УДК 004.9 https://doi.org/10.26160/2541-8637-2024-13-43-49

РАЗРАБОТКА МУЛЬТИАГЕНТНЫХ СИСТЕМ С ИСПОЛЬЗОВАНИЕМ АГЕНТНОЙ ПЛАТФОРМЫ SPADE

Пак А.О.
МИРЭА – Российский технологический университет, Москва

Ключевые слова: мультиагентные системы, библиотека SPADE, автономные агенты, протоколы связи, поведение агентов, диспетчеризация сообщений.
Аннотация. Обсуждаются вопросы разработки мультиагентных систем с использованием библиотеки SPADE. Рассматриваются модели агентов, поддерживаемые SPADE. Обсуждается протокол XMPP, обеспечивающий обмен структурированными, расширяемыми данными между агентами в режиме реального времени. Решается задача моделирования и поведения агентов.

DEVELOPMENT OF MULTI-AGENT SYSTEMS USING THE SPADE AGENT PLATFORM

Pak A.O.
MIREA – Russian Technological University, Moscow

Keywords: multi-agent systems, SPADE, autonomous agents, communication protocols, agent behavior, agent coordination, message dispatching.
Abstract. The issues of development and implementation of multi-agent systems using SPADE library are discussed. SPADE agent models and its working mechanisms are considered. The importance of XMPP as a key elejment for communication between agents is emphasized, allowing to exchange structured, extensible data in real time. The task of modeling agent behavior, message dispatching mechanisms, and a graphical interface for managing agents is posed, which increases the usability and flexibility of the system. An example of realization of agents' communication using SPADE is given.

Введение

Технология мультиагентных систем (МАС) подразумевает систему, состоящую из самостоятельных интеллектуальных агентов, способных общаться между собой [1]. Такая коммуникация позволяет агентам сотрудничать и взаимодействовать между собой, решать сложные задачи в группе. К настоящему времени создано множество программных средств и библиотек (платформ) для разработки МАС. Обычно, такие платформы предлагают определенные средства для коммуникации агентов, внутреннюю архитектуру агентов, набор инструментов для разработки и моделирования общения агентов. Следует отметить, что большая часть ранее созданных платформ перестала поддерживаться либо были адаптированы к новым требованиям к разработкам МАС.

Агенты в МАС могут варьироваться от простых, основанных на правилах, до сложных, способных принимать сложные решения [2]. Они могут действовать автономно, учиться в окружающей среде и даже демонстрировать поведение, имитирующее человеческий интеллект, такие как переговоры, сотрудничество и конкуренция. Такая автономность и интеллект делают МАС особенно

подходящими для требующих распределенного решения задач и принятия решений [3].

Среда для разработки мультиагентных систем SPADE

SPADE (Smart Python Agent Development Environment) – это программное средство на языке Python для разработки МАС, являющееся результатом эволюции платформ разработки МАС за счет тщательного отбора концепций и современных технологий в области распределенных систем, мгновенного обмена сообщениями, асинхронных и открытых систем [4].

Один из ключевых аспектов любого промежуточного программного обеспечения для МАС заключается в его агентной модели [5]. В случае с SPADE, эта модель схожа с используемыми на других платформах, таких как JADE [6]. Модель строится на ряде базовых абстракций и механизмов. Одним из первых является механизм регистрации, который позволяет каждому агенту зарегистрироваться в SPADE, используя уникальный идентификатор (в формате "username@server" и пароль). После регистрации агенты могут создавать одно или несколько моделей поведения, обеспечивающих выполнение задач агентом. Поведения делятся на несколько типов, каждый из которых определяет определённый шаблон действий, разработанный для выполнения стандартных задач в МАС. В частности, SPADE предлагает пять типов поведения: Cyclic, One-Shot, Periodic, Timeout и Finite State Machine.

Следующий основной механизм функционирования МАС – это диспетчер сообщений, который SPADE ассоциирует с каждым зарегистрированным агентом. Этот компонент действует как почтальон, перенаправляя любое сообщение агенту, которые могут ожидать это сообщение, а исходящие сообщения от агента - к системе коммуникации SPADE.

Говоря о внешней среде, отметим, что в некоторых МАС-приложениях агенты взаимодействуют в моделируемой или виртуальной среде, которая представляет собой контекст системы. SPADE по своей сути не предоставляет модуль моделирования среды, но при необходимости его можно интегрировать с внешними инструментами моделирования.

