УДК 004.4'242
DOI 10.17513/snt.40324

КОМПЛЕКС АВТОМАТИЧЕСКОГО ПРОЕКТИРОВАНИЯ АГЕНТОВ НА ОСНОВЕ ИЕРАРХИЧЕСКИХ КОНЕЧНЫХ АВТОМАТОВ

Чекан М.А.

ФГБУН «Институт динамики систем и теории управления имени В.М. Матросова Сибирского отделения Российской академии наук», Иркутск, e-mail: chekoopa@mail.ru

В исследовании рассмотрены вопросы автоматизации проектирования и реализации компонентов систем моделирования, основанных на мультиагентной архитектуре. Целью исследования является разработка инструментальных средств поддержки разработки таких систем. В ходе исследования были созданы инструменты, которые упрощают процесс создания агентов для платформы Java Agent DEvelopment Framework™. Особенностью этих инструментов является использование схем иерархических машин состояний для формирования каркаса агента с модулями-шаблонами, что значительно облегчает дальнейшую реализацию функциональности. Такой подход позволяет эффективно и надежно планировать архитектуру агента, учитывая поставленные задачи. В статье представлены ключевые компоненты комплекса инструментальных средств. Главный компонент – генератор каркаса агента, работающий в двух режимах. Первый подразумевает использование специально разработанных модулей для платформы Java Agent DEvelopment Framework™, реализующих поведение агента в логике иерархических машин состояний и настраиваемых в зависимости от схемы. Второй режим формирует самодостаточный модуль класса в парадигме расширенных иерархических машин состояний с возможностью выполнения кода при возникновении событий. Предлагается визуальный редактор с интерфейсом для проектирования схем и наполнения их кодом. Результаты исследования были применены при создании мультиагентной среды для моделирования взаимодействия микросетей. Фрагменты результатов работы комплекса для одного из агентов среды приведены в качестве примера.

Ключевые слова: иерархические машины состояний, мультиагентные среды, генерация кода

Исследование проведено при поддержке Министерства науки и высшего образования Российской Федерации, проект № FWEW-2021-0005 «Технологии разработки и анализа предметно-ориентированных интеллектуальных систем группового управления в недетерминированных распределенных средах».

AUTOMATIC AGENT DESIGN SYSTEM BASED ON HIERARCHICAL FINITE AUTOMATA

Chekan M.A.

Matrosov Institute for System Dynamics and Control Theory of Siberian Branch of the Russian Academy of Sciences, Irkutsk, e-mail: chekoopa@mail.ru

The study examines the issues of design automation and implementation of components for modeling systems based on a multi-agent architecture. The purpose of the study is to develop tools to support the development of such systems. During research, tools were developed that simplify the creation of agents for the Java Agent DEvelopment Framework™ platform. These tools use hierarchical state machine diagrams to form a framework with template modules that greatly facilitates implementation of functionality. This approach allows for efficient and reliable planning of agent architectures, taking into consideration the tasks assigned. The article presents key components of the toolset. The main component is an agent frame generator that operates in two modes: the first involves using specially designed modules for Java Agent DEvelopment Framework™ that implement agent behavior according to hierarchical state machines, and are configurable based on the schema. The second mode forms a self-sufficient class module within the paradigm of extended hierarchical state machines, with the ability to execute code on events occur. A visual editor with an interface for designing and filling diagrams with code is provided. The results of the study were applied in the creation of a multi-agent environment for modeling the interaction of microgrids. Fragments of the results of the complex's operation for one of the agents of the environment are given as an example.

Keywords: hierarchical state machines, multi-agent environments, code generation

The study was supported by the Ministry of Science and Higher Education of the Russian Federation, project No. FWEW 2021 0005 "Technologies for the development and analysis of domain-oriented intelligent group control systems in non-deterministic distributed environments".

Введение

Компьютерное моделирование играет одну из важнейших ролей в исследовании и проектировании сложных технологических, социальных и экономических процессов, и с ростом масштаба и сложности современных систем моделирование и использование цифровых двойников все чаще становятся базовыми инструментами для контролируемого исследования и проведения экспериментов для оценки последствий различных сценариев [1]. Особенно высокая ценность такого подхода лежит в критических инфраструктурных областях,

например в космической отрасли, энергетике, транспорте и т.д.

