Федеральное агентство по образованию
Уральский государственный технический университет – УПИ
имени первого Президента России Б. Н. Ельцина

М. В. Киселева

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ СИСТЕМ В СРЕДЕ ANYLOGIC

Учебно-методическое пособие

Научный редактор профессор, доктор технических наук Л. Г. Доросинский

Печатается по решению редакционно-издательского совета УГТУ-УПИ от 26.05.2009 г.

Екатеринбург
УГТУ-УПИ
2009

УДК 004.434:004.94(075.8)
ББК 32.973.26-018.2я73
К44

Рецензенты:
кафедра «Автоматизации и информационных технологий» – Уральский государственный колледж имени И. И. Ползунова
(доцент, канд. техн. наук В. В. Кийко);
С. Н. Киселев, ген. директор ЗАО «Уральский региональный межотраслевой центр научно-технического развития».

Киселева М. В.
К44 Имитационное моделирование систем в среде AnyLogic **:** учебно-методическое пособие / М. В. Киселёва. Екатеринбург : УГТУ - УПИ, 2009. 88 с.

Учебно-методическое пособие предназначено для изучения методов и средств построения имитационных моделей в инструментальной среде AnyLogic.

Содержит три лабораторные работы по созданию имитационных моделей – дискретно-событийной, системно-динамической и агентной. Для каждой модели приводится подробная постановка проблемы, разбирается структура и реализация модели в среде AnyLogic. Изучаются приемы наглядной визуализации исследуемого процесса, интерактивной анимации с возможностью изменения параметров системы по ходу моделирования процесса.

Библиогр.: 4 назв. Рис. 67.

УДК 004.434:004.94(075.8)

ББК 32.973.26-018.2я73

ВВЕДЕНИЕ

Имитационное моделирование

Моделирование – метод решения задач, при использовании которого исследуемая система заменяется более простым объектом, описывающим реальную систему и называемым моделью.

Моделирование применяется в случаях, когда проведение экспериментов над реальной системой невозможно или нецелесообразно, например, из-за высокой стоимости или длительности проведения эксперимента в реальном масштабе времени.

Различают физическое и математическое моделирование. Примером физической модели является уменьшенная копия самолета, продуваемая в потоке воздуха. При использовании математического моделирования поведение системы описывается с помощью формул. Особым видом математических моделей являются имитационные модели.

Имитационная модель – это компьютерная программа, которая описывает структуру и воспроизводит поведение реальной системы во времени. Имитационная модель позволяет получать подробную статистику о различных аспектах функционирования системы в зависимости от входных данных.

Имитационное моделирование – разработка компьютерных моделей и постановка экспериментов на них. Целью моделирования в конечном счете является принятие обоснованных, целесообразных управленческих решений. Компьютерное моделирование становится сегодня обязательным этапом в принятии ответственных решений во всех областях деятельности человека в связи с усложнением систем, в которых человек должен действовать и которыми он должен управлять. Знание принципов и возможностей имитационного моделирования, умение создавать и применять модели являются необходимыми требованиями к инженеру, менеджеру, бизнес-аналитику.

Назначение и возможности инструментальной среды AnyLogic

Современные системы моделирования поддерживают весь арсенал новейших информационных технологий, включая развитые графические оболочки для целей конструирования моделей и интерпретации выходных результатов моделирования, мультимедийные средства, анимацию в реальном масштабе времени, объектно-ориентированное программирование, Internet - решения и др. В данном пособии описываются методы и приемы построения моделей с помощью инструментальной системы AnyLogic.

Пакет AnyLogic – отечественный профессиональный инструмент нового поколения, который предназначен для разработки и исследования имитационных моделей. Разработчик продукта – компания «Экс Джей Текнолоджис» (XJ Technologies), г. Санкт-Петербург; электронный адрес: www.xjtek.ru.

AnyLogic был разработан на основе новых идей в области информационных технологий, теории параллельных взаимодействующих процессов и теории гибридных систем. Благодаря этим идеям чрезвычайно упрощается построение сложных имитационных моделей, имеется возможность использования одного инструмента при изучении различных стилей моделирования.

Программный инструмент AnyLogic основан на объектно-ориентированной концепции. Другой базовой концепцией является представление модели как набора взаимодействующих, параллельно функционирующих активностей. Активный объект в AnyLogic – это объект со своим собственным функционированием, взаимодействующий с окружением. Он может включать в себя любое количество экземпляров других активных объектов.

Графическая среда моделирования поддерживает проектирование, разработку, документирование модели, выполнение компьютерных экспериментов, оптимизацию параметров относительно некоторого критерия.

При разработке модели можно использовать элементы визуальной графики: диаграммы состояний (стейтчарты), сигналы, события (таймеры), порты и т.д.; синхронное и асинхронное планирование событий; библиотеки актив-

ных объектов.

Удобный интерфейс и многочисленные средства поддержки разработки моделей в AnyLogic делают не только использование, но и создание компьютерных имитационных моделей в этой среде моделирования доступными даже для начинающих.

При разработке модели на AnyLogic можно использовать концепции и средства из нескольких классических областей имитационного моделирования : динамических систем, дискретно-событийного моделирования, системной динамики, агентного моделирования. Кроме того, AnyLogic позволяет интегрировать различные подходы с целью получить более полную картину взаимодействия сложных процессов различной природы.

В данном пособии описываются три имитационные модели: дискретно-событийная, системно-динамическая и агентная. Для каждой модели приводится подробная постановка проблемы, разбирается структура модели, описывается процесс построения модели в среде AnyLogic и изучается ее поведение.

СРЕДСТВА ANYLOGIC ДЛЯ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ СИСТЕМ

Основные концепции

Две фазы моделирования. AnyLogic используется для разработки имитационных исполняемых моделей и последующего их прогона для анализа. Разработка модели выполняется в графическом редакторе AnyLogic с использованием многочисленных средств поддержки, упрощающих работу. Построенная модель затем компилируется встроенным компилятором AnyLogic и запускается на выполнение. В процессе выполнения модели пользователь может наблюдать ее поведение, изменять параметры модели, выводить результаты моделирования в различных формах и выполнять разного рода компьютерные эксперименты с моделью.

Для реализации специальных вычислений и описания логики поведения объектов AnyLogic позволяет использовать мощный современный язык Java.

Активные объекты, классы и экземпляры активных объектов. Основными строительными блоками модели AnyLogic являются *активные объекты*, которые позволяют моделировать любые объекты реального мира.

Класс в программировании является мощным средством, позволяющим структурировать сложную систему. Класс определяет шаблон, в соответствии с которым строятся отдельные экземпляры класса. Эти экземпляры могут быть определены как объекты других активных объектов.

Активный объект является *экземпляром класса* активного объекта. Чтобы создать модель AnyLogic, вы должны создать классы активных объектов (или использовать объекты библиотек AnyLogic) и задать их взаимосвязи. AnyLogic интерпретирует создаваемые вами графически классы активных объектов в классы *Java*, поэтому вы можете пользоваться всеми преимуществами объектно-ориентированного моделирования.

Активные объекты могут содержать вложенные объекты, причем уровень вложенности не ограничен. Это позволяет производить декомпозицию модели на любое количество уровней детализации.

Активные объекты имеют четко определенные интерфейсы взаимодействия – они взаимодействуют со своим окружением только посредством своих интерфейсных элементов.

Это облегчает создание систем со сложной структурой, а также делает активные объекты повторно используемыми. Создав класс активного объекта, вы можете создать любое количество объектов – экземпляров данного класса.

Каждый активный объект имеет структуру (совокупность включенных в него активных объектов и их связи), а также поведение, определяемое совокупностью переменных, параметров, стейтчартов и т.п. Каждый экземпляр активного объекта в работающей модели имеет свое собственное поведение, он может иметь свои значения параметров, функционирует независимо от других объектов, взаимодействуя с ними и с внешней средой.

Визуальная разработка модели. При построении модели используются средства визуальной разработки (введения состояний и переходов стейтчарта, введения пиктограмм переменных и т.п.), задания численных значений параметров, аналитических записей соотношений переменных и аналитических записей условий наступления событий. Основной технологией программирования в AnyLogic является визуальное программирование – построение с помощью графических объектов и пиктограмм иерархий структуры и поведения активных объектов.

Встроенный язык Java. AnyLogic является надстройкой над языком *Java* – одним из самых мощных и в то же время самых простых современных объектно-ориентированных языков. Все объекты, определенные пользователем при разработке модели с помощью графического редактора, компилируются в конструкции языка *Java*, а затем происходит компиляция всей собранной программы на *Java*, задающей модель, в исполняемый код. Хотя программирование сведено к минимуму, разработчику модели необходимо иметь некоторое представление об этом языке (например, знать синтаксически правильные конструкции).

Средства описания поведения объектов. Основными средствами описа-

ния поведения объектов являются переменные, события и диаграммы состояний. *Переменные* отражают изменяющиеся характеристики объекта. *События* могут наступать с заданным интервалом времени и выполнять заданное действие. *Диаграммы состояний (или стейтчарты)* позволяют визуально представить поведение объекта во времени под воздействием событий или условий, они состоят из графического изображения состояний и переходов между ними (т.е. по сути это конечный автомат). Любая сложная логика поведения объектов модели может быть выражена с помощью комбинации стейтчартов, дифференциальных и алгебраических уравнений, переменных, таймеров и программного кода на *Java*. Алгебраические и дифференциальные уравнения записываются аналитически.

Интерпретация любого числа параллельно протекающих процессов в модели AnyLogic скрыта от пользователя. Никаких усилий разработчика модели для организации *квазипараллелизма* интерпретации не требуется; отслеживание всех событий выполняется системой автоматически.

Модельное и реальное время. Понятие модельного времени является базовым в системах имитационного моделирования. *Модельное время* – это условное логическое время, в единицах которого определено поведение всех объектов модели. В моделях AnyLogic модельное время может изменяться либо непрерывно, если поведение объектов описывается дифференциальными уравнениями, либо дискретно, переключаясь от момента наступления одного события к моменту наступления следующего события, если в модели присутствуют только дискретные события. Моменты наступления всех планируемых событий в дискретной модели исполнительная система хранит в так называемом календаре событий, выбирая оттуда наиболее раннее событие для выполнения связанных с ним действий. Значение текущего времени в моделях AnyLogic может быть получено с помощью функции *time()*.

Единицу модельного времени разработчик модели может интерпретировать как любой отрезок времени: секунду, минуту, час или год. Важно только, чтобы все процессы, зависящие от времени, были выражены в одних и тех же

единицах. При моделировании физических процессов все параметры и уравнения должны быть выражены в одних и тех же единицах измерения физических величин.

Интерпретация модели выполняется на компьютере. Физическое время, затрачиваемое процессором на имитацию действий, которые должны выполняться в модели в течение одной единицы модельного времени, зависит от многих факторов. Поэтому единица физического времени и единица модельного времени не совпадают.

В AnyLogic приняты два режима выполнения моделей: режим виртуального времени и режим реального времени. В *режиме виртуального времени* процессор работает с максимальной скоростью без привязки к физическому времени. Данный режим используется для факторного анализа модели, набора статистики, оптимизации параметров модели и т.д. Поскольку анимация и другие окна наблюдения за поведением модели обычно существенно замедляют скорость интерпретации модели на компьютере, для повышения скорости выполнения эти окна нужно закрыть.

В *режиме реального времени* пользователь задает связь модельного времени с физическим временем, т.е. устанавливает ограничение на скорость процессора при интерпретации модели. В этом режиме задается количество единиц модельного времени, которые должны интерпретироваться процессором в *одну секунду*. Обычно данный режим включается для того, чтобы визуально представить функционирование системы в реальном темпе наступления событий, проникнуть в суть процессов, происходящих в модели.

Соотношение физического и модельного времени при работе модели можно понять на таком примере. При коэффициенте ускорения 4, если процессор успевает выполнить менее чем за 1 с все операции, которые в модели определены в течение 4-х единиц модельного времени, то он будет ждать до конца секунды. Если же процессор не успевает выполнить все операции, то у него не будет интервала ожидания, и коэффициент ускорения будет меньше того, который установлен пользователем.

Анимация поведения модели. AnyLogic имеет удобные средства представления функционирования моделируемой системы в живой форме динамической анимации, что позволяет «увидеть» поведение сложной системы. Визуализация процесса функционирования моделируемой системы позволяет проверить адекватность модели, выявить ошибки при задании логики.

Средства анимации позволяют пользователю легко создавать виртуальный мир (совокупность графических образов, ожившую мнемосхему), управляемый динамическими параметрами модели по законам, определенным пользователем с помощью уравнений и логики моделируемых объектов. Графические элементы, добавленные на анимацию, называются динамическими, поскольку все их параметры: видимость, цвет и т.п. – можно сделать зависимыми от переменных и параметров модели, которые меняются со временем при выполнении модели.

С помощью совершенной технологии визуализации работающих моделей AnyLogic можно создавать интерактивные анимации произвольной сложности, связывая графические объекты (в т.ч. импортированные чертежи) во встроенном редакторе с объектами модели. Как и модель, анимация имеет иерархическую структуру, которая может динамически изменяться. Возможно создание нескольких точек зрения или нескольких уровней детальности в пределах одной анимации. Элементы управления и развитая бизнес-графика превращают анимацию модели в настоящую панель управления для оценки эффективности решений. В AnyLogic поддерживается как двумерная, так и трёхмерная анимация.

Интерактивный анализ модели. Многие системы моделирования позволяют менять параметры модели только до запуска модели на выполнение. AnyLogic позволяет пользователю вмешиваться в работу модели, изменяя параметры модели в процессе ее функционирования. Примером таких средств являются *слайдеры*, которые могут быть введены в окно анимации.

Пользовательский интерфейс

После запуска AnyLogic открывается рабочее окно, в котором для продолжения работы надо создать новый проект или открыть уже существующий.

Начиная с версии 6.4 AnyLogic предоставляет пользователям возможность использовать шаблоны моделей при создании новых моделей. Чтобы создать новый проект, щелкните по соответствующей кнопке на панели инструментов или выберите пункт меню **Файл | Создать проект** и затем из ниспадающего меню – **Модель.** Откроется диалоговое окно **Новая модель**, где задается имя и местоположение нового проекта. Далее следуйте указаниям *Мастера создания модели*. Можно создавать новую модель «с нуля» или использовать шаблон.

При открытии проекта (нового или существующего) AnyLogic всегда открывает среду разработки проекта – графический редактор модели (рис. 1). Рассмотрим основные составляющие этого редактора.

Окно проекта обеспечивает легкую навигацию по элементам проекта, таким как пакеты, классы и т.д. Поскольку проект организован иерархически, то он отображается в виде дерева: сам проект образует верхний уровень дерева рабочего проекта, пакеты – следующий уровень, классы активных объектов и сообщений – следующий и т.д. Можно копировать, перемещать и удалять любые элементы дерева объектов, легко управляя рабочим проектом.

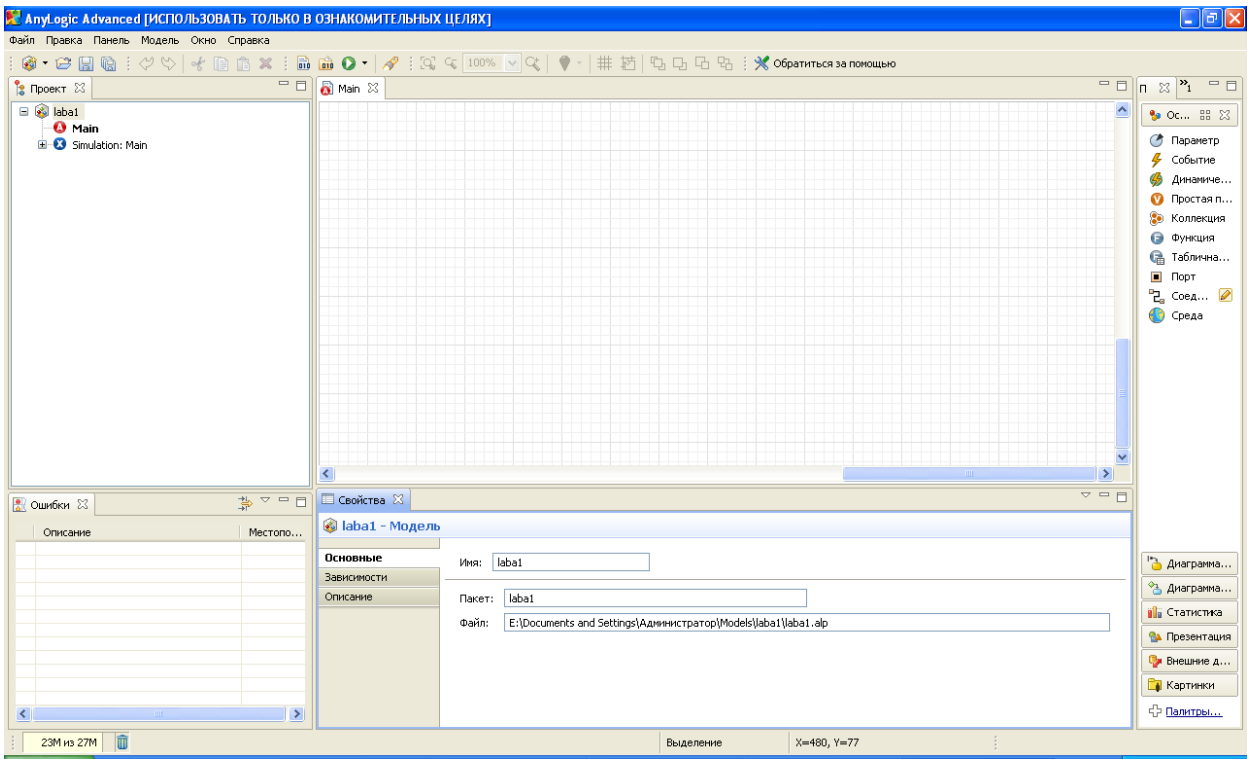


Рис. 1

Одна из ветвей в дереве проекта имеет название *Simulation:Main* (эксперимент). Эксперимент хранит набор настроек, с помощью которых конфигурируют работу модели. Один эксперимент создается автоматически при создании проекта. Это *простой эксперимент* с именем *Simulation*, позволяющий визуализировать модель с помощью анимации и поддерживающий инструменты для отладки модели. Простой эксперимент используется в большинстве случаев. Поддерживается ещё несколько типов экспериментов для различных задач моделирования.

Структурная диаграмма. При построении модели нужно задать ее структуру (т.е. описать, из каких частей состоит модель системы) и поведение отдельных объектов системы. В AnyLogic структурными элементами модели являются так называемые *активные объекты*. Активный объект имеет структуру и поведение. Элементы структуры – это другие активные объекты, включенные как составные элементы данного активного объекта, и связи, которые существуют между включенными активными объектами. Активные объекты могут содержать: события, стейтчарты, переменные, функции, уравнения, параметры.

Структура активного объекта задается графически на структурной диаграмме. Поведение задается с помощью стейтчарта и определяет реакции активного объекта на внешние события – логику его действий во времени.

Диаграмма состояний (или стейтчарт – *statechart)* – это модифицированные графы переходов конечного автомата. Стейтчарт позволяет графически задать пространство состояний алгоритма поведения объекта, а также события, которые являются причинами срабатывания переходов из одних состояний в другие, и действия, происходящие при смене состояний. Стейтчарты соответствуют стандарту UML. Они сохраняют графический вид, атрибуты и семантику выполнения, определенную в UML (*Unified Modeling Language*). Стейтчарты в AnyLogic поддерживают следующие типы событий: *сигнал* – объект может послать сигнал другому объекту, чтобы уведомить его о чем-то; *таймаут* – в течение заданного промежутка времени в стейтчарте ничего не происходит; *событие* – событие, при котором значение булево выражения становится «истина».

Кроме того, в окне редактора для модели можно построить двумерное или трехмерное анимационное представление, которое помогает понять, что происходит с моделью во времени. Именно в этом окне визуально представляется имитация поведения моделируемой системы. Элементы анимационной картинки имеют свои параметры, которые могут быть связаны с переменными и параметрами модели. Изменение переменных модели во времени ведет к изменению графического образа, что позволяет пользователю наглядно представить динамику моделируемой системы с помощью динамически меняющейся графики.

Окно свойств. В редакторе AnyLogic для каждого выделенного элемента модели существует свое окно свойств, в котором указываются свойства (параметры) этого элемента. При выделении какого-либо элемента в окне редактора снизу появляется окно свойств, показывающее параметры данного выделенного элемента. Окно свойств содержит несколько вкладок. Каждая вкладка содержит элементы управления, такие как поля ввода, флажки, переключатели, кнопки и

т.д., с помощью которых можно просматривать и изменять свойства элементов модели. Число вкладок и их внешний вид зависит от типа выбранного элемента.

Окно палитры. Содержит элементы (графические объекты), которые могут быть добавлены на структурную диаграмму. Элементы разбиты по группам, отображаемым на разных вкладках. Чтобы добавить объект палитры на диаграмму, щелкните сначала по элементу в палитре, а затем щелкните по диаграмме.

Параметры. Активный объект может иметь параметры. Параметры обычно используются для задания характеристик объекта. Вы можете задать различные значения параметров для разных объектов одного и того же класса, что требуется в тех случаях, когда объекты имеют одинаковое поведение, но их характеристики разные. Возможно описание параметров любых *Java*-классов.

Чтобы создать параметр класса активного объекта (рис. 2), в окне **Проект** щелкните мышью по классу активного объекта. В окне **Свойства** щелкните по кнопке **Новый параметр.** Задайте свойства параметра в открывшемся диалоговом окне **Параметр**.

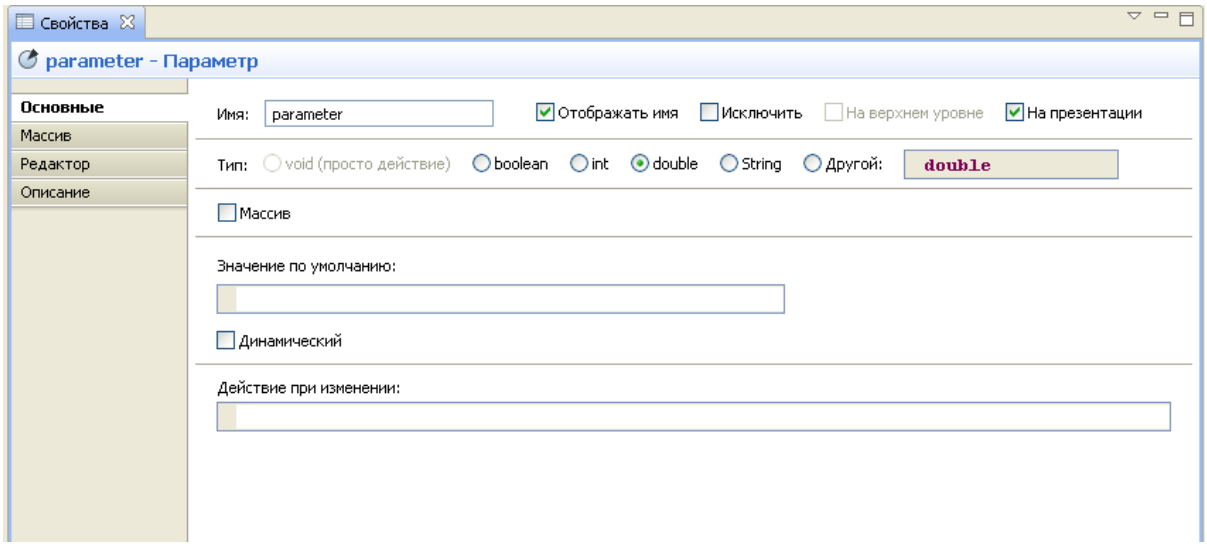


Рис. 2

Переменные. Активный объект может содержать переменные. Переменные могут быть либо внутренними, либо интерфейсными. Активный объект может иметь переменные, моделирующие, меняющиеся во времени величины. Переменные могут быть вынесены в интерфейс активного объекта и связаны с

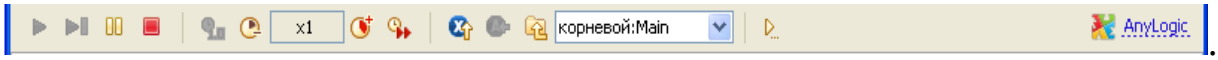
переменными других активных объектов. Тогда при изменении значения одной переменной будет немедленно меняться и значение связанной с ней зависимой переменной другого объекта. Этот механизм обеспечивает непрерывное и/или дискретное взаимодействие объектов.

