Таблица 3

Значения среднего и дисперсии числа $n(t)$ при $\gamma = 0.5$

	$\mu = 0.25$	$\mu = 0.5$	$\mu = 1$
$\lambda = 1$	$\alpha = 12.657$, $D = 40.477$	$\alpha = 8.013$, $D = 20.202$	$\alpha = 4.5$, $D = 8.5$
$\lambda = 2$	$\alpha = 23.313$, $D = 80.954$	$\alpha = 16.027$, $D = 40.404$	$\alpha = 9$, $D = 17.001$
$\lambda = 3$	$\alpha = 37.97$, $D = 121.431$	$\alpha = 24.04$, $D = 60.606$	$\alpha = 13.5$, $D = 25.501$

Заключение

В данной работе была получено выражение для характеристическаой функции числа $n(T)$ событий дополнительного потока, формируемого заявками, поступившими в систему на интервале $[0,T]$. Также были выведены формулы для характеристик числа $n(T)$ событий *d*-потока. Получен вид распределения вероятностей $P(n,t)$ в рамках численного эксперимента при различных значениях параметров, найдены значения математического ожидания и дисперсии при заданных параметрах распределений. Полученные результаты могут быть использованы для анализа деятельности различных экономических систем.

ЛИТЕРАТУРА

1. *Даммер Д.Д.* Исследование математической модели страховой компании в виде системы массового обслуживания в случайной среде и с учетом единовременных страховых выплат // Компьютерные науки и информационные технологии. Материалы Международной научной конференции 2018. С. 117 – 121.

2. *Жидкова Л.А., Моисеева С.П.* Математическая модель потоков покупателей двухпродуктовой торговой компании в виде системы массового обслуживания с повторным обращением к блокам // Известия Томского политехнического университета. 2013. Том 322, №6, С. 5 – 9.

3. *Balsamo S., De Nitti Persone V., Inverardi P.* A review on Queueing Network Models with Finite Capacity Queues for Software Architectures Performance Prediction // Performance Evaluat. 2003. V.51. Iss. 2. P. 269--288.

4. *Матальцкий М.А., Станкевич С.Э.* НМ-сети как новые стохастические модели прогнозирования доходов различных объектов // Вестник ГрГУ. Сер.5 Экономика. 2009. №1. С. 107 – 115.

5. *Narayan Bhat U.* An Introduction to Queueing Theory: Modeling and Analysis in Applications. Boston: Birkhauser, 2008.

6. *Клейнрок Л.* Теория массового обслуживания /Пер. И. И. Грушко; ред. В. И. Нейман. М.: Машиностроение, 1979.

7. *Bartlett M.S.* The spectral analysis of point processes // J. Royal Stat. Soc. 1963. B 25. P.264—296.

8. *Cox D.R., Lewis P.A.W.* The statistical analysis of series of events. London: Methuen, 1966.

9. *Назаров А.А., Терпугов А.Ф.* Теория вероятностей и случайных процессов. Томск: Изд-во НТЛ, 2010.

10. *Назаров А.А., Терпугов А.Ф.* Теория массового обслуживания. Томск: Изд-во НТЛ, 2010.

РАЗРАБОТКА ФРЕЙМВОРКА ДИСКРЕТНО-СОБЫТИЙНОГО МОДЕЛИРОВАНИЯ

Заварзин А.С., Осипов О.А.

Саратовский государственный университет

zavarzin.27100@mail.ru, oleg.alex.osipov@gmail.com

Введение

За последнее время имитационное моделирование стало одним из основных и наиболее распространённых инструментов исследования сложных систем. Роль имитационного моделирования в промышленном и информационном мире велика. Благодаря возможности построения имитационной модели реальной системы исследователи могут проводить большое количество экспериментов, анализировать поведение системы, собирать статистические данные и делать соответствующие выводы. Эксперименты над имитационной моделью, в отличие от экспериментов над реальной системой, позволяют сэкономить временные и денежные ресурсы. Таким образом, имитационное моделирование является важным инструментом исследования и анализа поведения реальных систем, который в последнее время активно развивается.

Для создания имитационных моделей используются различные подходы:

- создание компьютерных моделей с помощью универсальных языков программирования (Java, C++, Delphi);
- разработка компьютерных моделей с применением специализированных языков моделирования (SIMULA, AnyLogic);
- построение компьютерных моделей и проведение имитационных экспериментов при помощи специализированных компьютерных сред с графическим редактором (AnyLogic, Arena, SIMUL8, NetLogo);
- включение средств имитационного моделирования в стандартные математические компьютерные системы (пакет Simulink системы MATLAB).