В частности, мультиагентная парадигма моделирования приобретает все большую актуальность благодаря своей способности учитывать индивидуальные характеристики и взаимодействие множества автономных агентов [2]. В отличие от традиционных подходов, которые часто рассматривают системы как единое целое, агентное моделирование фокусируется на микроуровне, что позволяет более точно отражать реальные процессы. Это особенно важно в таких областях, как социальные науки (моделирование поведения групп), экономика (рыночная динамика), экология (взаимодействие видов), транспортные системы (потоки движения) и интернет вещей (координация устройств) [3]. В последнее время растет популярность киберфизических систем, представляющих собой совокупность интегрированных вычислительных, коммуникационных и физических компонентов, которые взаимодействуют для управления сложными процессами в реальном времени. Для киберфизических систем характерны самоорганизация и способность к адаптации под изменяющиеся условия окружающей среды. Эти характеристики являются необходимыми для систем инфраструктурных отраслей, строящихся на взаимодействии участников для организации жизненно необходимых процессов [4]. Это делает киберфизическую парадигму и агентный подход актуальными в решении современных задач моделирования.

С ростом масштаба и сложности моделируемых систем традиционные подходы к разработке моделей становятся все менее эффективными, что создает потребность в новых инструментах и методологиях. Современные системы, такие как умные города, глобальные экономические модели или экосистемы, включают тысячи или даже миллионы взаимодействующих элементов, каждый из которых обладает собственной логикой поведения. Для управления такой сложностью необходимы инструменты, которые позволяют предметным специалистам (например, экономистам, экологам или социологам) сосредоточиться на содержательной части модели, минимизируя затраты на программирование и техническую реализацию [5]. Это достигается за счет использования визуальных сред разработки, шаблонов проектирования и автоматизированных средств кодогенерации. Это обуславливает **цель исследования** – разработку инструментальных средств поддержки разработки мультиагентных сред моделирования.

Материалы и методы исследования

Мультиагентные среды обычно реализуются на специализированных платформах и фреймворках, таких как NetLogo, Repast, AnyLogic® или MATLAB® Simulink® [6], а также с использованием библиотек для универсальных языков программирования, таких как Python (например, Mesa или PyDy), Java или C++. Выбор платформы или фреймворка для разработки мультиагентной среды зависит от набора факторов, определяемых задачей моделирования: поддерживаемые операционные системы, язык описания агентов, гибкость переконфигурации, наличие модулей для обмена сообщениями и визуализации, соответствие стандартам, поддержка параллельных вычислений и т.д. В частности, параллелизм является важным аспектом для систем с большим количеством агентов [7, 8].

Результаты исследования и их обсуждение

В статье рассматривается платформа Java Agent DEvelopment Framework (JADE™) [9]. Она предоставляет готовую инфраструктуру для создания, управления и взаимодействия агентов, что значительно ускоряет процесс разработки и тестирования моделей. Платформа JADE™ реализует стандарты Foundation for Intelligent Physical Agents (FIPA), что обеспечивает совместимость с другими агентными платформами и поддерживает такие ключевые функции, как обмен сообщениями, управление жизненным циклом агентов и распределенные вычисления. JADE™ написана на Java, что обеспечивает кроссплатформенность и легкость интеграции с другими библиотеками и инструментами.

Агент в платформе JADE™ реализуется как объект класса, унаследованного от базового класса jade.core.Agent (далее – класс-агент). Базовый класс предоставляет основные методы для управления жизненным циклом агента, коммуникации с другими агентами и добавления моделей поведения – объектов классов, унаследованных от jade.core.behaviours.Behaviour (далее – класс-поведение). Модели поведения формируют полезную нагрузку агента и определяют логику его работы, включая циклические, одноразовые или сложные составные поведения. Данные модели уже включены в стандартную библиотеку фреймворка и позволяют на своей основе создавать необходимые модели агентов.

