ISSN 1992-6502 (Print) Вестник УГАТУ ISSN 2225-2789 (Online)
2026. Т. 30, № 1 (111). С. 22-30 http://journal.uust.ru/index.php/Vestnik

УДК 004.4'242 doi 10.54708/19926502_2026_30111122
Код ГРНТИ 28.17.31

Автоматизация разработки агентов моделирования микросетей на основе визуального программирования иерархических машин состояний

А.Г. Феоктистов, М.А. Чекан*

ФГБУН «Институт динамики систем и теории управления им. В.М. Матросова СО РАН», г. Иркутск, Россия

Аннотация. Статья посвящена вопросам разработки мультиагентной системы для исследования процессов совместно работающих микросетей. Важную роль в работе с подобными системами играет возможность снижения затрат на реализацию и перепроектирование отдельных агентов. Целью представленного исследования является разработка инструментального комплекса для автоматизации разработки поведения агента на основе расширенных иерархических машин состояний. Комплекс включает визуальный редактор диаграмм, генератор модуля поведения агента и библиотеку программных модулей. Ключевой особенностью представленного подхода является выделение типовой функциональности агентов в компоненты. Сочетание автоматной парадигмы с визуальными средствами и кодогенерацией существенно снижает затраты времени и необходимую квалификацию для предметного специалиста. Инструментальный комплекс успешно применен при автоматизации моделирования взаимодействия микросетей.

Ключевые слова: *мультиагентная система, иерархические машины состояний, генерация кода.*

***chekoopa@mail.ru**

Введение

В настоящее время компьютерное моделирование является неотъемлемой составляющей исследования сложных систем. Это особенно актуально для оценки состояния и прогнозирования развития критических инфраструктур, например, в энергетике, поскольку организация и проведение натурных экспериментов с энергетическими комплексами, как правило, сопряжены со значительными техническими рисками и финансовыми затратами [1].

Сегодня одним из перспективных направлений развития таких комплексов является создание микросетей [2], предполагающих взаимодействие большого числа самостоятельных сущностей: потребителей, производителей и накопителей электроэнергии, распределительных узлов и других вспомогательных инфраструктурных элементов, – а также их коммуникаций. В сочетании с критическим характером этих сущностей возрастает потребность в детальном моделировании работы таких сетей. Перспективным подходом к организации моделирования микросетей является использование мультиагентных технологий, позволяющих адекватно отразить взаимодействие всех элементов сети, которые обладают определенной степенью автономности работы и индивидуальными структурно-параметрическими характеристиками [3].

Распространенным способом поддержки мультиагентного моделирования является использование специализированных программных библиотек [4]. Однако их применение требует значительного объема низкоуровневого программирования при разработке мультиагентной системы (МАС). Рост числа агентов МАС и расширение набора их функциональных возможностей обуславливают существенное возрастание накладных расходов на создание конечного продукта [5]. Следовательно, необходимы инструменты, которые позволят избежать низкоуровневого программирования и дадут предметному специалисту возможность работать напрямую с высокоуровневыми компонентами: конфигурировать МАС, настраивать логику поведения и взаимодействия агентов, что в итоге

сделает возможным сосредоточение на исследовании самой сложной системы [6]. В таком контексте актуальным является применение абстракций, с помощью которых можно агрегировать и определить различные составляющие логики поведения агентов и их взаимодействий. Применение абстракций в совокупности с визуальными средствами, шаблонами проектирования и кодогенерацией имеет высокий потенциал с точки зрения снижения квалификационного порога вхождения предметных специалистов и накладных расходов при создании МАС.

Статья посвящена инструментальному комплексу (ИК) поддержки автоматизации разработки МАС для моделирования микросетей. Особое внимание уделено вопросам создания агентов на основе парадигмы программирования расширенных иерархических машин состояний (ПРИМС), в частности механизму кодогенерации поведения агента.

Парадигма ПРИМС основана на стандарте UML Statecharts и предоставляет механизм для описания программной логики на основе иерархических машин состояний (ИМС) с использованием переменных и поведения, приводимого непосредственно на языке целевой системы. Ключевой особенностью ПРИМС является гибкость описания содержимого диаграммы, позволяющая использовать как языки программирования, так и визуальное пиктографическое представление. Использование диаграмм состояний улучшает читаемость, сопровождаемость и масштабируемость моделей, обеспечивает наглядность и разделяемость логики работы агента в вычислительной среде. В сочетании с визуальными средствами ПРИМС также позволяет облегчить и ускорить разработку агентов, особенно на этапе их прототипирования [7].