Большинство компьютерных сред имитационного моделирования [1,2] обладают некоторыми недостатками. Например, многие продукты вынуждают пользователя использовать свой собственный язык программирования. Некоторые системы являются достаточно гибкими и многофункциональными (AnyLogic, SIMUL8), но обладают высокой стоимостью. Остальные же системы примитивны или сложны в использовании.

В данной работе представлен фреймворк, разработанный на универсальном языке программирования Java для дискретно-событийного моделирования систем. Данный фреймворк позволяет конструировать модели систем и осуществлять имитационное моделирование с вычислением основных характеристик.

1. Дискретно-событийное моделирование

Дискретно-событийное моделирование [3,4] используется для построения модели, отражающей развитие системы во времени, когда состояния переменных меняются мгновенно в конкретные моменты времени. В такие моменты времени происходят *события*, при этом событие определяется как мгновенное возникновение, которое может изменить состояние системы. Динамическая природа дискретно-событийных имитационных моделей обязует следить за текущим имитационным временем по мере функционирования имитационной модели.

Необходимо также иметь механизм для продвижения имитационного времени от одного значения к другому. В имитационной модели переменная, обеспечивающая текущее значение модельного времени, называется *часами модельного времени*. Существует два основных способа продвижения модельного времени: продвижение времени от события к событию и продвижение времени с постоянным шагом. При использовании подхода продвижения времени от события к событию часы модельного времени в исходном состоянии устанавливаются в 0 и определяется время возникновения будущих событий. После этого часы переходят на время возникновения ближайшего события и в этот момент обновляется состояние системы с учетом произошедшего события, а также обновляются сведения о времени возникновения будущих событий. Затем часы продвигаются ко времени возникновения следующего ближайшего события, обновляется состояние системы и определяется время будущих событий, и т.д. Процесс продвижения модельного времени от времени возникновения одного события ко времени возникновения другого продолжается до тех пор, пока не будет достигнуто какое-либо условие останова, указанное до начала имитационного моделирования.

Поскольку в дискретно-событийной имитационной модели все изменения происходят только во время возникновения событий, периоды простоя системы пропускаются. Длительность интервала продвижения модельного времени от одного события к другому может быть различной.

Все дискретно-событийные имитационные модели включают ряд общих компонентов. В частности, дискретно-событийная имитационная модель, которая использует механизм продвижения модельного времени от события к событию содержит следующие основные компоненты:

- *состояние системы* – совокупность переменных состояния, необходимых для описания системы в определенный момент времени;

- *часы модельного времени* – переменная, указывающая текущее значение модельного времени;
- *список событий* – список, содержащий время возникновения следующих событий;
- *статистические счётчики* – переменные, предназначенные для хранения статистической информации о характеристиках системы;
- *синхронизирующая программа* – подпрограмма, которая определяет следующее событие в списке событий и затем переводит часы модельного времени на время возникновения этого события;
- *программа обработки событий* – подпрограмма, обновляющая состояние системы, когда наступает определённое событие;
- *генераторы случайных чисел* – для моделирования случайных величин, определённых в имитационном моделировании;
- *генератор отчётов* – для расчётов характеристик системы и вывода отчётов по окончании моделирования;
- *основная программа* – подпрограмма, которая задаёт последовательность вызова остальных подпрограмм, и именно основная программа реализует имитационное моделирование системы.

2. Архитектура фреймворка дискретно-событийного моделирования

В данном разделе будет представлено краткое описание архитектуры разработанного ядра фреймворка дискретно-событийного моделирования.

Архитектурным подходом при разработке фреймворка стала архитектура, управляемая событиями (*event-driven architecture*, EDA) [5]. Эта архитектура завязана на производителях и потребителях событий. Главная идея состоит в том, чтобы разделить части системы так, чтобы каждая из частей активизировалась, когда необходимое событие происходит в другой. Производитель события не знает за каким из событий наблюдает какой из потребителей. Также и другие потребители не знают, кто из них за каким событием наблюдает. Таким образом, главная идея заключается в расщеплении частей системы.

Компоненты данного фреймворка спроектированы с учётом дальнейшей расширяемости, ядро фреймворка содержит минимально необходимый набор интерфейсов, абстрактных классов и базовых реализаций, что позволяет пользователю в дальнейшем расширить фреймворк под свои задачи. При разработки был поставлен упор на простоту и гибкость использования. Поведение компонентов самой системы моделируется с помощью событий и их обработчиков. Все компоненты модели находятся в специально отведенном для них окружении (**Environment**), фактически оно отражает текущее состояние модели. Компоненты могут быть связаны между собой и события одних компонентов могут влиять на состояния других (например, событие поступления требования генерирует в источнике новое требование и отправляет его в очередь системы обслуживания). Для имитационного моделирования необходимо объявить часы модельного времени (**Clock**). У каждого события (**Event**) в системе должен присутствовать обработчик, он будет выполнять действие, которое должно произойти при наступлении события, т.е. логику события.

