УДК 004.942

# КАКОЙ МОЖЕТ БЫТЬ ПЛАТФОРМА ДЛЯ ЭФФЕКТИВНОЙ РАЗРАБОТКИ ДИСКРЕТНО-СОБЫТИЙНЫХ ИМИТАЦИОННЫХ МОДЕЛЕЙ?

**А.А. Малыханов, В.Е. Черненко, А.Д. Шуравин (Ульяновск)**

**Введение**

Высокая конкуренция на мировом рынке имитационного моделирования ужесточает требования к эффективности проектирования, разработки и эксплуатации коммерческих имитационных моделей [1, 2]. Одним из путей повышения эффективности является создание специализированных платформ для имитационных моделей (ИМ). Однако для их разработки необходим обширный практический опыт, включающий репрезентативную выборку реализованных проектов [3].

Авторы располагают опытом выполнения 54 коммерческих проектов с применением имитационного моделирования, а также проведения предпроектных обследований ещё по нескольким десяткам случаев. Проекты охватывали различные отрасли: 14 – подземные и открытые горные работы, 5 – металлургические цеха, 25 – внутренняя логистика и транспортировка продукции, 4 – сельское хозяйство и пищевая промышленность, 3 – поточные производства и 2 – железнодорожные и портовые терминалы. Из них 28 проектов были выполнены в среде AnyLogic [4], 36 – на платформе Amalgama Platform [5], 10 из которых разработаны в инструменте моделирования горных работ MineTwin [6]. Объекты моделирования находились в 10 странах. Имитационные модели, разработанные на Amalgama Platform, успешно используются для проверки гипотез, оценки достижимости планов, затрат и помощи в принятии решений. Результаты проектов представлялись авторами на тематических конференциях, в том числе пять проектов представлены на Winter Simulation Conference, три на AnyLogic Conference, десять – на конференциях ИММОД [7, 8].

Созданные модели имели ряд общих особенностей:

- в рамках одной модели часто сочетались разнородные процессы, например, транспортировка материалов конвейерами и дискретное перемещение готовой продукции автомобильным и погрузочным транспортом;
- моделирование ограничивалось внешней, «логистической» стороной производственных процессов, без детальной проработки физических аспектов, таких как смешивание веществ, деформация тел, изменение свойств материалов, распространение сигналов;
- для адекватности модели часто требовалась имитация процессов планирования и диспетчеризации, включая управление оборудованием (например, диспетчеризация кранов в металлургическом цехе или составление оперативных графиков движения поездов).

Модели, обладающие упомянутыми особенностями, можно условно назвать процессно-ориентированными дискретно-событийными имитационными моделями. Именно о таких моделях и пойдет речь далее. Опираясь на накопленный опыт и с целью повышения эффективности разработки таких моделей, авторами была создана и развивается платформа Amalgama Platform, впервые представленная на конференции ИММОД-2019. В [9] подробно изложены её технические особенности.

В данной статье, опираясь на результаты развития платформы и реализации практических проектов, мы предлагаем наше видение ключевых принципов построения платформы для создания процессно-ориентированных дискретно-событийных имитационных моделей. Наша задача – осмыслить накопленный опыт, выявить особенности процесса разработки имитационных моделей описанного типа и

сформулировать требования к инструменту разработки таких моделей. Обсуждаемые технические и архитектурные решения мы будем иллюстрировать примерами из Amalgama Platform.

**Создание ИМ – это разновидность разработки программного обеспечения**

Наш опыт разработки коммерческих имитационных моделей показал, что создание ИМ по своей сути является разновидностью разработки программного обеспечения (ПО). К аналогичным выводам приходят и авторы других публикаций, причем некоторые уже в 1989 году [10, 11]. Рассмотрим основные сходства создания ИМ и разработки ПО.

Во-первых, создание ИМ, как и разработка ПО, сопровождается итеративным уточнением требований и декомпозиции предметной области, что позволяет выделять и понимать моделируемые объекты и процессы.

Во-вторых, оба процесса требуют формального описания логики в коде (или с помощью визуальных аналогов программного кода), обеспечивающего однозначность интерпретации и воспроизводимость.