1. Архитектура и состав инструментального комплекса

ИК представляет собой программную надстройку к фреймворку Java Agent DEvelopment Framework (JADE) [8] и включает в себя два программных средства и библиотеку программных модулей. Ключевым компонентом ИК является генератор модуля поведения агента (кодогенератор) CMLAgentGen – прикладная программа, работающая в режиме командной строки и формирующая Java-код на основе файла диаграммы ПРИМС. Кодогенератор детально рассматривается в следующем разделе.

В качестве базовой платформы для формирования агентов был выбран фреймворк JADE, реализованный на языке Java. Его определяющими особенностями являются открытый исходный код, кросс-платформенность и соответствие стандартам Foundation for Intelligent Physical Agents, что обеспечивает устойчивую и переносимую архитектуру агентов, гибкую коммуникацию между ними и, как следствие, высокую совместимость и простоту интеграции с внешними библиотеками и инструментами. Архитектура МАС в JADE строится на платформах, состоящих из связанных контейнеров, в которых выполняются агенты. Каждый агент представлен Java-модулем, реализованным через класс, наследующий базовый класс-агент (jade.core.Agent). Класс агента обеспечивает жизненный цикл и коммуникацию с другими агентами. Непосредственно поведение агента задается и подключается в агент в виде экземпляров классов, наследующих базовый класс-поведение (jade.core.behaviours.Behaviour). Стандартная библиотека модулей JADE содержит базовые классы-поведения, реализующие типовые модели (циклические, одноразовые, составные и др.).

ИК предоставляет библиотеку Java-модулей, предназначенную для использования с генерируемым кодом агентов в совокупности с библиотекой модулей платформы JADE. Библиотека модулей используется для обеспечения работы кодогенератора, реализации логики ПРИМС при исполнении агента и выполнения типовых операций в МАС.

Визуальную разработку агентов обеспечивает среда Lapki IDE [9] – пакет прикладных программ с открытым исходным кодом. ИК использует его визуальный редактор ИМС, позволяющий не только сформировать граф состояний и переходов, но и детализировать поведение целевой платформы, используя ее язык программирования либо пиктограммы (если платформа поддерживает эту возможность). Интерфейс редактора приведен на Рис. 1.

В рамках ИК редактор Lapki IDE может использоваться в пиктографическом режиме, а также в полнотекстовом режиме с доступом к прямому редактированию событий и действий.

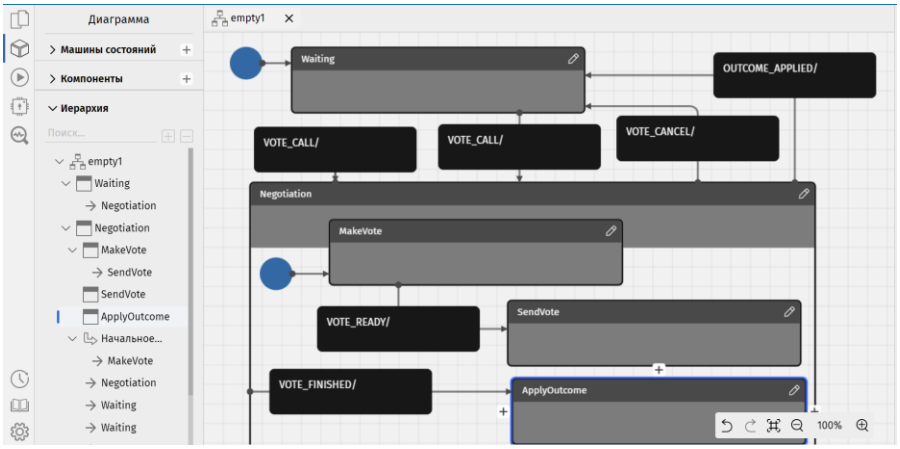

Рисунок 1. Визуальный редактор диаграмм ИМС.

Диаграммы ПРИМС хранятся и передаются между редактором и кодогенератором в формате CyberiadaML, основанном на языке GraphML [10], который применяется для описания графов. Нотация CyberiadaML расширяет этот язык и регламентирует способ описания элементов ИМС и поведения целевой платформы в текстовом виде.