Передача сообщений. AnyLogic позволяет передавать информацию от одного объекта другому путем передачи специальных пакетов данных – сообщений. С помощью передачи сообщений можно реализовать механизм оповещения – сообщения будут представлять команды или сигналы, посылаемые системой управления. Можно также смоделировать поток заявок, в этом случае сообщения будут представлять собой заявки – объекты, которые производятся, обрабатываются, обслуживаются или еще каким-нибудь образом подвергаются воздействию моделируемого процесса (документы в модели бизнес-процесса, клиенты в модели системы массового обслуживания, детали в производственных моделях).

Сообщения принимаются и посылаются через специальные элементы активных объектов – *порты*. Обмен сообщениями возможен только между портами, соединенными соединителями – элементами, играющими роль путей движения сообщений.

Чтобы соединить порты вложенных объектов, щелкните мышью по кнопке панели инструментов **Соединитель**, а затем щелкните мышью поочередно по обоим портам. Чтобы добавить точку изгиба**,** щелкните мышью по кнопке панели инструментов **Редактировать точки**.

Запуск и просмотр модели. Запускать и отлаживать модель можно с помощью меню **Модель** и панели инструментов:



При исполнении модели запустится компилятор, который построит исполняемый код модели в языке *Java*, скомпилирует его и затем запустит модель на исполнение.

Для запуска модели щелкните по кнопке **Выполнить**, затем выберите эксперимент из выпадающего списка. После этого откроется окно презентации,

отображающее созданную презентацию для запущенного эксперимента. Щелкните по кнопке, чтобы запустить модель и перейти на презентацию.

В окне презентации можно увидеть: анимированную диаграмму модели, окна инспекта элементов модели, ожившую анимацию, диаграммы состояний, графики статистики.

Стохастическое моделирование

Существуют различные способы описания стохастического поведения: использование различных законов распределения и/или статистики для описания времени между событиями; реализация недетерминированных алгоритмов, например, утеря с определенной вероятностью заявки. В случае, если несколько событий должны произойти в одно и то же время, AnyLogic позволяет выбрать событие, которое должно произойти, случайно, используя равномерное распределение.

AnyLogic поддерживает *Stat::Fit* – специализированное программное обеспечение для обработки статистики, которое позволяет подбирать распределения по имеющейся выборке.

Рассмотрим средства для разработки стохастических моделей в AnyLogic.

В AnyLogic включено 37 генераторов случайных величин с наиболее часто встречающимися вероятностными распределениями: равномерным, экспоненциальным, Бернулли, биномиальным и т.п. Их описание можно найти в *руководстве пользователя* AnyLogic.

Все классы вероятностных распределений унаследованы от класса *Distr*. Они называются *DistrExponential*, *DistrChi*, *DistrNormal* и т.д. Класс *Distr* имеет только один абстрактный метод *get()*, возвращающий случайное значение, сгенерированное по этому закону распределения. Пользователь может определить свое вероятностное распределение, для чего нужно создать свой класс распределения и унаследовать его от базового класса *Distr*. Методы классов распределений подробно описаны в *справочнике классов* AnyLogic.

Вызвать метод очень просто, например, *exponential*(0.6) или *uniform*(-1,1),

который вернет соответствующее случайное значение.

В общем случае все параметры производительности систем, функционирующих в условиях неопределенности, являются стохастическими. AnyLogic включает средства, позволяющие выполнять анализ случайных величин и визуализировать их распределения. Случайная величина (СВ) здесь задается набором данных, в котором хранятся не только все ее конкретные значения, но и автоматически подсчитываются статистические характеристики: количество реализаций, среднее, минимальное и максимальное значения, дисперсия, средне-квадратичное отклонение (СКО) и доверительный интервал для среднего значения. На выходе стохастической модели СВ легко представить графически в виде гистограммы.

Проведение экспериментов

С помощью экспериментов задаются конфигурационные настройки модели. AnyLogic поддерживает несколько типов экспериментов: простой эксперимент, эксперимент для варьирования параметров, оптимизационный и др. На рис. 3 показано окно выбора эксперимента.

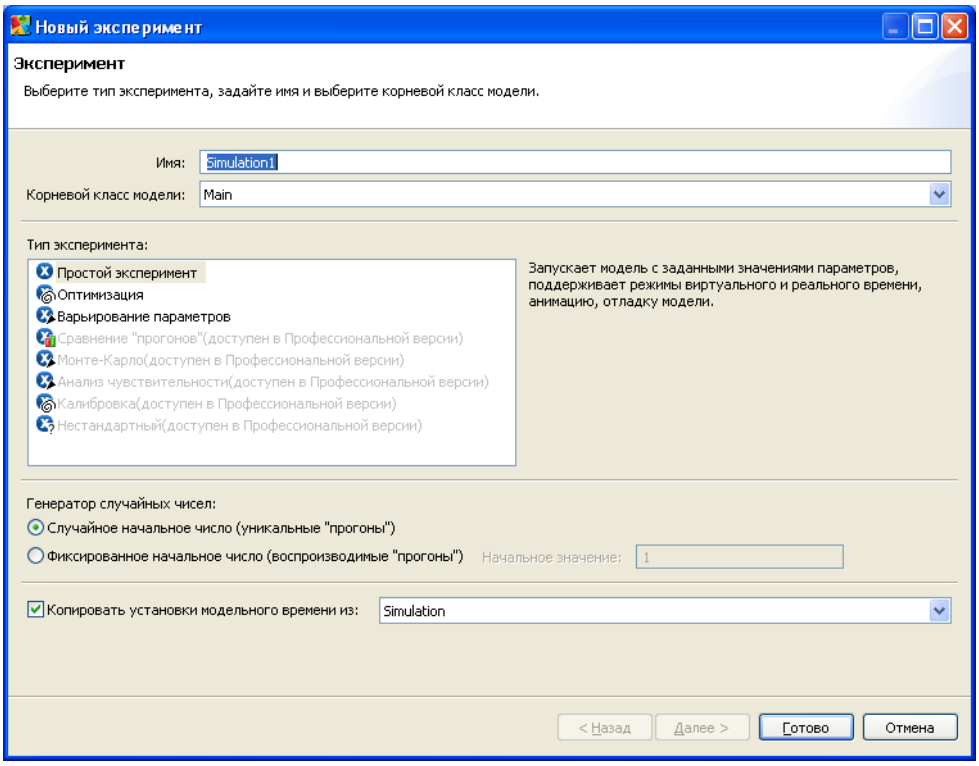


Рис. 3

Простой эксперимент. Задачи вида «что – если» (так называемая *прямая задача имитационного моделирования ИМ*) в AnyLogic решаются с помощью *простого эксперимента*. Простой эксперимент (с именем *Simulation*) создается автоматически при создании проекта. Он позволяет визуализировать модель с помощью анимации, графиков (диаграмм) и т.п. Широкие возможности для отображения данных предоставляет библиотека бизнес-графики (Business Graphics Library).

Для построения, например, графика зависимости переменных от времени в поле анимации сначала нужно построить прямоугольник, в переделах которого будет размещаться график, после чего в любое место поля редактора перенести экземпляр объекта *ChartTime* из Business Graphics Library. Затем в окне свойств данного объекта следует настроить параметры, определяющие цвет и толщину линий, имена отображаемых переменных, названия переменных, которые будут отображаться, цвет текста и т.д.

Простой эксперимент используется в большинстве случаев при разработке и анализе моделей, созданных в AnyLogic. В частности, он поддерживает средства для отладки модели. Можно организовать несколько простых экспериментов с различными значениями исходных факторов и, сделав один из этих экспериментов *текущим*, запустить модель на выполнение.

Эксперимент для варьирования параметров. Анализ чувствительности модели. Анализ чувствительности модели – процедура оценки влияния исходных гипотез и значений ключевых факторов на выходные показатели модели. Обычно эксперимент с варьированием параметров и анализом реакции модели помогает оценить, насколько чувствительным является выдаваемый моделью прогноз к изменению гипотез, лежащих в основе модели. При анализе чувствительности обычно рекомендуется выполнять изменение значений факторов по отдельности, что позволяет ранжировать их влияние на результирующие показатели.

В AnyLogic доступен механизм автоматического запуска модели заданное

количество раз с изменением значений выбранных параметров – это *эксперимент для варьирования параметров*. При запуске данного эксперимента пользователь может изучить и сравнить поведение модели при разных значениях параметров с помощью графиков.

Чтобы запустить такой эксперимент, нужно выполнить следующее:

- создать эксперимент для варьирования параметров;
- сконфигурировать эксперимент, выбрав параметры модели, которые вы хотите варьировать, и, задав значения, которые эти параметры должны будут принять за определенное вами количество прогонов модели, в окне свойств данного эксперимента;
- запустить модель, выбрав данный эксперимент в качестве текущего.

Такой вид эксперимента не поддерживает визуализацию работы модели с помощью анимации.

Оптимизационный эксперимент. Используется для решения задач *количественного анализа* (расчет показателей эффективности системы). Поиск тех значений факторов, которые определяют наиболее предпочтительный вариант решения, называется *обратной задачей ИМ*. Обратные задачи моделирования отвечают на вопрос о том, какое решение из области допустимых решений обращает в максимум показатель эффективности системы. Для решения обратной задачи многократно решается прямая задача. В случае, когда число возможных вариантов решения невелико, решение обратной задачи сводится к простому перебору всех возможных решений. Сравнивая их между собой, можно найти оптимальное решение.

Если перебрать все варианты решений невозможно, то используются *методы направленного перебора с применением эвристик*. При этом оптимальное или близкое к оптимальному решение находится после многократного выполнения последовательных шагов (решений прямой задачи и нахождения для каждого набора входных параметров модели вектора результирующих показателей). Правильно подобранная эвристика приближает эксперимент к оптимальному решению на каждом шаге.

В качестве блока регистрации значений выходных показателей и выбора очередного приближения при оптимизации (рис. 4) пользователь может использовать любой внешний оптимизатор или же оптимизатор *OptQuest*, встроенный в AnyLogic. Оптимизатор *OptQuest* разработан недавно на основе метаэвристик рассеянного поиска (scatter search) и поиска «табу» (tabu search). Этот оптимизатор является лучшим из предлагаемых на рынке профессиональных пакетов оптимизации для решения сложных проблем оптимизации.

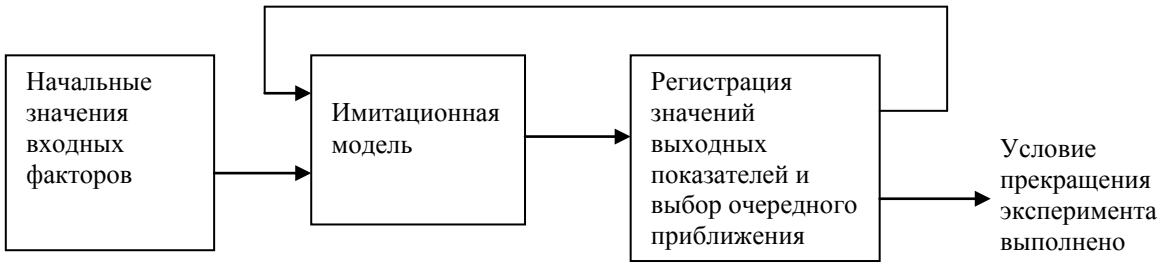


Рис. 4

Оптимизатор *OptQuest* запускается прямо из среды разработки модели. Чтобы настроить оптимизацию в AnyLogic необходимо выполнить следующее:

1) создать в разработанной модели оптимизационный эксперимент;
2) задать оптимизационные параметры и области их изменения;
3) задать условие остановки модели после каждого прогона. Это может быть либо остановка по времени выполнения прогона, либо остановка по условиям, накладываемым на переменные модели;
4) задать целевую функцию, т.е. исследуемую реакцию системы;
5) задать ограничения, которые в конце каждого прогона определяют, допустимо ли значение вектора исходных входных факторов. Ограничения можно не задавать (т.е. это опционально);
6) задать условия прекращения эксперимента.

После запуска модели оптимизационный эксперимент найдет наилучшие значения входных параметров, при которых заданная целевая функция обратится в минимум или максимум.

МОДЕЛИРОВАНИЕ СИСТЕМЫ МАССОВОГО ОБСЛУЖИВАНИЯ В ANYLOGIC (ЛАБОРАТОРНАЯ РАБОТА № 1)

Цель работы

Изучить пользовательский интерфейс и инструментальные средства пакета AnyLogic для имитационного моделирования систем массового обслуживания.

Порядок выполнения работы

Лабораторные работы предусмотрены для версии 6.x продукта AnyLogic, для версии 5.x они могут отличаться.

Модель AnyLogic представляет собой файл с именем, заданным пользователем, и расширением *.alp*. При создании новой модели можно сразу указать полный путь и имя каталога, в котором будет находиться ваша модель.

В ходе выполнения лабораторной работы необходимо научиться создавать дискретно-событийные модели с помощью библиотеки Enterprise Library пакета AnyLogic. Для этого активно используйте справочное руководство по Enterprise Library и учебное пособие по Enterprise Library (меню **Справка**).

При выполнении лабораторной работы студент сначала выполняет общее задание, а затем индивидуальное задание по варианту, предлагаемому преподавателем.

Общая информация о создании моделей в Enterprise Library

Для создания новой модели щелкните мышью по кнопке **Создать проект**. Появится диалоговое окно, в котором вы должны будете дать имя файлу вашей модели и выбрать каталог, где он будет храниться.

Рассмотрим рабочее окно AnyLogic. В левой части рабочей области находится панель «Проект». Панель «Проект» обеспечивает легкую навигацию по элементам моделей, открытых в текущий момент времени. Поскольку модель организована иерархически, то она отображается в виде дерева: сама модель образует верхний уровень дерева; эксперименты, классы активных объектов и

Java-классы образуют следующий уровень; элементы, входящие в состав активных объектов, вложены в соответствующую подветвь дерева класса активного объекта и т.д.

В правой рабочей области отображается панель «Палитра», а внизу – панель «Свойства». Панель «Палитра» содержит разделенные по категориям элементы, которые могут быть добавлены на диаграмму класса активного объекта или эксперимента. Панель «Свойства» используется для просмотра и изменения свойств выбранного в данный момент элемента (или элементов) модели.

В центре рабочей области AnyLogic открывается графический редактор диаграммы класса активного объекта *Main*.

Чтобы добавить объект на блок-схему модели, щелкните по объекту в окне палитры **Enterprise Library** и перетащите его мышью на структурную диаграмму. При этом его свойства будут отображены на панели «Свойства». В этом окне вы можете изменять свойства элемента в соответствии с требованиями вашей модели. Позднее для изменения свойств элемента нужно будет сначала щелчком мыши выделить его на диаграмме или в дереве проекта.

Объекты должны взаимодействовать между собой, поэтому вы должны будете соединять их друг с другом. Можно соединять объекты с помощью мыши, перетаскиванием порта одного объекта на порт другого или с помощью специального средства «Соединитель». Чтобы соединить порты объектов, щелкните мышью по кнопке панели инструментов **Соединитель**, а затем щелкните мышью поочередно по обоим портам. Для добавления точки изгиба щелкните мышью по кнопке панели инструментов **Редактировать точки**.

Модель выполняется в соответствии с набором конфигурационных установок, называемым экспериментом. Вы можете создать несколько экспериментов и изменять рабочую конфигурацию модели, просто меняя текущий эксперимент модели. Один эксперимент, названный *Simulation*, создается автоматически. Выберите его щелчком мыши по элементу дерева и измените настройки модели в окне **Свойства** (рис. 5). Окно **Свойства** имеет вкладки: ос-

новные, дополнительные, модельное время, презентация, окно, параметры, описание.

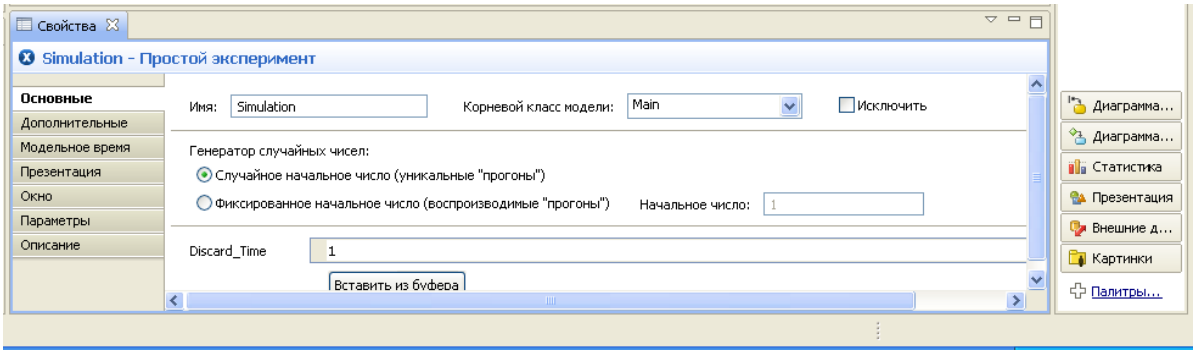


Рис. 5

На вкладке **Основные** можно выбрать класс, который будет запущен при запуске модели. По умолчанию в качестве корневого объекта выбран объект класса *Main*, автоматически создаваемого в каждой модели. Вы можете переименовывать классы модели. Для этого нужно выделить класс щелчком мыши по значку класса в дереве модели и затем изменить его имя в окне **Свойства.**

На вкладке **Модельное время** можно:

1) задать режим моделирования. В режиме реального времени задается связь модельного времени с физическим, т.е. задается количество единиц модельного времени, выполняемых в одну секунду. Режим реального времени лучше всего подходит для показа анимации. В режиме виртуального времени модель выполняется без привязки к физическому времени – она выполняется так быстро, как это возможно. Данный режим лучше всего подходит, когда требуется моделировать работу системы в течение достаточно длительного периода времени;

2) запустить модель так, чтобы она работала бесконечно, но можно и остановить ее в заданный момент времени. Вы можете остановить модель по достижении переменной заданного значения или по выполнении какого-нибудь определенного условия.

Дополнительные свойства эксперимента (вкладка **Дополнительные**) позволяют управлять выполнением модели. Можно задать действие перед и после запуска модели, а также задать численные методы для прогона и точность

получаемых значений.

На вкладе **Презентация** можно определить вид и скорость выполнения прогона.

Задание к лабораторной работе

Построим с помощью элементов библиотеки Enterprise Library модель простой системы массового обслуживания – модель банковского отделения.

В банковском отделении находятся банкомат и стойки банковских кассиров, которые предназначены для быстрого и эффективного обслуживания посетителей банка. Операции с наличностью клиенты банка производят с помощью банкомата, а более сложные операции, такие как оплата счетов, – с помощью кассиров.

Необходимо произвести оценку затрат операций и определить, сколько денег тратится на обслуживание одного клиента и какую часть этой суммы составляют накладные расходы на оплату работы персонала банка, а какую – на обслуживание посетителей.

1. Создание нового проекта.

Создайте новую модель. Переименуйте класс *Main* в *Model*. В свойствах эксперимента *Simulation* задайте выполнение модели в режиме реального времени с выполнением одной единицы модельного времени в одну секунду. В этой модели под единицей модельного времени мы будем понимать одну минуту работы банковского отделения.

2. Создание блок-схемы.

Создайте блок-схему модели, которая пока будет состоять только из банкомата. Для этого перетащите в окно структуры элементы библиотеки Enterprise Library и соедините их так, как показано на рис. 6.

Объект *source* генерирует заявки (entities) определенного типа через заданный временной интервал. Заявки представляют собой объекты, которые производятся, обрабатываются, обслуживаются или еще каким-нибудь образом подвергаются воздействию моделируемого процесса: это могут быть клиенты в

системе обслуживания, детали в модели производства, документы в модели документооборота и др. В нашем примере заявками будут посетители банка, а объект *source* будет моделировать их приход в банковское отделение.

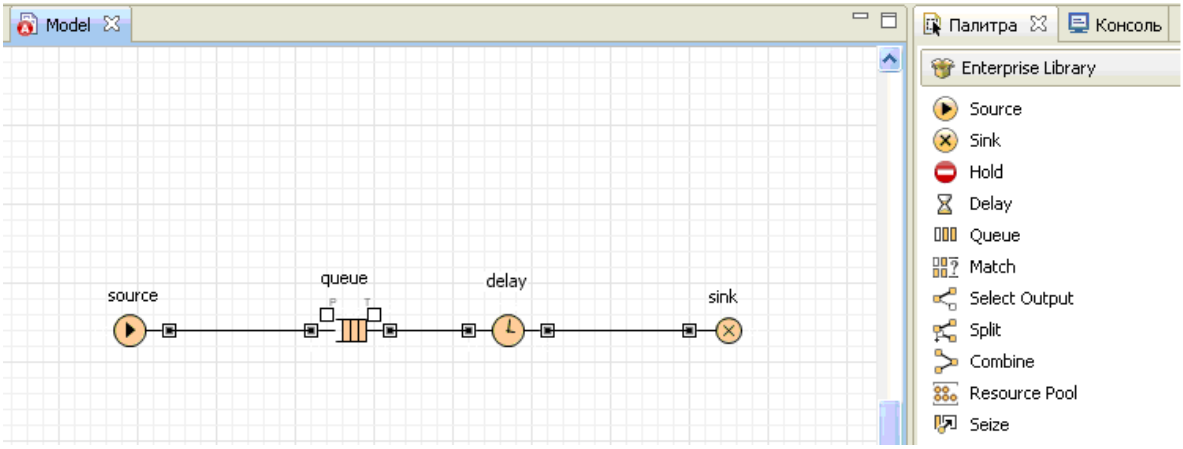


Рис. 6

Объект *queue* моделирует очередь клиентов, ожидающих обслуживания.

Объект *delay* моделирует задержку. В нашем примере он будет имитировать банкомат, тратящий определенное время на обслуживание клиента.

Объект *sink* обозначает конец блок-схемы.

3. Запуск модели.

Для каждой модели, созданной в Enterprise Library, автоматически создается блок-схема с наглядной визуализацией процесса, с помощью которой вы можете изучить текущее состояние модели, например, длину очереди, количество обслуженных человек и т.д.

Для запуска модели (рис. 7) щелкните мышью по кнопке **Запустить.** Откроется окно с презентацией запущенного эксперимента. AnyLogic автоматически помещает на презентацию каждого простого эксперимента заголовок и кнопку, позволяющую запустить модель и перейти на презентацию, нарисованную вами для главного класса активного объекта этого эксперимента (*Main*).

Щелкните по этой кнопке. AnyLogic переключится в режим работы модели. С помощью визуализированной блок-схемы вы можете проследить, сколько человек находится в очереди, сколько человек в данный момент обслуживается и т.д.

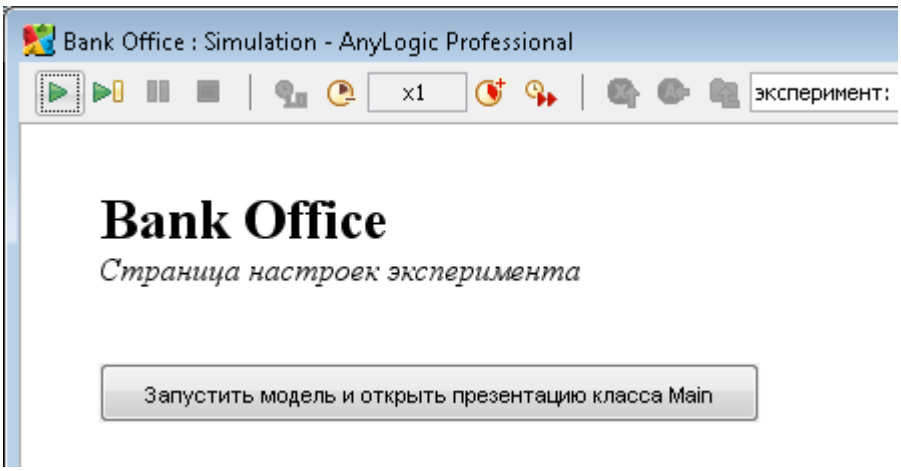


Рис. 7

На рис. 8 видно, что 4 человека стоят в очереди, а 23 человека покинули очередь (блок *queue*), из них 22 обслужили (блок *sink*), а один еще обслуживается у банкомата (блок *delay*).

