Министерство образования и науки Российской Федерации
Федеральное агентство по образованию
Южно-Уральский государственный университет
Кафедра «Информационно-аналитическое обеспечение управления в социальных и экономических системах»

681.3(07)
К681

А.М. Коровин

МОДЕЛИРОВАНИЕ СИСТЕМ

Учебное пособие к лабораторным работам

Челябинск
Издательский центр ЮУрГУ
2010

УДК 681.3.066(076.5)
К681

Одобрено
учебно-методической комиссией
приборостроительного (КТУР) факультета.

Рецензенты:
В.С. Жабреев, М.Ю. Катаргин.

Коровин, А.М.

К681 Моделирование систем: учебное пособие к лабораторным работам / А.М. Коровин. – Челябинск: Издательский центр ЮУрГУ, 2010. – 47 с.

В учебном пособии изложены теоретические предпосылки, необходимые для практического решения в ходе лабораторного практикума по курсу «Моделирование» задач имитационного моделирования сложных систем на основе создания гибридных моделей с помощью новой отечественной инструментальной системы AnyLogic. Приведены примеры моделей дискретно-событийных и динамических систем.

Пособие предназначено для студентов, обучающихся в рамках направления подготовки 230100 – «Информатика и вычислительная техника», а также будет полезно студентам других родственных направлений и специальностей.

УДК 681.3.066(076.5)

ВВЕДЕНИЕ

Компьютерное имитационное моделирование становится сегодня одним из обязательных этапов при принятии ответственных решений в процессе управления сложными системами. Поэтому знание концепций, принципов и возможностей компьютерного моделирования, умение использовать существующий программный инструментарий для создания и использования моделей являются необходимыми требованиями, предъявляемыми к менеджеру, бизнес-аналитику.

Однако широкому распространению имитационного моделирования до сих пор препятствует несколько причин. Во-первых, системы, для которых необходима разработка моделей, обычно сложны, что затрудняет построение для них моделей даже с использованием специализированных пакетов. Во-вторых, разработка сложных моделей часто требует от пользователя написания программного кода и знания языка программирования, с которым совместима среда разработки. Традиционный подход в этой области основную проблему имитационного моделирования видел именно в разработке компьютерной программы, реализующей модель, программ генерации и анализа случайных величин, а не в разработке и анализе модели как таковой. В результате разработка полезных моделей в традиционной технологии может тянуться годами. Потому сроки разработки модели являются критическим фактором. Среда разработки имитационных моделей, скрывающая от разработчика все эти проблемы, может качественно ускорить построение и анализ имитационных моделей.

Лозунг революции, которая уже происходит в имитационном моделировании, следующий: «*Не нужно быть ни профессиональным программистом, ни профессиональным математиком, чтобы разрабатывать имитационные модели»* [2, 8, 10]. Это совсем не значит, что строить модели могут ничего не понимающие в программировании и математике люди: создание полезных моделей требует подготовки в обеих этих областях.

В данном пособии основное внимание уделено вопросам практического применения современной отечественной инструментальной системы AnyLogic, недавно разработанной на основе выше указанных новых подходов в моделировании.

В первой главе анализируются существующие основные подходы к имитационному моделированию и применяемое программное обеспечение. Во второй главе описаны возможности и средства имитационного моделирования в среде AnyLogic. Примеры решения прямой и обратной задач моделирования с помощью AnyLogic версии 5.4.1 изложены в третьей главе.

Указанная система оказалось очень удобным и мощным средством для решения широкого круга задач имитационного моделирования для процессов и систем различной природы в производстве и бизнесе. Возможность использования одного упомянутого инструмента при изучении различных парадигм моделирования делает программный продукт AnyLogic незаменимым при изучении курса моделирования для направления «Информатика и вычислительная техника».

Глава 1. ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ

1.1. Понятие имитационного моделирования

Имитационное моделирование – это разработка и выполнение на компьютере программной системы, отражающей структуру и функционирование (поведение) моделируемого объекта или явления во времени. Такую программную систему называют имитационной моделью этого объекта или явления. Объекты и сущности имитационной модели представляют объекты и сущности реального мира, а связи структурных единиц объекта моделирования отражаются в интерфейсных связях соответствующих объектов модели. Таким образом, *имитационная модель* – это упрощенное подобие реальной системы, либо существующей, либо той, которую предполагается создать в будущем. Имитационная модель обычно представляется компьютерной программой, выполнение программы можно считать имитацией поведения исходной системы во времени [1, 2, 6, 8, 9, 10].