SPADE поддерживает разработку масштабируемых МАС, которые могут адаптироваться к изменяющимся условиям и требованиям. Гибкость SPADE в сочетании с надежностью применения XMPP для связи между агентами позволяет разработчикам создавать системы, способные эффективно решать широкий спектр сложных распределенных задач.

Взаимодействие агентов и архитектура агентной системы

В SPADE используются сложные механизмы, обеспечивающие беспрепятственное взаимодействие между агентами. Они могут общаться, используя различные типы сообщений и протоколы, что позволяет им обмениваться информацией, координировать задачи и эффективно принимать коллективные решения. Этому способствует развитая система обмена сообщениями SPADE, которая поддерживает синхронный и асинхронный обмен, обеспечивая взаимодействие агентов в режиме реального времени или в течение длительного времени, в зависимости от требований приложения.

Опишем структуру агентной системы SPADE (рис. 1).

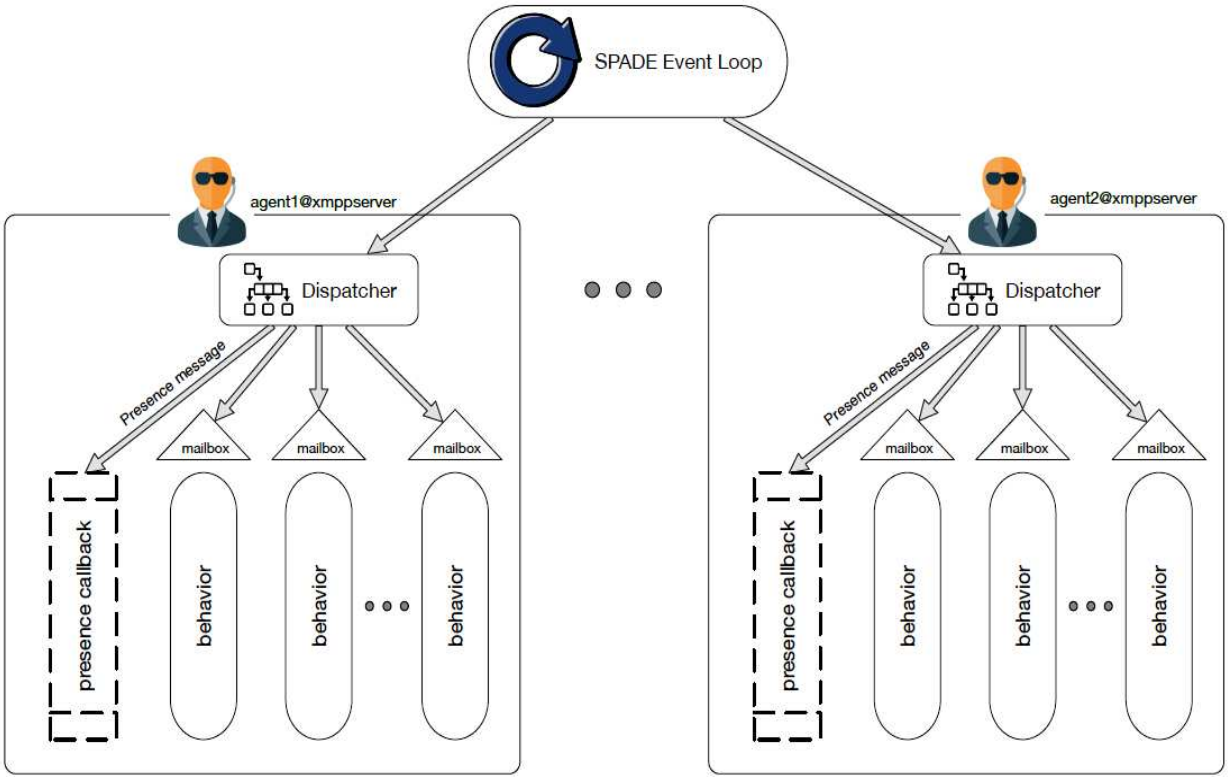


Рис. 1. Структура агентной системы SPADE

Архитектура SPADE организована вокруг центрального контура событий SPADE Event Loop – сложного механизма, который эффективно управляет событиями и передачей сообщений между агентами. Этот цикл событий действует как сердце системы, обеспечивая правильную отправку сообщений и реагирование агентов на динамичную среду MAC.

Каждый **агент** в SPADE имеет уникальный XMPP – идентификатор, что позволяет ему работать независимо, оставаясь при этом частью большой сети.