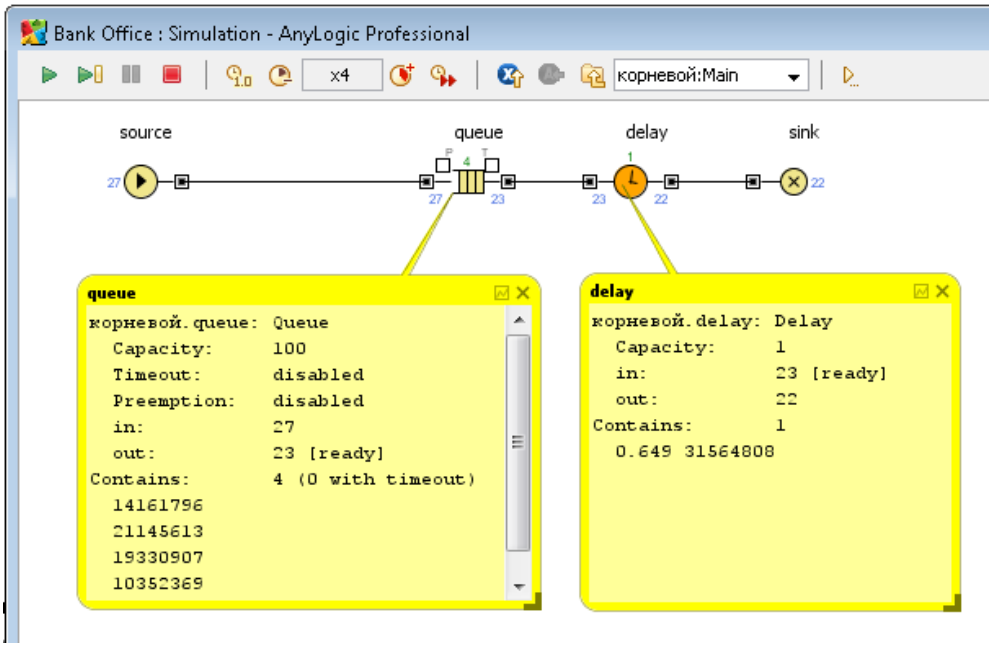


Рис. 8

С помощью кнопок панели инструментов **Замедлить** и **Ускорить** можно изменить скорость выполнения модели. Во время выполнения модели можно следить за состоянием любого блока диаграммы процесса с помощью окна *инспекта* этого объекта. Чтобы открыть окно инспекта, щелкните мышью по значку блока. В окне инспекта будет отображена базовая информация по выделенному блоку, например, для блока *Queue* будет отображена вместимость очереди, количество заявок, прошедшее через каждый порт объекта, и т.д.

4. Изменение данных модели.

Задайте данные модели, изменяя свойства созданных объектов (рис. 9).

В свойстве *interarrivalTime* объекта *source* укажите, как часто в отделение приходят клиенты – *exponential*(0.67).

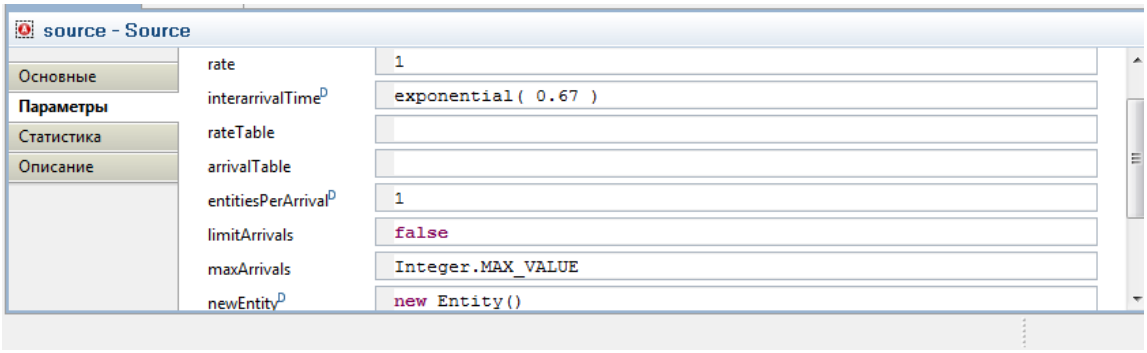


Рис. 9

Интервал между приходом клиентов распределен экспоненциально со средним значением, равным 1.5 единицы модельного времени. Заметьте, что аргумент функции *exponential()* равен 0.67, потому что в качестве аргумента задается интенсивность прихода клиентов.

Функция *exponential()* является стандартной функцией генератора случайных чисел AnyLogic. AnyLogic предоставляет функции и других случайных распределений, таких как нормальное, равномерное, треугольное и т.д. За детальным описанием функций и их параметров обращайтесь к руководству пользователя или справочнику классов (см. методы класса *Func*). Для вызова руководства пользователя, справочника классов AnyLogic выберите соответствующие пункты меню **Справка**.

В свойстве *capacity* объекта *queue* (рис. 10) задайте максимальную длину очереди – 15.

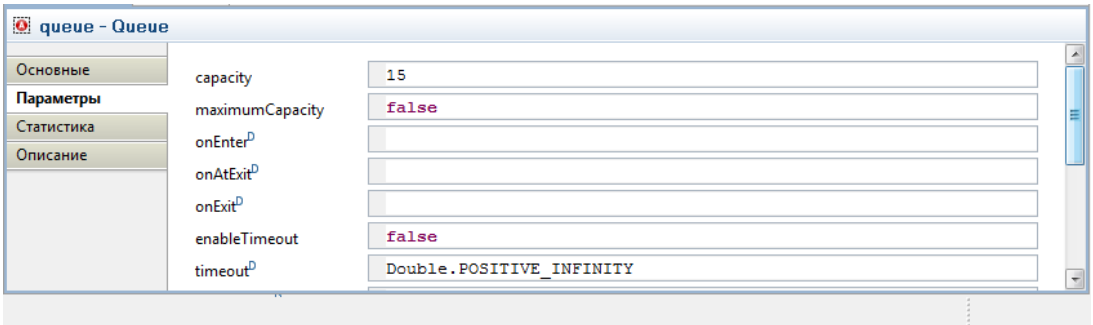


Рис. 10

В свойстве *delayTime* объекта *delay* (рис. 11) задайте время задержки (время обслуживания) – *triangular*(0.8, 1, 1.3).

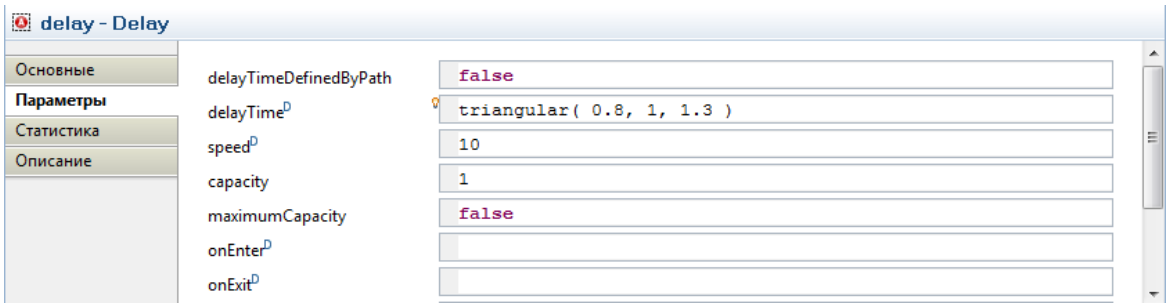


Рис. 11

Обслуживание одного клиента занимает примерно 1 минуту. Здесь время обслуживания распределено по треугольному закону со средним значением, равным 1 минуте, минимальным – 0.8 и максимальным – 1.3 минуты.

Запустите модель и проанализируйте ее работу.

5. Сбор статистики.

AnyLogic позволяет производить сбор сложной статистики. Для этого нужно лишь включить у объекта режим сбора статистики, поскольку по умолчанию он отключен для повышения скорости выполнения модели.

В системе собирается статистика по длине очереди для блока *queue* (*length*) и статистика по коэффициенту использования для блока *delay* (*utilization*). Чтобы включить сбор статистики для объекта, установите переключатель **Включить сбор статистики** на вкладке **Основные** свойств объекта.

Запустите модель и просмотрите в окне инспекта статистику для блоков *queue* и *delay*. Можно также просмотреть собранную статистику с помощью диаграмм и графиков или путем вывода числовых значений на анимацию.

6. Моделирование банковских кассиров.

Усложним модель, добавив в нее банковских кассиров. Можно моделировать число кассиров, как и банкомат, с помощью объектов *delay*. Но куда более удобным представляется моделирование числа кассиров с помощью ресурсов. Ресурс – это специальный объект Enterprise Library, который может потребоваться заявке для выполнения какой-то задачи. В нашем примере посетителям банковского отделения (заявкам) необходимо получить помощь у банков-

ских служащих (ресурсов).

Добавьте на диаграмму следующие объекты:

1) *selectOutput* – является блоком принятия решения. В зависимости от заданного вами условия, заявка, поступившая в этот объект, будет поступать на один из двух выходов объекта. Оставьте свойство ***selectCondition*** – *uniform*() < 0.5, тогда к кассирам и банкомату будет приходить примерно равное количество клиентов;

2) *Service* – моделирует занятие заявкой ресурса на определенное время. С помощью этого объекта мы промоделируем обслуживание клиента кассиром. Задайте следующие свойства объекта: назовите объект *tellerLines* (свойство **Имя**); укажите, что в очереди к кассирам может находиться до 20 человек (свойство ***queueCapacity***); задайте время обслуживания (свойство ***delayTime***). Будем полагать, что время обслуживания имеет треугольное распределение с минимальным средним значением 2.5, средним – 6 и максимальным – 11 минут;

3) *ResourcePool* – задает ресурсы определенного типа. Он должен быть подсоединен к объектам, моделирующим занятие и освобождение ресурсов (в нашем случае это объект *Service*). Задайте следующие свойства объекта: назовите объект *tellers*; задайте число кассиров (свойство ***capacity***) – 4.

Измените имя объекта *delay* на *ATM* (банкомат).

Соедините объекты соответствующим образом (рис. 12).

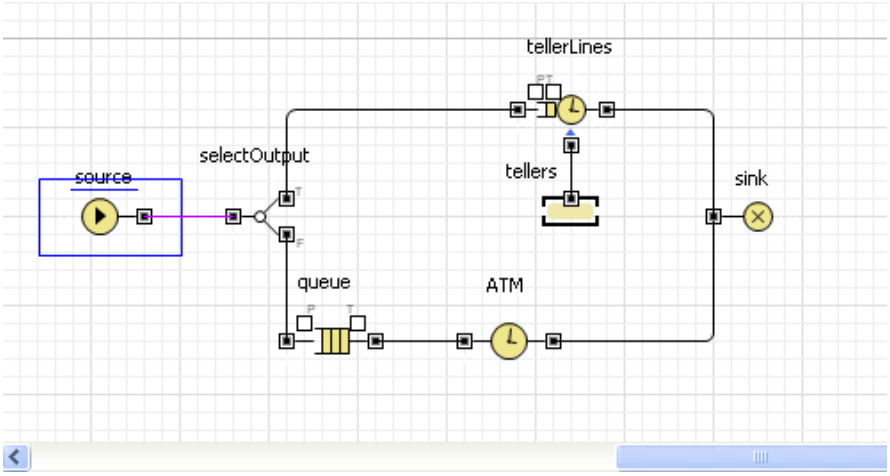


Рис. 12

Запустите модель и изучите ее поведение.

7. Сбор статистики о времени обслуживания клиента.

Необходимо определить, сколько времени клиент проводит в банковском отделении и сколько времени он теряет, ожидая своей очереди. Соберем эту статистику с помощью специальных объектов сбора данных и отобразим собранную статистику распределения времени обслуживания клиентов с помощью гистограмм.

Создадим класс сообщения *Customer*. Сообщения этого класса будут представлять клиентов банковского отделения. Выберите базовый класс *Entity* (сообщения), добавьте параметры для хранения информации о проведенном времени:

1) в панели **Проект**, щелкните правой кнопкой мыши по элементу модели и выберите **Создать | Java класс** из контекстного меню (рис. 13);

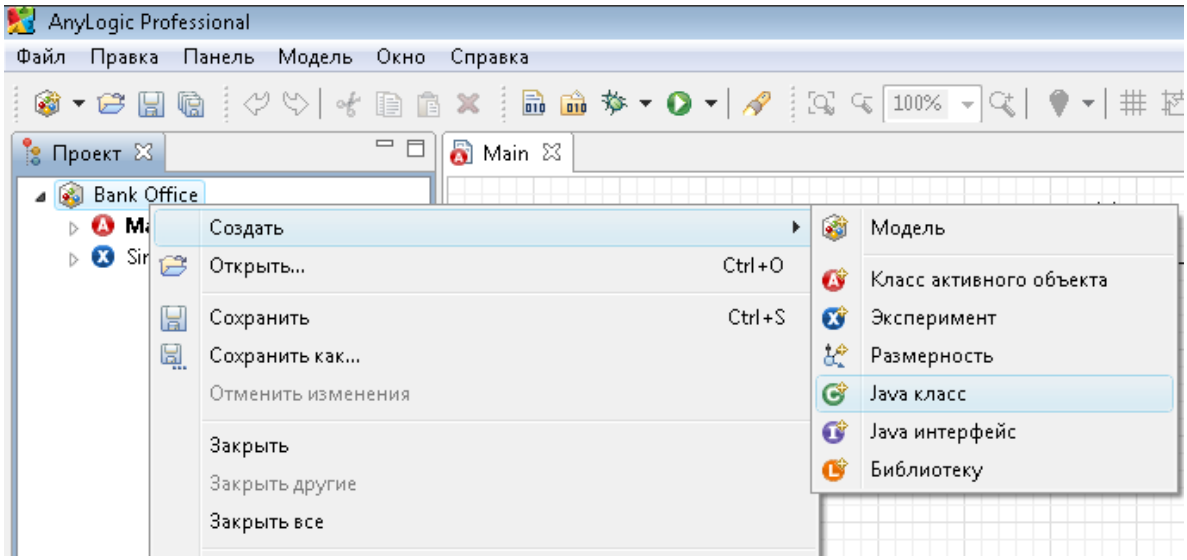


Рис. 13

2) появится диалоговое окно **Новый Java класс**. В поле **Имя** введите имя нового класса *Customer*;

3) сделайте так, чтобы этот класс наследовался от базового класса заявки *Entity* (рис. 14): выберите из выпадающего списка **Базовый класс** полное имя данного класса: *com.xj.anylogic.libraries.enterprise.Entity*;

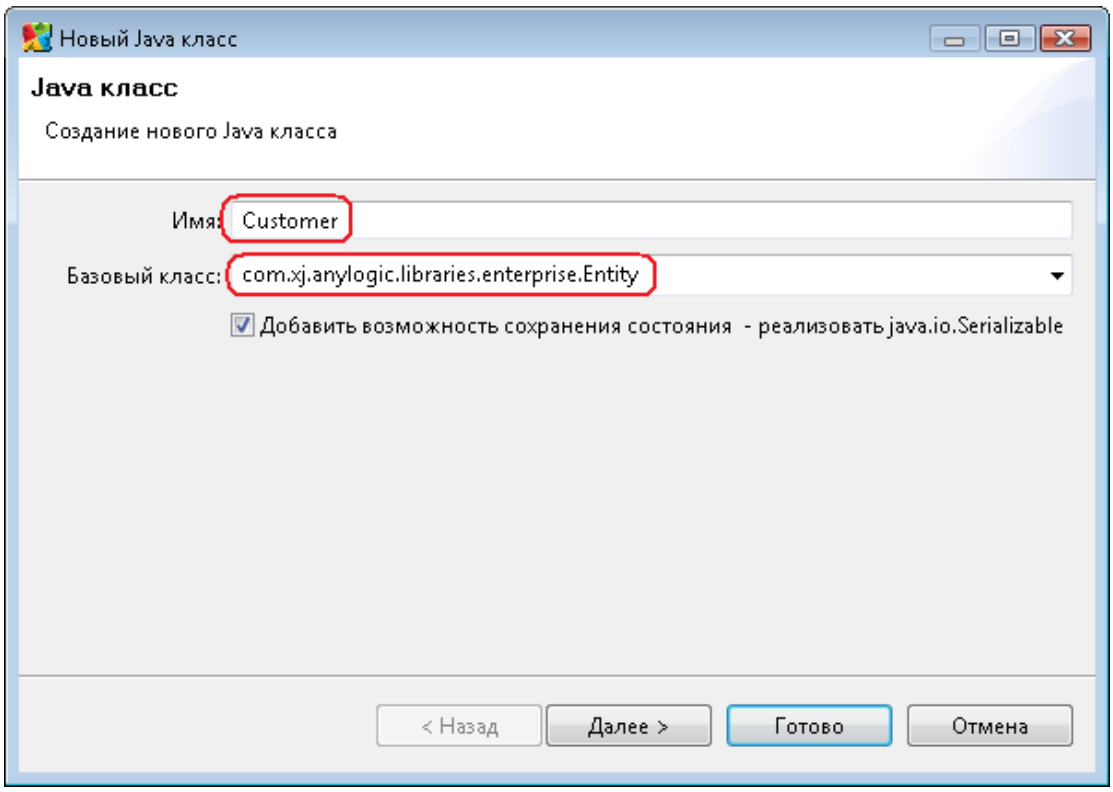


Рис. 14

4) щелкните мышью по кнопке **Далее**. На второй странице Мастера вы можете задать параметры создаваемого Java-класса. Создайте параметры:

- *enteredSystem* типа *double* для сохранения момента времени, когда клиент пришел в банковское отделение;
- *startWaiting* типа *double* для сохранения момента времени, когда клиент встал в очередь к банкомату;

5) щелкните мышью по кнопке **Готово**. Вы увидите редактор кода созданного класса. Можете закрыть его, щелкнув мышью по крестику в закладке с его названием.

Теперь вычислим время, которое тратится персоналом банка на обслуживание клиентов, и время, которое клиенты тратят на ожидание своей очереди.

Добавьте элементы сбора статистики по времени ожидания клиентов и времени пребывания клиентов в системе. Эти элементы будут запоминать соответствующие значения времени для каждого клиента и предоставят пользователю стандартную статистическую информацию: среднее, минимальное, максимальное из измеренных значений, среднеквадратичное

отклонение, доверительный интервал для среднего и т.п.:

1) чтобы добавить объект сбора данных гистограммы на диаграмму, перетащите элемент **Данные гистограммы** с палитры **Статистика** на диаграмму активного класса;

2) задайте свойства элемента (рис. 15).

- Измените **Имя** на *waitTimeDistr*.
- Измените **Заголовок** на *Waiting time distribution*.
- Сделайте **Кол-во интервалов** равным 50.
- Задайте **Начальный размер интервала**: 0.01;

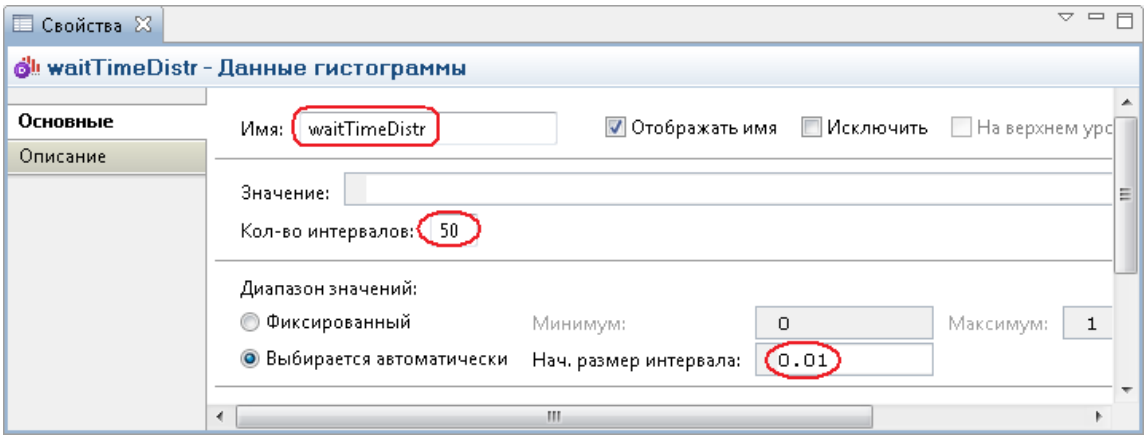


Рис. 15

3) создайте еще один элемент сбора данных гистограммы (**Ctrl** + перетащите только что созданный объект данных гистограммы, чтобы создать его копию). Измените **Имя** этого элемента на *timeInSystemDistr*, а **Заголовок** на *Time in system distribution*.

Измените свойства блоков вашей диаграммы процесса. Задайте следующие свойства объектов диаграммы:

1) блок *source*, свойство **Новая заявка** – введите `new Customer()`. Введите `Customer` в поле **Класс заявки**. Это позволит напрямую обращаться к полям класса заявки *Customer* в коде динамических параметров этого объекта. Введите `entity.enteredSystem = time();` в поле **Действие при выходе**. Этот код будет сохранять время создания заявки-клиента в переменной `enteredSystem` нашего класса заявки *Customer*.

Функция `time()` возвращает текущее значение модельного времени;

2) блок *tellerLines* (блок *Service*) – введите `Customer` в поле **Класс заявки**. Добавьте код в поля:

Действие при входе: `entity.startWaiting = time();`

Действие при выходе:

```
waitTimeDistr.add(time() - entity.startWaiting);
```

3) блок *queue* – введите `Customer` в поле **Класс заявки**. Добавьте код в поля:

Действие при входе: `entity.startWaiting = time();`

Действие при выходе:

```
waitTimeDistr.add(time()-entity.startWaiting);
```

Данный код добавляет время, в течение которого клиент ожидал обслуживания, в объект сбора данных *waitTimeDistr*;

4) блок *ATM* (блок *delay*) – введите `Customer` в поле **Класс заявки**;

5) блок *sink* – введите `Customer` в поле **Класс заявки**. Напишите следующий код, чтобы сохранить в наборах данных данные о клиенте, покидающем банковское отделение (**Действие при входе**):

```
timeInSystemDistr.add(time()-entity.enteredSystem);
```

Данный код добавляет полное время пребывания клиента в банковском отделении в объект сбора данных гистограммы *timeInSystemDistr*.

Добавьте две гистограммы для отображения распределений времен ожидания клиента и пребывания клиента в системе.

Чтобы добавить гистограмму на диаграмму класса активного объекта, перетащите элемент **Гистограмма** из палитры **Статистика** в то место графического редактора, куда вы хотите ее поместить. Укажите, какой элемент сбора данных хранит данные, которые хотите отображать на гистограмме: щелкните мышью по кнопке **Добавить данные** и введите в поле **Данные** имя соответствующего элемента – *waitTimeDistr*.

Аналогичным образом добавьте еще одну гистограмму и расположите ее под ранее добавленной. В поле **Данные** введите *timeInSystemDistr*.

Измените заголовки отображаемых данных.

Запустите модель. Включите режим виртуального времени и посмотрите, какой вид примет распределение времени ожидания и времени пребывания клиента в системе.

8. Оценка затрат операций.

Enterprise Library предоставляет инструменты для проведения оценки затрат операций. Метод оценки затрат операций (activity-based costing, ABC-метод) оценивает процесс и эффективность операций, определяет стоимость обслуживания/производства и указывает возможности для усовершенствования продуктивности и эффективности процесса. С помощью этого метода производится количественная оценка стоимости и производительности операций, эффективности использования ресурсов и стоимости объектов.

Проведем учет затрат операций в нашем примере, чтобы понять, во сколько в среднем обходится обслуживание одного клиента и какие накладные расходы связаны с обслуживанием клиентов, ожидающих своей очереди.

Сначала необходимо написать вспомогательную функцию для пересчета почасовой зарплаты в поминутную.

Создайте математическую функцию (перетащите элемент **Функция** с палитры **Основные** на диаграмму активного класса), назовите ее *toMinute*. На странице свойств задайте тип, аргументы и выражение функции. Тип возвращаемого значения должен быть *double*. Аргумент функции *perHour* типа *double*. Напишите выражение функции (тело функции) – `return perHour/60;`. Функция должна быть статической, поскольку эта функция не использует значения, специфичные для конкретного экземпляра класса *Model*. В противном случае нам пришлось бы вызывать ее со ссылкой на конкретный экземпляр класса *Model*.

Добавьте необходимые параметры в классе сообщений *Customer*:

- *serviceCost* будет хранить информацию о том, во сколько компании обходится обслуживание этого клиента. Тип – *double*. Значение по умолчанию – 0;

- *waitCost* – затраты на ожидание клиента в очереди. Тип – *double*. Значение по умолчанию – 0. Заметим, что банк несет издержки, связанные с обслуживанием клиентов, ожидающих своей очереди;
- *existenceCostPerHour* задает, во сколько обходится компании пребывание клиента в банке. Тип – *double*. Значение по умолчанию – `Main.toMinute(1.5)`.