В русскоязычной литературе термин «*моделирование*» соответствует американскому «*modeling*» и имеет смысл создание модели и ее анализ, причем под термином «*модель*» понимается объект любой природы, упрощенно представляющий исследуемую систему. Слова «*имитационное моделирование*» и «*вычислительный (компьютерный) эксперимент*» соответствуют англоязычному термину «*simulation*». Эти термины подразумевают разработку модели именно как компьютерной программы и исполнение этой программы на компьютере.

Итак, *имитационное моделирование* – это деятельность по разработке программных моделей реальных или гипотетических систем, выполнение этих программ на компьютере и анализ результатов компьютерных экспериментов по исследованию поведения моделей. Имитационное моделирование имеет существенные *преимущества перед аналитическим моделированием* в тех случаях, когда:

– отношения между переменными в модели нелинейны, и поэтому аналитические модели трудно или невозможно построить;

– модель содержит стохастические компоненты;

– для понимания поведения системы требуется визуализация динамики происходящих в ней процессов;

– модель содержит много параллельно функционирующих взаимодействующих компонентов.

Во многих случаях имитационное моделирование – это единственный способ получить представление о поведении сложной системы и провести ее анализ.

Имитационное моделирование может использоваться при принятии решений на стадиях проектирования и анализа производственных систем (например, конвейерных линий или складских помещений), транспортных систем (автомагистралей, портов, метрополитена), различных организаций, предоставляющих сервисы массового обслуживания (парикмахерских, центров обработки заказов по телефону, больниц, автозаправок, банков), социальных и финансовых систем и т. п.

1.2. Этапы имитационного моделирования

Имитационное моделирование состоит из двух больших этапов: создания модели и анализа построенной модели с целью принятия решения. Но построение действительно полезной имитационной модели, как отмечено в [1, 2] требует большой работы.

Сначала разработчик модели должен определить, какие задачи будут решаться с ее помощью, т. е. моделированию в любой его форме должна предшествовать *формулировка цели моделирования*. От цели зависит то, какие процессы в реальной системе следует выделить и отразить в модели, а от каких процессов абстрагироваться, какие характеристики этих процессов учитывать, а какие – нет, какие соотношения между переменными и параметрами модели должны быть отражены в модели.

Следующий этап можно охарактеризовать как создание *концептуальной (содержательной)* модели. На нем происходит *структуризация* модели, т. е. выделение отдельных подсистем, определение элементарных компонентов модели и их связей на каждом уровне иерархии. В имитационном моделировании структура модели отражает структуру реального объекта моделирования на некотором уровне абстракции, а связи между компонентами модели являются отражением реальных связей.

Элементы системы, их связи, параметры и переменные, а также их соотношения и законы их изменения должны быть выражены средствами *среды моделирования*, т. е. в этой среде должны быть определены переменные и параметры модели, построены процедуры вычисления изменения переменных и характеристик модели во времени.

При необходимости для большего понимания процессов, протекающих в модели, должно быть разработано *анимационное* представление этих процессов.

Затем построенная модель должна быть *проверена* с точки зрения *корректности* ее реализации.

Следующий этап – это *калибровка или идентификация* модели, т. е. сбор данных и проведение измерений тех характеристик в реальной системе, которые должны быть введены в модель в виде значений параметров и распределений случайных величин.

Далее, необходимо выполнить *проверку правильности* модели (ее *валидацию),* которая состоит в том, что выход модели проверяется на нескольких тестовых режимах, в которых характеристики поведения реальной системы известны либо очевидны.

Последним этапом работы с моделью является *компьютерный эксперимент,* т.е. собственно то, ради чего и создавалась модель. В простейшем случае компьютерный эксперимент – это запуск на исполнение модели при различных значениях ее существенных параметров (факторов) и наблюдение ее поведения с регистрацией характеристик поведения. Этот вид использования модели называется прогнозом, или экспериментом типа «*что будет, если...*». Компьютерное моделирование позволяет не только получить прогноз, но и

определить, какие управляющие воздействия на систему приведут к благоприятному развитию событий.

В табл. 1.1 перечислены этапы компьютерного имитационного моделирования.

Таблица 1.1

Этапы компьютерного имитационного моделирования

№ п/п	Название этапа	Результат
1	Понимание системы	Понимание того, что происходит в системе, подлежащей анализу, какова ее структура, какие процессы в ней протекают
2	Формулировка цели моделирования системы	Список задач, которые предполагается решить с помощью будущей модели. Список входных и выходных параметров модели, список исходных данных, критерии завершенности будущего исследования
3	Разработка концептуальной структуры модели	Структура модели, состав существенных процессов, подлежащих отображению в модели, зафиксированный уровень абстракции для каждой подсистемы модели (список допущений), описание управляющей логики для подсистем
4	Реализация модели в среде моделирования	Реализованные подсистемы, их параметры и переменные, их поведение, реализованная логика и связи подсистем
5	Реализация анимационного представления модели	Анимационное представление модели, интерфейс пользователя
6	Проверка корректности реализации модели	Убеждение в том, что модель корректно отражает те процессы реальной системы, которые требуется анализировать
7	Калибровка модели	Фиксация значений параметров, коэффициентов уравнений и распределений случайных величин, отражающих те ситуации, для анализа которых модель будет использоваться
8	Планирование и проведение компьютерного эксперимента	Результаты моделирования – графики, таблицы и т. п., дающие ответы на поставленные вопросы