В-третьих, ключевыми этапами как создания ИМ, так и разработки ПО являются тестирование, верификация и валидация. Эти этапы гарантируют техническую корректность реализации и соответствие программной реализации целям исследования.

В-четвертых, прошли времена, когда создание имитационных моделей было делом мастеров-одиночек. Современные коммерческие имитационные модели создаются командами. Как и в индустрии «большого ПО», работа в команде требует наличия систем контроля версий и применения практик командной разработки.

В-пятых, подобно другим аналитическим системам современные имитационные модели часто требуют интеграции с другими ИТ-системами.

Наконец, любая имитационная модель должна включать средства доставки результатов пользователям, будь то отчёты, визуализации или экспорт результатов во внешние системы.

Таким образом, имитационное моделирование воспроизводит процессы промышленной разработки ПО, оставаясь при этом ориентированным на специфические задачи сценарного анализа и прогнозирования.

**При создании моделей целесообразно использовать стандартные средства разработки ПО**

Большинство подходов и инструментов, применяемых в имитационном моделировании, целесообразно заимствовать из мира разработки программного обеспечения. Повторное изобретение решений там, где уже существуют зрелые практики, снижает эффективность и качество конечного продукта. Приведем несколько примеров таких подходов и инструментов.

Алгоритмы сортировки нужны в логике большинства нетривиальных ИМ, но их нет смысла реализовывать заново – стандартные библиотеки современных языков программирования предоставляют оптимизированные реализации. Например, в стандартной библиотеке языка Java есть метод Collections.sort [12].

Сбор статистики моделирования и анализ данных лучше доверить готовым библиотекам, так как они предоставляют богатый набор методов и обеспечивают совместимость с BI-инструментами. Примером такого инструмента является Metabase [13] – система с открытым кодом, позволяющая настраивать шаблоны анализа данных (например, логов ИМ) и выводить результаты анализа в виде графиков и таблиц.

Облачные вычисления уже давно реализованы с помощью средств контейнеризации и оркестрации, а с точки зрения возможности удаленного выполнения имитационные эксперименты не отличаются от других расчетных задач. Таким образом,

инструменты типа Docker могут использоваться для организации серверного и облачного исполнения имитационных моделей.

Пользовательский интерфейс может строиться на основе существующих UI-библиотек. 2D- и 3D-анимация, таблицы, графики, диаграммы – все это не является уникальной особенностью ИМ и уже реализовано в мире «большого ПО». Так, для построения кроссплатформенных десктоп-приложений может использоваться Eclipse RCP [14], а для создания веб-приложений – библиотеки React [15] и Vue [16].

Средства автоматизации тестирования также не нужно создавать специально для имитационных моделей – можно заимствовать их напрямую из промышленной разработки ПО. Экосистема языка Java, например, предоставляет инструмент Junit [17], позволяющий организовать модульное и интеграционное тестирование.

Таким образом, для системы имитационного моделирования стоит разрабатывать только специфические программные компоненты, нужные именно в имитационных моделях. Смежные компоненты, которые используются не только в имитационных моделях, можно взять из уже существующих библиотек.

### Для дискретно-событийных процессно-ориентированных моделей достаточно небольшого количества простых модулей

Каковы же модули, логические блоки, специфичные именно для процессно-ориентированных дискретно-событийных ИМ? Наша практика показывает, что таких модулей нужно сравнительно немного. В Amalgama Platform мы используем Java как платформу и основной язык программирования. Поэтому модули, описываемые в этой главе, являются Java-классами.

**Симуляционный движок** [18] (Simulation Engine) позволяет разработчикам запускать дискретно-событийные имитационные модели в приложениях на языках программирования, например на Java. Движок представляет собой класс, предоставляющий программный интерфейс для планирования событий, предстоящих к выполнению в заданные моменты модельного времени.

Во время работы движок выполняет цикл из трёх повторяющихся шагов:

1. определение времени следующего по хронологии события;
2. перенос модельного времени к этому моменту;
3. выполнение действия, связанного с данным событием.