2. Генерация кода агентов

Кодогенератор предусматривает два режима работы: базовый и расширенный. Оба этих режима принимают на вход диаграмму ПРИМС и формируют набор Java-модулей, реализующих поведение на основе описанной машины состояний. Разница заключается в функциональной полноте сгенерированного поведения.

В базовом режиме генерируемая машина состояний работает на основе классов HSMBehaviour и CyberiadaHSMBehaviour, включенных в библиотеку модулей ИК. Класс представляет собой аналог FSMBehaviour из библиотеки JADE: каждое состояние – это отдельное поведение (тип определяется названием состояния), которое по завершении выполнения передает код выхода, определяющий следующее выполняемое состояние-поведение. HSMBehaviour расширяет этот принцип иерархичностью: состояния вкладываются друг в друга, и необработанный сигнал передается на родительский уровень. CyberiadaHSMBehaviour в свою очередь выступает оберткой для подключения файла диаграммы и формирования графа состояний на этапе инициализации агента.

В качества примера рассматривается агент микросети туристической базы, взаимодействующей с соседними турбазами для обмена мощностями. Диаграмма ИМС агента для базового режима работы кодогенератора приведена на Рис. 2, фрагмент результата генерации представлен на Рис. 3.

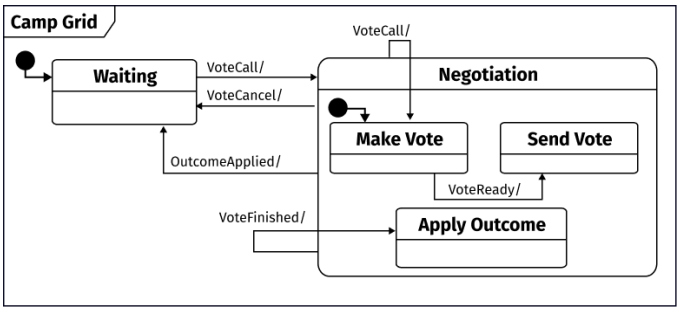

Рисунок 2. ИМС микросети турбазы.

Генератор формирует класс-оболочку, в котором загружается файл диаграммы и производится преобразование связываемых состояний в экземпляры классов, а сигналов –

в конкретные значения типа-перечисления. Состояниям с одинаковыми названиями соответствуют разные экземпляры одного класса. Классы-поведения, используемые в ИМС, также генерируются автоматически в виде шаблонов поведений, что позволяет программисту перейти к непосредственному описанию действий агента или задействовать готовые программные модули.

```
CyberiadaMLBehaviour behaviour = new CyberiadaMLBehaviour(this) {
  private Behaviour createState(String name) {
    switch (name) {
      case "Waiting" -> return new WaitingBhv();
      case "Negotiation" -> return new NegotiationBhv();
      case "MakeVote" -> return new MakeVoteBhv();
      case "SendVote" -> return new SendVoteBhv();
      case "ApplyOutcome" -> return new ApplyOutcomeBhv(); } }
  private int identifySignal(String name) {
    switch (name) {
      case "VoteCall" -> return S.VoteCall;
      case "VoteCancel" -> return S.VoteCancel;
      case "VoteFinished" -> return S.VoteFinished;
      case "VoteReady" -> return S.VoteReady;
      case "OutcomeApplied" -> return S.OutcomeApplied; } } }
behaviour.loadFile("campgrid.graphml");
```

Рисунок 3. Фрагмент кода, сгенерированного в базовом режиме.

Основным недостатком такого подхода является функциональная ограниченность результирующей машины состояний, обусловленная особенностями класса HSMBehaviour. Также отсутствует возможность описания поведения агента в самой диаграмме и применения условий-ограничений на переходы. Кроме того, в настоящее время не поддерживается миграция сгенерированных агентов между контейнерами JADE.

Расширенный режим генератора позволяет устранить большую часть указанных недостатков. Он создает самодостаточный класс-поведение, где каждый узел диаграммы представлен отдельной функцией-обработчиком, передающей управление исполнением агента в соответствии с описанной в диаграмме структурой состояний и событий.