Более сложные эксперименты позволяют выполнить *анализ чувствительности модели,* оценку рисков различных вариантов управляющих решений, а также *оптимизацию* для определения параметров и условий рационального функционирования модели.

Один из важных вопросов – представление и анализ результатов моделирования. Для этого в инструментальной среде могут быть использованы специальные средства для обработки статистической информации, для представления в структурированном или графическом виде полученных данных, интеграция с внешними базами данных и т. п.

Часто имитационная модель используется в качестве модуля большей системы поддержки принятия решения (СППР), получающей в режиме реального времени данные мониторинга состояния управляемой системы, оценивающей, к каким последствиям может привести текущая ситуация, и предлагающей оптимальное (или просто рациональное) управляющее решение для минимизации отрицательных последствий развития системы в будущем.

1.3. Анализ подходов к имитационному моделированию и применяемого программного обеспечения

Прогноз состояния ресурсов и средств подразумевает ответы на вопросы «что будет, если...» и как расходуются ресурсы и средства при выполнении текущего плана производства, при поступлении новых заказов, изменении сроков и объемов поставки ресурсов, поломках средств. Решается задача путем моделирования текущей ситуации и проигрыванием сценариев ее развития.

Одним из базовых элементов структуры современных СППР являются системы моделирования, для построения которых в настоящее время используются следующие подходы:

- системная динамика,
- дискретно-событийное моделирование,
- динамические системы,
- агентное моделирование,
- объединенный подход.

Системная динамика (программные продукты: VenSim, PowerSim, iThink, ModelMaker и др.). Основная парадигма данного подхода заключается в математическом описании систем дифференциальных уравнений, приведенных к форме Коши. Системно-динамический подход был разработан и предложен Дж. Форрестером в конце 1950-х как «исследование информационных обратных связей в промышленной деятельности с целью показать как организационная структура, усиления (в политиках) и задержки (в принятии решений и действиях) взаимодействуют, влияя на успешность предприятия». Приложения системной динамики включают также социальные, урбанистические, экологические системы. Системная динамика абстрагируется от отдельных объектов и событий и предполагает «агрегатный» взгляд на процессы, концентрируясь на политиках,

этими процессами управляющих. Моделируя в стиле системной динамики, вы представляете структуру и поведение системы как множество взаимодействующих положительных и отрицательных обратных связей и задержек.

Важно отметить следующие моменты системной динамики:

1) поскольку модель оперирует только количествами, агрегатами, объекты, находящиеся в одном накопителе, неразличимы, лишены индивидуальности;

2) аналитику предлагается рассуждать в терминах глобальных структурных зависимостей и, соответственно, ему необходимы соответствующие данные.

Дискретно-событийное моделирование (программные продукты: Arena, GPSS, Extend, SimProcess, AutoMod, PROMODEL, Enterprise Dynamics, FlexSim, eMPlant и др.). Основная парадигма данного подхода заключается в использовании транзактов, отображающих динамические объекты моделирования (заявки), и блоков-объектов, обрабатывающих эти заявки. Дискретно-событийное моделирование наиболее развито и имеет огромную сферу приложений – от логистики и систем массового обслуживания до транспортных и производственных систем. Этот вид моделирования наиболее подходит для моделирования производственных процессов. Идея моделирования систем с дискретными событиями была сформулирована в виде Системы общецелевого моделирования (General Purpose Systems Simulator) Джеффри Гордоном (Geoffrey Gordon) в 1961 году. Сейчас GPSS исполнилось 49 лет.

Это был один из самых удачных на то время проблемно-ориентированных языков программирования. Проблемной областью GPSS являются системы массового обслуживания (системы с очередями).

Историю GPSS можно условно разделить на два больших этапа. Первый этап – это GPSS на "больших" ЭВМ, так называемых мэйнфреймах (типа IBM/360 и ЕС ЭВМ). Второй этап – на персональных ЭВМ.

Первая версия системы появилась в 1961 году и называлась GPSS. Далее последовательно друг за другом появились GPSS II (1963 г.), GPSS III (1965 г.), GPSS/360 (1967 г., GPSS V (1971 г.). Все эти версии были разработаны и поддерживались фирмой IBM.