Основной особенность имитационного движка является возможность его встраивания в код любой Java-программы. Это позволяет, например, добавлять небольшие имитационные модели в код уже существующих бизнес-систем без изменения структуры их кода. Пример создания минималистичной имитационной модели с помощью имитационного движка Amalgama Platform показан на листинге 1.

Листинг 1 – Создание минималистичной имитационной модели с помощью имитационного движка Amalgama Platform

```
Engine engine = new Engine();
engine.setTemporal(LocalDateTime.of(2022, 1, 1, 0, 0), ChronoUnit.DAYS);
engine.scheduleAbsolute(10, () -> {

      // This is the code of first event's action
      System.out.println("Event 1 is executed at date" + endigne.date());
      engine.scheduleRelative(1, () -> System.out.println("Event 2 is executed at
      date" + endigne.date());

});
engine.setFastMode(true);
engine.run();
```

**Машина состояний** [19] (State Machine) – это абстракция, часто используемая в имитационных моделях. Она состоит из состояний и переходов между ними. Переходы

определяют направления возможных изменений состояния. Переход может соединять одно состояние с другим либо замыкаться на то же самое состояние. При выполнении перехода текущее состояние машины изменяется с исходного состояния на целевое. Машина состояний является частным случаем более общего понятия конечного автомата (finite-state machine) [20]. В платформе Amalgama Platform машина состояний реализована с помощью Java-класса StateMachine, который:

- явно определяет, какие переходы возможны, а какие нет;
- предоставляет простой программный интерфейс для обработки входа и выхода из состояний;
- имеет встроенный программный интерфейс для сбора статистики о времени пребывания в каждом состоянии;
- содержит средства визуализации.

Визуальное представление простой машины состояний проиллюстрировано на рис. 1.

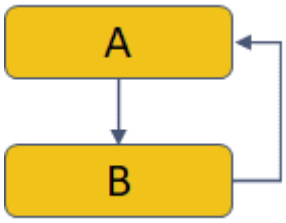


**Рис. 1. Пример машины состояний с двумя состояниями и переходами между ними**

Машина состояний из приведённой иллюстрации может быть создана с помощью кода, показанного на листинге 2.

Листинг 2 – Пример создания простой машины состояний

```
Engine engine = new Engine();
// Create a state machine with 2 states of type String - "A" and "B"
// And let "A" be the initial state
var stateMachine = new StateMachine<>(new String[] {"A", "B"}, "A", engine);
// Add all default transitions between all different states
stateMachine.addAllTransitions();
```

**Сервис** [21] (Service) – это абстракция, предназначенная для моделирования элементарной системы массового обслуживания. Сервис включает как очередь заявок, ожидающих обслуживания, так и процесс непосредственного обслуживания (рис. 2). При этом зона ожидания может содержать неограниченное число заявок, а обслуживающее устройство способно работать одновременно с одной или несколькими заявками. Класс *Service* автоматически поддерживает очередь заявок и определяет, какая из них должна быть обслужена следующей.

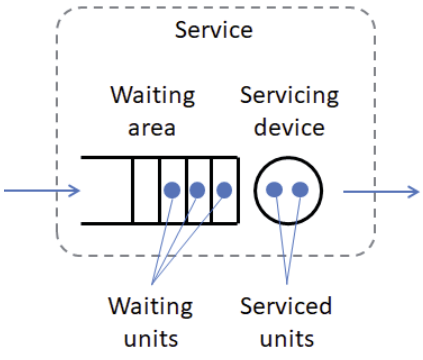


Рис. 2. Иллюстрация абстракции сервиса

Необходимость воспроизведения таких процессов возникает в большинстве имитационных моделей. Примеры типичны: грузовики, ожидающие разгрузки у терминала, или покупатели, стоящие в очереди к кассам супермаркета.

**Библиотека агентов в графах** [22] (Graph Agent Library) – это инструмент, предоставляемый платформой *Amalgama Platform* для поддержки моделей, в которых агенты «живут» в некоторой сети или графе. Такой тип декомпозиции предметной области характерен для многих коммерческих имитационных моделей.