Диаграмма ПРИМС для расширенного режима работы кодогенератора приведена на Рис. 4. В отличии от базового режима, она включает как внешние переходы, меняющие состояние, так и внутренние. Внутренние переходы не изменяют состояние системы и описываются идентично. Косая черта в конце строки используется как разделитель между сигналом и поведением – последовательностью действий (операторов языка программирования, вызовов функций), выполняемой при получении соответствующего сигнала. Рядом с сигналом в квадратных скобках может приводиться ограничивающее условие (guard condition), при невыполнении которого событие игнорируется.

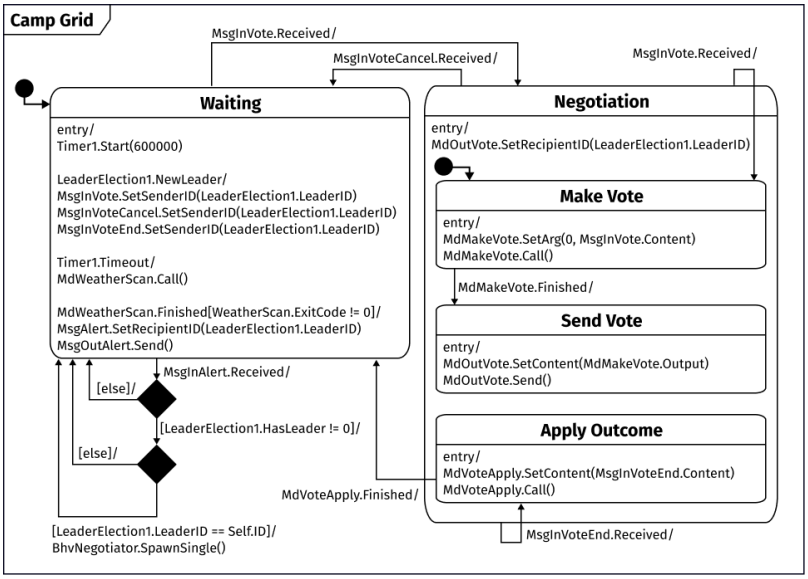

Рисунок 4. Расширенная ИМС микросети турбазы.

Как правило, для описания поведения системы одного логического условия бывает недостаточно, и требуется применение логических операторов. Однако, их пиктографическое представление в визуальных средах сопряжено со сложностями, и в текущей версии Lapki IDE такая возможность не реализована. Тем не менее, для моделирования составных условий предусмотрено специальное псевдосостояние выбора, которое позволяет обойти это ограничение. Псевдосостояние (ПС) – это узел МС, используемый для передачи управления в другое состояние, но не способный сам его удерживать. В данном случае, ПС выбора по сигналу получает управление из другого состояния и передает его по цепочке переходов в зависимости от условия. ПС выбора можно выстраивать в цепочки, а на промежуточных переходах – прописывать поведение.

При рассмотрении диаграммы ПРИМС можно заметить, что описание поведения, сигналов и условий следует схеме «объект-действие», характерной для объектно-ориентированного программирования. Это обеспечивает естественную основу для компонентной архитектуры, что является главным преимуществом расширенного режима кодогенератора.

Вместе с иерархией состояний пользователь описывает набор компонентов – объектов классов, реализующих типовые действия агента в МАС, например отправку или прием сообщений, вызов прикладных модулей, базовую арифметику и т. п. Каждому компоненту соответствует Java-класс, элементы которого отражены на диаграмме ПРИМС. А именно: процедуры соответствуют действиям, вызываемым в блоках поведения, переменные-поля отображаются как атрибуты, используемые в качестве аргументов действий и в ограничивающих условиях, аргументы конструктора – это параметры, которые пользователь указывает в редакторе при создании компонента, сигналы (маркеры возникновения события в компоненте) формируются при вызове специальных «сигнальных» функций-предикатов, при истинном значении которых генерируется соответствующий сигнал в ИМС.

Одним из представленных в ИК компонентов является «Приемник ACL-сообщений» (ACLReciever). В качестве входных параметров он принимает четыре опциональных поля: перформатив (коммуникативный тип сообщения), онтологию, язык и ID отправителя. Все эти поля можно изменять во время выполнения ПРИМС соответствующими действиями (SetSenderID, SetPerformative, SetOntology и т. д.). Они определяют фильтр сообщений, используемый при проверке в главном цикле агента. При получении сообщения по заданному фильтру предикат Received возвращает истину, а в атрибуты SenderID, Ontology, Performative, Content и прочие помещаются соответствующие поля полученного сообщения.