Одной из таких моделей является машина состояний (МС). Это математическая модель, используемая для описания поведения

системы, которая может находиться в одном из конечного числа состояний и переходить между ними в ответ на события. Каждое состояние определяет, как система реагирует на входные данные, а переходы между состояниями задаются правилами, которые могут зависеть от условий или триггеров. Машины состояний широко применяются в разработке программного обеспечения, робототехнике, игровых движках, а также в мультиагентных системах, так как они позволяют четко структурировать поведение и упрощают управление сложными процессами.

JADE™ предоставляет класс FSMBehaviour, реализующий составное поведение на основе конечного автомата, где каждое состояние представляется объектом класса-поведения, а переходы между состояниями определяются результатом завершения поведения. Схема автомата формируется во время исполнения с помощью методов registerState() и registerTransition(), добавляющих соответственно состояния и переходы. FSMBehaviour автоматически управляет выполнением текущего состояния и переходом к следующему, что упрощает разработку сложных сценариев поведения агента [10].

Иерархические машины состояний (ИМС, Hierarchical State Machines, HSM) расширяют концепцию обычных машин состояний, добавляя возможность вложенности [11]. В ИМС состояния могут содержать внутри себя подсостояния, образуя иерархию. Это позволяет моделировать более сложное поведение, разбивая его на уровни абстракции. Например, состояние «Работа» может включать подсостояния «Ожидание», «Обработка» и «Завершение», каждое из которых имеет свои собственные переходы и логику, но при этом переходы из состояния «Работа» являются общими для всех подсостояний. Иерархические машины состояний особенно полезны в мультиагентных системах, где агенты могут иметь сложное поведение, требующее декомпозиции на более простые компоненты. Такой подход улучшает читаемость, поддерживаемость и масштабируемость моделей [12].

Предлагаемый программный комплекс выполняет задачу генерации шаблона класса-агента, структура и базовая логика которого определяется иерархической машиной состояний. В рамках надстройки предлагается два подхода. Первый – модульный, он предполагает генерацию каркаса из переключающихся состояний-поведений, где логику последних разработчик далее реализует самостоятельно с помощью других программных средств. Второй подход – расширенный, где генерация самодостаточного класса-агента на основе расширенных иерархических машин состояний, где логика выполнения приводится непосредственно в схеме в виде кода, используемого при генерации класса.

Прежде всего, комплекс включает в себя набор библиотек-модулей для платформы JADE™, написанных на языке Java и обеспечивающих реализацию модульного подхода. Модуль HSMBehaviour – это класс-поведение, реализующий иерархическую машину состояний, где каждое состояние является объектом класса-поведения, и переключение между состояниями осуществляется на основе кода завершения очередного состояния.