Dispatcher (коммуникационный узел агента). Каждый агент оснащен диспетчером – важнейшим компонентом, отвечающим за управление потоком сообщений. Диспетчер сортирует и определяет приоритеты входящих сообщений, гарантируя, что они будут обработаны соответствующим поведением агента. Он действует как коммуникационный узел агента, взаимодействуя с циклом событий SPADE для получения и отправки сообщений.

Behaviour (определение действий агента). Поведение – это предопределенные действия или задачи, которые может выполнять агент. SPADE позволяет реализовать целый ряд модели поведений, от простых одноразовых действий до сложных и постоянно выполняемых задач.

Mailbox (почтовый ящик). Для каждого агента служит очередью для входящих сообщений. Сообщения, отправленные агенту, будь то сообщения от других агентов или системные уведомления, хранятся в почтовом ящике до тех пор, пока диспетчер не сможет их обработать. Это гарантирует, что связь не будет потеряна даже в периоды высокой активности.

Presence callback (обратный вызов присутствия – реакция на состояния агента). Одной из отличительных особенностей SPADE является использование обратного вызова присутствия. Функция presence callback запускается в ответ на события, связанные с присутствием, такие как вход или выход агента из системы

или изменение его статуса. Presence callback позволяет агентам адаптировать свое поведение в зависимости от доступности других агентов, что способствует созданию более динамичной и системы.

Каждый агент в SPADE предоставляет графический интерфейс, пример приведен на картинке. На рисунке 2 показана страница агента, где можно посмотреть его имя, список его поведения и список его контактов.

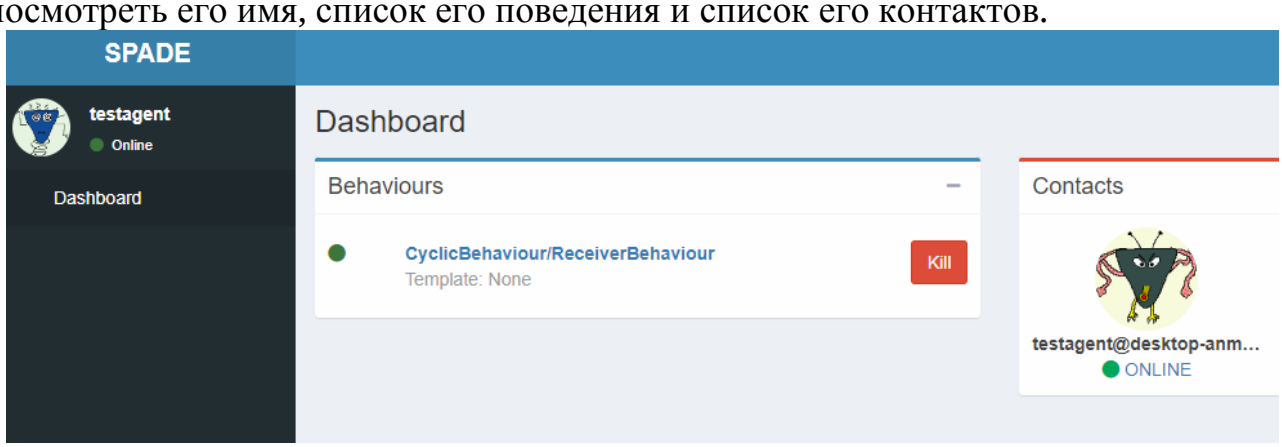


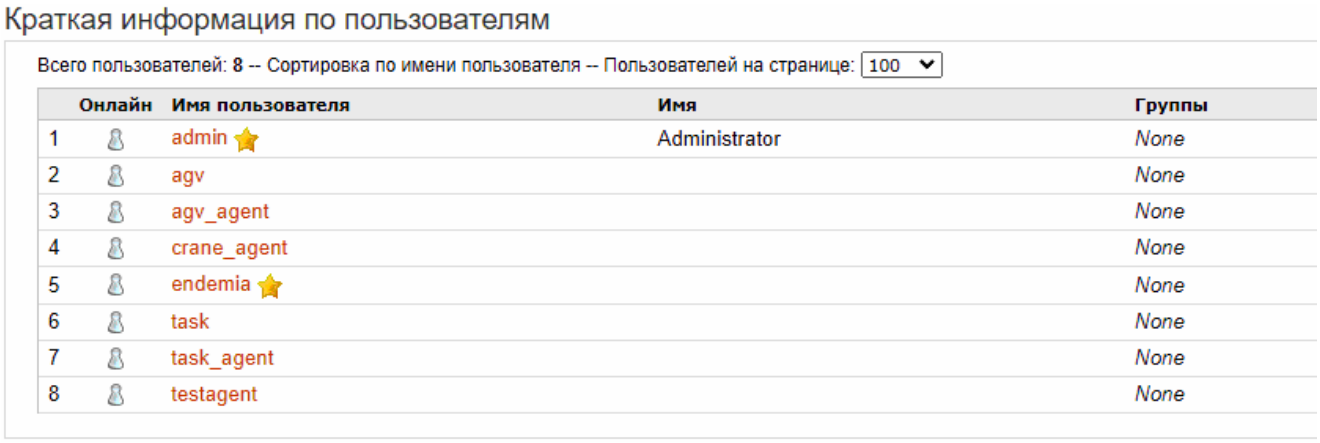
Рис. 2. Страница агента в графическом интерфейсе