Теперь необходимо добавить код в свойства блоков:

1) блок *tellerLines* (блок *Service*) – добавьте код в **Действие при выходе**:

```
waitTimeDistr.add(time() - entity.startWaiting);
Costwait.add(entity.existenceCostPerHour*(time() -
entity.startWaiting));
Costservice.add ((time()-entity.enteredSystem)*
(entity.existenceCostPerHour + Model.busyCostRate));
```

2) блок *queue* добавьте код в **Действие при выходе**:

```
waitTimeDistr.add(time() - entity.startWaiting);
Costwait.add(entity.existenceCostPerHour*(time()-
entity.startWaiting));
Costservice.add((time()-entity.enteredSystem)*
(entity.existenceCostPerHour + Model.busyCostRate));
```

Добавьте в модель вспомогательные элементы, собирающие статистику затрат компании, для чего необходимо создать:

1) переменные, задающие заработную плату кассиров. Для этого перетащите элемент **Простая переменная** с палитры **Основные** на диаграмму активного класса:

- *busyCostRate*, тип – *double*, значение по умолчанию – `toMinute(6.5)`, глобальный;
- *idleCostRate*, тип – *double*, значение по умолчанию – `toMinute(4.0)`, глобальный.

Мы платим кассиру 6.5 дол. в час, если он был занят обслуживанием клиентов и 4.0 дол., если он был свободен;

2) простую переменную, задающую расходы, которые связаны с работой банкомата:

ATMCostPerUse, тип – *double*, значение по умолчанию – 0.3, глобальный.

Одна операция банкомата обходится компании в 0.30 дол.;

3) переменную *timeUpdateCosts*. Создайте еще несколько переменных и назовите их так:

- *tellersIdleTime*;

- *tellersBusyTime*;

- *tellersIdleCost*;

- *tellersBusyCost*.

Эти переменные будут хранить информацию о том, сколько времени кассиры были заняты обслуживанием клиентов и сколько им требуется выплатить за работу;

5) два набора данных гистограммы (палитра **Статистика**) и назовите их так:

- *Costwait*;

- *Costservice*.

Данные наборы данных будут хранить статистику затрат компании на обслуживание клиентов;

6) функцию, которая будет обновлять статистику (палитра **Основные**). Назовите ее *UpdateCosts*.

```
double dt = time()-timeUpdateCosts;
tellersIdleTime += tellers.idle()*dt;
tellersBusyTime+=(tellers.capacity-tellers.idle())*dt;
tellersIdleCost += tellersIdleTime*idleCostRate;
tellersBusyCost += tellersBusyTime*busyCostRate;
timeUpdateCosts = time();
return 0;
```

Сосчитайте затраты на обслуживание клиента:

1) измените свойства объекта *tellers* (рис. 16). Добавьте следующий код:

onSeize - `UpdateCosts();`

onRelease - `UpdateCosts();`

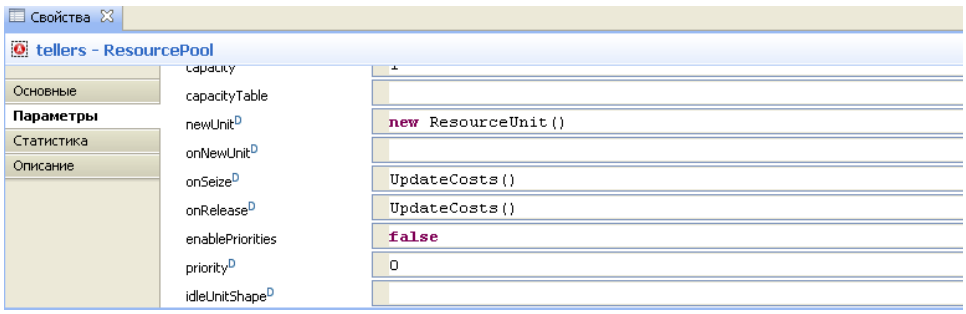


Рис. 16

2) измените свойства объекта *ATM*. Добавьте строки, чтобы учесть затраты на обслуживание клиента:

```
Costservice.add((time()-entity.enteredSystem)*
(entity.existenceCostPerHour + Model.ATMCostPerUse));
```

3) запустите модель. Посмотрите статистику затрат компании на обслуживание клиентов. Покажите, во сколько в среднем обходится обслуживание одного клиента и какие накладные расходы связаны с обслуживанием клиентов, ожидающих своей очереди.

Внесите изменения в модель банковского отделения согласно варианту.

Вариант	Распределение вероятности прихода клиентов в банк	Вероятность обращения к кассиру/ к банкомату	Время обслуживания клиента кассиром	Количество кассиров
1	Экспоненциальное	1/1	4 ± 2	1
2	Экспоненциальное	1/2	6 ± 2	2
3	Экспоненциальное	2/1	8 ± 2	3
4	Экспоненциальное	1/3	7 ± 2	4
5	Экспоненциальное	3/1	9 ± 2	5
6	Нормальное	1/1	8 ± 2	1
7	Нормальное	1/2	7 ± 2	2
8	Нормальное	2/1	9 ± 2	3
9	Нормальное	1/3	4 ± 2	4
10	Нормальное	3/1	6 ± 2	5
11	Треугольное	1/1	2 ± 2	1
12	Треугольное	1/2	7 ± 2	2
13	Треугольное	2/1	9 ± 2	3
14	Треугольное	1/3	4 ± 2	4
15	Треугольное	3/1	6 ± 2	5
16	Равномерное	1/1	4 ± 2	1
17	Равномерное	1/2	6 ± 2	2
18	Равномерное	2/1	7 ± 2	3
19	Равномерное	1/3	8 ± 2	4
20	Равномерное	3/1	9 ± 2	5

Проанализируйте поведение модели. Постройте графики и диаграммы

переменных, характеризующих поведение модели, и поместите их в отчет. Для создания графиков (диаграмм) используйте палитру **Статистика**.

Результаты работы

Студент должен предоставить отчет по лабораторной работе с выводами, продемонстрировать работу модели, ответить на вопросы преподавателя.

ПОСТРОЕНИЕ СИСТЕМНО-ДИНАМИЧЕСКОЙ МОДЕЛИ В ANYLOGIC (ЛАБОРАТОРНАЯ РАБОТА № 2)

Цель работы

Изучить интерфейс и возможности пакета AnyLogic для построения моделей системной динамики.

Порядок выполнения работы

В ходе лабораторной работы необходимо создать и изучить типичную системно-динамическую модель, представляющую интерес в экономике. Это модель распространения среди населения инноваций и новых продуктов, разработанная Франком Бассом (Frank Bass, 1969). Среди бизнес-аналитиков она ляется одной из самых популярных моделей исследования рынка новых продуктов.

В работе дано подробное описание этой модели, инструкции по созданию модели в пакете AnyLogic и предложены некоторые ее расширения. При возникновении трудностей при создании модели используйте учебное пособие по системной динамике пакета AnyLogic (находится в меню **Справка**).

Модель жизненного цикла продукта

Модель представляет собой динамику процесса превращения потенциальных покупателей нового продукта (*Potential_Adopters*) во владельцев продукта (*Adopters*). Изначально продукт никому не известен, и для того, чтобы люди начали его приобретать, он рекламируется. В итоге люди покупают продукт либо под воздействием рекламы, либо узнав о нем от знакомых, по «сарафанному радио». Эффективность рекламы пропорциональна числу людей, на которых она действует, т.е. числу потенциальных покупателей. В свою очередь, эффективность «сарафанного радио» зависит от числа людей, уже купивших продукт. Иными словами, в данной модели должна быть отражена структура взаимных зависимостей характеристик и параметров системы.

Для описания модели в терминах системной динамики необходимо опре-

делить ключевые переменные модели и их влияние друг на друга, а затем создать потоковую диаграмму модели. При создании потоковой диаграммы нужно учесть, какие переменные должны быть представлены накопителями, какие потоками, а какие – вспомогательными переменными.

Накопители (также называемые уровнями или фондами) представляют собой такие объекты реального мира, в которых сосредотачиваются некоторые ресурсы; их значения изменяются непрерывно.

Потоки – это активные компоненты системы, они изменяют значения накопителей. В свою очередь, накопители системы определяют значения потоков.

Вспомогательные переменные помогают преобразовывать одни числовые значения в другие; они могут произвольно изменять свои значения или быть константами.

При создании потоковой диаграммы выявляются переменные, которые накапливают значения с течением времени. В данной модели численности потребителей и потенциальных потребителей продукта являются накопителями, а процесс приобретения продукта – потоком.

Системно-динамическое представление данной модели показано на рис. 17. Накопители обозначаются прямоугольниками, поток – вентилем, а вспомогательные переменные – кружками. Стрелки обозначают причинно-следственные зависимости в модели.

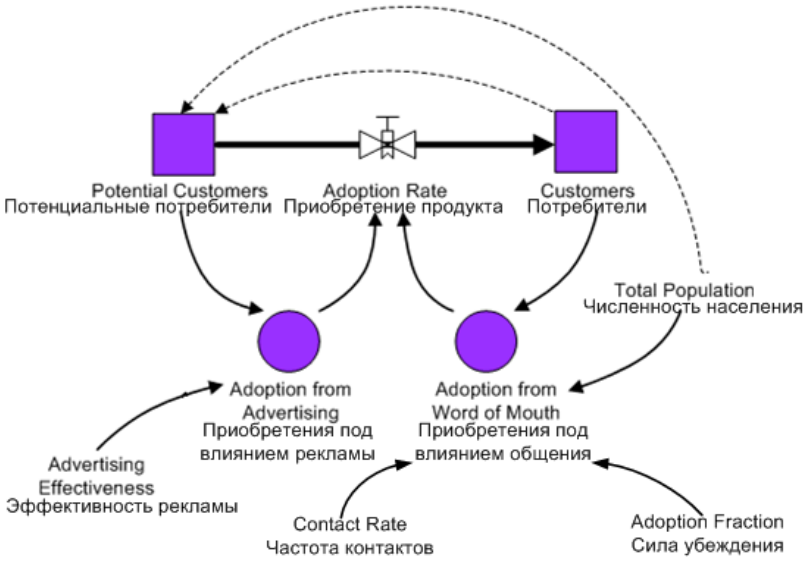


Рис. 17

Задание к лабораторной работе

В AnyLogic потоковая диаграмма создается с помощью структурной диаграммы. На диаграмме графически задаются накопители, потоки и вспомогательные переменные.

1. Создайте новый проект для будущей модели и сохраните его в своей папке. Откройте структурную диаграмму двойным щелчком мыши по элементу дерева *Main* в окне **Проект.**

2. Создайте два накопителя для того, чтобы смоделировать численности потребителей и потенциальных потребителей продукта. Для этого перетащите элемент **Накопитель** из палитры **Системная динамика** на диаграмму класса активного объекта. На диаграмме появится маленький голубой прямоугольник, обозначающий переменную-накопитель (что соответствует классической нотации системной динамики).

Измените имя накопителя – окно **Свойства,** вкладка **Основные**, введите *PotentialAdopters* в поле редактирования **Имя.**

Таким же образом создайте еще один накопитель, назовите его *Adopters* (рис. 18).

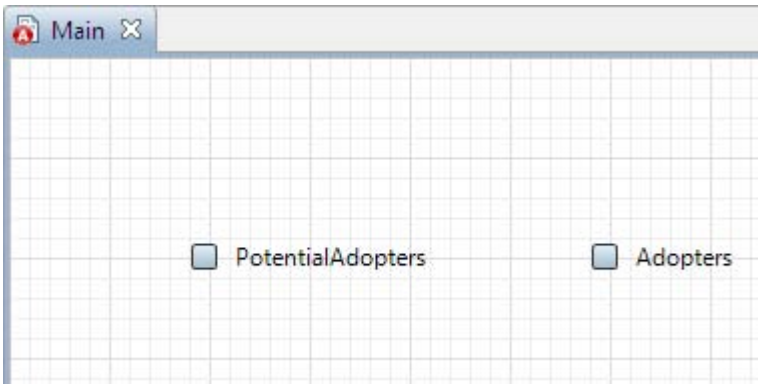


Рис. 18

3. Создайте поток приобретения продукта, увеличивающий число потребителей продукта и уменьшающий численность потенциальных потребителей.

В AnyLogic поток создается с помощью специального инструмента задания потоков. Инструмент задания потоков создает переменную-поток и автоматически конфигурирует накопители, в которые входит или из которых исходит данный поток, таким образом, что их значения изменяются в соответствии

со значением этого потока.

Чтобы создать поток, сделайте двойной щелчок мыши по накопителю *PotentialAdopters*, а потом щелкните по накопителю *Adopters*. AnyLogic создаст новую переменную-поток и сделает ее исходящим потоком для накопителя *PotentialAdopters* и входящим – для *Adopters*. На диаграмме появятся стрелки, которые будут обозначать образовавшиеся зависимости между потоком и этими накопителями (рис. 19). Выделите созданную переменную в графическом редакторе и измените имя этого потока на *AdoptionRate*.

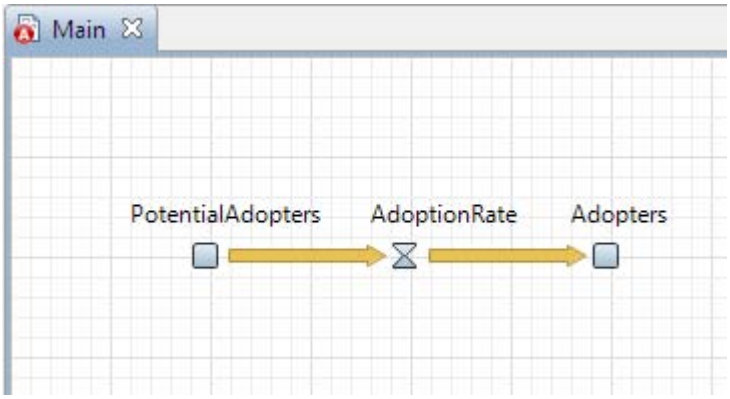


Рис. 19

4. Посмотрите свойства накопителей. Формулы накопителей должны выглядеть следующим образом (рис. 20).

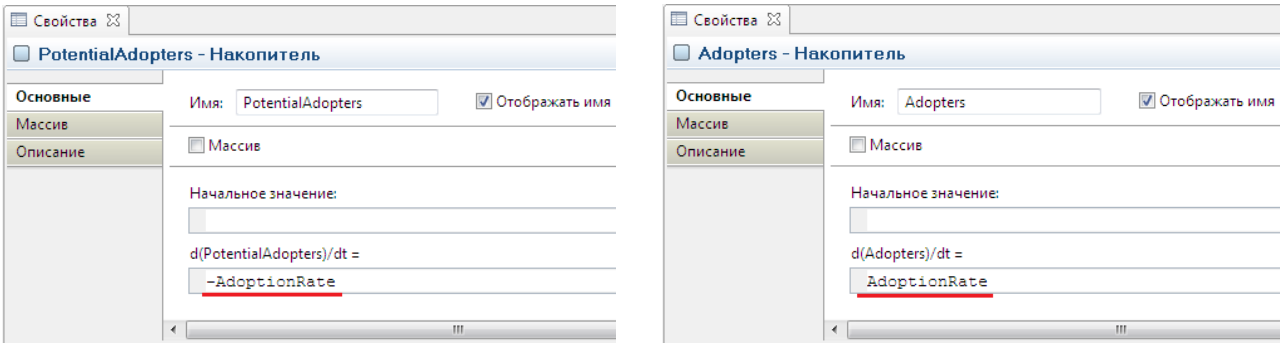


Рис. 20

Эти формулы были автоматически заданы инструментом задания потоков.

5. Создайте константы модели. Перетащите элемент **Параметр** из палитры **Системная динамика** на диаграмму класса активного объекта.

1) Создайте константу, задающую общую численность населения. Для

этого создайте новый параметр и измените его свойства. В поле **Имя** введите *TotalPopulation*. В поле **По умолчанию** введите 100000. Можно задать также краткое описание константы в поле **Описание**;

2) в нашей модели интенсивность рекламы и вероятность того, что продукт будет приобретен под ее влиянием, полагаются постоянными. Создайте константу, задающую эффективность рекламы. Назовите ее *AdEffectiveness*. Задайте значение по умолчанию 0.011;

3) частота, с которой потенциальные потребители общаются с потребителями, принимается как постоянная величина. Поэтому задайте частоту контактов константой. Назовите константу *ContactRate*. Предположим, что каждый потенциальный потребитель в среднем встречается со 100 постоянными потребителями в год. Задайте значение по умолчанию 100;

4) задайте константой силу убеждения владельцев продукта, определяющую ту долю контактов, которая приводит к продажам продукта. Назовите константу *AdoptionFraction*. Задайте значение 0.015.

6. Задайте начальные значения накопителей.

Начальное число потребителей нашего продукта равно нулю, поэтому в окне свойств накопителя *Adopters* введите 0 в поле редактирования **Начальное значение**.

Начальное количество потенциальных потребителей будет равно общей численности населения. В окне свойств накопителя *PotentialAdopters* введите *TotalPopulation* в поле редактирования **Начальное значение.** Вы можете сделать это с помощью Мастера (**Ctrl** + пробел).

7. Создайте две вспомогательные переменные, которые будут соответствовать двум составляющим потока приобретения продукта – приобретениям, совершенным под влиянием рекламы и под влиянием потребителей продукта соответственно.

1) Перетащите элемент **Вспомогательная переменная** из палитры **Системная динамика** на диаграмму класса активного объекта и назовите ее *AdoptionFromAd*. В поле **AdoptionFromAd =** введите

`AdEffectiveness * PotentialAdopters`.

Влияние рекламы моделируется следущим образом: некий постоянный процент потенциальных клиентов *AdEffectiveness* всё время переходит в разряд клиентов. Их доля в *AdoptionRate* равна *PotentialAdopters * AdEffectiveness*;

2) создайте еще одну переменную и назовите ее *AdoptionFromWOM*. Задайте формулу интенсивности продаж продукта под влиянием устного общения потребителей продукта с теми, кто данный продукт еще не приобрел:

`ContactRate * AdoptionFraction * PotentialAdopters * Adopters / TotalPopulation`.

Проанализируйте данную формулу.

8. Задайте формулу для потока приобретения продукта. Значение потока определяется суммой двух его независимых составляющих – продаж в результате рекламного влияния и продаж под влиянием общения с потребителями продукта. В окне свойств переменной *AdoptionRate* на вкладке **Основные** введите формулу, по которой будет вычисляться значение потока, в поле **AdoptionRate**= : `AdoptionFromAd + AdoptionFromWOM`

Создание модели завершено. Диаграмма накопителей и потоков должна выглядеть, как показано на рис. 21.

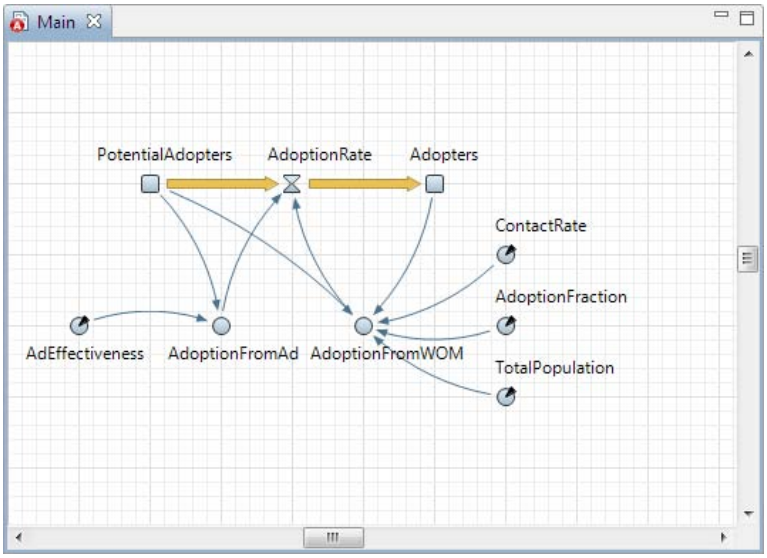


Рис. 21

9. Просмотрите причинно-следственные зависимости между накопителями, потоками и вспомогательными переменными в модели.

Зависимости будут показаны стрелками, как в общепринятой системно-динамической нотации. Стрелка, направленная от потока к накопителю, означает, что этот поток является входящим потоком для данного накопителя. Стрелка, направленная от накопителя к потоку, означает, что поток является исходящим. Тонкая стрелка, направленная от переменной *A* к переменной *B*, означает, что изменение значения переменной *A* вызовет изменение значения переменной *B*.

Можно увидеть, что модель содержит два цикла с обратной связью: компенсирующий и усиливающий. *Компенсирующий цикл* с обратной связью воздействует на поток приобретения продукта, вызванный рекламой. Поток приобретения продукта сокращает число потенциальных потребителей, что приводит к снижению интенсивности приобретения продукта. *Усиливающий цикл* с обратной связью воздействует на поток приобретения продукта, вызванный общением с потребителями продукта. Поток приобретения продукта увеличивает численность потребителей продукта, что приводит к росту интенсивности приобретения продукта под влиянием общения с потребителями продукта и, следовательно, к росту интенсивности приобретения продукта.

10. Просмотрите код модели. Для этого постройте модель (клавиша **F7**), а потом на панели **Проект** выберите **Модель,** нажмите правую кнопку мыши и из ниспадающего меню выберите **Открыть в…** и затем – в **Java редакторе.**

11. Сконфигурируйте выполнение модели, для этого необходимо настроить текущий эксперимент модели.

Если сейчас запустить модель, то она будет работать бесконечно. Поскольку мы хотим наблюдать поведение модели только тогда, когда происходит процесс распространения продукта, постольку нам нужно остановить модель, когда система придет в точку равновесия. Процесс распространения продукта в этой модели длится примерно 8 лет. Поэтому задайте останов модели после 8 единиц модельного времени.

Для этого в окне свойств эксперимента *Simulation:Main* перейдите на вкладку **Модельное время,** выберите **В заданное время** из выпадающего спи-

ска **Остановить**. В расположенном ниже поле введите 8. Модель остановится после того, как истекут 8 единиц модельного времени.

Задайте выполнение модели в режиме реального времени (вкладка **Презентация** окна свойств эксперимента). Задайте скорость выполнения – 2.

Вы можете сменить метод, используемый для решения системы дифференциальных уравнений. Если вы не укажете никакого конкретного метода, т. е. оставите выбранный по умолчанию метод Automatic, то во время работы модели AnyLogic будет автоматически выбирать численный метод в соответствии с поведением системы. На вкладке **Дополнительные** окна свойств эксперимента выберите метод RK4 из выпадающего списка **Дифф. уравнения**.

11. Проверьте ошибки и запустите модель. Для проверки ошибок постройте проект с помощью кнопки панели инструментов **Построить** (или клавиша **F7**). В окне **Ошибки** появится список всех ошибок, обнаруженных в проекте, если таковые имеются. Двойным щелчком мыши по ошибке в этом списке вы можете перейти к предполагаемому месту ошибки, чтобы исправить ее. После построения проекта запустите модель.

12. Просмотрите значения переменных в окне работающей модели.

13. Исследуйте динамику обеих составляющих потока продаж. Для этого откройте окно инспекта для переменной *AdoptionFromAd* в окне презентации. Вы можете переключить окно инспекта в режим графика – оно будет отображать временной график изменений значения переменной в модельном времени. Текущее значение переменной будет отображаться рядом с началом координат графика. Окно инспекта автоматически масштабируется таким образом, чтобы полностью вместить кривые графиков от начала до конца периода моделирования.

Откройте окно инспекта переменной *AdoptionFromWOM* и переключите его в режим графика (рис. 22).

Можно увидеть, что при внедрении нового продукта на рынок, когда число потребителей равно нулю, реклама будет являться единственным источником продаж. Наибольший рекламный эффект отмечается в начале процесса

распространения продукта; он неуклонно падает по мере уменьшения численности потенциальных потребителей.