Преимущество компонентного подхода в ИК заключается в единой архитектуре, обеспечивающей лаконичное представление как элементов компонентов, так и поведения агента с помощью пиктограмм (Рис. 5). Подобная визуализация способствует более простому и интуитивному пониманию схемы пользователем.

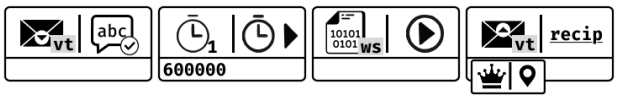

Рисунок 5. Пример пиктограмм компонентов агента.

Таким образом, значительная часть логики агента описывается через композицию готовых компонентов и их взаимодействие в рамках диаграмм ПРИМС. Для сценариев, выходящих за рамки стандартных компонентов и требующих реализации на Java, предусмотрен специальный компонент для прямого встраивания программного кода.

Для иллюстрации процесса генерации на Рис. 6 представлен фрагмент кода для состояния Waiting. Программный код включает обработку действий входа в состояние и выхода из него, а также обработку внутренних и внешних событий. Каждому возможному сигналу соответствует константа из типа-перечисления, которая передается при проверке предикатов в функции action класса Behaviour. Результатом выполнения функции-обработчика

является специальный маркер, определяющий дальнейшие действия системы: обработка завершена, не завершена, сигнал передан в родительское состояние, необходимо выполнить переход в другой узел.

```
QState Root_Waiting(QEvt e) { /* сигнал в состоянии Waiting */
  QState status_ = null;
  switch (e.sig) {
    case Q_ENTRY_SIG : /* обработка входа */
      status_ = q_handled(); stateChanged = false; inVertex = false;
      on_Root_Waiting_Entry(); /* вызов поведения для события */
      break;
    case Q_EXIT_SIG: /* обработка выхода */
      status_ = q_handled(); break;
    case SIG_LeaderElection1_NewLeader:
      status_ = q_handled();
      on_Root_Waiting_LeaderElection1_NewLeader();
      break;
    /* ... */
    case SIG_MdWeatherScan_Finished:
      status_ = q_handled();
      if (WeatherScan.ExitCode != 0) {
        on_Root_Waiting_MdWeatherScan_Finished();
      } else { status_ = q_super(Root); }
      break;
    /* ... */
    case SIG_MsgInAlert_Received:
      on_Root_Waiting_MsgInAlert_Received();
      status_ = q_tran(Root_Choice1); stateChanged = true; break;
    /* ... */
    default: /* необработанный сигнал передается на верхний уровень */
      status_ = q_super(Root); break; }
    return status_; }
```

Рисунок 6. Фрагмент кода, сгенерированного в расширенном режиме.

Ограничивающие условия непосредственно подставляются в условные конструкции if–else. Для событий с одним сигналом, но разными условиями, генератор формирует соответствующую цепочку if–else if–else. Поведение в состояниях реализуется через вызов выделенных функций-ячеек. Такой подход позволяет пользователю модифицировать логику, не углубляясь в детали функций-обработчиков сигналов. Дополнительно, если пользователь предпочитает реализовать поведение самостоятельно, генератор предоставляет закомментированные шаблоны для ручной доработки.

Как упоминалось ранее, ПС выбора также являются узлами ИМС и имеют собственных обработчиков. Пример сгенерированного кода для ПС выбора представлен на Рис. 7.

```
QState Root_Choice1(QEvt e) { /* сигнал в узле выбора */
  QState status_ = null;
  switch (e.sig) {
    case Q_ENTRY_SIG:
      status_ = q_handled(); inVertex = true; /* флаг псевдосостояния */
      break;
    case Q_EXIT_SIG: /* обработка выхода */
      status_ = q_handled(); inVertex = false;
      break;
    case Q_VERTEX_SIG: /* полезная нагрузка */
      if (LeaderElection1.HasLeader != 0) {
        q_tran(Root_Choice2);
      } else { status_ = q_tran(Root_Waiting); }
      /* при отсутствии [else] вызывается q_super */
      inVertex = false;
      break; }
    return status_; }
```

Рисунок 7. Функция обработки ПС выбора.

Обработчик сигнала для ПС отличается от обработчика обычного состояния наличием сигнала, автоматически вызываемого при переходе в ПС и приводящему к выполнению полезной нагрузки – в данном случае для проверки условий всех исходящих переходов.

