-future-of-ai-customizing-ai-agents-with-the-semantic-kernel\hyphenation-agent-framework/

[7] Y. Zhang *et al.*, "Review of Autonomous Systems and Collaborative AI Agent Frameworks," *SSRN Electronic Journal*, 2025, in-depth review and comparison of frameworks such as LangGraph, CrewAI, OpenAI Swarm, AutoGen, and IBM Watsonx.AI. [Online]. Available: https://papers.ssrn.com/sol3/Delivery.cfm/5142205.pdf?abstractid=5142205&mirid=1

[8] World Journal of Advanced Engineering and Technology Sciences, "A Comprehensive Review of AI Agent Frameworks, Challenges and Opportunities," *WJAETS*, 2024, comparative analysis of LangGraph, CrewAI, AutoGen, and other frameworks; discusses strengths, limitations, and enterprise use cases. [Online]. Available: https://journalwjaets.com/sites/default/files/fulltext_pdf/WJAETS-2025-0071.pdf

[9] Turing.com, "A Detailed Comparison of Top 6 AI Agent Frameworks in 2025," 2025, feature-by-feature comparison of LangGraph, LlamaIndex, CrewAI, Semantic Kernel, AutoGen, and OpenAI Swarm. [Online]. Available: https://www.turing.com/resources/ai-agent-frameworks

[10] AI21 Labs, "12 AI Agent Frameworks for Enterprises in 2025," 2025, comparison and selection criteria for enterprise AI agent frameworks. [Online]. Available: https://www.ai21.com/blog/ai-agent-frameworks/

[11] 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

[12] ——, "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

[13] IBM Think, "AI Agents in 2025: Expectations vs. Reality," *IBM Insights*, 2025, overview of the state, expectations, and enterprise adoption of AI agents in 2025. [Online]. Available: https://www.ibm.com/think/insights/ai-agents-2025-expectations-vs-reality

[14] BitMart Research, "AI Agents: 2024 Status and 2025 Outlook," *BitMart Research Reports*, 2025, comprehensive report on the evolution of AI agents, multi-agent ecosystems, and future trends. [On-

Relation Identifier	ScAgent (ScActionInitiatedAgent) Method	ScAgentBuilder Method
`nrel_primary_initiation_condition`	`ScAddr GetEventClass() const` and `ScAddr GetEventSubscriptionElement() const`	`ScAgentBuilder * SetPrimaryInitiationCondition( std::tuple<ScAddr, ScAddr> const & primaryInitiationCondition) noexcept`
`nrel_sc_agent_action_class`	`ScAddr GetActionClass() const`	`ScAgentBuilder * SetActionClass(ScAddr const & actionClassAddr) noexcept`
`nrel_initiation_condition_and_result`	`ScAddr GetInitiationCondition() const` and `ScAddr GetResultCondition() const`	`ScAgentBuilder * SetInitiationConditionAndResult( std::tuple<ScAddr, ScAddr> const & initiationConditionAndResult) noexcept`
`nrel_sc_agent_key_sc_elements`	–	–
`nrel_sc_agent_program`	`ScResult DoProgram(ScAction & action)`	–
`nrel_inclusion`	–	–

Table IV
AGENT SPECIFICATION RELATIONS AND CORRESPONDING API METHODS

Specification attribute	Static agent specification	Dynamic agent specification	Semi-dynamic agent specification
Definition location	In the agent's class (by overriding public getters of the `ScAgent` or `ScActionInitiatedAgent` classes).	In the knowledge base or initially in code (using the API of `ScModule` class and the API of the `ScAgentBuilder` class) but automatically saved into the knowledge base.	In the knowledge base or initially in code (using the API of `ScModule` class and the API of the `ScAgentBuilder` class) and supplemented externally (via overriding public getters of `ScAgent` or `ScActionInitiatedAgent` classes).
Persistence	Not stored in the knowledge base.	Stored in the knowledge base.	Partially stored in the knowledge base, partially defined in the code.
Mutability	Changes in the knowledge base do not affect the specification, as it is defined in the code.	Other agents can modify the specification.	Some parts of the specification can be changed dynamically, others are defined in the code.
Use Case	• Implementing an agent in C++ for the first time. • Minimizing the number of searches in the knowledge base.	Analyzing and modifying the specification by other agents.	Changing some parts of this specification, while allowing other parts of specification to have fast access.
Implementation method	Overriding public getters of the `ScAgent` or `ScActionInitiatedAgent` classes.	Using the API of the `ScModule` class and the API of the `ScAgentBuilder` class.	Combination of defining the specification in the knowledge base and overriding public getters of agent classes.
Example scenario	An agent that always performs the same task with predefined parameters.	An agent that changes its behavior depending on the data in the knowledge base.	An agent that uses part of the specification from the knowledge base and defines part in the code for optimization.
Key characteristics	Requires overriding the `GetActionClass` and `DoProgram` methods.	Provides the ability to analyze and modify the agent's specification by other agents.	Combines the advantages of static and dynamic specifications.
Event handling	The agent reacts to events in the knowledge base, checks the initiation condition, generates and executes an action.	The agent reacts to events in the knowledge base based on its dynamically changing specification.	The agent reacts to events based on a combination of static and dynamic specifications.
Applicability	When the agent specification should not change dynamically and is defined in the code.	When the agent specification should be able to change dynamically during system operation.	When it is needed to change some parts of the specification, but quick access to other parts defined in the code is required.