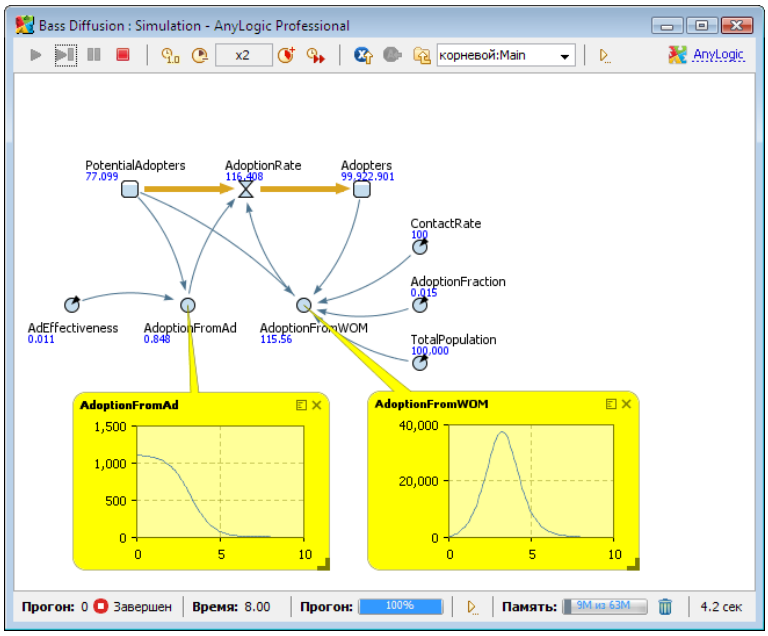


Рис. 22

14. Изучите динамику изменения численностей потребителей и потенциальных потребителей продукта с помощью диаграмм. Для этого создайте диаграмму для отображения переменных *Adopters* и *PotentialAdopters*.

Перетащите элемент **Временной график** из палитры **Статистика** на диаграмму класса *Main* и измените размер графика, как показано на рис. 23.

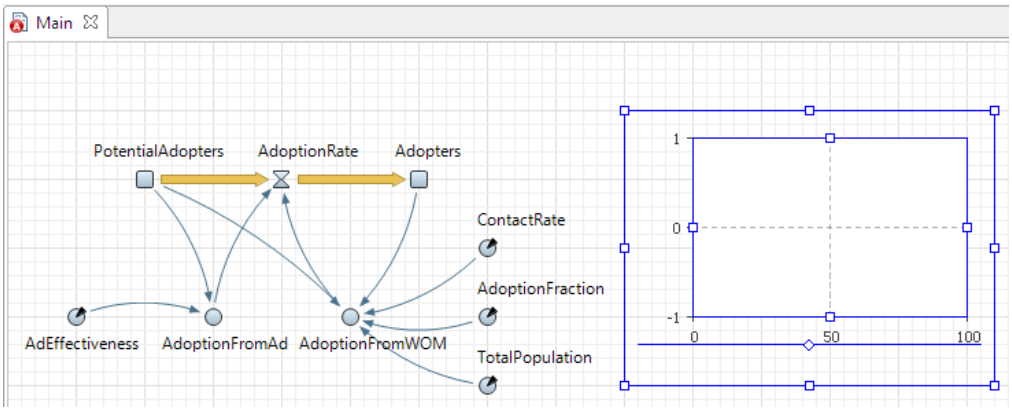


Рис. 23

Перейдите на вкладку **Основные** панели **Свойства**. В поле **Временной диапазон** задайте диапазон временной оси диаграммы – 8. Диаграмма будет отображать график только для заданного временного интервала.

Добавьте элементы данных, историю изменения значений которых вы хотите отображать на временном графике: щелкните мышью по кнопке **Добавить элемент данных**. Введите в поле **Выражение** имя соответствующего накопителя – *PotentialAdopters*. В поле **Заголовок** введите *Potential adopters*. Данная строка будет отображаться в легенде диаграммы для этого элемента данных. Выберите первую опцию из выпадающего списка **Стиль маркера**, чтобы наносимые на кривую графика точки не отображались дополнительными точками – маркерами.

Добавьте на график еще один элемент данных, который будет отображать значение накопителя *Adopters*.

15. Добавьте график, отображающий изменение интенсивности продаж. Для этого добавьте на диаграмму еще один временной график, поместите его под добавленным ранее графиком. Измените свойства графика. В качестве **Выражения** должно быть задано имя потока *AdoptionRate*.

16. Запустите модель. Первая диаграмма показывает, как изменяются переменные *PotentialAdopters* и *Adopters* во время «прогона» модели. Они представляют собой классические *S*-образные кривые (рис. 24).

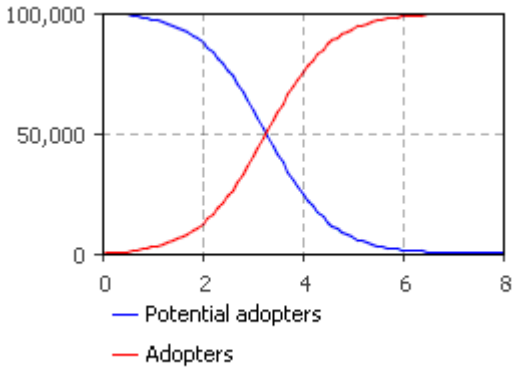


Рис. 24

На втором графике (рис. 25) вы увидите классическую колоколообразную кривую.

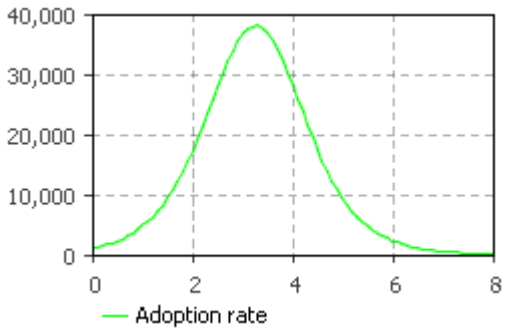


Рис. 25

17. Проанализируйте характеристики модели для своего варианта. Поместите графики и диаграммы в отчет. Сделайте выводы.

Вариант	Эффективность рекламы	Сила убеждения	Длительность процесса
1	0,010	0,01	6
2	0,010	0,02	7
3	0,010	0,03	8
4	0,005	0,04	9
5	0,005	0,05	10
6	0,005	0,01	11
7	0,015	0,02	10
8	0,015	0,03	9
9	0,015	0,04	8
10	0,010	0,05	7
11	0,010	0,01	6
12	0,010	0,02	7
13	0,005	0,03	8
14	0,005	0,04	9
15	0,005	0,05	10
16	0,015	0,01	11
17	0,015	0,02	10
18	0,015	0,03	9
19	0,020	0,04	8
20	0,020	0,05	7

Расширение модели жизненного цикла продукта

Расширенная модель поможет спланировать стратегию выпуска продукта на рынок, сориентироваться на конкретного потребителя и спрогнозировать спрос на продукт для того, чтобы выработать более рациональную и эффективную рекламную стратегию.

Моделирование повторных покупок

Созданная модель не учитывает того, что со временем продукт может

быть израсходован или прийти в негодность, что вызовет необходимость его повторного приобретения. Смоделируем повторные покупки, полагая, что потребители продукта снова становятся потенциальными потребителями, когда продукт, который они приобрели, становится непригоден.

Определите константу, задающую среднее время жизни продукта, – *ProductLifeTime* (рис. 26). Пусть средняя продолжительность использования нашего продукта равна двум годам.

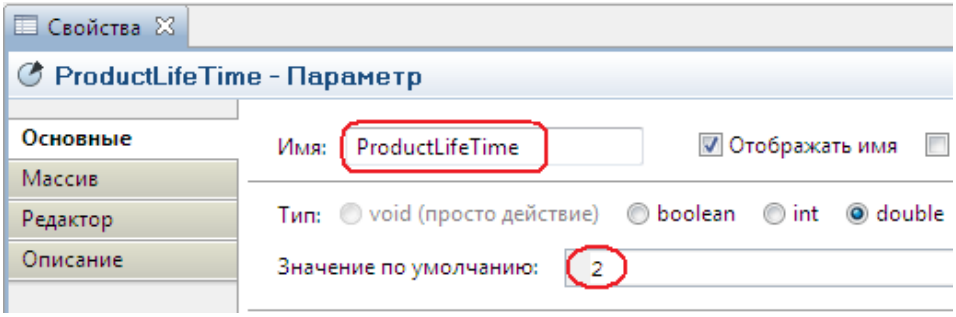


Рис. 26

Потребители продукта снова становятся потенциальными потребителями тогда, когда продукт, который они приобрели, расходуется и перестает использоваться. Поэтому поток прекращения использования продукта является ничем иным, как потоком приобретения, задержанным на среднее время пригодности продукта.

Создайте поток прекращения использования продукта, ведущий из *Adopters* в *PotentialAdopters* (сделайте двойной щелчок мыши по накопителю *Adopters*, а потом щелкните по накопителю *PotentialAdopters*). Назовите поток *DiscardRate* (нажмите **Ctrl+Enter** сразу после того, как вы закончите вводить новое имя). Формулы накопителей должны будут выглядеть следующим образом (рис. 27).

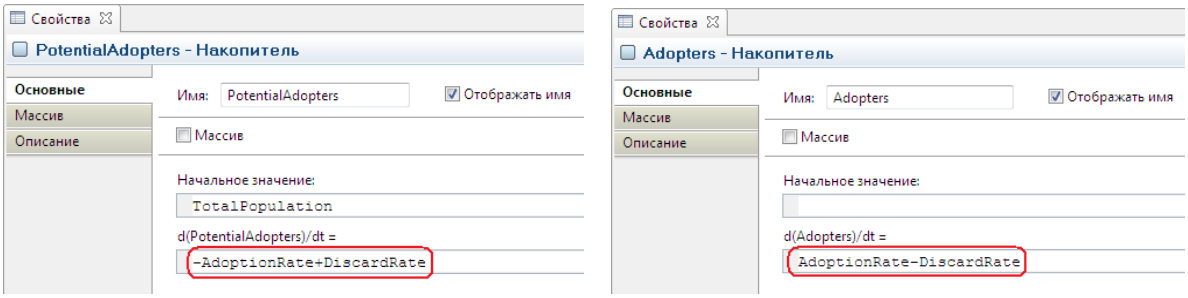


Рис. 27

Поместите поток на структурной диаграмме над потоком приобретения продукта (рис. 28).

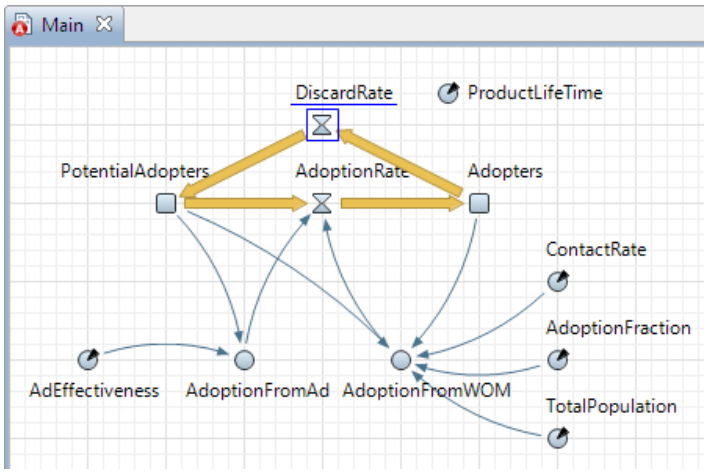


Рис. 28

Задайте следующую формулу для потока *DiscardRate*: `delay(AdoptionRate, ProductLifeTime)`

Функция `delay()` реализует временную задержку; она имеет следующую нотацию:

`delay(<задерживаемый поток>, <значение задержки>, <начальное значение>)`

В нашем случае функция представляет собой *AdoptionRate* с временной задержкой *ProductLifeTime*. Пока не истекло время использования первого приобретенного продукта, поток равен нулю.

Проверить работу функции задержки проще всего с помощью диаграммы. Для этого добавьте на график, отображающий динамику изменения интенсивности продаж, еще одну величину – интенсивность отказа от продукта, определяемую нашим потоком *DiscardRate*.

Запустите модель. Проверьте, как работает функция задержки (рис. 29). На диаграмме видно, что поток прекращения использования продукта является потоком приобретения продукта, задержанным на 2 года – время пригодности продукта.

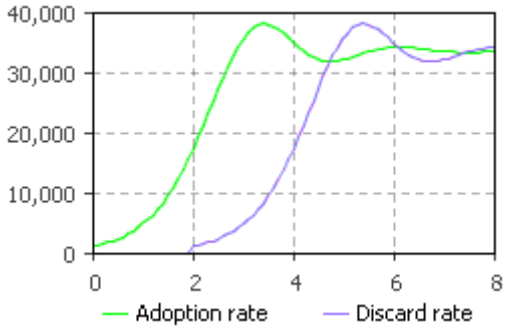


Рис. 29

С помощью другой диаграммы (рис. 30) проследите динамику изменения численностей потребителей.

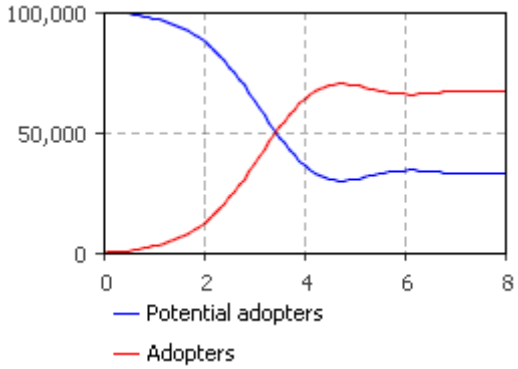


Рис. 30

Теперь численность потенциальных потребителей не уменьшается до нуля, а постоянно пополняется по мере того, как потребители заново покупают продукты взамен непригодных. Интенсивность приобретения продукта растет, падает и в итоге принимает какое-то значение, зависящее от средней жизни продукта и параметров, которые опрделяют интенсивность этого потока. Наличие в модели прекращения использования продукта означает, что какая-то доля населения всегда будет оставаться потенциальными потребителями.

Моделирование цикличности спроса

В текущей модели процент контактов потребителей продукта с потенциальными потребителями, который приводит к продажам продукта, полагается постоянным. На самом деле он изменяется, поскольку спрос на наш продукт зависит от текущего времени года. Продукт пользуется наибольшим спросом летом, зимой спрос на товар резко падает, за исключением небольшого предпраздничного периода в декабре. Необходимо смоделировать сезонную цик-

личность спроса.

Предположим, что у вас есть экспериментальные данные того, как изменяется средний спрос на продукт в течение года. Добавьте эти данные в модель с помощью табличной функции. Табличная функция – это функция, заданная в табличной форме, которая может быть сделана непрерывной с помощью интерполяции и экстраполяции.

1. Промоделируйте кривую спроса табличной функцией. Создайте новую табличную функцию. В появившемся окне свойств задайте имя табличной функции – *demand*. На вкладке **Общие** окна свойств функции введите **Данные** функции. В поле **Аргумент** введите значение нового аргумента функции, в поле **Функция** введите значение функции для этого аргумента. Задайте параметры функции, как на рис. 31.

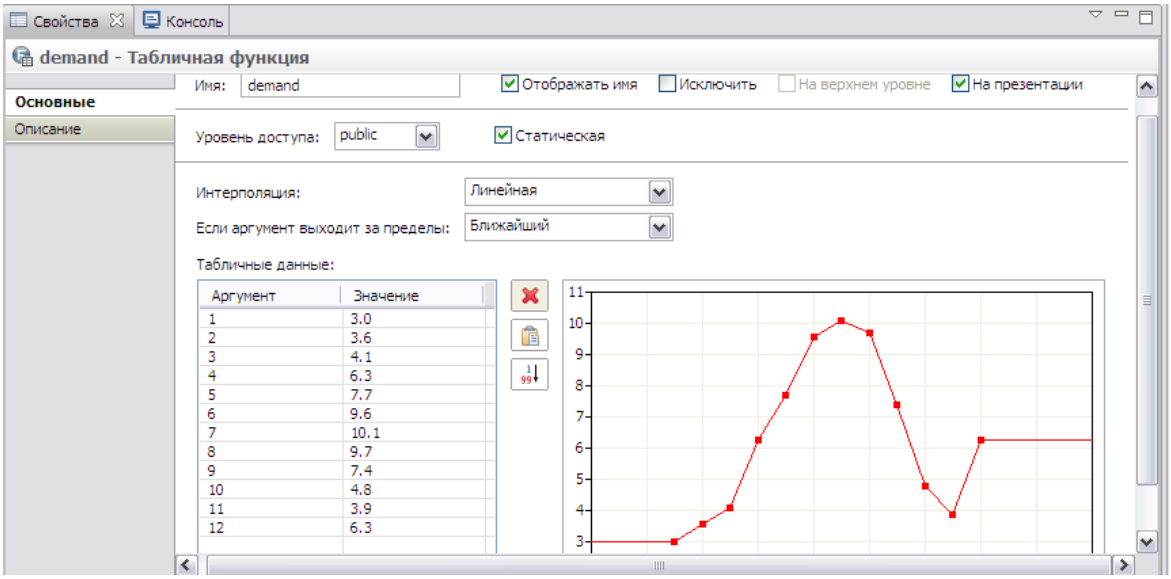


Рис. 31

Задайте тип интерполяции – **Линейная интерполяция** из группы элементов **Интерполяция**. Интерполяция будет производиться соединением табличных точек линиями.

Задайте тип реакции на аргументы, лежащие за пределами области допустимых значений функции. Выберите элемент **Использовать ближайший корректный** из группы элементов **Если аргумент выходит за пределы**. Щелкните мышью по кнопке **График…**. В появившемся диалоговом окне бу-

дет показан получившийся график кривой спроса.

2. Промоделируйте, как спрос влияет на интенсивность приобретения продукта. Для этого создадим специальную математическую функцию и заменим параметр *AdoptionFraction* вспомогательной переменной, значение которой будет считаться этой функцией.

Создайте новую функцию. В появившемся диалоговом окне задайте имя функции *adoptFraction*. Убедитесь, что функция возвращает значение типа *real*. Создайте аргумент, передающий функции текущее значение времени. В таблице **Аргументы** добавьте аргумент и назовите его *time*. Оставьте выбранный по умолчанию тип аргумента *double*. Задайте выражение функции. В поле редактирования **Выражение** введите

```
Return (demand((time-floor(time))*12+1)/200.0);.
```

Это выражение вычисляет номер текущего месяца и передает его табличной функции *demand*. Табличная функция возвращает значение спроса на продукт для данного месяца. В заключение для получения значения доли людей, покупающих продукт под влиянием общения, значение спроса делится на коэффициент преобразования. Функция *floor()* является предопределенной функцией AnyLogic.

Вводя выражения, пользуйтесь «Мастером Функций», в котором предопределенные функции присутствуют наряду с переменными, аргументами функций и табличными функциями модели. За детальным описанием функций и их параметров обращайтесь к руководству пользователя или справочнику классов (нужно смотреть методы класса *Func*).

3. Замените константу, задающую силу убеждения потребителей продукта, вспомогательной переменной, значение которой будет вычисляться по созданной функции.

Удалите параметр *AdoptionFraction*. Создайте вспомогательную переменную *AdoptionFraction*. Задайте формулу *adoptFraction(time())*. Теперь значение вспомогательной переменной будет вычисляться нашей функцией. Функция принимает один аргумент – *time()*, т.е. текущее модельное время.

4. Задайте останов модели в момент времени 25 и запустите модель. Теперь поведение модели колеблется около точки равновесия в силу того, что колеблются значения потока приобретения и потока прекращения использования продукта (рис. 32).

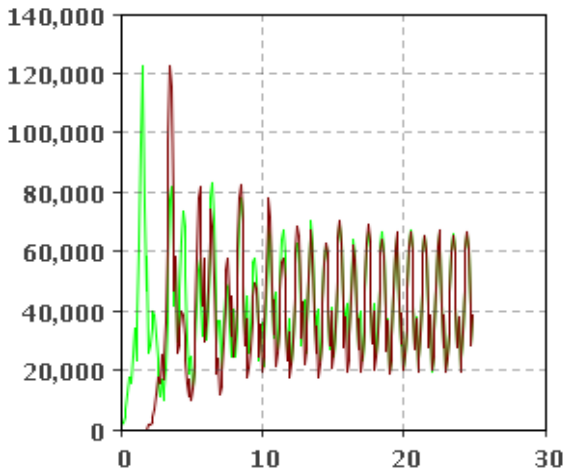


Рис. 32

Моделирование стратегии рекламной кампании

На данный момент эффективность рекламы в модели полагается постоянной. На самом деле она зависит от текущих расходов компании на рекламу. Улучшим нашу модель, чтобы иметь возможность управлять расходами на рекламную кампанию. Изменяя месячные расходы на рекламу, можно будет влиять на текущую эффективность рекламы.

1. Создайте константу, задающую месячные расходы компании. Назовите параметр *MonthlyExpenditures*. Установите значение по умолчанию – 1100.

2. Замените константу *AdEffectiveness* вспомогательной переменной. Удалите параметр *AdEffectiveness*. Создайте простую переменную *AdEffectiveness* с формулой `MonthlyExpenditures/10000.0`. Мы полагаем, что именно так эффективность рекламы зависит от текущих рекламных расходов компании.

Затем необходимо вести статистику всех расходов компании. Это может быть сделано созданием специальной переменной для хранения информации о том, сколько денег было потрачено на рекламу продукта. Каждый месяц таймер будет обновлять значение переменной, добавляя значение запланированных на предстоящий месяц расходов на рекламную кампанию продукта.

3. Добавьте статическую переменную (используйте палитру *Основные*) *TotalExpenditures*. Убедитесь, что у переменной нет уравнения. Задайте начальное значение – 0.0.

4. Создайте таймер для обновления значения переменной *TotalExpenditures*, щелкнув мышью по кнопке панели инструментов **Событие** и поместив его на структурную диаграмму. В окне **Свойства** назовите таймер *monthlyTimer*.

Сделайте так, чтобы событие срабатывало каждый месяц. Убедитесь, что событие **Циклическое**. Поскольку одна единица модельного времени в нашей модели соответствует одному году, то одному месяцу будет соответствовать выражение 1.0/12.0. Введите 1.0/12.0 в поле **Таймаут**. Установите **Время первого срабатывания** =0 (будет срабатывать при старте). В поле **Действие** введите `TotalExpenditures+=MonthlyExpenditures;` . Этот код будет выполняться каждый раз по истечении таймаута таймера. Он выполняет сбор статистики, а именно: добавляет значение запланированных рекламных расходов на предстоящий месяц к значению переменной *TotalExpenditures*.

Поскольку реклама играет значительную роль только в начальной стадии процесса завоевания рынка, постольку необходимо в какой-то момент времени, скажем, через 3 года, остановить рекламную кампанию. В результате приостановки кампании мы сэкономим деньги, бесцельно тратящиеся на рекламу тогда, когда насыщение рынка будет определяться практически исключительно покупками продукта, вызванными общением потребителей с потенциальными потребителями.

5. Добавьте константу, задающую время переключения, с именем *SwitchTime* и значением параметра по умолчанию 3.0.

6. Создайте стейтчарт для моделирования рекламной стратегии.

Для того чтобы создать новый стейтчарт, выберите кнопку панели инструментов **Диаграмма**. Нарисуйте следующий стейтчарт (рис. 33).

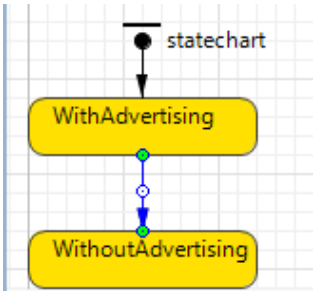


Рис. 33

Создайте первое состояние стейтчарта, щелкнув мышью по кнопке **Состояние**, переименуйте состояние в *WithAdvertising*. Добавьте еще одно состояние под только что созданным. Назовите его *WithoutAdvertising*. Когда стейтчарт перейдет в это состояние, мы должны будем остановить рекламную кампанию. Для реализации этого введите `MonthlyExpenditures=0.0;` в поле **Действие при входе**.

Добавьте переход из состояния *WithAdvertising* в состояние W*ithoutAdvertising* (рис. 34). Укажите, что переход произойдет по истечении времени *SwitchTime*: в свойствах перехода выберите **По таймауту** из выпадающего списка **Происходит** и введите *SwitchTime* в поле **Таймаут**.