Второй этап. Появление персональных ЭВМ и принципиально новых идей и подходов взаимодействия человека с ЭВМ не могло не отразиться на GPSS. Он несколько утратил свою привлекательность. Появились новые системы моделирования, использующие возможности новой техники – оперативность, интерактивность, наглядность при разработке моделей и проведении исследований. Среди огромного множества разработок в последующие годы (1982-2001 гг.) сейчас можно выделить несколько основных. Это системы: GPSS/PC и GPSS World (Minuteman Software), MicroGPSS и WebGPSS.

Язык GPSS ввел в моделирование парадигму потокового или сетевого моделирования (flowchart или network-based modeling). Поток сущностей (транзакций) продвигается по структурной диаграмме, представляющей систему. Транзакции ожидают в очередях, конкурируют за использование ресурсов и блоков, осуществляющих их обработку (обслуживание), и в конце концов

покидают систему. Парадигма потокового моделирования оказалась достаточно общей и использовалась во многих программных продуктах.

Агентное моделирование (программные продукты: Swarm, RePast, библиотеки на Java или C++, разработанные в различных университетах, и др.). Основная парадигма заключается в том, что модель представляется в виде множества отдельных активных объектов (агентов), каждый из которых взаимодействует с другими агентами, образующими для него внешнюю среду. Агентное моделирование – относительно новое направление в имитационном моделировании, которое появилось в 90-х годах и используется для исследования децентрализованных систем, динамика функционирования которых определяется не глобальными правилами и законами (как в других парадигмах моделирования), а когда эти глобальные правила и законы являются результатом индивидуальной активности членов группы. Цель агентных моделей – получить представление об этих глобальных правилах, общем поведении системы, исходя из предположений об индивидуальном, частном поведении ее отдельных активных объектов и взаимодействии этих объектов в системе. Агент – некая сущность, обладающая активностью, автономным поведением, может принимать решения в соответствии с некоторым набором правил, взаимодействовать с окружением, а также самостоятельно изменяться.

В отличие от системной динамики или дискретно-событийных моделей, здесь нет такого места, где централизованно определялось бы поведение (динамика) системы в целом. Вместо этого, аналитик определяет поведение на индивидуальном уровне, а глобальное поведение возникает как результат деятельности многих (сотен, тысяч, миллионов) агентов, каждый из которых следует своим собственным правилам, живёт в общей среде и взаимодействует со средой и с другими агентами.

В случае моделирования систем, содержащих большие количества активных объектов (людей, животных, машин, предприятий или даже проектов, активов, товаров и т. п.), которые объединяет наличие элементов индивидуального поведения, от сложных (цели, стратегии) до самых простых (временные ограничения, события, взаимодействия) агентное моделирование является подходом более универсальным и мощным, так как оно позволяет учесть любые сложные структуры и поведения. Другое важное преимущество агентного моделирования в том, что разработка модели возможна в отсутствии знания о глобальных зависимостях: вы можете знать очень немного о том, как вещи влияют друг не друга на глобальном уровне, или какова глобальная последовательность операций, и т. п., но, понимая индивидуальную логику поведения участников процесса, можно построить агентную модель и вывести из неё глобальное поведение. Таким образом, иногда, даже если в принципе и существует, скажем, системно-динамическая модель системы, построить агентную модель может быть проще. И, наконец, агентную модель проще поддерживать: уточнения обычно делаются на локальном уровне и не требуют глобальных изменений. В таком случае, введение новых объектов (предприятий, объектов социальной

инфраструктуры, планов стратегического развития) в агентной имитационной модели не изменяет её структуры, что может потребоваться при использовании моделей системной динамики.

Динамические системы (программные продукты: MATLAB, VisSim, LabView, Easy 5 и др.). Основная парадигма данного подхода заключается, как и в системной динамике, в описании системы соответствующей математической моделью, состоящей из набора переменных состояния и системы алгебро-дифференциальных уравнений над ними. Но, в отличие от системной динамики, переменные состояния имеют прямой «физический» смысл: координата, скорость и т. д. и не являются агрегатами дискретных объектов.

Практически все выше перечисленные программные инструменты имитационного моделирования разработаны для поддержки одного определённого подхода.

Среди систем имитационного моделирования, предоставляющих *объединенный подход*, стоит выделить систему AnyLogic, объединяющую возможности создания гибридных моделей на основе моделей системной динамики, дискретно-событийных моделей и агентного подхода.

