СРЕДСТВА РАЗРАБОТКИ АГЕНТОВ В ПРОГРАММНОЙ ПЛАТФОРМЕ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Зотов Н. В.
Кафедра интеллектуальных информационных технологий,
Белорусский государственный университет информатики и радиоэлектроники
Минск, Республика Беларусь
E-mail: n.zotov@bsuir.by

В работе рассмотрены программные средства разработки агентов в ostis-системах.

Введение

Развитие современных интеллектуальных систем характеризуется переходом от монолитных архитектур к модульным подходам, основанным на взаимодействии агентов. Ключевым требованием к таким системам является обеспечение семантической совместимости компонентов и возможность их динамической интеграции для решения комплексных задач.

Платформа OSTIS [1, 2] предлагает решение указанных проблем через использование единого языка представления знаний – SC-кода – и sc-памяти для всех компонентов системы. Такой подход обеспечивает унифицированную основу для создания семантически совместимых агентов, способных к эффективному взаимодействию и совместному решению задач.

I. Средства разработки агентов в OSTIS-системах

Платформа OSTIS предоставляет инструменты для создания, управления и интеграции агентов.

Агенты реагируют на события в sc-памяти. Агент активируется, когда происходит событие в sc-памяти, на которое он подписан. Основное условие инициирования определяет событие в sc-памяти, которое пробуждает агента. После пробуждения агент проверяет полное условие своего инициирования. В случае успеха он инициирует и выполняет действие, используя свою программу. После агент проверяет свой результат.

В рамках программного интерфейса для создания агентов и управления ими введены классы и методы для работы с агентами:

1. Два базовых класса для всех типов агентов:
   - класс `ScAgent` для реализации классов агентов, которые реагируют на любые элементарные события в sc-памяти;
   - класс `ScActionInitiatedAgent` для реализации агентов, которые реагируют на события инициированных действий в sc-памяти.
2. Класс `ScAgentContext` для работы с событиями `ScEvent`, подписками `ScEventSubscription` и ожидателями `ScWait` и `ScEventWaiter`.
3. Класс `ScAction` для обработки действий в sc-памяти.
4. Класс `ScAgentBuilder` для управления динамическими спецификациями агентов.
5. Класс `ScKeynodes` для указаний ключевых sc-элементов, используемых в агентах.
6. Класс `ScModule` для регистрации агентов в sc-памяти.

Пример использования класса `ScAgent` для создания агента представлен в листинге 1.

```
// Файл my_agent.hpp
#pragma once

#include <sc-memory/sc_agent.hpp>

// Класс агента должен наследовать класс
ScAgent и указывать шаблонный аргумент
как класс sc-события.
Здесь ScEventAfterGenerateIncomingArc
    <ScType::ConstPermPosArc> -
тип события, на которое реагирует агент.
class MyAgent : public ScAgent<
  ScEventAfterGenerateIncomingArc<
      ScType::ConstPermPosArc>>
{
public:
  // Здесь указывается класс действий,
  которые выполняет данный агент.
  // Здесь `GetActionClass`
  переопределяет `GetActionClass`
  в классе `ScAgent`.
  Это переопределение обязательно.
  ScAddr GetActionClass() const
      override;
  // Здесь указывается программа данного
  агента. Это переопределение обязательно.
  ScResult DoProgram(
    ScEventAfterGenerateIncomingArc<
      ScType::ConstPermPosArc> const
          & event,
    ScAction & action) override;
    // ...
    // Другие пользовательские методы.
};
```

Листинг 1 – Определение агента, наследующего класс `ScAgent`

Для подписки агентов на события необходимо реализовать модуль (листинг 2) и вызвать метод `Agent` для подписки агентов (листинг 3).