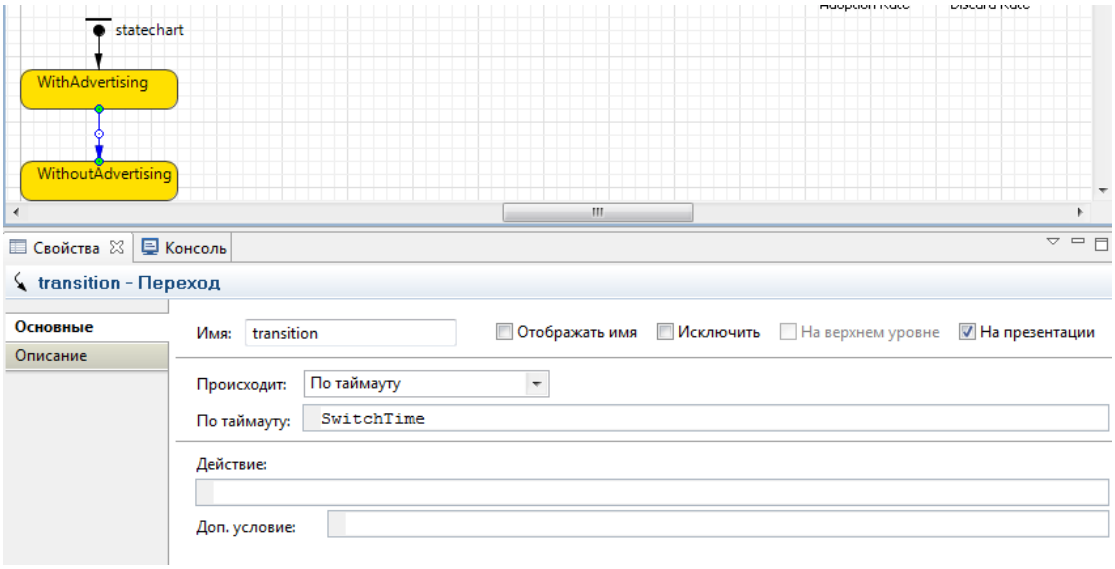


Рис. 34

Теперь, когда стейтчарт находится в начальном состоянии *WithAdvertising*, рекламные расходы кампании определяются переменной *MonthlyExpenditures*. Как только стейтчарт покидает данное состояние в момент времени *SwitchTime*, компания перестает рекламировать продукт.

7. Запустите модель и убедитесь, что рекламная кампания длится только

три года.

Оптимизация рекламной стратегии

Рыночная стратегия в данной модели предельно проста: в определенный момент времени компания прекращает рекламировать продукт. Мы же хотим найти оптимальную рыночную стратегию для достижения требуемого количества потребителей к определенному моменту времени при минимальных затратах на рекламу. Можно решить эту проблему, используя оптимизацию, при которой выбранные параметры модели будут систематически изменяться для минимизации или максимизации значения целевого функционала.

1. Определите константу, задающую необходимый порог насыщения рынка, например 80 процентов от общей численности населения. Назовите параметр *ExpectedSaturation* и задайте значение по умолчанию `TotalPopulation*0.8`.

2. Добавьте константу, задающую момент времени, когда должно быть достигнуто требуемое количество потребителей. Назовите параметр *SaturationTime* и задайте значение по умолчанию 1.5.

3. Измените стейтчарт, чтобы выполнить проверку насыщения рынка.

Откройте диаграмму стейтчарта. Измените стейтчарт так, чтобы он выглядел как на рис. 35.

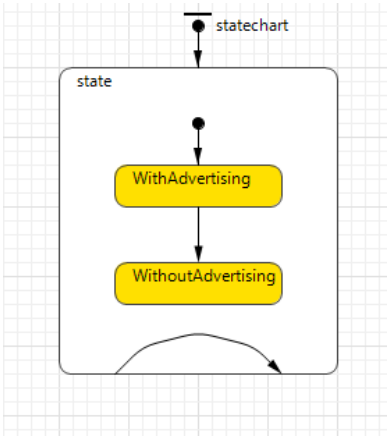


Рис. 35

Добавьте сложное состояние, включающее в себя два существующих состояния. Добавьте еще один указатель начального состояния, указывающий на сложное состояние. Добавьте внутренний переход в сложное состояние. Сде-

лайте так, чтобы переход выполнял проверку насыщения рынка продукта по истечении времени *SaturationTime*. Для этого выберите **По таймауту** из выпадающего списка **Происходит** и введите *SaturationTime* в поле **Таймаут**. В поле **Доп. условие** введите: `Adopters<=ExpectedSaturation`. Это условие проверяет, достигнуто ли необходимое число потребителей. Если выражение, заданное в поле **Доп. условие** истинно, то происходит переход и выполняется код, указанный в поле **Действие**. В поле **Действие** введите `TotalExpenditures=50000;`.

Мы увеличиваем значение переменной *TotalExpenditures* для того, чтобы показать, что наше требование не было выполнено.

4. Создайте оптимизационный эксперимент AnyLogic (рис. 36).

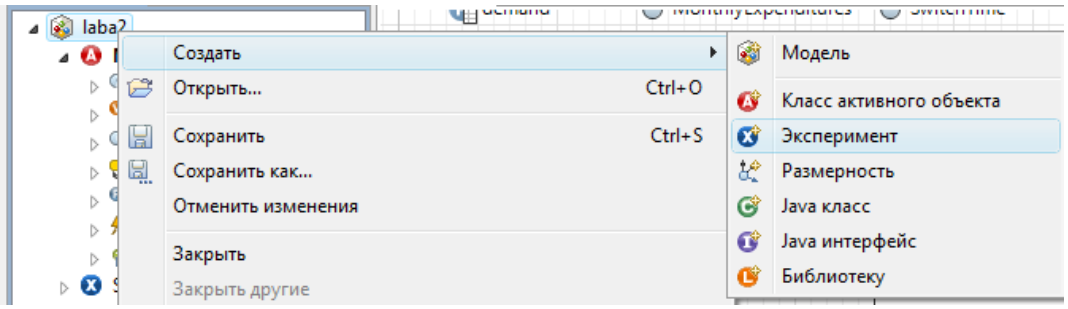


Рис. 36

В появившемся диалоговом окне задайте имя эксперимента и выберите **Оптимизационный эксперимент** как тип нового эксперимента.

5. Настройте созданный эксперимент (рис. 37-38).

На вкладке **Модельное время** окна свойств эксперимента задайте **Стоп по времени** 1.6.

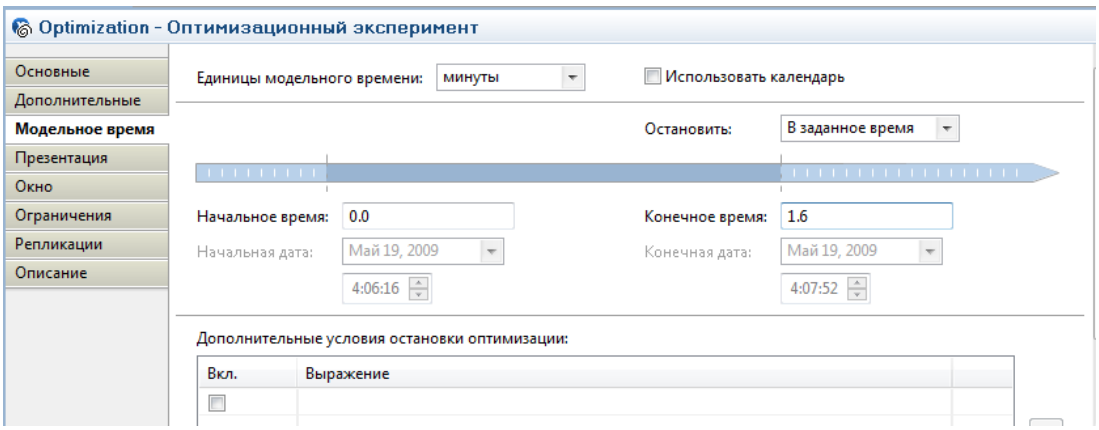


Рис. 37

На вкладке **Основные** задайте число «прогонов» модели 500.

Мы хотим минимизировать деньги, затраченные на рекламу продукта. На вкладке **Основные** выберите переменную *TotalExpenditures* в качестве **Целевого функции** и убедитесь, что установлен флажок **Минимизировать**.

Измените оптимизационные параметры в таблице **Параметры** на вкладке **Основные** окна свойств:

- тип параметра *SwitchTime* на непрерывный и установите максимальное 1.5 и минимальное 0.0 значения;
- тип параметра *MonthlyExpenditures* на непрерывный и установите максимальное 1500 и минимальное 1000 значения.

Во время оптимизации, значения параметров модели будут систематически изменяться, чтобы определить наименьшее значение переменной *TotalExpenditures*, выбранной в качестве целевого функционала.

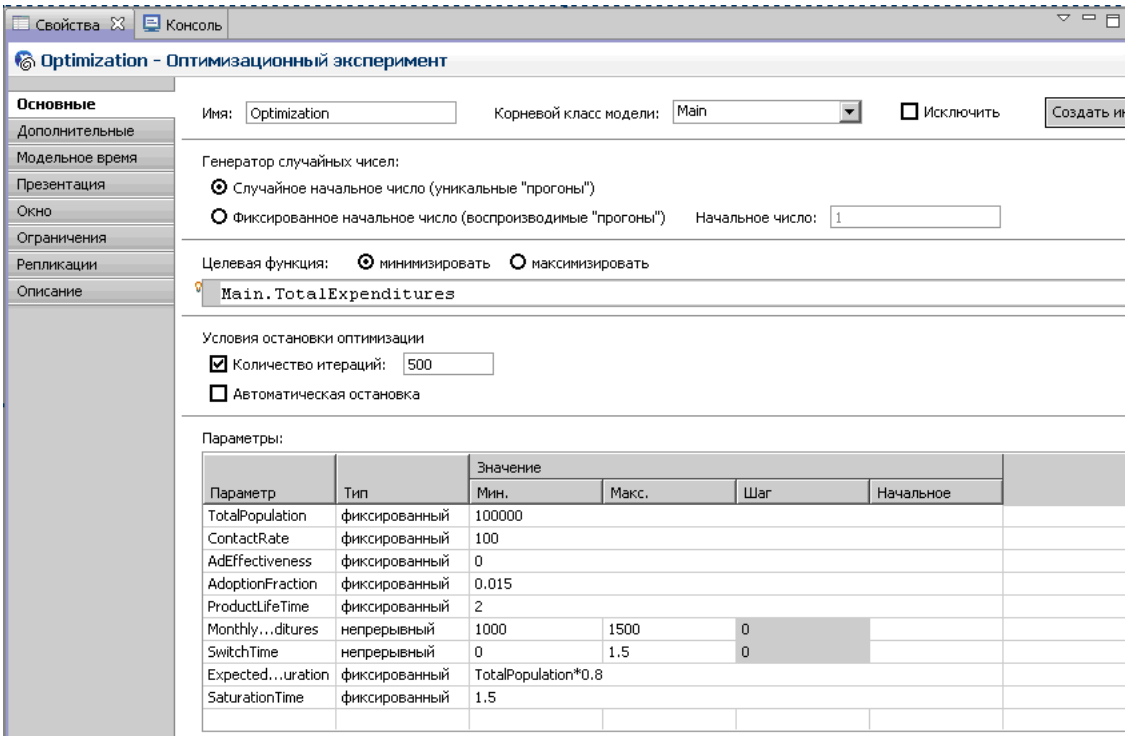


Рис. 38

Нажмите кнопку **Создать интерфейс –** будет создан стандартный интерфейс для данного эксперимента.

6. Запустите модель, выбрав оптимизационный эксперимент. AnyLogic запустит модель 500 раз с разными значениями параметров *MonthlyExpenditures*

и *SwitchTime*. Итоговая статистика оптимизации отображается в окне оптимизационного эксперимента.

На рис. 39 видно, что значение функционала примерно равно 11 250, наилучшие значения оптимизационных параметров следующие: *SwitchTime* = 0.75; *MonthlyExpenditures* = 1 250.

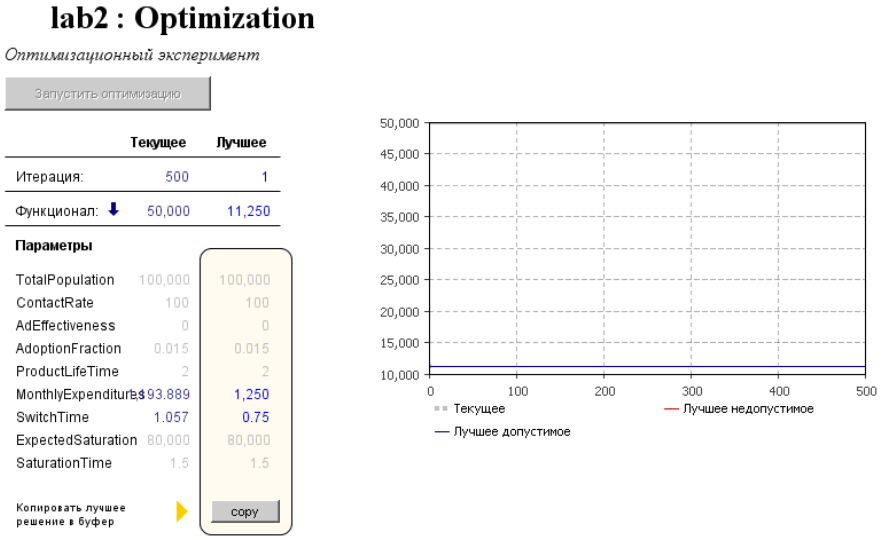


Рис. 39

Сделайте вывод о том, как спланировать стратегию завоевания рынка, чтобы рекламная кампания была наиболее рациональной и эффективной.

7. Примените результаты оптимизации.

В окне **Оптимизация** щелкните мышью по кнопке **copy**. В появившемся диалоговом окне выберите эксперимент, в который вы хотите скопировать результаты оптимизации. Оставьте выбранный по умолчанию эксперимент *Simulation* и щелкните мышью по кнопке **OK**.

Запустите модель с оптимизированными значениями параметров для того, чтобы убедиться в том, что к моменту времени *SaturationTime* будет достигнуто требуемое количество потребителей.

Результаты работы

Студент должен предоставить отчет по лабораторной работе с выводами, продемонстрировать работу модели, ответить на вопросы преподавателя.

РАЗРАБОТКА МНОГОАГЕНТНОЙ МОДЕЛИ В ANYLOGIC (ЛАБОРАТОРНАЯ РАБОТА № 3)

Цель работы

Изучить методологию агентного моделирования. Приобрести практические навыки работы с системой AnyLogic при построении агентных моделей.

Порядок выполнения работы

В данной лабораторной работе предлагается изучить агентный подход моделирования сложных систем. В ходе работы создается классическая модель распространения инноваций Басса и те ее расширения, которые демонстрируют возможности AnyLogic для создания агентных моделей.

Агенты в AnyLogic

Агент – это некоторая сущность, которая обладает активностью, автономным поведением, может принимать решения в соответствии с некоторым набором правил, может взаимодействовать с окружением и другими агентами, а также может изменяться (эволюционировать). Многоагентные (или просто агентные) модели используются для исследования децентрализованных систем, динамика функционирования которых определяется не глобальными правилами и законами, а, наоборот, эти глобальные правила и законы являются результатом индивидуальной деятельности членов группы. Цель агентных моделей – получить представление об общем поведении системы исходя из знаний о поведении ее отдельных активных объектов и взаимодействии этих объектов в системе. Агентная модель может содержать десятки и даже сотни тысяч активных агентов.

При помощи агентов моделируют рынки (агент – потенциальный покупатель), конкуренцию и цепочки поставок (агент – компания), население (агент – семья, житель города или избиратель) и мн. др.

В среде AnyLogic можно легко и быстро создавать модели с агентами. Агент естественно реализовывать с помощью базового элемента AnyLogic – ак-

тивного объекта. В модели можно создавать классы активных объектов и далее использовать в модели любое число экземпляров этих классов. Активный объект имеет параметры, которые можно изменять извне, переменные, которые можно считать памятью агента, а также поведение (рис. 40).

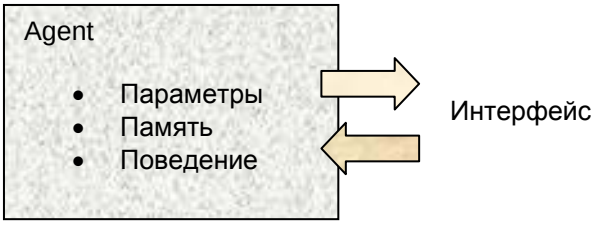


Рис. 40

Параметры могут указывать пол агента, дату рождения и т.д. Переменными можно, например, выразить возраст агента, его координаты в пространстве, социальные свойства.

Стейтчарты и таймеры могут выражать поведение: состояния агента и изменение состояний под воздействием событий и условий. Например, переходы в разные возрастные или социальные группы, изменения образования или дохода и т.д.

Кроме того, агент может иметь интерфейс для взаимодействия с окружением, который реализуется с помощью интерфейсных объектов: портов и интерфейсных переменных.

Задание к лабораторной работе

Создайте новый проект для будущей модели и сохраните его в своей папке.

Первым шагом при создании агентной модели является создание агентов. Для каждого агента задается набор правил, согласно которым он взаимодействует с другими агентами; это взаимодействие и определяет общее поведение системы. В нашей модели агентами будут люди. Создадим агентную модель с помощью **Мастера создания модели.**

Создание агентной модели

Шаг 1. Щелкните мышью по кнопке панели инструментов **Создать**. Поя-

вится диалоговое окно **Новая модель**. Задайте имя новой модели. Щелкните мышью по кнопке **Далее**.

Шаг 2. Выберите шаблон модели (рис. 41). В связи с тем, что мы хотим создать новую агентную модель, то нужно установить флажок **Использовать шаблон модели** и выбрать **Агентная модель** в расположенном ниже списке **Выберите метод моделирования**. Щелкните мышью по кнопке **Далее**.

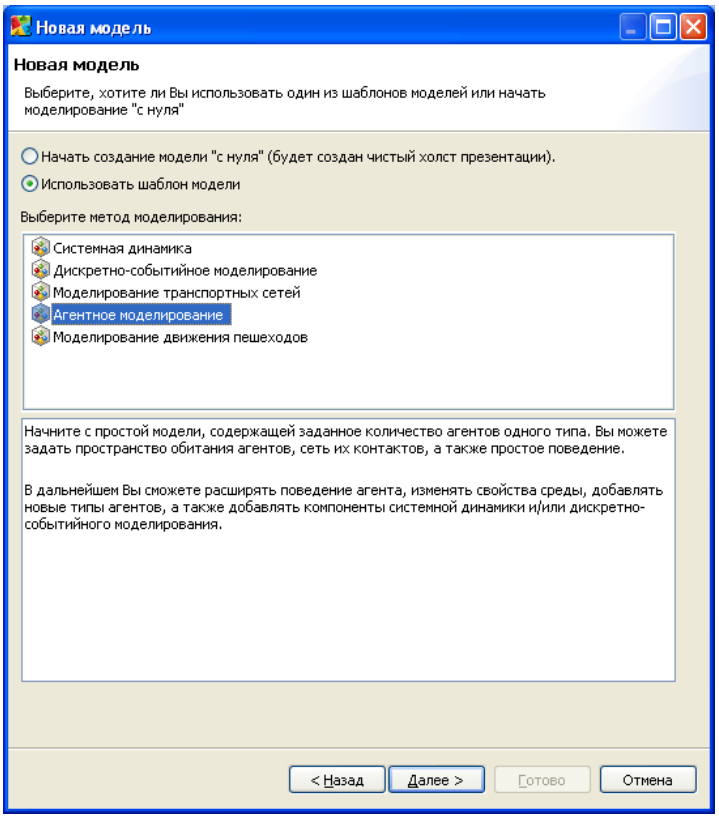


Рис. 41

Шаг 3. Создайте агентов, т.е. задайте имя класса агента и количество агентов, которое будет изначально создано в нашей модели. Задайте в качестве имени класса *Person* и введите в поле **Начальное количество агентов** 1000. Щелкните мышью по кнопке **Далее**.

Шаг 4. Задайте свойства пространства, в котором будут обитать агенты и выберите фигуру анимации агента. Установите флажок **Добавить пространство** и выберите ниже тип этого пространства – **Непрерывное**. Задайте размерности данного пространства: введите в поле **Ширина** – 600, а в поле **Высота** – 350. В результате наши агенты будут располагаться каким-то образом в пределах непрерывного пространства, отображаемого на презентации модели

областью размером 600×350 пикселей. Не меняйте значения, выбранные в выпадающих списках **Начальное расположение** и **Анимация**; пусть агенты изначально расставляются по пространству случайным образом, а анимируются с помощью фигурки человечка. Щелкните мышью по кнопке **Далее**.

Шаг 5. Задайте сеть взаимосвязей агентов (рис. 42). Установите флажок **Использовать сеть** и оставьте выбранной опцию **Случайное**. Ниже можно установите флажок **Показывать связи**, чтобы отображать на презентации связи между знакомыми (или потенциально могущими встретиться и пообщаться) агентами с помощью линий. Щелкните мышью по кнопке **Далее**.

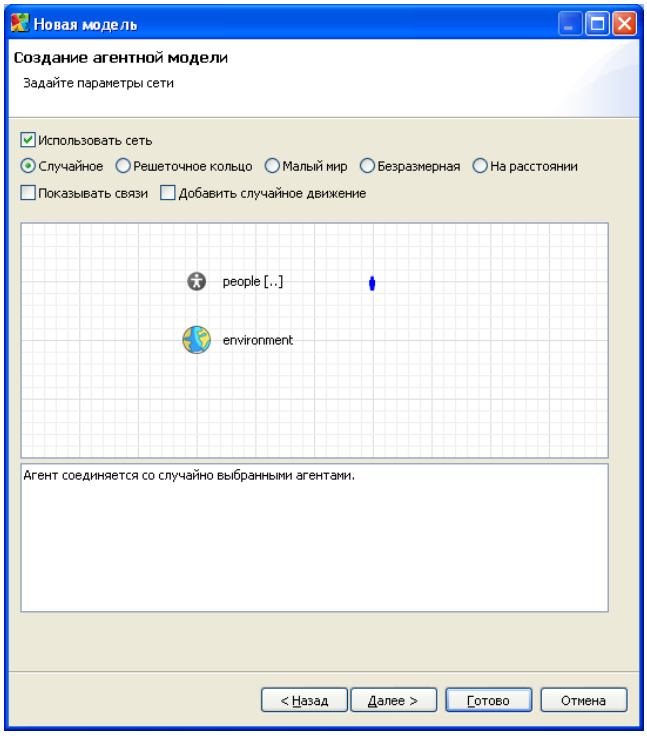


Рис. 42

Шаг 6. Установите флажок **Добавить простое поведение**. В результате у агента будет создана диаграмма состояний.

Задание характеристик агента

Характеристики агента задаются с помощью параметров класса. Все агенты обладают общей структурой, поскольку все они задаются объектами одного класса. Параметры же позволяют задавать характеристики индивидуально для каждого агента. Создадим параметр, который задает подверженность человека влиянию рекламы.

Откройте структурную диаграмму класса *Person*. Перетащите элемент **Параметр** из палитры **Основная** на диаграмму класса, в окне свойств параметра задайте имя *AdEffectiveness*, значение по умолчанию – 0.011.

Задание поведения агента

Поведение агента обычно описывается в классе этого агента (в нашей модели это класс *Person*) с помощью *диаграммы состояний* (*стейтчарт*).

Мастер создания моделей уже создал простейшую диаграмму состояний из двух состояний, между которыми существует два разнонаправленных перехода. Изменим данный стейтчарт.

1. Откройте структурную диаграмму класса *Person*. На диаграмме класса вы увидите следующую диаграмму состояний (рис. 43).

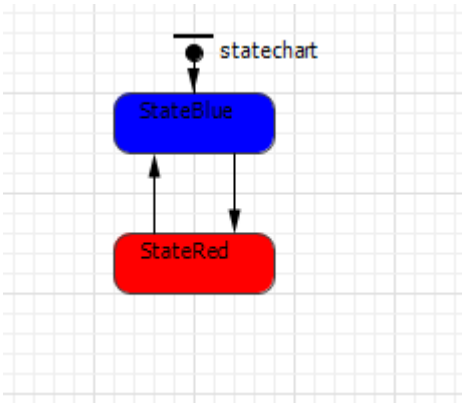


Рис. 43

2. Откройте свойства верхнего состояния, переименуйте верхнее состояние в *PotentialAdopter*. Это начальное состояние. Нахождение стейтчарта в данном состоянии означает, что человек еще не купил продукт.

3. Нижнее состояние назовите *Adopter* (т.е. человек уже купил продукт).

4. Измените свойства перехода из состояния *PotentialAdopter* в состояние *Adopter*. Этот переход будет моделировать покупку продукта.

В окне свойств перехода выберите **С заданной интенсивностью** из выпадающего списка **Происходит** и введите *AdEffectiveness* в расположенном ниже поле **Интенсивность**. Время, через которое человек купит продукт, экспоненциально зависит от эффективности рекламы продукта.