Table V
AGENT SPECIFICATION TYPES

line]. Available: https://medium.com/@BitMartResearch/ai-agent-current-status-in-2024-and-outlook-for-2025-9b9f8492d9db
[15] Sifted, "AI agents (2025)," *Sifted Pro Briefings*, 2025, market analysis and future outlook for AI agents across industries. [Online]. Available: https://sifted.eu/pro/briefings/ai-agents-2025
[16] R. Patel *et al.*, "Understanding Agentic Frameworks in AI Development: A Technical Analysis," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 2025, technical analysis of agentic frameworks, architectural components, and industry applications. [Online]. Available: https://ijsrcseit.com/index.php/home/article/view/CSEIT25111249
[17] T. Grenawalt, "Agentic AI Architecture: A Deep Dive," *Markovate Research*, 2025, explores the building blocks and layered architecture of agentic AI systems, detailing perception, cognitive, orchestration, and governance modules. [Online]. Available: https://markovate.com/blog/agentic-ai-architecture/
[18] GetStream.io, "Best 5 Frameworks To Build Multi-Agent AI Applications," 2024, overview of agent architecture, memory, tools, and reasoning in multi-agent frameworks. [Online]. Available: https://getstream.io/blog/multiagent-ai-frameworks/
[19] SmythOS, "AI Agent Frameworks: A Comprehensive Guide," 2025, explains the core components and design principles of AI agent frameworks. [Online]. Available: https://smythos.com/ai-integrations/tool-usage/ai-agent-frameworks/
[20] Z. Duan, "Agent AI with LangGraph: A Modular Framework for Enhancing Machine Translation Using Large Language Models," *arXiv preprint arXiv:2412.03801*, 2024, explores LangGraph's modular, graph-based orchestration for AI agents and its integration with LLMs. [Online]. Available: https://arxiv.org/abs/2412.03801
[21] LangChain Team, "Top 5 LangGraph Agents in Production 2024," *LangChain Blog*, 2025, case studies and best practices for deploying LangGraph agents in real-world applications. [Online]. Available: https://blog.langchain.dev/top-5-langgraph-agents-in-production-2024/
[22] MarkTechPost, "Creating An AI Agent-Based System with LangGraph: A Beginner's Guide," *MarkTechPost*, 2025, detailed technical introduction to LangGraph, its architecture, and use cases. [Online]. Available: https://www.marktechpost.com/2025/01/29/creating-an-ai-agent-based-system-with-langgraph-a-beginners-guide/
[23] Towards AI, "Building LLM Agents with LangGraph #1: Introduction to LLM Agents & LangGraph," *Towards AI*, 2025, step-by-step guide to building LLM agents and workflows with LangGraph. [Online]. Available: https://pub.towardsai.net/building-llm-agents-with-langgraph-1-introduction-to-llm-agents\hyphenation-langgraph-d94648aad62f
[24] Stackademic, "Navigating the AI Agent Landscape: In-Depth Analysis of Autogen, CrewAI, LlamaIndex, and LangChain," *Stackademic Blog*, 2024, comprehensive analysis of AutoGen's multi-agent conversational architecture and integration capabilities. [Online]. Available: https://blog.stackademic.com/navigating-the-ai-agent-landscape-in-depth-analysis-of-autogen\hyphenation-crewai-llamaindex-and-langchain-2a3bcd932abc
[25] Shakudo, "Top 9 AI Agent Frameworks as of April 2025," *Shakudo Blog*, 2025, comparative review including AutoGen's strengths, architecture, and use cases. [Online]. Available: https://www.shakudo.io/blog/top-9-ai-agent-frameworks
[26] DataCamp, "CrewAI: A Guide With Examples of Multi AI Agent Systems," 2024, overview and practical guide to CrewAI's role-based, multi-agent system features and architecture. [Online]. Available: https://www.datacamp.com/tutorial/crew-ai
[27] LlamaIndex Team, "A Powerful Framework for Building Production Multi-Agent AI Systems," 2024, introduction to llama-agents, distributed service-oriented architecture, and orchestration flows.
[28] Visual Studio Magazine, "Semantic Kernel Agent Framework Graduates to Release Candidate," 2025, news and technical summary of Semantic Kernel's agent framework, plugins, and enterprise integration. [Online]. Available: https://visualstudiomagazine.com/Articles/2025/03/04/Semantic-Kernel-Agent-Framework-Graduates-to-Release-Candidate.aspx
[29] N. Zotov, "Semantic Theory of Programs in Next-Generation Intelligent Computer Systems," in *Open Semantic Technologies for Intelligent Systems (OSTIS-2022): Collection of Scientific Papers*. Minsk, Belarus: Belarusian State University of Informatics and Radioelectronics, 2022, pp. 145–160. [Online]. Available: https://libeldoc.bsuir.by/handle/123456789/49326
[30] 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
[31] 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
[32] M. Orlov, "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
[33] OSTIS-AI, "OSTIS-AI: Open Semantic Technology for Intelligent Systems," 2025, accessed: 15.03.2025. [Online]. Available: https://ostis-ai.github.io
[34] 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
[35] 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.
[36] 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
[37] V. Ivashenko, "Semantic space integration of logical knowledge representation and knowledge processing models," in *Open Semantic Technologies for Intelligent Systems: Research Papers Collection*, vol. 7. Minsk: Belarusian State University of Informatics and Radioelectronics, 2023, pp. 95–114. [Online]. Available: https://libeldoc.bsuir.by/bitstream/123456789/51250/3/Ivashenko_Semantic.pdf
[38] 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.
[39] M. Orlov, A. Makarenko, and K. Petrochuk, "Current State of OSTIS-systems Component Design Automation Tools," in *Open Semantic Technologies for Intelligent Systems: Research Papers Collection*, vol. 8. Minsk: Belarusian State University of Informatics and Radioelectronics, 2024, pp. 49–62.
[40] 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