Работа имитационной модели прекращается при выполнении некоторого условия останова. Это может быть достижение некоторого заданного времени (**TimeStopCondition**), или отсутствие новых событий в системе (**EmptyStopCondition**). Также пользователь фреймворка может сам определить дополнительные условия останова реализовав интерфейс **Predicate**.

Перечислим основные интерфейсы и базовые имплементации в разработанном фреймворке:

- **Event** – интерфейс события, содержит методы отсрочки события (postpone), метод активации события (activate), который необходимо реализовать в дочерних классах;
- **AbstractEvent** – абстрактный класс, имплементирует интерфейс **Event**, содержит время активации события;
- **HandlerEvent** – наследует **AbstractEvent**, позволяет определить множество обработчиков, которые применяются для обработки событий, что позволяет разбить логику обработки на части;
- **EventProvider** – интерфейс контейнера для событий, содержит базовые методы работы с контейнерами (add, addAll, peek, getNext, remove, count, isEmpty), где метод getNext возвращает ближайшее событие, удаляя его из контейнера;
- **EventProviderImpl** – базовая реализация интерфейса **EventProvider**, использующая ArrayList<**Event**> в качестве контейнера;
- **EventHandler** – функциональный интерфейс для обработки событий;
- **TimeOut** – класс имплементирующий интерфейс **EventHandler**, содержит логику откладывания события;
- **Environment** – интерфейс окружения для имитационной модели;
- **EnvironmentImpl** – базовая реализация интерфейса **Environment**;
- **SimulationModel** – интерфейс для имитационной модели, содержит метод запуска (run) и метод для перехода к следующему состоянию (step);
- **AbstractSimulationModel** – абстрактный класс, реализует интерфейс **SimulationModel**, является базовой реализацией для всех моделей;
- **SimulationModelImpl** – наследует класс **AbstractSimulationModel**, не добавляет никакой новой логики, также является базовой реализацией для всех моделей;
- **TimerSimulationModel** – наследует класс **AbstractSimulationModel**, описывает модель таймера;
- **SimulationContext** – интерфейс контекста имитационной модели;
- **SimulationContextImpl** – базовая реализация интерфейса **SimulationContext**, содержит компоненты для системы (**Environment**, **Clock**, **EventProvider**) имитационного моделирования.

На данный момент реализованы следующие модули, изображённые на рис. 1.

- *simulation-core* – модуль, содержащий основные компоненты для дискретно-событийной модели (события, контейнеры для событий, обработчики событий, часы модельного времени и т.д.);
- *random-variable* – модуль, содержащий датчики псевдослучайных чисел с различными распределениями (экспоненциальное, эрланговское, нормальное и т.д.);
- *queueing-component-library* – модуль, содержащий необходимый набор компонентов для создания имитационных моделей процессов обслуживания;
- *examples* – примеры имитационных моделей.