Конечное состояние, предназначенное для обозначения окончания работы машины состояний, реализовано аналогичным образом. В полезной нагрузке обработчика возвращается сигнал Q_TERMINATE_SIG. Он передается на родительский уровень ИМС, где устанавливает флаг завершения поведения. Этот флаг, в свою очередь, обрабатывается функцией done класса Behaviour, что приводит к завершению данного поведения.

3. Экспериментальный анализ

В рамках экспериментального анализа была проведена оценка временных и трудовых затрат на реализацию трех агентов, моделирующих локальную энергосистему турбазы, источник генерации электроэнергии и потребителя электроэнергии. Агенты разработаны для программного комплекса моделирования взаимодействующих микросетей [11]. На Рис. 8 приведены полученные средние оценки времени (в человеко-часах), затраченного на создание одного агента. Затраты оценены для типовых этапов разработки: построение логического каркаса, описание поведения агента внутри МАС, тестирование функциональности и формирование модулей для интеграции с мультиагентной платформой.

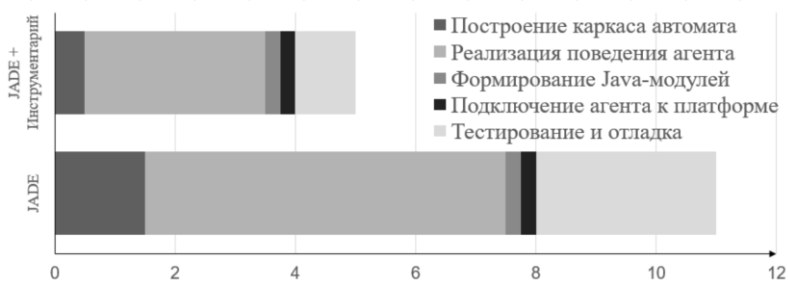

Рисунок 8. Трудозатраты на создание агента (в человеко-часах).

Анализ показал, что предложенный ИК эффективно сокращает трудозатраты на самых сложных этапах – спецификации поведения агента и отладки. Трудоемкость этапа подключения агента к JADE, слабо зависящий от реализации, требует сопоставимых трудозатрат. Общая оценка трудозатрат по разработке агента в целом согласуется с данными, приведенными в работе [12].

Заключение

В статье представлен подход к визуальной разработке агентов, основанный на использовании иерархических машин состояний. Данный подход реализован в виде ИК, включающего графический редактор диаграмм и кодогенератор, который формирует модули поведения для платформы JADE.

В настоящее время ИК применяется в составе программного комплекса для мультиагентного моделирования взаимодействующих микросетей. Этот комплекс используется в исследованиях живучести автономных энергосистем инфраструктурных объектов Байкальской природной территории. Разработанный инструментарий также может быть адаптирован для проектирования агентов в МАС другого назначения.

Дальнейшие исследования и разработки направлены на расширение поддержки конструкций ПРИМС, предусмотренных соответствующим стандартом. В частности, планируется реализовать поддержку локальной и глубокой истории состояний. Особое внимание будет уделено механизмам параллельного выполнения, таким как разделение состояния на области и создание узлов ветвления и слияния.

Литература:

1. Vorobev S., Edelev A. Problems of searching for critically important combinations of gas industry objects from the position of ensuring energy security // E3S Web of Conferences. 2023. Vol. 461. Art. 01003. DOI: 10.1051/e3sconf/202346101003.