XMPP-коммуникации в SPADE

Протокол XMPP (Extensible Messaging and Presence Protocol) играет ключевую роль в SPADE, обеспечивая основу для взаимодействия агентов.

XMPP, изначально разработанный для обмена мгновенными сообщениями, превратился в ключевой протокол для коммуникации между приложениями в режиме реального времени. Гибкость и расширяемость XMPP делают его идеальным выбором для коммуникационных потребностей агентов SPADE. Он позволяет обмениваться структурированными, но расширяемыми данными между агентами практически в режиме реального времени.

Аналогично с веб-интерфейсом, при настройке XMPP-сервера для обмена сообщениями между агентами некоторые доступные варианты XMPP-серверов (по типу OpenFire) позволяют пользоваться такими же функциями, как и во встроенной версии, за исключением того, что OpenFire позволяет тонко настроить систему под необходимые задачи. Пример списка пользователей сервера (включая профили для администрирования сервера) представлен на рисунке 3.



	Онлайн	Имя пользователя	Имя	Группы
1		admin	Administrator	None
2		agv		None
3		agv_agent		None
4		crane_agent		None
5		endemia		None
6		task		None
7		task_agent		None
8		testagent		None

Рис. 3. Список пользователей сервера

Пример использования SPADE

Создадим 2-х агентов для нашего проекта – EchoAgent и SenderAgent. EchoAgent – агент, который ожидает входящих сообщений, передает их эхом отправителю и выводит содержимое сообщения и ответное действие на консоль. SenderAgent (агент-отправитель): агент, который активируется после задержки, отправляет одно сообщение EchoAgent и регистрирует свое действие.

Программный код для создания агентов изображен на рисунках 4 и 5.

```
class EchoAgent(Agent):
    class EchoBehaviour(CyclicBehaviour):
        async def run(self):
            msg = await self.receive(timeout=10)
            if msg:
                print(f"EchoAgent получил сообщение: {msg.body}")
                reply = msg.make_reply()
                reply.body = f"Echo: {msg.body}"
                await self.send(reply)
                print(f"EchoAgent ответил: {reply.body}")
            else:
                print("EchoAgent не получил сообщение.")

    async def setup(self):
        self.add_behaviour(self.EchoBehaviour())
```

Рис. 4. Программный код для реализации EchoAgent

```
class SenderAgent(Agent):
    class SendMessageBehaviour(OneShotBehaviour):
        async def run(self):
            msg = Message(to="testagent@localhost")
            msg.set_metadata("performative", "inform")
            msg.body = "I am here"
            await self.send(msg)
            print("SenderAgent отправил сообщение: I am here")

    async def setup(self):
```

Рис. 5. Программный код для реализации SenderAgent

EchoAgent. В коде осуществляется определение класса агента EchoAgent. Он наследует свойства от spade.agent.Agent.

EchoBehaviour. Здесь имеем подкласс CyclicBehaviour, который непрерывно проверяет наличие входящих сообщений каждые 10 сек. При получении сообщения EchoAgent печатает его содержимое, отправляет ответ, повторяющий полученное сообщение, и регистрирует действие ответа.

Обработка сообщений. Используется await self.receive(timeout = 10) для асинхронного ожидания сообщений, демонстрируя использование паттернов асинхронного программирования для обработки взаимодействия агентов.

Механизм ответа. Использует msg.make_reply() для генерации ответного сообщения, демонстрируя встроенные в SPADE утилиты обработки сообщений для упрощения процессов взаимодействия.

SenderAgent. Определяется его класс spade.agent.Agent, обозначая его как другого агента SPADE в системе.

SendMessageBehaviour. Это подкласс OneShotBehaviour, предназначенный для выполнения единственной задачи - отправки сообщения EchoAgent. Этот тип поведение показывает, как реализовать агенты, которые выполняют определенное действие, а затем прекращают активность, что полезно для инициирования процессов или отправки уведомлений в MAC.