Для определения подхода реализации системы моделирования экономических и социальных систем (ЭСС) (например, городской среды) необходимо определиться с возможностью создания в рамках каждого из них объектов как с более, так и менее детальным уровнем абстракции, так как указанные ЭСС представляет собой систему, содержащую большие количества активных объектов (людей, предприятий), часть которых объединяет наличие определенных моделей поведения, при этом каждый объект может реализовать как сложную, более детальную (поведение конкретного предприятия или человека), так и простую, менее детальную (поведение производственной отрасли или группы людей) модели поведения.

В таблице 1.2 представлены возможности реализации в каждом из подходов объектов с разными уровнями абстракции.

Таблица 1.2

Уровни абстракций в различных подходах имитационного моделирования

Имитационный подход	Уровень абстракции		
	Высокий	Средний	Низкий
Системная динамика	+	–	–
Дискретно-событийное моделирование	–	+	+
Динамические системы	–	–	+
Агентное моделирование	+	+	+

Таким образом, в результате рассмотрения парадигм имитационного моделирования и анализа таблицы 1.2 были отмечены преимущества

агентно-ориентированного подхода для анализа динамики развития ЭСС. При использовании данного подхода, в отличие от остальных, отдельные элементы ЭСС модели могут быть представлены объектами с любым уровнем абстракции. Чем более детально в модели будут реализованы агенты, представляющие собой определенные субъекты ЭСС, тем более глубокой и точной будет результирующая информация о моделируемой системе. Глобальное поведение мультиагентных систем возникает как результат деятельности многих (десятков, сотен, тысяч, миллионов) агентов, поведение которых определяется на индивидуальном уровне.

Наиболее мощным из инструментов, поддерживающих агентное моделирование, является отечественный инструментарий AnyLogic компании XJ Technologies, доказавший в последнее время свою мощь и удобство за счет применения объектно-ориентированного подхода, визуального проектирования, дружественного пользовательского графического интерфейса, платформонезависимого языка Java, агентных технологий, технологии гибридных систем. Все это делает AnyLogic мощным средством для решения очень широкого круга проблем в области имитационного моделирования, предоставляющего гибкость для решения одной и той же задачи на одной платформе с применением различных уровней абстракции, а также возможность применения различных стилей создания модели и их комбинирования.

1.4. Компьютерный эксперимент на имитационной модели

Исследование систем с помощью имитационного моделирования состоит в организации и проведении компьютерного эксперимента на имитационной модели. Такое исследование отличается как от натурного эксперимента с реальным объектом, так и умозрительного эксперимента. Компьютерный эксперимент с построенной имитационной моделью лежит между этими двумя крайностями. Более того, он имеет преимущества перед обоими этими подходами.

Действительно, правильно организованный натурный эксперимент обычно дает точный ответ на поставленный вопрос, но он часто дорог или экономически неэффективен. Иногда такой эксперимент попросту невозможен, например в случае, если системы еще не существует. С компьютерной моделью можно проигрывать ситуации, которые трудно или нежелательно получить в натурном эксперименте. Модель можно выполнять многократно, постепенно усложнять, обращая внимание на все более тонкие детали, которые при наблюдении реального процесса могут и ускользнуть от наблюдения или быть незарегистрированными. Прогоняя компьютерную модель в различных масштабах времени, можно часто получить значительно больше информации из наблюдения анимационной картины процесса, чем наблюдая реальный эксперимент.

С другой стороны, умозрительный эксперимент – это волюнтаристское решение проблемы на основе «здравого смысла» и общих интуитивных предположений о поведении системы. Но для сложных систем очевидные на

первый взгляд решения зачастую оказываются неверными. Поэтому основанные на интуиции методы прогноза и традиционные методы «волевого» принятия решения во многих случаях оказываются неадекватными. Моделирование и проведение компьютерного эксперимента позволяет избежать недостатков обеих крайностей, в связи с чем этот подход завоевывает все большую популярность.

При постановке задач различают *прямые и обратные задачи* имитационного моделирования.

Начнем с определения *прямой задачи* имитационного моделирования. При компьютерном эксперименте среди множества параметров модели должны быть выделены те из них, которые будут считаться «факторами», влияние которых на выходные переменные модели должно быть проанализировано. Все факторы, которые могут изменяться при поиске «хорошего» варианта, составляют набор *<x1, x2,… ,xn>*, который можно считать вектором *x* длины *n*.

Обозначим через *x* все множество возможных наборов факторов. Компьютерный эксперимент с моделью состоит в том, что модель запускается на компьютере при различных значениях факторов и/или различных структурных характеристиках, которые, конечно, тоже можно считать факторами (например, представив логический параметр «*включено/не включено»* для какой-либо ветви структуры как фактор с двумя значениями: 1 и 0). Каждый прогон модели приведет к получению вектора исходов, например плотности народонаселения через 20 лет в некотором районе, вектору коэффициентов загрузки оборудования и т. п. В результате таких экспериментов исследователь может получить ответ на вопрос:

«К каким последствиям могло бы привести данное изменение в реальной системе с течением времени?»