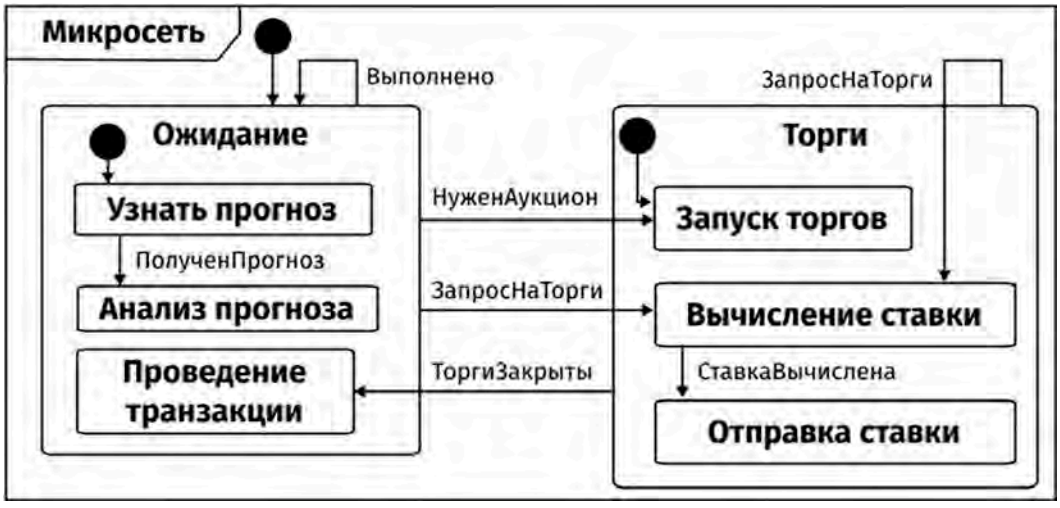

Рис. 1. Иерархическая машина состояний взаимодействующей микросети
Источник: составлено автором

```
HSMBehaviour hsm = new HSMBehaviour(this);
hsm.registerFirstState(new FallthroughBehaviour(), “Waiting”);
hsm.registerState(new AskWeather(), “AskWeather”, “Waiting”);
hsm.registerState(new CheckWeather(), “CheckWeather”, “Waiting”);
hsm.registerState(new ApplyTrade(), “ApplyTrade”, “Waiting”);
hsm.registerState(new FallthroughBehaviour(), “Trading”);
hsm.registerState(new InitTrade(), “InitTrade”, “Trading”);
hsm.registerState(new CalculateBid(), “CalculateBid”, “Trading”);
hsm.registerState(new SendBid(), “SendBid”, “Trading”);
hsm.registerDefaultTransition(“Waiting”, “AskWeather”);
hsm.registerDefaultTransition(“Trading”, “InitTrade”);
hsm.registerTransition(“AskWeather”, “CheckWeather”, S.GOT_WEATHER);
hsm.registerTransition(“Waiting”, “AskWeather”, S.DONE);
hsm.registerTransition(“Waiting”, “InitTrade”, S.NEED_TRADE);
hsm.registerTransition(“Waiting”, “CalculateBid”, S.TRADE_REQUEST);
hsm.registerTransition(“Trading”, “CalculateBid”, S.TRADE_REQUEST);
hsm.registerTransition(“CalculateBid”, “SendBid”, S.BID_CALCULATED);
hsm.registerTransition(“Trading”, “ApplyTrade”, S.TRADE_FINISHED);
```

Листинг 1. Создание машины состояний на основе HSMBehaviour

Таким образом, логика и интерфейс класса максимально приближены к встроенному в платформу FSMBehaviour, что позволяет применять модуль в сценариях одноуровневых машин состояний, при этом имея возможность описывать надсостояния и общие переходы. Такой подход снижает порог освоения модуля, а также позволяет быстро адаптировать существующие наработки в виде обычных МС. С другой стороны, такой подход менее гибок, в том числе предполагает использование изолированных состояний.

В качестве примера была рассмотрена иерархическая машина состояний, описывающая функционирование микросети, взаимодействующей с другими микросетями для обмена энергоресурсами в рамках аукциона. Схема ИМС приведена на рис. 1. Код, формирующий эту ИМС с помощью библиотеки HSMBehaviour, приведен в листинге 1. В отличие от FSMBehaviour, функция registerState имеет вариант, позволяющий указать надсостояние. Здесь используется вспомогательный модуль FallthroughBehaviour, который представляет собой мгновенно завершающееся поведение и тем самым реализует переход в начальное состояние составного состояния с помощью перехода по умолчанию. Стоит отметить, что модуль реализован с нуля, так как существующая реализация этой функциональности [13] отсутствовала в открытом доступе на момент написания статьи.

Для более эффективного и доступного составления графа состояний предлагается использовать визуальную среду разработки Lapki IDE [14]. Этот пакет прикладных программ с открытым исходным кодом предоставляет редактор ИМС с визуальным интерфейсом и возможностью описания поведения технической системы в текстовом или пиктографическом виде (в зависимости от наличия поддержки платформы). Для хранения схем Lapki IDE использует специально разработанный формат файла CyberiadaML. В его основе лежит язык описания графов GraphML, расширенный для описания поведения целевой системы с помощью стандартизованной системы тегов. В рамках программного комплекса Lapki IDE используется в полнотекстовом режиме, предоставляющем возможность работы с графом состояний и указания событий и действий в виде непосредственного текста.