Асинхронная установка и выполнение. Агент ожидает 30 сек. (await asyncio.sleep(30)) перед активацией, что подчеркивает использование asyncio для синхронизации и планирования задач в асинхронной среде MAC.

Функция main. Выполняется инициализация агентов. Оба агента инстанцируются с соответствующими JID и паролями, готовые к подключению к XMPP-серверу. Программный код функции main изображен на рисунке 6.

```
async def main():
    echo_agent = EchoAgent("testagent@localhost", "password")
    sender_agent = SenderAgent("testagent2@localhost", "password")

    await echo_agent.start()
    print("EchoAgent в сети.")

    await asyncio.sleep(30)  # Ожидание 30 секунд перед запуском SenderAgent
    await sender_agent.start()
    print("SenderAgent в сети и отправил сообщение.")

    await asyncio.sleep(5)
    await echo_agent.stop()
    await sender_agent.stop()
    print("Оба агента были выключены.")


if __name__ == "__main__":
    asyncio.run(main())
```

Рис.6. Программный код функции main

Последовательное выполнение. Код демонстрирует управление жизненным циклом агентов в асинхронном контексте – запуск EchoAgent, ожидание в течение 30 секунд, затем запуск SenderAgent. Эта последовательность демонстрирует временную координацию в MAC.

После короткого периода после отправки сообщения оба агента останавливаются, иллюстрируя, как правильно завершать деятельность агентов и очищать ресурсы в приложениях SPADE.

Этот код представляет собой краткий пример создания и координации MAC с помощью SPADE, используя асинхронное программирование, реализацию поведения и коммуникацию на основе XMPP. Он дает представление

о практических аспектах разработки приложений МАС, от простого эха сообщений до выполнения задач по таймеру, в рамках надежного фреймворка SPADE. Рисунок 7 демонстрирует результат работы программы.

```
EchoAgent в сети.
EchoAgent не получил сообщение.
EchoAgent не получил сообщение.
EchoAgent не получил сообщение.
SenderAgent в сети и отправил сообщение.
SenderAgent отправил сообщение: I am here
EchoAgent получил сообщение: I am here
EchoAgent ответил: Echo: I am here
Оба агента были выключены.

Process finished with exit code 0
```

Рис. 7. Результат работы программы

Заключение

В работе показаны возможности создания МАС на основе фреймворка SPADE и их интеграции с XMPP-серверами, такими как OpenFire.

Архитектура SPADE базируется на механизме событийного цикла и автономии, предоставляемой каждому агенту с помощью уникальных XMPP-идентификаторов, и гарантирует, что проектируемые МАС способны к сложным коммуникациям и процессам принятия решений. Это обеспечивается с помощью обширного набора функций, включая разнообразные модели поведения агентов и функции диспетчеризации сообщений, что способствует созданию среды, в которой агенты могут действовать, учиться и адаптироваться автономно, достигая при этом коллективных целей.

Список литературы

1. Shi Z., Zhang J., Yue J., Yang X. A cognitive model for multi-agent collaboration // International Journal of Intelligence Science. 2013, vol. 4, no. 01, pp. 1-6. DOI: 10.4236/ijis.2014.41001.
2. Li Q., Chu H., Zhang M., Li M., Diao L. Collaboration strategy for software dynamic evolution of multi-agent system // Journal of Central South University. 2015, vol. 22, no. 7. pp. 2629-2637. DOI: 10.1007/s11771-015-2793-2.
3. Golpayegani F. Multi-agent collaboration in distributed self-adaptive systems // 2015 IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops. – 2015, pp. 146-151. DOI: 10.1109/SASOW.2015.29.
4. Dorrie A., Kanhere S., Jourdak R. Multi-agent systems: A survey // IEEE Access. 2018, vol. 6, pp. 28573-28593. DOI: 10.1109/ACCESS.2018.2831228.
5. Palanca J., Terrasa A., Julian V., Carrascosa C. Spade 3: Supporting the new generation of multi-agent systems // IEEE Access. 2020, vol. 8, pp. 182537-182549. DOI: 10.1109/ACCESS.2020.3027357.
6. Palanca J., Rincon JA, Carrascosa C., Julian V., Terrasa A. A flexible agent architecture in SPADE // International Conference on Practical Applications of Agents and Multi-Agent Systems. – Cham: Springer International Publishing, 2022. – P. 320-331. – DOI: 10.1007/978-3-031-18192-4_26.

Сведения об авторе:
Пак Александр Олегович – аспирант.