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

#include <sc-memory/sc_module.hpp>

class MyModule : public ScModule
```

```
{
};
```

Листинг 2 – Определение класса модуля, наследующего класс ScModule

```
// Файл 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)
  // Инициализирует статический объект
  класса `MyModule`, который может быть
  использован для вызова методов подписки
  агентов на sc-события.
  ->Agent<MyAgent>();
  // Этот метод подписывает агента и
  возвращает объект MyModule.
```

Листинг 3 – Подписка агента через класс модуля

Каждый агент имеет спецификацию. Спецификация агента представляет собой формализованный подход к управлению агентами.

Спецификация агента включает:

- его первичное условие инициирования,
- класс действий, которые он может выполнять,
- полное условие инициирования,
- условие результата,
- ключевые sc-элементы, используемые во время выполнения действия,
- и другие детали.

Спецификация агента может быть представлена в базе знаний с использованием SC-кода или программно с использованием программного интерфейса sc-памяти на C++.

Программный интерфейс для создания агентов и управления ими предоставляет два метода для реализации агентов: (1) когда спецификация агента представлена в базе знаний; (2) когда спецификация агента представлена непосредственно в коде.

Спецификации агентов могут быть статическими, динамическими или полудинамическими.

1. *Статическая спецификация агента* предоставляется извне в классе агента (путём переопределения публичных методов-геттеров классов `ScAgent` или `ScActionInitiatedAgent`). Она не хранится в базе знаний.
2. *Динамическая спецификация агента* хранится в базе знаний, либо определяется непосредственно на SCs/SCg-коде, либо загружается через API `ScAgentBuilder`.
3. *Полудинамическая спецификация агента* предоставляется в базе знаний или изначально в коде и дополняется извне.

Пример реализации агента с полудинамической спецификацией показан в листинге 4.

```
// Файл my_agent.hpp
#pragma once

#include <sc-memory/sc_agent.hpp>

class MyAgent : public ScAgent<
    ScEventAfterGenerateIncomingArc<
    ScType::ConstPermPosArc>>
{
public:
  // Статическая часть: класс действий
  агента задаётся в коде.
  ScAddr GetActionClass() const
      override
  {
    return MyKeynodes::my_action;
  }

  // Статическая часть: программа агента
  задаётся в коде.
  ScResult DoProgram(ScAction &
      action) override
  {
    // Реализация логики агента.
    return action.FinishSuccessfully
        ();
  }

  // Динамическая часть: условие
  инициирования агента извлекается из
  базы знаний.
  ScAddr GetInitiationCondition()
      const override
  {
    ScAddr result =
        SearchInitiationCondition(
        GetAbstractAgent());
    return result.IsValid() ? result
         : ScAgent::
        GetInitiationCondition();
  }

  // Другие методы могут быть реализованы
  аналогично.
};
```

Листинг 4 – Пример полудинамической спецификации агента

Заключение

Программный интерфейс для создания агентов и управления ими в общей семантической памяти, реализованный в рамках технологии OSTIS, является ключевым элементом программной платформы для разработки ostis-систем. Этот интерфейс обеспечивает создание, спецификацию и взаимодействие агентов, работающих в sc-памяти, что позволяет эффективно решать задачи обработки знаний и построения семантически совместимых систем.

1. Zotov, N. OSTIS Platform – a Framework for Developing Intelligent Agents Based on Semantic Networks / N. Zotov // Open Semantic Technologies for Intelligent Systems (OSTIS) : сборник научных трудов / Белорусский государственный университет информатики и радиоэлектроники ; редкол. : В. В. Голенков [и др.]. – Минск, 2025. – Вып. 9. – С. 113-134.
2. Зотов, Н. В. Модель управления процессами в общей семантической памяти интеллектуальных систем / Н. В. Зотов // Информационные технологии и системы 2023 (ИТС 2023) : материалы Международной научной конференции, Минск, 22 ноября 2023 / Белорусский государственный университет информатики и радиоэлектроники ; редкол. : Л. Ю. Шилин [и др.]. – Минск : БГУИР, 2023. – С. 53-54.