Для полноценной реализации модульного подхода используется модуль CyberiadaHSMBehaviour. Это класс-поведение, наследующий HSMBehaviour и расширяющий его функцией загрузки CyberiadaML-файла, на основе которого создается граф состояний агента. Название состояния определяет класс-поведение, объект которого исполняется в данном состоянии, причем одному названию в нескольких состояниях будут соответствовать разные экземпляры класса. Содержимое перехода определяет название сигнала, преобразующееся в целочисленную константу типа-перечисления для кодов завершения поведений. Оба вышеперечисленных фактора предполагают наличие функций, связывающих текстовое обозначение с фактическим значением в коде, для чего предусмотрены две соответствующие функции – createState и identifySignal. Пример кода обвязки вышеописанной машины состояния приведен в Листинге 2. В данном режиме не учитываются условия и действия при переходах и событиях внутри состояния.

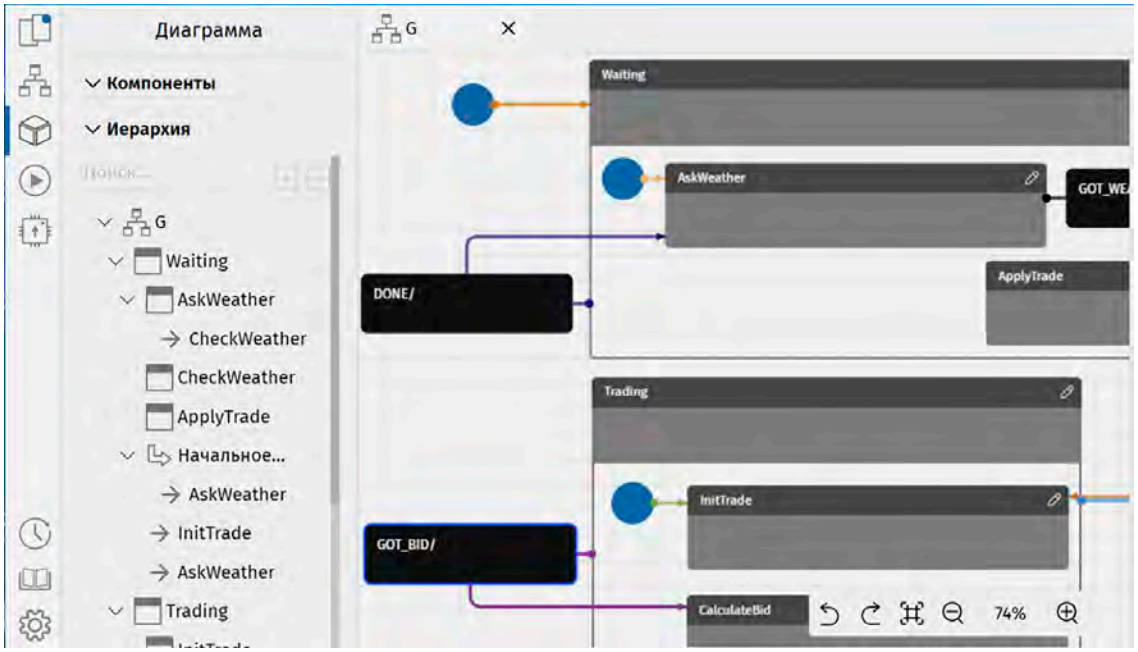

Рис. 2. Редактор Lapki IDE и фрагмент диаграммы состояний
Источник: составлено автором

```
CyberiadaMLBehaviour behaviour = new CyberiadaMLBehaviour(this) {
    private Behaviour createState(String name) {
        switch (name) {
            case “Waiting” -> return new Waiting();
            case “Trading” -> return new Trading();
            case “AskWeather” -> return new AskWeather();
            case “CheckWeather” -> return new CheckWeather();
            case “ApplyTrade” -> return new ApplyTrade();
            case “InitTrade” -> return new InitTrade();
            case “CalculateBid” -> return new CalculateBid();
            case “SendBid” -> return new SendBid(); } }
    private int identifySignal(String name) {
        switch (name) {
            case “GOT_WEATHER” -> return S.GOT_WEATHER;
            case “NEED_TRADE” -> return S.NEED_TRADE;
            case “BID_CALCULATED” -> return S.BID_CALCULATED;
            case “TRADE_REQUEST” -> return S.TRADE_REQUEST;
            case “TRADE_FINISHED” -> return S.TRADE_FINISHED;
            case “DONE” -> return S.DONE; } } }
behaviour.loadFile(“microgrid.graphml”);
```

Листинг 2. Инициализация машины состояний в CyberiadaMLBehaviour

Генератор каркаса агента CMLAgentGen – прикладная программа, работающая в командной строке и на основе переданного CyberiadaML-файла формирует Java-код агента. При модульном подходе код оборачивает CyberiadaHSMBehaviour, как указано в листинге 2, и подготавливает шаблоны классов-поведений для каждого состояния, тем самым создавая пакет модулей, на основе которого специалист далее формирует функциональность агента.