5. Удалите переход, ведущий из нижнего состояния в верхнее, поскольку мы пока создаем простейшую модель, в которой человек, однажды приобрев-

ший продукт, навсегда остается его потребителем, и соответственно перехода из состояния *Adopter* в состояние *PotentialAdopter* пока что быть не должно (рис. 44). Чтобы удалить переход, выделите его на диаграмме и нажмите ***Del***.

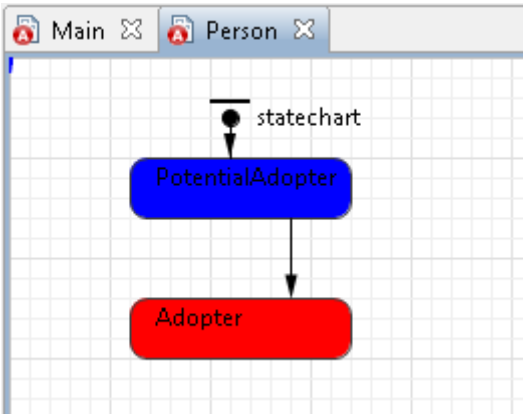


Рис. 44

6. Настройте выполнение модели (рис. 45). В окне свойств эксперимента перейдите на вкладку **Модельное время** и задайте останов модели после 8 единиц модельного времени.

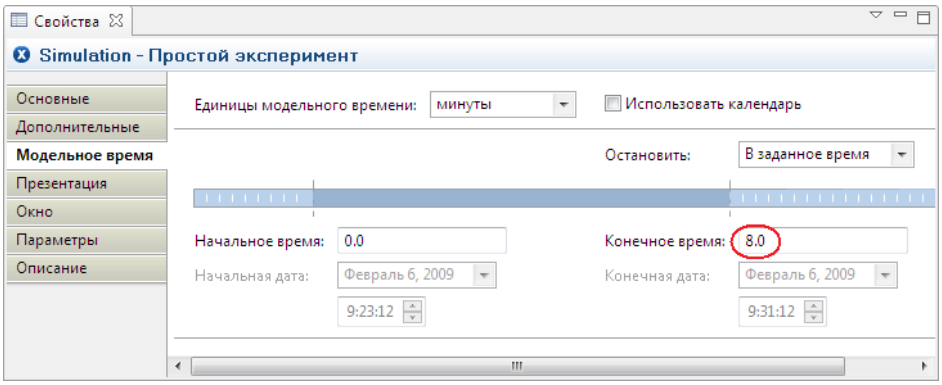


Рис. 45

7. Постройте проект с помощью кнопки панели инструментов **Построить** (клавиша **F7**). Если ошибок в проекте нет, то запустите модель. Вы увидите, как число потенциальных покупателей (синих) переходит в разряд покупателей (красных).

Подсчет потребителей продукта

Главная задача модели распространения продукта – изучение того, как быстро люди покупают новый продукт. Для этого будем подсчитывать число потребителей и потенциальных потребителей продукта, что можно сделать с помощью функций сбора статистики.

Создадим функции сбора статистики для подсчета потенциальных потребителей продукта.

1. Откройте диаграмму класса *Main*. Выделите на диаграмме вложенный объект *people*.

2. Перейдите на вкладку **Статистика** панели свойств объекта *people*. Щелкните мышью по кнопке **Добавить функцию сбора статистики**. Откроется секция свойств для задания свойств новой функции сбора статистики по элементам этого реплицированного объекта (*people*).

3. Задайте имя функции – *potentialAdopters*. Оставьте выбранный по умолчанию **Тип функции** – кол-во.

Задайте **Условие**:

```
item.statechart.isStateActive(item.PotentialAdopter)
```

Эта функция будет вести подсчет количества агентов, для которых выполняется заданное условие, т.е. тех агентов, которые находятся в текущий момент времени в состоянии *PotentialAdopter* (являются потенциальными потребителями продукта). Здесь `item` – это агент (элемент реплицированного объекта *people*).

4. Создайте еще одну функцию сбора статистики (рис. 46). Назовите ее *adopters*. **Тип функции** – кол-во. Условие:

```
item.statechart.isStateActive(item.Adopter)
```

Данная функция будет вести подсчет количества агентов, которые находятся в состоянии *Adopter* (т.е. уже приобрели продукт).

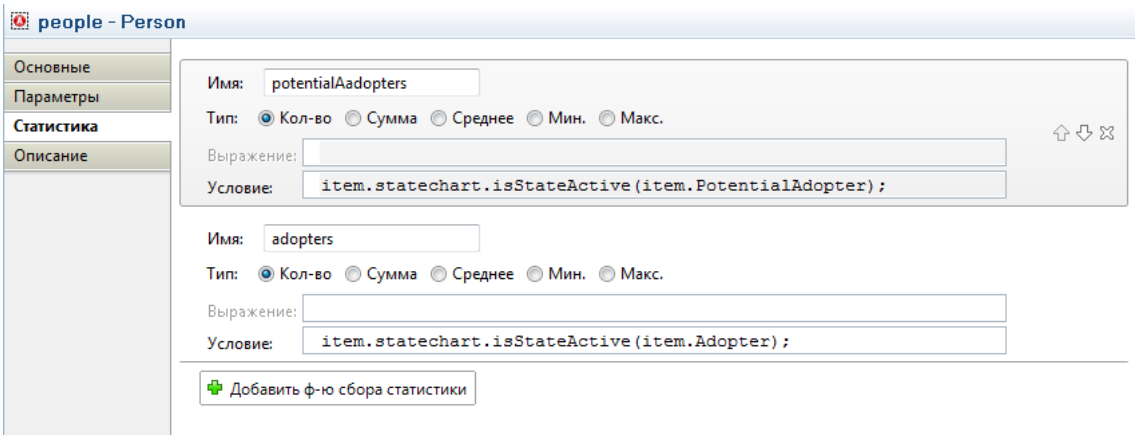


Рис. 46

5. Добавьте временной график, отображающий динамику изменения численностей потребителей и потенциальных потребителей продукта. Расположите его, как показано на рис. 47.

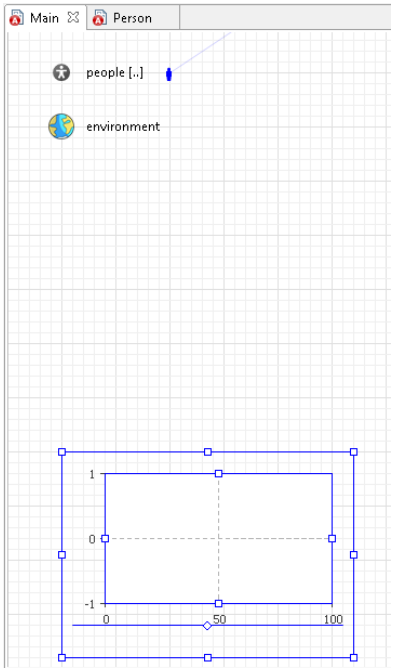


Рис. 47

Настройте свойства графика (рис. 48).

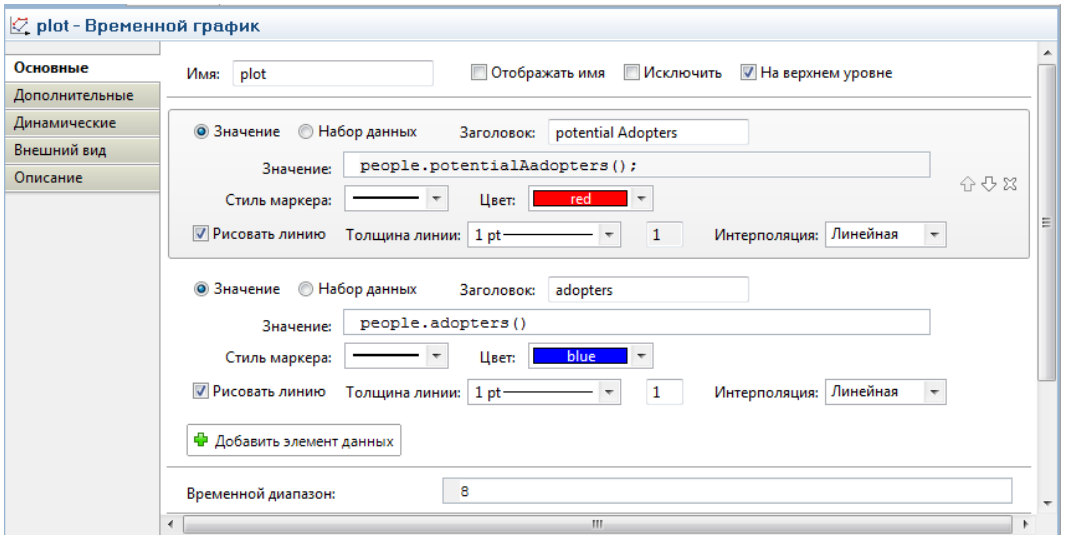


Рис. 48

6. Запустите модель. На графике (рис. 49) просмотрите динамику моделируемого процесса.

Вы увидите, что под влиянием рекламы каждую единицу времени постоянная доля от общей численности потенциальных потребителей продукта приобретает изучаемый нами продукт.

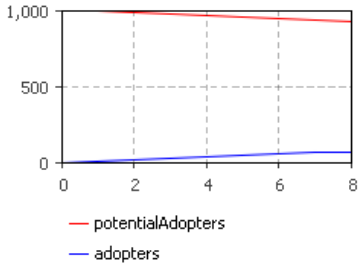


Рис. 49

Учет влияния общения людей

В текущей модели люди приобретают продукт только под влиянием рекламы. На самом деле рекламный эффект играет значительную роль только в момент выпуска продукта на рынок. В дальнейшем все большую роль будет играть общение людей со своими знакомыми, которые этот продукт уже приобрели. В основном люди приобретают новые продукты именно под влиянием убеждения своих знакомых; данный процесс чем-то схож с распространением эпидемии.

Чтобы учесть влияние общения людей, внесем в нашу модель небольшие изменения.

Откройте диаграмму класса *Person* и создайте два новых параметра:

1) параметр *ContactRate* – среднегодовое количество встреч человека. Значение по умолчанию – 100. Тип – *int*. Предположим, что человек в среднем встречается со 100 людьми в год;

2) параметр *AdoptionFraction* – сила убеждения человека, влияющая на то, сколько людей он сможет убедить в необходимости купить продукт. Значение по умолчанию – 0.015. Тип – *double*.

Измените стейтчарт агента:

1) откройте диаграмму стейтчарта *adoption*;

2) добавьте в состояние *Adopter* внутренний переход (рис. 50). Для этого щелкните мышью по кнопке панели инструментов **Переход**, затем поочередно щелкните по любым двум сторонам состояния *Adopter*;

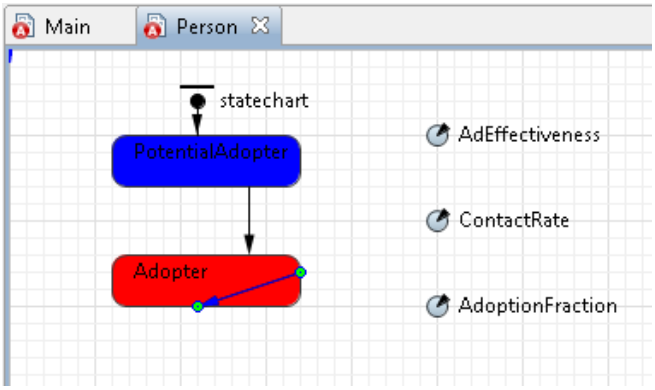


Рис. 50

3) задайте интенсивность, по которому происходит переход, – `ContactRate`.

Данный переход будет моделировать покупку продукта знакомым этого человека. От того, насколько быстро владелец продукта сумеет убедить своего знакомого в необходимости покупки, будет зависеть от силы убеждения этого человека и от того, сколько знакомых он встречает за год;

4) задайте **Действие** перехода:

```
send("Buy!", RANDOM);
```

Такой переход посылает сообщение случайно выбранному человеку. Метод *send()* отсылает сообщение другому агенту. Первый аргумент задает сообщение, которое будет послано, а второй задает агента, которому сообщение будет адресовано. В нашем случае мы посылаем сообщение какому-то случайно выбранному агенту, поэтому в качестве значения аргумента мы используем специальную константу RANDOM (рис. 51).

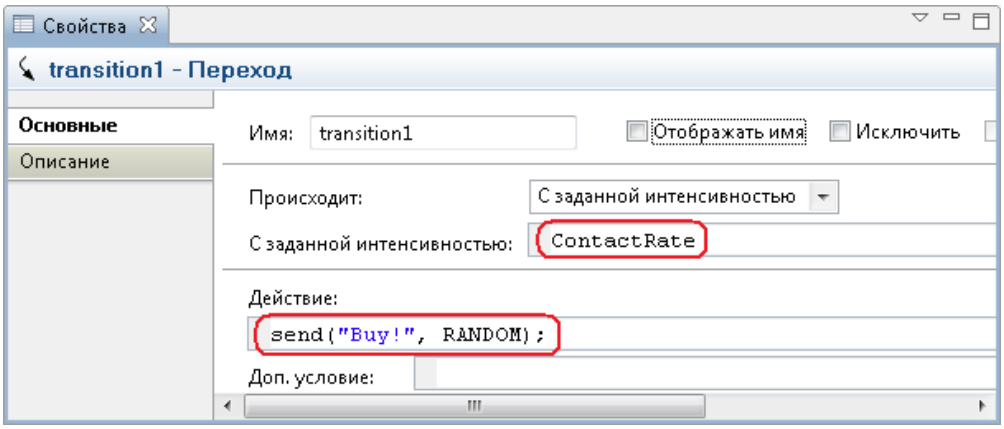


Рис. 51

Данный переход генерирует сигнал для стейтчарта какого-то знакомого.

Затем срабатывает переход стейтчарта, моделирующий покупку продукта этим знакомым;

5) добавьте еще один переход из состояния *PotentialAdopter* в состояние *Adopter* (рис. 52). Он будет срабатывать по сигналу, который будет генерироваться внутренним переходом состояния *Adopter*;

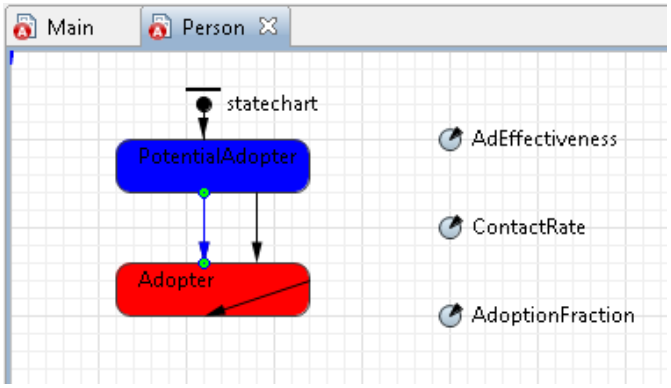


Рис. 52

6) измените свойства этого перехода.

Вероятность принятия решения о покупке продукта будет зависеть от силы убеждения человека. В нашей модели данная характеристика задается параметром *AdoptionFraction*.

Введите `randomTrue(AdoptionFraction)` в поле **Доп. условие**. В результате введения дополнительного условия продукт будет приобретаться с вероятностью, задаваемой параметром *AdoptionFraction*.

Переход будет срабатывать, когда диаграмма состояний этого агента получит сообщение "Buy!" («Купи!») от другого агента – своего знакомого. Чтобы этот переход срабатывал при получении сообщения, на странице свойств данного перехода выберите из выпадающего списка **Происходит** *При получении сообщения* (рис. 53). Укажите, что переход будет срабатывать только при получении сообщения соответствующего содержания. Для этого выберите из группы **Тип сообщения** опцию **String**, далее – опцию **Если сообщение равно** и введите `"Buy!"` в расположенном ниже поле.

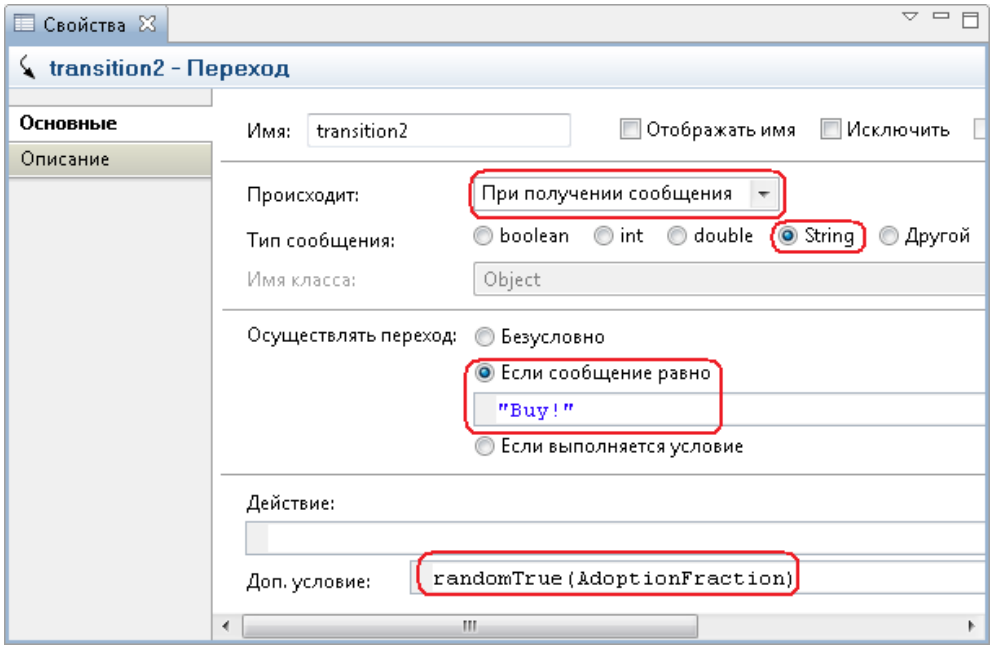


Рис. 53

Измените свойства агента.

Откройте свойства класса *Person* (рис. 54). В поле **Действие при получении** сообщения введите `statechart.receiveMessage(msg);`

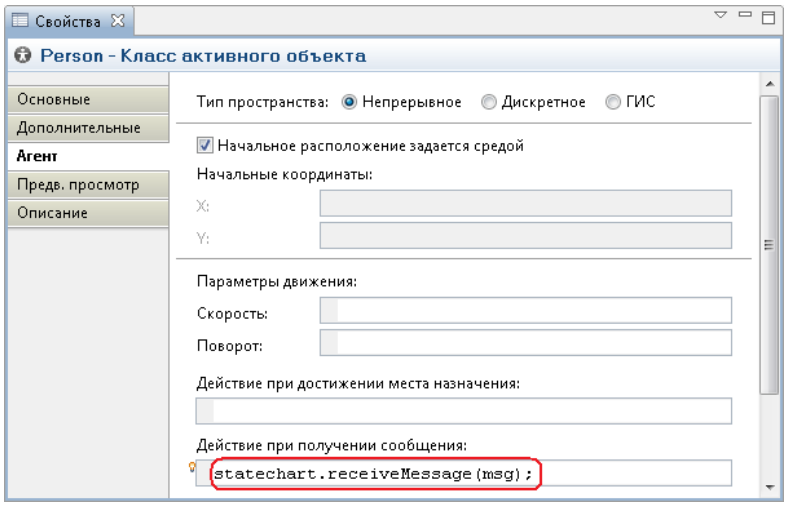


Рис. 54

Запустите модель. Изучите динамику изменения числа потребителей и потенциальных потребителей продукта. Графики переменных должны представлять собой *S*-образные кривые.

Создание пространственной модели

Сделаем нашу модель более реалистичной, допустив возможность общения только тех людей, которые находятся друг от друга на расстоянии, не превышающем 25 километров.

Свойства формирования сетей контактов агентов, как и многие другие свойства агентной модели, задаются в объекте *среда*.

1. Откройте диаграмму класса *Main*. Выделите на диаграмме объект *environment*  , задающий настройки среды, в которой обитают агенты.

2. На странице свойств **Дополнительные** измените тип сети контактов (рис. 55). Выберите **Согласно расстоянию** из выпадающего списка **Тип сети** и введите 25 в расположенном ниже поле **Радиус соединения**.

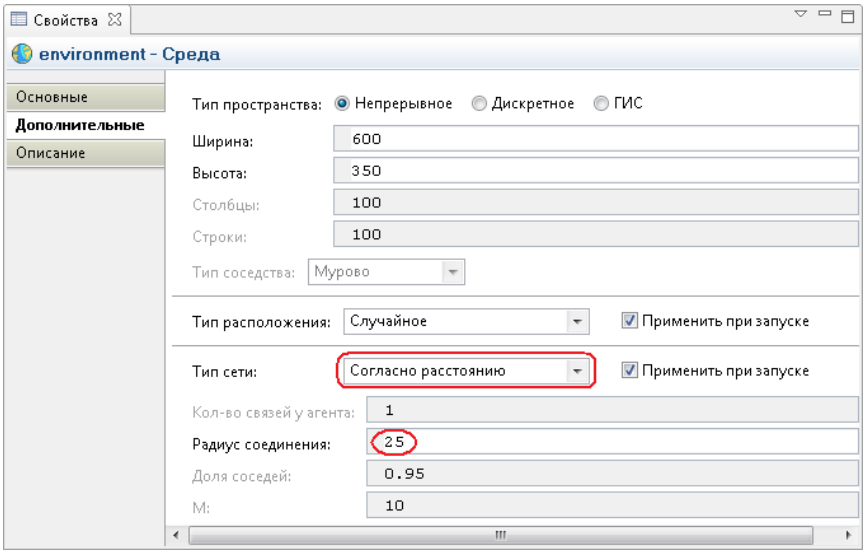


Рис. 55

Люди будут общаться не с любыми своими знакомыми, вне зависимости от места их проживания, а только с теми, кто живет поблизости.

3. Измените диаграмму состояний агента. Откройте диаграмму класса *Person*. Измените свойства внутреннего перехода состояния *Adopter* (рис. 56). Здесь в качестве значения последнего аргумента метода *send* используется специальная константа RANDOM_CONNECTED. Теперь этот переход посылает сообщение случайно выбранному знакомому этого человека.

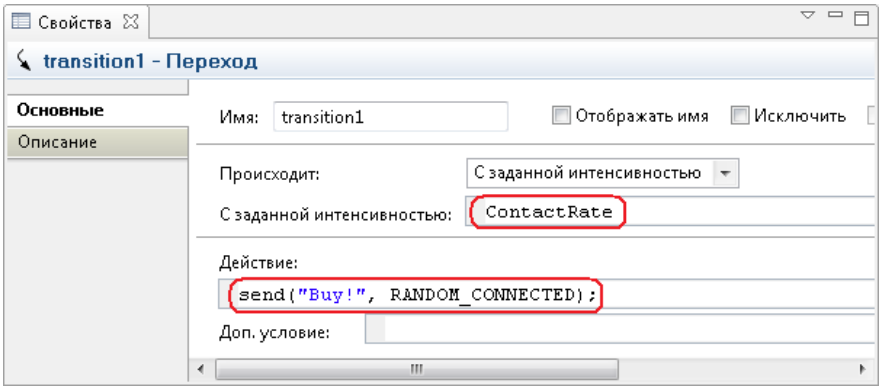


Рис. 56

4. Запустите модель и посмотрите, как изменилась динамика приобретения продукта. Убедитесь, что теперь агенты соединены только с теми, которые находятся от них на расстоянии, не превышающем 25 единиц, а сам процесс распространения продукта происходит медленнее.

Моделирование повторных покупок

Промоделируем повторные покупки, полагая, что потребители продукта снова становятся потенциальными потребителями, когда продукт, который они приобрели, становится непригоден.

1. Задайте средний срок службы продукта. Откройте структурную диаграмму класса *Main* и создайте параметр *DiscardTime* (рис. 57). Пусть средний срок службы продукта равен одному году.

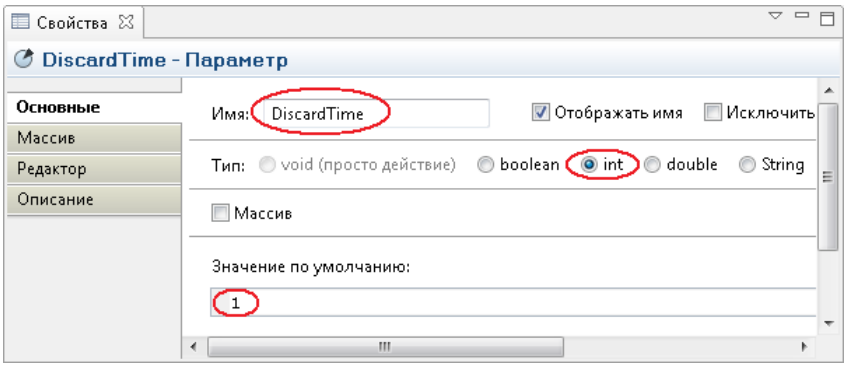


Рис. 57

2. Измените стейтчарт агента. Добавьте переход из состояния *Adopter* в состояние *PotentialAdopter* (рис. 58).