или по-другому:

«Что будет, если в заданных условиях мы примем конкретное решение k из области допустимых решений x?»

Это так называемая *прямая* задача имитационного моделирования (задача типа «what-if», или «что-если»). Например, для примера моделирования телефонной станции при проведении экспериментов с моделью можно изменять число телефонных каналов и тип станции. Определение влияния этих изменений на интересующие нас выходные значения (например, на процент отвергнутых заявок) является примером прямой задачи моделирования.

Формально прямая задача имитационного моделирования может быть представлена следующим образом. Обозначим *y=<y1, y2,..., ym>* вектор значений интересующих исследователя признаков. Пусть множество *Y* обозначает множество всех возможных результатов, *X* и *Y* в общем случае множества вещественных векторов конечной размерности. Если построенная имитационная модель не содержит неопределенностей и позволяет однозначно определить связь каждого конкретного набора параметров системы *x=<x1, x2,..., xn>* из множества допустимых решений *X* с вектором результирующих показателей *y=<yl, y2,…, ym>* из множества *Y*, то такой наиболее простой случай назовем *детерминированным.*

В детерминированном случае имитационная модель выступает как функциональное отображение $x \rightarrow y$, а сам имитационный эксперимент можно рассматривать как реализацию этого отображения: один прогон имитационной модели для каждого набора x параметров системы дает набор $y \in Y$ интересующих пользователя выходных значений. Эти выходные значения могут зависеть также и от времени.

Инструмент имитационного моделирования при выполнении компьютерного эксперимента в этом случае должен обеспечить удобный интерфейс для задания значений исходных параметров (факторов) и регистрации соответствующих значений выходных показателей и их изменения во времени.

Задачи вида «что-если» в AnyLogic решаются с помощью так называемого *простого эксперимента.* Такой эксперимент позволяет визуально отображать результаты работы модели с помощью анимации, графиков (диаграмм) и т. п. Широкие возможности для отображения данных предоставляет библиотека бизнес-графики (Business Graphics Library).

Другой тип компьютерного эксперимента – это *анализ чувствительности*, то есть процедура оценки влияния исходных гипотез и значений ключевых факторов на выходные показатели модели. Обычно эксперимент с варьированием параметров на рис. 1.1. 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.

Рис. 1.1. Имитационная модель при решении прямых задач

Для того чтобы оценить влияние изменения отдельных факторов на поведение модели на AnyLogic, пользователь не должен многократно запускать модель раз за разом, вручную меняя значения параметров между запусками и пытаясь отследить закономерности поведения модели, анализируя результаты каждого запуска по отдельности. Механизм автоматического запуска модели заданное количество раз с варьированием значений выбранных параметров (рис. 1.1) доступен в AnyLogic – это *эксперимент для варьирования параметров* [2]. При запуске данного эксперимента пользователь может изучить и сравнить поведение модели при разных значениях параметров с помощью графиков.

Чтобы запустить процесс варьирования параметров, нужно выполнить следующие шаги:

1. Создать эксперимент для варьирования параметров и сделать его текущим.

2. Сконфигурировать эксперимент, выбрав параметры, которые вы хотите варьировать, и задать значения, которые эти параметры должны будут принять за определенное вами количество прогонов модели.

3. Запустить модель.

Дадим формулировку *обратной задачи* имитационного моделирования. Чаще всего имитационное моделирование используется для того, чтобы оценить качество возможных решений по управлению сложной системой. *Решением* будем называть выбор одной альтернативы из некоторого множества рассматриваемых вариантов. *Рациональным решением* называется такое решение, которое максимизирует ожидаемую выгоду или полезность системы. Это решение всегда выбирается в соответствии с некоторым критерием. Иными словами, если мы хотим количественно оценить возможные альтернативы выбора, кроме множеств X и Y следует ввести *показатель эффективности.* Обозначим w показатель эффективности системы, который может быть подсчитан по вектору y выходных значений, и пусть W есть множество всех возможных оценок качества решения. Функция Φ из множества Y векторов результирующих показателей во множество W оценок качества модели дает оценку каждому возможному вектору результирующих показателей. Оценка вектора исходов может быть выполнена в самой имитационной модели, поэтому можно считать, что имитационная модель в детерминированном случае позволяет непосредственно вычислить значение показателя эффективности $w \in W$ каждого прогона модели. При этом функция Φ называется функцией полезности или целевой функцией.