При расширенном подходе генератор создает самодостаточный класс с системой сигналов и функций-состояний согласно описанной структуре и их переходам. Он также создает функции-ячейки с участками кода, указанного в действиях событий и переходов. При этом код может размещаться как комментарий (для дальнейшей доработки специалистом) или как есть, позволяя описывать функциональность агента непосредственно в схеме.

```
QState G_AskWeather(QEvt e) { /* обработка сигнала в состоянии AskWeather */
  QState status_ = null;
  switch (e.sig) {
    case Q_ENTRY_SIG : /* вход в состояние */
      stateChanged = false;
      inVertex = false;
      status_ = q_handled();
      on_AskWeather_Entry(); /* вызов функции-ячейки */
      break;
    case Q_EXIT_SIG: /* выход из состояния */
      status_ = q_handled();
      on_AskWeather_Exit();
      break;
    case GOT_WEATHER_SIG:
      if (true) {
        on_AskWeather_GOT_WEATHER();
        stateChanged = true;
        status_ = q_tran(G_CheckWeather);
      } else { status_ = q_unhandled(); }
      break;
    default: /* передача сигнала в надсостояние */
      status_ = q_super(G_Waiting);
      break; }
    return status_; }
void on_AskWeather_Entry() { /* функция-ячейка, вынесена для удобства */
    App.run("WeatherForecast", GOT_WEATHER); }
void on_AskWeather_Exit() {
    App.cancel("WeatherForecast"); }
```

Листинг 3. Фрагмент кода класса-агента в расширенном режиме

Фрагмент функций класса для состояния «Узнать прогноз» приведен в листинге 3.

Разработанный комплекс задействован в пакете прикладных программ для моделирования микросетей, взаимодействующих с применением экономического механизма регулирования спроса и предложений [5, 15]. Применение комплекса в сочетании с визуальным редактором позволило снизить трудозатраты при описании логики агента. Использование визуальных средств проектирования и кодогенерации в разработке агентных систем значительно упрощает и ускоряет процесс создания, тестирования и модификации агентов. Схемы ИМС позволяют разработчикам наглядно проектировать поведение агентов, не углубляясь в низкоуровневый код, а также служат наглядной документацией, понятной всем участникам разработки. Это особенно полезно для исследователей и инженеров, которые могут не обладать глубокими навыками программирования, но при этом нуждаются в создании сложных моделей. Среда Lapki IDE предоставляет интуитивно понятный интерфейс для визуального проектирования, что снижает порог входа и ускоряет разработку. Кодогенерация, в свою очередь, автоматизирует процесс преобразования визуальных моделей в программный код, что минимизирует вероятность ошибок и экономит время. Это особенно важно при перепроектировании агентов, когда необходимо быстро адаптировать модель к изменяющимся требованиям или новым данным. В совокупности эти подходы повышают эффективность разработки, снижают затраты на поддержку и делают агентное моделирование более доступным для широкого круга специалистов. В дальнейшем предполагается расширение средств создания более функционально насыщенных МС. Планируется реализация механизма условий и псевдосостояний выбора.

Заключение

В рамках исследования разработаны новые инструментальные средства автоматизации разработки агентов для платформы JADE™ на основе иерархических машин состояний, включающие в себя визуальную среду разработки. Результаты исследования обеспечивают снижение сложности и времязатрат на реализацию агента, а также позволяют легче перепроектировать его логику в зависимости от задач моделирования.

Список литературы

1. Еделев А.В., Карамов Д.Н., Башарина О.Ю. Анализ уязвимости автономных микросетей // Информационные и математические технологии в науке и управлении. 2024. № 1 (33). С. 112–121. DOI: 10.25729/ESI.2024.33.1.010.