Граф – это сеть, в которой размещаются агенты. Она состоит из узлов и дуг, соединяющих эти узлы. Узел соответствует точке на плоскости с координатами X и Y. Дуга соответствует ломаной линии, соединяющей некоторые два узла. Каждая дуга имеет направление. Двунаправленные связи представляются двумя дугами с противоположными направлениями. Библиотека предоставляет программный интерфейс для следующих действий:

- перемещение агентов в любую позицию на узле или дуге графа;
- отправку агентов в заданную точку графа по кратчайшему или произвольному маршруту;
- обработку событий, связанных с входом агента в узел или на дугу, а также с достижением цели;
- обработку столкновений агентов друг с другом;
- предоставление доступа к траекториям и путям движения агентов.

Пример создания простой имитационной модели движения агента по графу показан на листинге 3, а иллюстрация работы имитационной модели – на рис. 3.

Листинг 3 – Пример создания простой имитационной модели движения агента по графу

```
// Create a simulation engine instance
Engine engine = new Engine();
// Create a new instance of graph agent
var agent = new GraphAgent<>(engine);
// Place the agent into the graph environment
agent.setGraphEnvironment(environment);
// Jump the agent into the nodeA of the environment
agent.jumpTo(nodeA);
// The code below will print true
System.out.println(agent.getCurrentNode() == nodeA);
agent.moveTo(nodeB, 1);
```

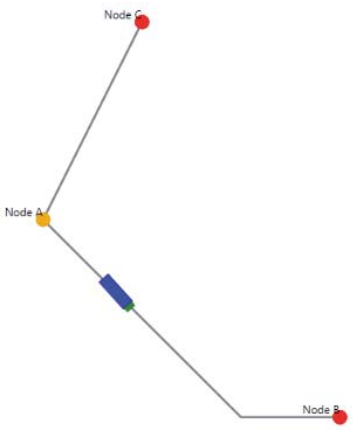


Рис. 3. Иллюстрация движения агента по графу

**Библиотека дискретных потоков** [23] (Discrete Rate Library) предназначена для моделирования поведения системы взаимосвязанных потоковых элементов, которые

могут принимать, накапливать или передавать непрерывное количество некоторого материала. Материал – это однородное вещество в моделируемой системе. Примеры материалов: жидкости («сырая нефть», «питьевая вода» и т.п.) и сыпучие грузы («песок», «гравий» и др.).

Потоковые элементы могут принимать, содержать или передавать смеси различных материалов. Библиотека содержит несколько типов потоковых элементов: резервуары, бункеры, разделители, объединители, конвейеры и клапаны. На рис. 4 показана простая имитационная модель распределения вещества по двум хранилищам, построенная с помощью библиотеки дискретных потоков.

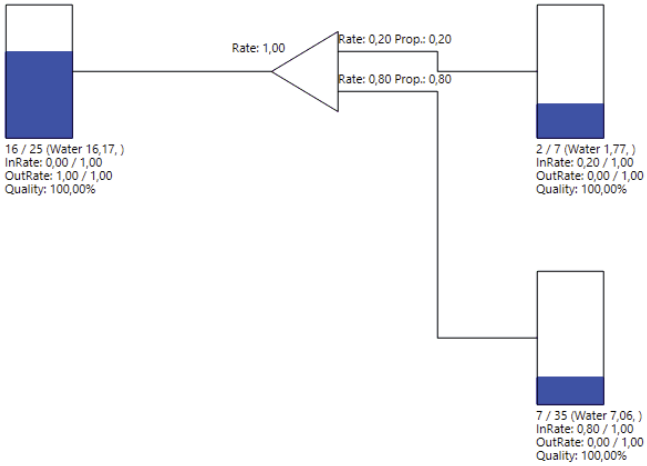


Рис. 4. Иллюстрация простой модели, построенной с помощью библиотеки дискретных потоков

**Значительная часть усилий тратится на алгоритмы планирования и диспетчеризации ресурсов**

Практика показывает, что при разработке процессно-ориентированных дискретно-событийных моделей существенная доля трудозатрат связана не с описанием транспортных или производственных операций, а с моделированием построения планов работы и алгоритмов диспетчеризации ресурсов. Причины этого следующие:

- **Настоящие предприятия планируют.** Даже если фактические процессы далеки от «идеальных», на любом предприятии существуют процедуры планирования. Поэтому для адекватного воспроизведения поведения моделируемой системы необходимо учитывать взаимодействие операционных процессов с планами на различных горизонтах;
- **Нет готовых инструментов планирования, заточенных под ИМ.** Существующие промышленные решения, такие как системы APS (Advanced Planning and Scheduling), разрабатывались для эксплуатации на предприятии, а не для экспериментов в имитационных моделях. Поэтому интеграция с ними часто не обеспечивает гибкости, необходимой для проведения сценарных анализов;
- **ИМ можно использовать как «тестовый стенд» для планирования.** Именно имитационные модели предоставляют уникальную возможность обкатки алгоритмов планирования в безопасной среде. В отличие от реального производства, модель позволяет проверять десятки и сотни альтернативных стратегий без риска срыва реального процесса. Таким образом, ИМ может быть не только потребителем планов, но и инструментом их разработки, отладки и сравнения.

Опыт авторов показывает, что нет универсальных подходов по реализации алгоритмов планирования в ИМ. Однако все же можно выделить основные принципы такой реализации:

- **Алгоритмы планирования лучше задавать в коде**, используя библиотеки оптимизации (например, линейного и целочисленного программирования) и эвристические методы, это дает возможность генерировать планы хорошего качества;
- **Имитационная модель и планировщик должны работать в одном временном и событийном контексте; э**то упрощает разработку ИМ и исключает расхождения между миром модели и миром планирования.

Мы рассматриваем ИМ как естественную среду для постепенной эволюции планирования: от воспроизведения фактических правил — к поиску новых алгоритмов и их интеграции в реальные системы.

**Программный код обладает преимуществами по сравнению с визуальными языками разработки ИМ**

Мы рассмотрели ключевые элементы, из которых строятся процессно-ориентированные дискретно-событийные модели, и подчеркнули, что их реализация возможна на основе стандартных средств промышленного программирования. Следующий вопрос – каким образом разработчики ИМ взаимодействуют с этими элементами: через код или через визуальные интерфейсы?

На наш взгляд, визуальные средства разработки ИМ во многом повторяют путь, который прошли визуальные инструменты в программировании, только с временным лагом. Они полезны для прототипирования и обучения, но не подходят для создания сложных промышленных моделей: снижают гибкость, плохо интегрируются с системами контроля версий и усложняют масштабирование [24].

Наш опыт показывает, что для разработчика ИМ программный код остаётся лучшим средством формализации задачи и организации мыслительного процесса. Код обеспечивает чёткую структуру, воспроизводимость и прозрачность решений, а также интегрируется в современные процессы разработки и позволяет легко использовать сторонние модули и библиотеки.

История подтверждает вывод о преимуществе программного кода для создания ИМ. Объектно-ориентированные языки возникли именно как инструмент моделирования сложных систем. Пионерский язык Simula был создан для дискретно-событийной симуляции и положил начало ООП [25]. Впоследствии язык Smalltalk закрепил объектную парадигму как основную в индустрии [26].

Сегодня ООП является основным подходом к разработке ПО, а такие языки, как Java, входят в число самых распространённых в мире. Таким образом, объектно-ориентированные языки сами по себе являются выразительными средствами моделирования, позволяющими формализовать структуру системы и описывать динамику поведения объектов при построении сложных моделей [27].