Назовем вектор x тех значений факторов, которые определяют наиболее предпочтительный вариант решения, как $x_{опт}$. Поиск вектора $x_{опт}$ называется *обратной задачей имитационного моделирования.* Обратные задачи моделирования отвечают на вопрос:

«Какое решение из области допустимых решений обращает в максимум показатель эффективности системы?»

Для решения обратной задачи обычно многократно решается прямая задача. В случае, когда число возможных вариантов решений прямой задачи невелико, решение обратной задачи сводится к простому перебору всех возможных решений прямой задачи. Сравнивая между собой все полученные решения, можно найти одно или несколько наиболее предпочтительных решений (наборов исходных факторов x), для которых величина $\Phi(y)$ достигает максимума.

Если перебрать все варианты для получения искомого оптимального решения в сложной системе невозможно, используются методы направленного перебора с применением эвристик. При этом оптимальное или близкое к оптимальному решение находится после многократного выполнения последовательных шагов согласно схеме на рис.1.2. На каждом таком шаге решается прямая задача моделирования, то есть получение для каждого нового набора входных параметров модели вектора результирующих показателей. Правильно подобранная эвристика для выбора очередного варианта набора входных параметров приближает эксперимент к оптимальному решению на каждом шаге.

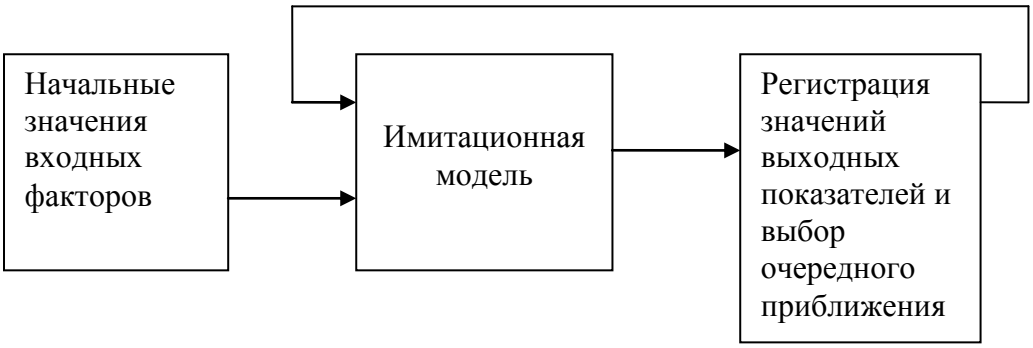

Рис.1.2. Имитационная модель при решении обратных детерминированных задач

Многие модели в бизнесе, науке и технике включают существенные нелинейности, комбинаторные зависимости и неопределенности, которые легко представимы в имитационных моделях, но являются слишком сложными для представления формальным математическим аппаратом, наборами математических формул. Это препятствует непосредственному применению классических методов оптимизации в таких системах. Для оптимизации систем, которые не могли быть формализованы в рамках этих математических моделей, использовался метод случайного поиска с эвристиками, которые обычно разрабатывались специально для каждой конкретной задачи.

Целью пакета OptQuest, встроенного в AnyLogic, как раз и является оптимизация систем, которые не могут быть представлены как математические модели и оптимизация в которых не может быть выполнена с помощью классических алгоритмов. В пакете реализованы современные мощные алгоритмы оптимизации [2, 10-13]. Пакет использует подход «черного ящика» для вычисления значений целевой функции: он устанавливает входные параметры и обращается к имитационной модели, которая по набору входных параметров возвращает значение целевой функции.

Глава 2. ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ В СРЕДЕ ANYLOGIC. ОСНОВНЫЕ ПОНЯТИЯ

Группа учёных из Санкт-Петербургского Политехнического университета разрабатывала программное обеспечение для имитационного моделирования. Акцент при разработке ставился на прикладные методы: моделирование стохастических систем, оптимизацию и визуализацию модели [2, 10]. Программное обеспечение, выпущенное в 2000 г., было основано на последних преимуществах информационных технологий: объектно-ориентированный подход, элементы стандарта UML, языка программирования Java, современного GUI и т. д.

Рис. 2.1. Три подхода имитационного моделирования

Продукт получил название AnyLogic 4.0, потому что он поддерживал все три известных метода моделирования:

– системная динамика;
– дискретно-событийное (процессное) моделирование;
– агентное моделирование,
– любая комбинация этих подходов в пределах одной модели.

Огромный шаг вперёд был сделан в 2003 году, когда был выпущен AnyLogic 5, ориентированный на бизнес-моделирование. С помощью AnyLogic стало возможным разрабатывать модели в следующих областях:

– производство;
– логистика и цепочки поставок;

– рынок и конкуренция;
– бизнес-процессы и сфера обслуживания;
– здравоохранение и фармацевтика;
– управление активами и проектами;
– телекоммуникации и информационные системы;
– социальные и экологические системы;
– пешеходная динамика;
– оборона.