2. Ahmad S., Shafiullah M., Ahmed C.B., Alowaifeer M. A review of microgrid energy management and control strategies // IEEE Access. 2023. Vol. 11. P. 21729–21757. DOI: 10.1109/ACCESS.2023.3248511.
3. Khan M.W., Wang J., Xiong L., Ma M. Modelling and optimal management of distributed microgrid using multi-agent systems // Sustainable cities and society. 2018. Vol. 41. P. 154–169. DOI: 10.1016/j.scs.2018.05.018.
4. Cardoso R.C., Ferrando A. A review of agent-based programming for multi-agent systems // Computers. 2021. Vol. 10. No. 2. P. 16. DOI: 10.3390/computers10020016.
5. Kostromin R., Feoktistov A. Agent-based DevOps of software and hardware resources for digital twins of infrastructural objects // Proceedings of the 4th International Conference on Future Networks and Distributed Systems (ICFNDS 2020). 2020. Vol. 8. P. 1–6. DOI: 10.1145/3440749.3442599.
6. Kuhail M.A., Farooq S., Hammad R., Bahja M. Characterizing visual programming approaches for end-user developers: A systematic review // IEEE Access. 2021. Vol. 9. P. 14181–14202. DOI: 10.1109/ACCESS.2021.3051043.
7. Bazydło G. Designing Reconfigurable Cyber-Physical Systems Using Unified Modeling Language // Energies. 2023. Vol. 16. No. 3. P. 1273. DOI: 10.3390/en16031273.
8. Bellifemine F., Poggi A., Rimassa G. JADE: A FIPA2000 compliant agent development environment // Proceedings of the 5th International Conference on Autonomous Agents (AGENTS'01). 2001. No. 153. P. 216–217. DOI: 10.1145/375735.376120.
9. Lapki IDE GitHub repository [Webpage]. URL: https://github.com/kruzhok-team/lapki-client (accessed: December 2, 2025)
10. The GraphML File Format [Webpage]. URL: http://graphml.graphdrawing.org. (accessed: November 2, 2025)
11. Feoktistov A.G., Bodnyuk M.E., Karamov D.N. Monitoring system of a physical model for a microgrid // Optoelectronics, Instrumentation and Data Processing. 2025. Vol. 61. No. 2. P. 5–12. DOI: 10.15372/AUT20250201.
12. Kruger K., Basson A.H. Evaluation of JADE multi-agent system and Erlang holonic control implementations for a manufacturing cell // International Journal of Computer Integrated Manufacturing. 2019. Vol. 32. No. 3. P. 225–240. DOI:10.1080/0951192X.2019.1571231.

Благодарности:
Работа выполнена в рамках гранта № 075-15-2024-533 Министерства науки и высшего образования РФ на выполнение крупного научного проекта по приоритетным направлениям научно-технологического развития (рег. № 124052100088-3).

Об авторах:
ФЕОКТИСТОВ Александр Геннадьевич, д.т.н., зав. лабораторией 5.1 параллельных и распределенных вычислительных систем, Институт динамики систем и теории управления им. В.М. Матросова СО РАН, Россия, 664033 Иркутск, ул. Лермонтова, 134, agf@icc.ru, 0000-0002-9127-6162, U-5369-2017.
ЧЕКАН Михаил Андреевич, стажер-исследователь лаборатории 5.1 параллельных и распределенных вычислительных систем, Институт динамики систем и теории управления им. В.М. Матросова СО РАН, Россия, 664033 Иркутск, ул. Лермонтова, 134, chekoopa@mail.ru, 0000-0001-7622-421X, PAQ-9167-2025, 58872348500.

Metadata:
Title: Development automation for microgrid modeling agents based on visual programming of hierarchical state machines.
Author 1: Alexander Gennadievich Feoktistov, Doctor of Sciences, Head of the Laboratory 5.1 of Parallel and Distributed Computing Systems, Matrosov Institute for System Dynamics and Control Theory SB RAS, 134 Lermontova St., 664033 Irkutsk, Russia, agf@icc.ru, 0000-0002-9127-6162, U-5369-2017, 8416831500.

Author 2: Mikhail Andreevich Chekan, Intern Researcher of the Laboratory 5.1 of Parallel and distributed computing systems, Matrosov Institute for System Dynamics and Control Theory SB RAS, 134 Lermontov St., 664033 Irkutsk, Russia, chekoopa@mail.ru, 0000-0001-7622-421X, PAQ-9167-2025, 58872348500.

Abstract: The article is devoted to the development of a multi-agent system for studying the processes of joint operation of microgrids. An important role in working with such systems is played by the ability to reduce the cost of implementation and redesign of individual agents. The purpose of the presented research is to develop a toolkit for automating the creation of agent behaviour based on extended hierarchical state machines. The toolkit includes a visual diagram editor, a module generator for agent behaviours, and a library of software modules. The key feature of the presented approach is the separation of typical agent functionality into components. Combining the automata paradigm with visual tools and code generation greatly reduces the time required and the necessary expertise of the subject specialist. The toolkit has been successfully used to automate the simulation of interactions between microgrids.

Keywords: multi-agent system, hierarchical state machines, code generation.