**Основные направления развития платформ для создания ИМ – стандартизация и интеграция со смежными инструментами**

Несколько направлений видятся наиболее перспективными для развития средств создания процессно-ориентированных дискретно-событийных ИМ:

- интеграция средств планирования и составления расписаний с ИМ. Например, интеграция ИМ со средствами решения задач линейного и целочисленного программирования; такая интеграция упростит имитацию планирования и диспетчеризации в моделях сложных систем;
- внедрение больших языковых моделей (large language models, LLM) в процесс разработки моделей с помощью программного кода; LLM помогут в декомпозиции предметной области, генерации шаблонов кода, реализации типовых алгоритмов и автоматизированном тестировании;

- переход от академических стандартов разработки процессно-ориентированных дискретно-событийных ИМ к стандартам де-факто, то есть таким, которые не обязательно были бы официально утверждены международными организациями стандартизации, но получили бы широкое распространение и признание в профессиональном сообществе, поскольку они доказали удобство и эффективность на практике. В смежной области моделирования физических систем и гибридных систем таким стандартом является FMI [28] / Modelica [29].

Эти шаги обеспечат дальнейшую интеграцию имитационного моделирования в индустриальные практики и повысят его применимость в решении сложных инженерных и управленческих задач.

**Литература**


1. **Bergström, F.** (2015). Simulation-Based Product Development and Manufacturing.
2. Simulation Software Market Size, Share, Trends Analysis Report [Электронный ресурс] // Coherent Market Insights. – Режим доступа: https://www.coherentmarketinsights.com/industry-reports/simulation-software-market (дата обращения: 20.09.2025).
3. **Santos, F.** Quantitatively assessing the benefits of model-driven development in agent-based simulation [Electronic resource] / F. Santos, I. Nunes, A. L. C. Bazzan // Simulation Modelling Practice and Theory. 2020. Vol. 102. Article 102063. URL: https://www.sciencedirect.com/science/article/abs/pii/S1569190X20300654 (обращение: 20.09.2025).
4. Официальный сайт компании AnyLogic Company – Режим доступа http://www.anylogic.ru/ (дата обращения: 20.09.2025).
5. Официальный сайт Amalgama Platform – Режим доступа https://platform.amalgamasimulation.com/ (дата обращения: 20.09.2025).
6. Официальный сайт MineTwin – Режим доступа https://mine-twin.com/ (дата обращения: 20.09.2025).
7. **Малыханов А.А.** Имитационное моделирование объединенного логистического комплекса по упаковке и отгрузке полимерных материалов / А. А. Малыханов, М. Е. Черненко, А. Л. Морозов // Имитационное моделирование. Теория и практика (ИММОД-2023): Сборник трудов одиннадцатой всероссийской научно-практической конференции по имитационному моделированию и его применению в науке и промышленности, Казань, 18–20 октября 2023 года. – Казань: Издательство АН РТ, 2023. – С. 400-406. – EDN TBXHYO.
8. **Малыханов А.А.** Использование гибкой архитектуры инструмента ИМ MineTwin для решения задач флюоритового рудника / А. А. Малыханов, М. Е. Черненко // Имитационное моделирование. Теория и практика (ИММОД-2023): Сборник трудов одиннадцатой всероссийской научно-практической конференции по имитационному моделированию и его применению в науке и промышленности, Казань, 18–20 октября 2023 года. – Казань: Издательство АН РТ, 2023. – С. 150-154. – EDN WKVDKL.
9. **Малыханов А.А.** От имитационной модели к цифровому двойнику: анализ опыта выполнения коммерческих проектов / А. А. Малыханов, В. Е. Черненко // Девятая всероссийская научно-практическая конференция по имитационному моделированию и его применению в науке и промышленности : Труды конференции, Екатеринбург, 16–18 октября 2019 года. – Екатеринбург: Издательство Уральского государственного педагогического университета, 2019. – С. 37-46. – EDN WFKLKE.A
10. **B. J. Schroer, F. T. Tseng, and S. X. Zhang.** Combining software engineering principles with discrete event simulation, in Proceedings of the 1989 Winter Simulation Conference, 1989.