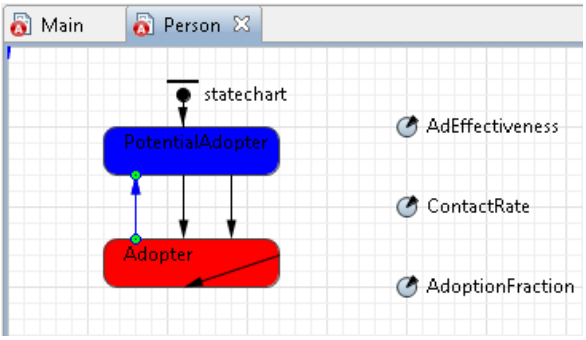


Рис. 58

Задайте свойства перехода как показано на рис. 59.

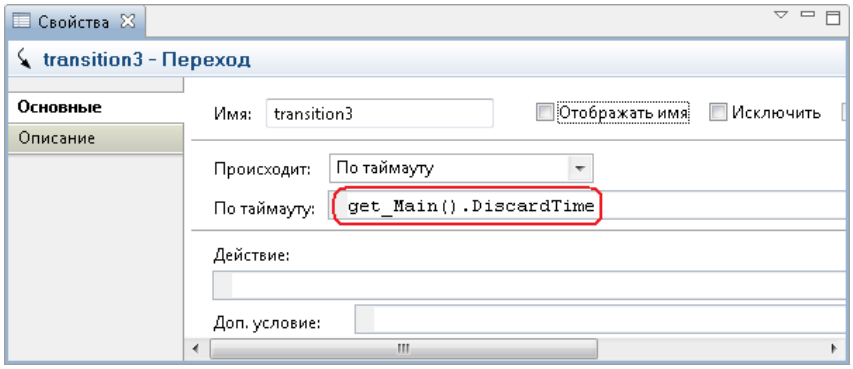


Рис. 59

3. Удалите условие остановки модели по времени и запустите модель. С помощью диаграммы проследите динамику изменения числа потребителей продукта. На диаграмме видно, что насыщение рынка в модели с повторными покупками не достигается.

Создание анимации

AnyLogic позволяет создавать интерактивную анимацию с возможностью изменения параметров системы по ходу моделирования процесса.

Для создания анимации добавим в модель необходимые переменные.

1. Откройте структурную диаграмму класса *Person*. Создайте вспомогательную переменную *isAdopter*; тип переменной – *boolean*; начальное значение – *false*. С помощью данной переменной будем определять, является ли данный человек владельцем продукта или нет.

2. Откройте структурную диаграмму класса *Main* и создайте две переменные: переменную *adopters* с начальным значением 0 и переменную *potadopters* с начальным значением 0. Эти переменные будем использовать для подсчета

численности покупателей и потенциальных покупателей.

3. Откройте диаграмму стейтчарта и внесите изменения. Щелкните мышкой по состоянию. В окне свойств состояния *PotentialAdopter* задайте **Действие при входе**

```
Main.potadopters++;
isAdopter=false;
```

и **Действие при выходе** из этого состояния:

```
Main.potadopters--;
```

При входе в состояние увеличивается на единицу значение переменной, подсчитывающей количество потенциальных потребителей продукта. При выходе из состояния уменьшается значение этой переменной. Кроме того, изменяется значение переменной *isAdopter*, чтобы показать, купил ли этот человек наш продукт (если да, то значение переменной равно *true*, если нет, то *false*).

4. Задайте свойства состояния *Adopter* (рис. 60).

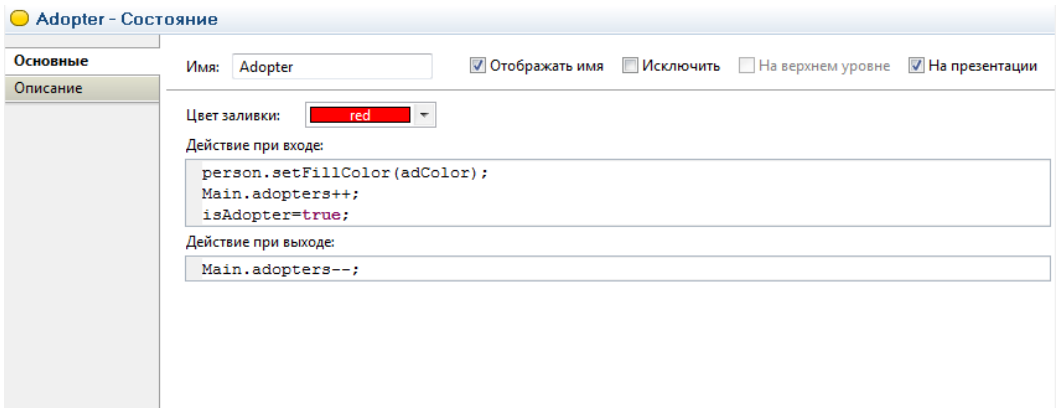


Рис. 60

5. Нарисуйте анимацию, как показано на рис. 61. Для этого добавьте **Столбцовый индикатор** числа потенциальных потребителей продукта (координаты: (350, 80), ширина – 30, высота – 200; переменная, которую будет отображать этот индикатор – *potadopters*; максимальное отображаемое значение – 1000, цвет индикации – желтый; сбросьте флажок **Отображать шкалу**).

Аналогичным образом добавьте **Столбцовый индикатор** числа потребителей (можно использовать копирование). Расположите его правее выше описанного индикатора, измените соответствующие свойства: переменную для

отображения, цвет индикации.

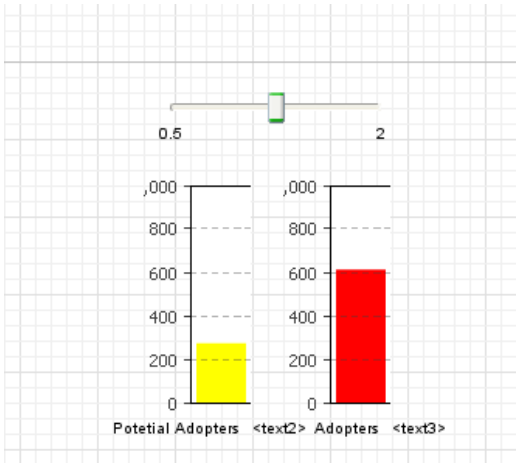


Рис. 61

Добавьте подписи к индикаторам *Potential Adopters* и *Adopters*, установите шрифт Times New Roman размером 8.

Добавьте элемент управления **Бегунок**, с помощью которого будем изменять срок службы продукта (координаты: (330, 10), ширина – 150, высота – 30; переменная, значение которой будет изменяться с помощью этого элемента управления, – *DiscardTime*; минимальные и максимальные значения бегунка – 0.5 и 2).

Поместите под бегунком три метки: *Discard Time* – под бегунком, 0.5 – у левого края бегунка и 2 – у правого.

6. Запустите модель.

7. Проанализируйте характеристики модели по своему варианту. Сделайте выводы.

Вариант	Эффективность рекламы	Сила убеждения	Длительность процесса
1	0,010	0,01	6
2	0,010	0,02	7
3	0,010	0,03	8
4	0,005	0,04	9
5	0,005	0,05	10
6	0,005	0,01	11
7	0,015	0,02	10
8	0,015	0,03	9
9	0,015	0,04	8
10	0,010	0,05	7
11	0,010	0,01	6
12	0,010	0,02	7

Вариант	Эффективность рекламы	Сила убеждения	Длительность процесса
13	0,005	0,03	8
14	0,005	0,04	9
15	0,005	0,05	10
16	0,015	0,01	11
17	0,015	0,02	10
18	0,015	0,03	9
19	0,020	0,04	8
20	0,020	0,05	7

Расширение модели жизненного цикла продукта

1. Моделирование рекламных кампаний

1) Откройте диаграмму класса *Main*.

2) Создайте новые переменные: *adX*, *adY*, тип *double*; *adTime*, тип *double*, начальное значение -1; *adRange*, тип *double*, начальное значение 50.

3) Создайте новое событие. Назовите таймер *adTimer*. Событие должно быть циклическим. Задайте **Таймаут** – `uniform(0.5,1)`. Задайте **Действие при срабатывании**:

```
adX=uniform(adRange,300-adRange);
adY=uniform(adRange,300-adRange);
adTime=time();
for (AgentContinuous2D people: environ-
ment.getAgentCollection() )
    if (people.distanceTo(adX,adY) < adRange)
      people.send("buy!", people);
```

Данный код производит циклический перебор всех агентов модели. У агентов, попавших в область проведения кампании (в круг с радиусом *adRange*), генерируется событие стейтчарта, в результате чего срабатывает переход стейтчарта, моделирующий покупку продукта.

В связи с тем, что теперь покупки моделируются с помощью таймера, то нужно удалить переход стейтчарта, который моделирует покупку продукта под влиянием рекламы.

4) Откройте диаграмму стейтчарта. Удалите переход из состояния *PotentialAdopter* в состояние *Adopter* (переход с таймаутом `AdEffectiveness`).

Необходимо также внести изменения в анимацию: будем подсвечивать области проведения рекламных кампаний на анимации сиреневыми кругами.

5) Откройте анимационную диаграмму. Нарисуйте круг (кнопка панели инструментов **Овал**) со свойствами, показанными на рис. 62-63.

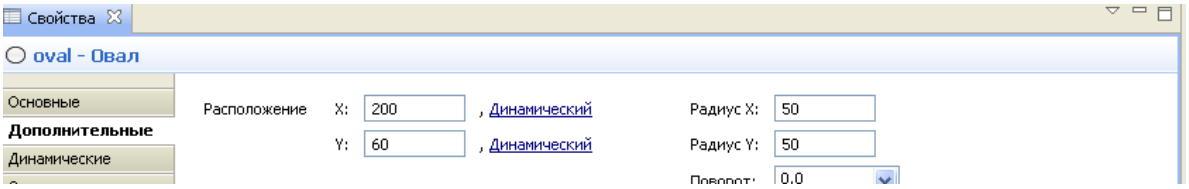


Рис. 62

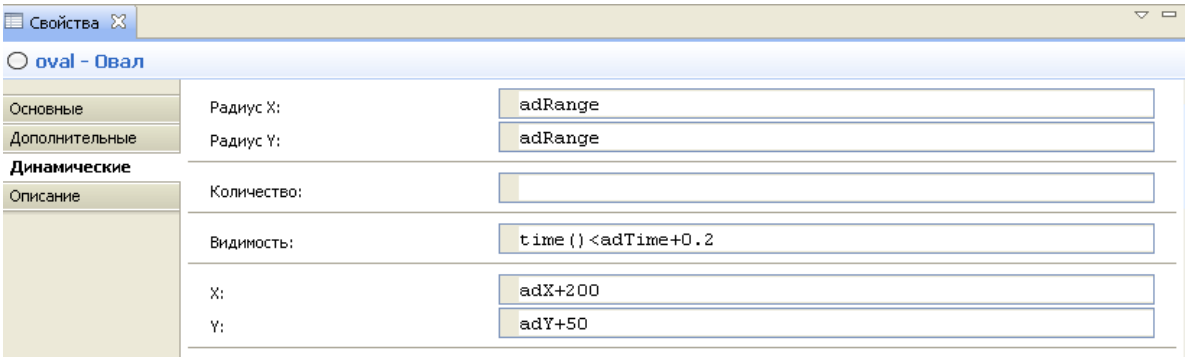


Рис. 63

Местоположение круга на диаграмме определяется переменными *adX* и *adY*, хранящими координаты центра области проведения текущей кампании. Круг будет виден в течение 0.2 единиц модельного времени после момента проведения кампании.

6) Запустите модель и изучите ее поведение.

2. Изучение распространения нескольких продуктов

Созданная модель изучает динамику распространения только одного продукта. Расширим модель, чтобы иметь возможность изучать распространение сразу нескольких различных продуктов.

1) Создайте вспомогательную переменную, которая будет определять тип рекламируемого продукта. Откройте структурную диаграмму класса *Main* и создайте переменную *adColor* класса *Color*.

Будем различать продукты по цвету: владельцев одного продукта отобразим на анимации синим цветом, а владельцев другого – красным. Области проведения рекламных кампаний также будут отображаться различными цветами в

зависимости от рекламируемого продукта.

2) Измените анимацию – измените свойства сиреневого круга: выберите переменную *adColor* в качестве **Цвета заливки** фигуры.

Пусть потенциальные покупатели продукта будут отображаться серым цветом, владельцы – красным или синим в зависимости от того, какой продукт они приобрели.

3) Измените код, который выполняется при срабатывании события *adTimer*, моделирующего проведение рекламных кампаний.

Откройте свойства события и измените **Действие при срабатывании**. Добавьте выделенные строки:

```
adX=uniform(adRange,400-adRange);
adY=uniform(adRange,400-adRange);
adTime=time();
adColor = uniform() > 0.5 ? Color.blue : Color.red;
for( Agent people: environment.getAgentCollection() )
  if (people.distanceTo(adX,adY) < adRange)
  people.send(adColor,people);
```

Теперь с равной вероятностью будут рекламироваться разные продукты.

4) Откройте диаграмму стейтчарта *adoption* и измените стейтчарт следующим образом (рис. 64).

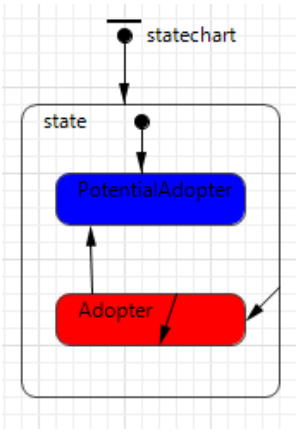


Рис. 64

Добавьте сложное состояние, включающее в себя оба состояния стейтчарта. Сделайте это состояние начальным: добавьте указатель начального состояния, направленный в него.

Удалите переход из состояния *PotentialAdopter* в состояние *Adopter*.

Измените **Действие** внутреннего перехода состояния *Adopter*. Замените на строку `send(color,RANDOM_CONNECTED);`.

Добавьте переход из сложного состояния в состояние *Adopter*. В окне **Свойства** задайте свойства перехода: переход должен срабатывать **По сигналу** *Color* и выполнять **Действие** `color=msg;`. Для этого ***тип сигнала*** ставим *другой* и пишем *Color*, а в поле **Действие** пишем `Main.adColor=msg;`.

Выделите щелчком мыши состояние *PotentialAdopter*. Добавьте в **Действия при входе** строки:

```
Main.adColor = Color.lightGray;
person.setFillColor(Main.adColor);
```

Выделите щелчком мыши состояние *Adopter*. Добавьте в **Действия при входе** строку `person.setFillColor(Main.adColor);`.

5) Запустите модель. Изучите распространение сразу двух продуктов (рис. 65).

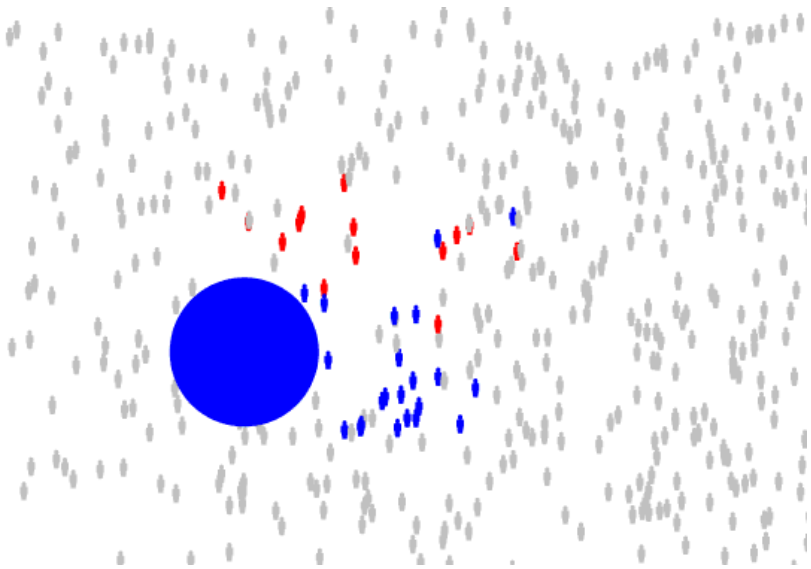


Рис. 65

3. Моделирование передвижения людей

В модели на данный момент люди неподвижно находятся в определенных местах. Чтобы сделать модель более реалистичной, смоделируем передвижение людей по рассматриваемой области.

1) Откройте структурную диаграмму класса *Person*.

2) Создайте вспомогательные переменные *oldx* и *oldy* типа *double*. Для обеих переменных задайте **Начальное значение** `uniform(300)`. Создайте переменную *tmoved* типа *double*. Задайте **Начальное значение** -10.

3) Создайте новое **Событие**. Назовите его *moveTimer*. Событие должно быть циклическим. Задайте **Таймаут** `uniform(5,10)`. Задайте **Действие при срабатывании** таймера:

```
oldx=person.x;
oldy=person.y;
tmoved=time();
person.x=triangular(0,150,300);
person.y=triangular(0,150,300);
```

При срабатывании таймера текущее модельное время, возвращаемое функцией *time()*, запоминается переменной *tmoved*. Переменные *oldx* и *oldy* запоминают текущие координаты агента *x* и *y*. Новые координаты генерируются по треугольному закону распределения с минимальным значением, равным 0, средним – 150 и максимальным – 300.

4) Создайте две алгоритмические функции, которые помогут более плавно рисовать движение людей на анимации.

Для этого перетащите соответствующий элемент с палитры **Основные** на структурную диаграмму. Назовите функцию *animationX*. Задайте **Тело функции**:

```
if (time() > tmoved+0.5)
    return person.x;
return oldx + (person.x-oldx)*(time()-tmoved)/0.5;
```

Аналогично создайте алгоритмическую функцию *animationY*:

```
if (time() > tmoved+0.5)
    return person.y;
return oldy + (person.y-oldy)*(time()-tmoved)/0.5;
```

5) Откройте свойство *Person* (рис. 66) и на вкладке **Агент** укажите:

```
animationX();
animationY();
```

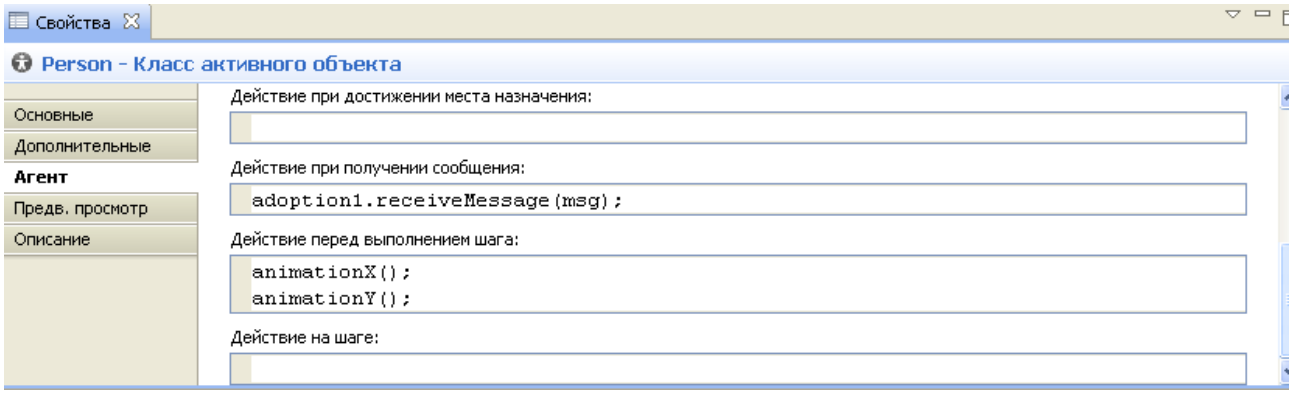


Рис. 66

6) Запустите модель и проследите за перемещением людей с помощью анимации.

В созданной модели число людей, проживающих в изучаемой нами области, является постоянным. На самом деле всегда происходит миграция людей – они переезжают из одной области в другую, и это тоже необходимо учесть в модели. Смоделируем миграцию людей с помощью таймеров.

7) Откройте структурную диаграмму класса *Main* и создайте новое **Событие**. Назовите его *inMigration*. Событие должно быть циклическим.

Задайте **Таймаут** `exponential(20)`. В среднем в нашу область будут приезжать 20 человек в год (в качестве аргумента функции *exponential()* задается частота происхождения события). Задайте **Действие** `add_people();`.

8) Для моделирования того, как люди уезжают из области, создайте еще одно событие.

Откройте структурную диаграмму класса *Person* и создайте новое **Событие**. Назовите его *outMigration*. Событие должно срабатывать только один раз, поэтому установите флажок **Срабатывает один раз**. Задайте **Таймаут** `exponential(0.02)`. Задайте **Действие**:

```
get_Main().remove_people( this );
if (isAdopter)
    get_Main().adopters--;
else
    get_Main().potadopters--;
```

9) Добавьте на анимацию метку, отображающую число жителей области.

Для этого в области просмотра анимации добавьте текстовую метку, отображающую следующий текст:

```
people.size()+" people".
```

10) Запустите модель и изучите ее поведение. Окно просмотра анимации показано на рис. 67.

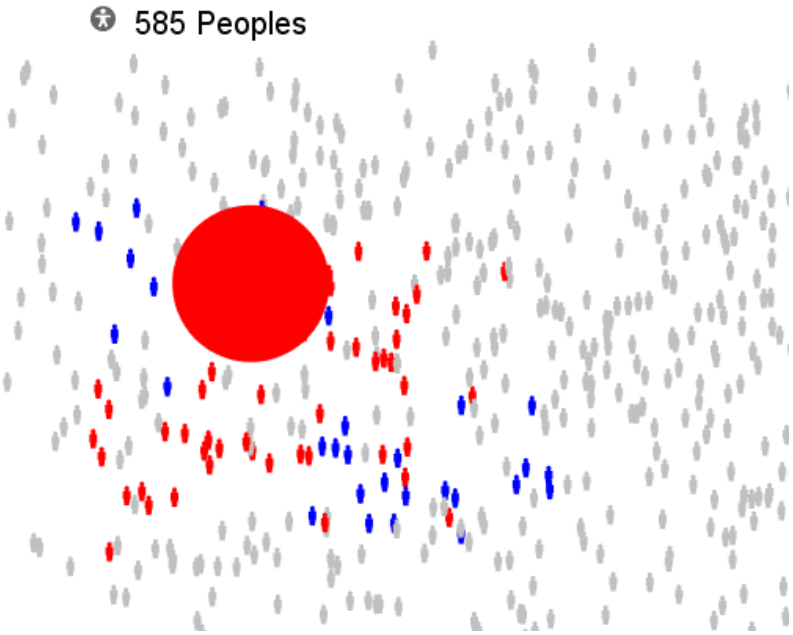


Рис. 67

Результаты работы

Студент должен предоставить отчет по лабораторной работе с выводами, продемонстрировать работу модели, ответить на вопросы преподавателя.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Карпов Ю. Г. Имитационное моделирование систем. Введение в моделирование с AnyLogic 5 / Ю. Г. Карпов. СПб. : БХВ Петербург, 2006. 400 с.
2. AnyLogic User's Manual. XJ Technologies : [электрон. ресурс]. Режим доступа : http://www.xjtek.com
3. AnyLogic Tutorial. XJ Technologies : [электрон. ресурс]. Режим доступа : http://www.xjtek.com
4. Многоподходное имитационное моделирование в AnyLogic. XJ Technologies : [электрон. ресурс]. Режим доступа : http://www.xjtek.ru

ОГЛАВЛЕНИЕ

Учебное издание

Киселева Марина Васильевна

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ СИСТЕМ В СРЕДЕ ANYLOGIC

Редактор *И.В. Меркурьева*

Компьютерная верстка *М. В. Киселевой*

Подписано в печать 02.12.2009 Формат 60 ×841/16
Бумага писчая Плоская печать Усл.печ.л. 5,12
Уч.-изд.л. 4,4 Тираж 50 экз. Заказ

Редакционно-издательский отдел УГТУ-УПИ

620002, Екатеринбург, ул. Мира, 19

rio@mail.ustu.ru

ОАО «Издательство УМЦ УПИ»

620002, Екатеринбург, ул. Мира, 17