[41] 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
[42] 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
[43] 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
[44] N. Zotov, “A formal model of shared semantic memory for next-generation intelligent systems,” in *Open Semantic Technologies for Intelligent Systems: Research Papers Collection*, vol. 8. Minsk: Belarusian State University of Informatics and Radioelectronics, 2024, pp. 63–78. [Online]. Available: https://libeldoc.bsuir.by/bitstream/123456789/55531/1/A_Formal.pdf
[45] 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
[46] ——, “C++ Events API for sc-machine,” 2025, accessed: 15.03.2025. [Online]. Available: https://ostis-ai.github.io/sc-machine/sc-memory/api/cpp/extended/agents/events/
[47] ——, “C++ Agents API for sc-machine,” 2025, accessed: 15.03.2025. [Online]. Available: https://ostis-ai.github.io/sc-machine/sc-memory/api/cpp/extended/agents/agents/
[48] ——, “scp-machine: Software implementation of semantic network program interpreter,” 2025, accessed: 15.03.2025. [Online]. Available: https://github.com/ostis-ai/scp-machine
[49] ——, “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/
[50] ——, “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/
[51] ——, “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/
[52] D. V. Shunkevich, *Semantic Technologies for Designing Problem Solvers*. Minsk: Belarusian State University of Informatics and Radioelectronics, 2022. [Online]. Available: https://libeldoc.bsuir.by/handle/123456789/48018
[53] 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

ПЛАТФОРМА OSTIS – ФРЕЙМВОРК ДЛЯ РАЗРАБОТКИ ИНТЕЛЛЕКТУАЛЬНЫХ АГЕНТОВ НА БАЗЕ СЕМАНТИЧЕСКИХ СЕТЕЙ

Зотов Н.В.

Данная статья рассматривает фреймворки для создания ИИ-агентов и представляет *OSTIS Platform* как решение существующих ограничений современных подходов. В работе анализируются принципы построения ИИ-агентов, проводится оценка таких фреймворков, как LangGraph, CrewAI, AutoGen, Semantic Kernel и LlamaIndex, а также подробно описываются преимущества технологии OSTIS. К этим преимуществам относятся единая семантическая основа и глубокое представление знаний. В статье также рассматривается реализация платформы на базе Технологии OSTIS и моделей, управляемых агентами, акцентируя внимание на их потенциал в развитии интеллектуальных систем.

Received 15.03.2024