2. Cardoso R.C., Ferrando A. A review of agent-based programming for multi-agent systems // Computers. 2021. Vol. 10, Is. 2. P. 16. DOI: 10.3390/computers10020016.

3. Antelmi A., Cordasco G., D'Ambrosio G., De Vinco D., Spagnuolo C. Experimenting with agent-based model simulation tools // Applied Sciences. 2022. Vol. 13, Is. 1. P. 13. DOI: 10.3390/app13010013.

4. Томин Н.В., Домышев А.В., Барахтенко Е.А. Обзор методов моделирования и управления киберфизическими системами в мультиэнергетических микросетях // iPolytech Journal. 2023. Т. 27, № 4. С. 773–789. DOI: 10.21285/1814-3520-2023-4-773-789.

5. Бычков И.В., Феоктистов А.Г., Чекан М.А. Модель поведения агента микросети // Вычислительные технологии. 2023. Т. 28, № 6. С. 108–117. DOI: 10.25743/ICT.2023.28.6.010.

6. Lemmassi A., Derouich A., Hanafi A., Benmessaoud M., El Ouanjli N. Design and conception of an electrical power system for 1U CubeSat using MATLAB/Simulink // The European Physical Journal Plus. 2025. Vol. 140, Is. 1. P. 45. DOI: 10.1140/epjp/s13360-024-05934-1.

7. Феоктистов А.Г., Костромин Р.О. Разработка и применение проблемно-ориентированных мультиагентных систем управления распределенными вычислениями // Известия ЮФУ. Технические науки. 2016. № 11. С. 65–74. DOI: 10.18522/2311-3103-2016-11-6575.

8. Чекан М.А. Сравнительный анализ программного обеспечения для автоматизации процесса моделирования микросетей // Современные наукоемкие технологии. 2022. № 9. С. 33–38. DOI: 10.17513/snt.39305.

9. Bergenti F., Caire G., Monica S., Poggi A. The first twenty years of agent-based software development with JADE // Autonomous Agents and Multi-Agent Systems. 2020. Vol. 34. P. 1–19. DOI: 10.1007/s10458-020-09460-z.

10. Bellifemine F., Poggi A., Rimassa G. Developing multi-agent systems with a FIPA-compliant agent framework // Software: Practice and Experience. 2001. Vol. 31, Is. 2. P. 103–128. DOI: 10.1002/1097-024X(200102)31:2<103::AID-SPE358>3.0.CO;2-O.

11. Ivanchev J., Deboeser C., Braud T., Knoll A., Eckhoff D., Sangiovanni-Vincentelli A. A hierarchical state-machine-based framework for platoon manoeuvre descriptions // IEEE Access. 2021. Vol. 9. P. 128393–128406. DOI: 10.1109/ACCESS.2021.3106455.

12. Rocha M., Simão A., Sousa T. Model-based test case generation from UML sequence diagrams using extended finite state machines // Software Quality Journal. 2021. Vol. 29, Is. 3. P. 597–627. DOI: 10.1007/s11219-020-09531-0.

13. Griss M.L., Fonseca S., Cowan D., Kessler R. Using UML State Machine Models for More Precise and Flexible JADE Agent Behaviors // Agent-Oriented Software Engineering III. 2003. Vol. 2585. P. 113–125. DOI: 10.1007/3-540-36540-0_9.

14. Чекан М. Среда программирования киберфизических систем в парадигме машин состояний // Материалы VI Международного семинара по информационным, вычислительным и управляющим системам для распределенных сред (ICCS-DE 2024). Иркутск: ИДСТУ СО РАН, 2024. С. 217–219. [Электронный ресурс]. URL: https://iccs-de.icc.ru/files/2024/Proceedings_ICCS-DE-2024.pdf (дата обращения: 05.02.2025).

15. Feoktistov A., Edelev A., Tchernykh A., Gorsky S., Basharina O., Fereferov E. An Approach to Implementing High-Performance Computing for Problem Solving in Workflow-based Energy Infrastructure Resilience Studies // Computation. 2023. Vol. 11, Is. 12. P. 243. DOI: 10.3390/computation11120243.