11. **G. Wagner,** Tutorial: Information and process modeling for simulation, in Proceedings of the 2014 Winter Simulation Conference, 2014.
12. Collections (Java Platform SE 8) [Электронный ресурс]: документация Java / разработчик Oracle Corporation. – Режим доступа: https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html#sort-java.util.List- (дата обращения: 20.09.2025).
13. Metabase: open-source business intelligence platform [Электронный ресурс] / разработчик Metabase Inc.– Режим доступа: https://www.metabase.com/ (дата обращения: 20.09.2025).
14. Rich Client Platform [Электронный ресурс]: официальная документация Eclipse RCP / The Eclipse Foundation. – Режим доступа: https://wiki.eclipse.org/Rich_Client_Platform (дата обращения: 20.09.2025).
15. React: официальная документация JavaScript-библиотеки для создания пользовательских интерфейсов [Электронный ресурс] / разработчик Meta Platforms, Inc.– Режим доступа: https://react.dev/ (дата обращения: 20.09.2025).
16. Vue.js: официальная документация JavaScript-фреймворка для создания пользовательских интерфейсов [Электронный ресурс] / разработчик Evan You и сообщество разработчиков. – Режим доступа: https://vuejs.org/ (дата обращения: 20.09.2025).
17. JUnit 5: официальный сайт фреймворка для тестирования программного обеспечения на Java [Электронный ресурс] / разработчики JUnit Team. – Режим доступа: https://junit.org/ (дата обращения: 20.09.2025).
18. Amalgama Simulation Engine: концепции и архитектура движка моделирования [Электронный ресурс] / разработчик Amalgama. – Режим доступа: https://platform.amalgamasimulation.com/amalgama/KeyConcepts/engine.html (дата обращения: 20.09.2025).
19. State Machines in Amalgama: концепция конечных автоматов в платформе моделирования [Электронный ресурс] / разработчик Amalgama. – Режим доступа: https://platform.amalgamasimulation.com/amalgama/KeyConcepts/state_machine.html (дата обращения: 20.09.2025).
20. **Хопкрофт Дж.Э.** Введение в теорию автоматов, языков и вычислений / Дж. Э. Хопкрофт, Р. Мотвани, Дж. Д. Ульман. 2-е изд. Москва : Вильямс, 2002. С. 61. 528 с. : ил. – ISBN 5-8459-0261-4. – Текст : непосредственный.
21. Queues and Services in Amalgama: концепция очередей и сервисов в платформе моделирования [Электронный ресурс] / разработчик Amalgama. – Режим доступа: https://platform.amalgamasimulation.com/amalgama/KeyConcepts/queues_and_services.html (дата обращения: 20.09.2025).
22. Graph Agent Library: библиотека графовых агентов для моделирования [Электронный ресурс] / разработчик Amalgama. – Электрон. дан. – Режим доступа: https://platform.amalgamasimulation.com/amalgama/Libraries/graph_agent_library.html (дата обращения: 20.09.2025).
23. Discrete Rate Library: библиотека дискретных скоростей для моделирования процессов [Электронный ресурс] / разработчик Amalgama. – Режим доступа: https://platform.amalgamasimulation.com/amalgama/Libraries/discrete_rate_library.html
24. **Robert Schaefer**. 2011. On the limits of visual programming languages. SIGSOFT Softw. Eng. Notes 36, 2 (March 2011), 7–8. DOI https://doi.org/10.1145/1943371.1943373 (дата обращения: 20.09.2025).
25. **Dahl, Ole-Johan.** (2004). The Birth of Object Orientation: the Simula Languages. Lecture Notes in Computer Science. 2635. 15-25. 10.1007/978-3-540-39993-3_3. Kay, Alan. (1993).

26. The Early History of Smalltalk. Sigplan Notices - SIGPLAN. 28. 69-95. 10.1145/155360.155364.
27. **Dassen, J. & Groenewegen, Luuk & Koopman, Pieter.** (1999). Formalising Object-Oriented Modelling Languages. – Режим доступа: https://www.researchgate.net/publication/2645735_ Formalising_Object-Oriented_Modelling_Languages (обращение: 20.09.2025).
28. Functional Mock-Up Interface. The leading standard to exchange dynamic simulation models. – Электрон. дан. – Режим доступа: https://fmi-standard.org/ (дата обращения: 20.09.2025).
29. OpenModelica. – Электрон. дан. – Режим доступа: https://openmodelica.org/doc/OpenModelicaUsersGuide/v1.11.0/fmi.html.