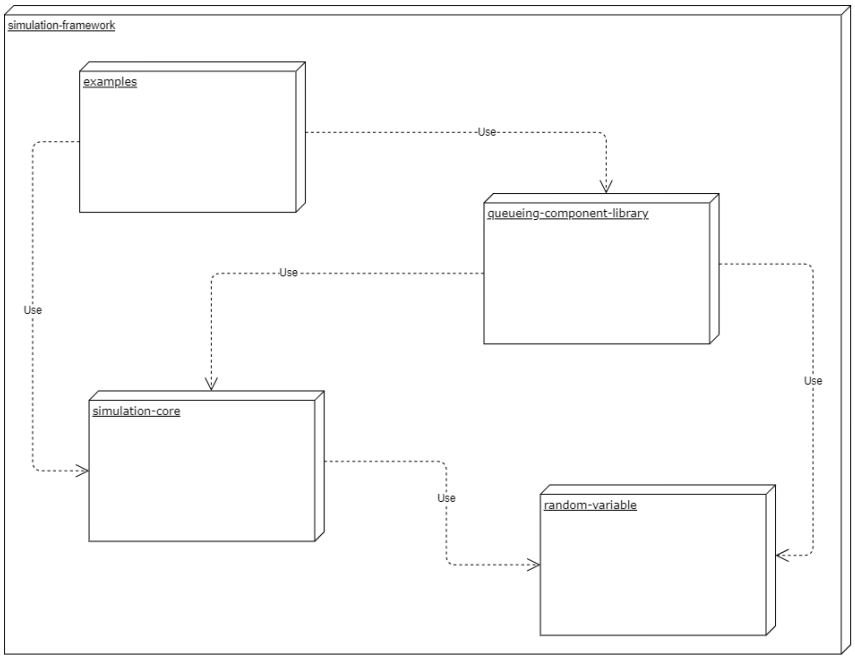

Рис. 1. Модули фреймворка дискретно-событийного моделирования

3. Пример использования

Рассмотрим пример использования фреймворка дискретно-событийного моделирования. Для начала, чтобы сконфигурировать модель, необходимо создать объекты, отвечающие за контекст модели, т.е. окружение, где будут находиться все объекты системы; часы модельного времени, отвечающие за синхронизацию и продвижение времени; провайдер (контейнер) для событий, позволяющий получать события в порядке их наступления. Данное действие можно описать так:

```
Environment env = new EnvironmentImpl();
Clock clock = new ClockImpl();
EventProvider eventProvider =
   new EventProviderImpl(Collections.emptyList());
SimulationContext simulationContext =
   new SimulationContextImpl(env, clock, eventProvider);
```

Затем необходимо инициализировать события и логику их обработки. Для данного примера были выбраны два таймера. Интервалы между срабатываниями первого таймера имеют экспоненциальное распределение. Срабатывание второго таймера происходят через фиксированные интервалы. Логика обработки событий передаётся через лямбда-выражение в функцию addHandler:

```
HandledEvent randomPeriodic =
   new HandledEvent.HandledEventBuilder(simulationContext)
      .periodic(new ExponentialRandomVariable(new Random(), 1))
      .addHandler(event -> System.out.println(
         "Message from periodic random event: " +
            event.getActivateTime()))
      .build();

HandledEvent constPeriodic =
     new HandledEvent.HandledEventBuilder(simulationContext)
```

```
.periodic(3)
.addHandler(event -> System.out.println(
   "Message from periodic const event: " +
      event.getActivateTime()))
.build();
```

После этого помещаем эти два события в провайдер, созданный ранее:

```
eventProvider.add(randomPeriodic);
eventProvider.add(constPeriodic);
```

Остаётся только инициализировать модель созданным выше контекстом, добавить условие останова и запустить имитационную модель:

```
SimulationModelImpl model =
   new SimulationModelImpl(simulationContext);
model.setStopCondition(new TimeStopCondition(10));
model.run();
```

Заключение

В данной работе был представлен разработанный фреймворк дискретно-событийного моделирования. В дальнейшем планируется доработка фреймворка, добавление компонентов для имитационного моделирования систем и сетей массового обслуживания, создание удобного пользовательского интерфейса для быстрого конструирования систем, добавление автоматизированного расчёта необходимой статистики и вывода результатов.

ЛИТЕРАТУРА

1. *Журавлёв С.С.* Краткий обзор методов и средств имитационного моделирования производственных систем // Журнал: Проблемы информатики // Рубрика: Имитационное моделирование технических систем и технологических процессов, 2009. 47 – 53.

2. *Рыжиков Ю.И., Соколов Б.В., Юсупов Р.М.* Проблемы теории и практики имитационного моделирования // Сб. докл. III Всерос. науч.-прак. конф. «Имитационное моделирование. Теория и практика» (ИММОД-2007). Санкт-Петербург, 17-19 окт. 2007. Т. 1. 58 – 70.

3. *Кельтон В., Лоу А.* Имитационное моделирование. Классика CS. 3-е изд. – СПб.: Питер: Киев: Издательская группа BHV, 2004. 847 с.

4. *Wagner G.* AOR modelling and simulation: Towards a general architecture for agent-based discrete event simulation // International Bi-Conference Workshop on Agent-Oriented Information System. – Springer, Berlin, Heidelberg, 2003. 174 – 188.

5. *Welsh M.* The staged event-driven architecture for highly-concurrent server applications // University of California, Berkeley. – 2000.

ИССЛЕДОВАНИЕ ЦИКЛИЧЕСКОЙ СИСТЕМЫ С ПОВТОРНЫМИ ВЫЗОВАМИ

Ключникова П.Н., Пауль С.В.

Томский государственный университет

polya.klyuch@gmail.com, paulsv82@mail.ru

Введение

В настоящее время телекоммуникационные системы, такие как компьютерные, телефонные сети, мобильная связь, call-центры, играют всё большую роль в нашей жизни. Отметим, что системы массового обслуживания являются адекватными математическими моделями вышеперечисленных реальных систем [1,2].

На данный момент есть много работ о моделировании различных телекоммуникационных систем [3,4,5], но большинство из них не рассматривает возможность повторного обращения к прибору. Системы массового обслуживания с повторными вызовами