В марте 2010 года была выпущена версия программы AnyLogic 6.5. AnyLogic 6 написан на языке программирования Java в популярной среде разработки Eclipse. Anylogic 6 является кросс-платформенным программным обеспечением, работает как под управлением операционной системы Windows, так и под Mac OS и Linux.

2.1. Методы имитационного моделирования

Модели AnyLogic могут быть основаны на любой из основных парадигм имитационного моделирования: дискретно-событийное моделирование, системная динамика и агентное моделирование.

При разработке моделей в AnyLogic можно использовать концепции и средства из нескольких методов моделирования, например, в агентной модели использовать методы системной динамики для представления изменений состояния среды или в непрерывной модели динамической системы учесть дискретные события. Например, управление цепочками поставок при помощи имитационного моделирования требует описания участников цепи поставок агентами: производители, продавцы, потребители, сеть складов. При этом производство описывается в рамках дискретно-событийного (процессного) моделирования, где продукт или его части – это заявки, а автомобили, поезда, штабелёры – ресурсы. Сами поставки представляются дискретными событиями, но при этом спрос на товары может описываться непрерывной системно-динамической диаграммой. Возможность смешивать подходы позволяет описывать процессы реальной жизни, а не подгонять процесс под доступный математический аппарат.

Графическая среда моделирования AnyLogic включает в себя следующие элементы:

- ***Stock & Flow Diagrams*** (диаграмма потоков и накопителей) применяется при разработке моделей, используя метод системной динамики.
- ***Statecharts*** (карты состояний) в основном используется в агентных моделях для определения поведения агентов. Но также часто используется в дискретно-событийном моделировании, например для симуляции машинных сбоев.
- ***Action charts*** (блок-схемы) используется для построения алгоритмов. Применяется в дискретно-событийном моделировании (маршрутизация звонков) и агентном моделировании (для логики решений агента).

- ***Process flowcharts*** (процессные диаграммы) основная конструкция, используемая для определения процессов в дискретно-событийном моделировании.

Среда моделирования также включает в себя: низкоуровневые конструкции моделирования (переменные, уравнения, параметры, события и т.п.), формы представления (линии, квадраты, овалы и т.п.), элементы анализа (базы данных, гистограммы, графики), стандартные картинки и формы экспериментов.

Среда моделирования AnyLogic поддерживает проектирование, разработку, документирование модели, выполнение компьютерных экспериментов с моделью, включая различные виды анализа – от анализа чувствительности до оптимизации параметров модели относительно некоторого критерия.

AnyLogic включает в себя набор следующих стандартных библиотек:

- ***Enterprise Library*** разработана для поддержки дискретно-событийного моделирования в таких областях как производство, цепи поставок, логистика и здравоохранение. Используя Enterprise Library, Вы можете смоделировать системы реального мира с точки зрения заявок (англ. *entity*) (сделок, клиентов, продуктов, транспортных средств, и т. д.), процессов (последовательности операций, очередей, задержек) и ресурсов. Процессы определены в форме блочной диаграммы.
- ***Pedestrian Library*** создана для моделирования пешеходных потоков в "физической" окружающей среде. Это позволяет Вам создавать модели с большим количеством пешеходного трафика (как станции метро, проверки безопасности, улицы и т. д.). Модели поддерживают учёт статистики плотности движения в различных областях. Это гарантирует приемлемую работу пунктов обслуживания с ограничениями по загруженности, оценивает длину простаивания в определённых областях, и обнаруживает потенциальные проблемы с внутренней геометрией – такие как эффект добавления слишком большого числа препятствий – и другими явлениями. В моделях, созданных с помощью Pedestrian Library, пешеходы двигаются непрерывно, реагируя на различные виды препятствий (стены, различные виды областей) так же как и обычные пешеходы. Пешеходы моделируются как взаимодействующие агенты со сложным поведением. Для быстрого описания потоков пешеходов Pedestrian Library обеспечивает высокоуровневый интерфейс в виде блочной диаграммы.
- ***Rail Yard Library*** поддерживает моделирование, имитацию и визуализацию операций сортировочной станции любой сложности и масштаба. Модели сортировочной станции могут использовать комбинированные методы моделирования (дискретно-событийное и агентное моделирование), связанные с действиями при транспортировке: погрузками и разгрузками, распределением ресурсов, обслуживанием, различными бизнес-процессами.

2.2. Основные возможности и средства имитационного моделирования в среде AnyLogic

2.2.1. Две фазы имитационного моделирования

В AnyLogic две фазы имитационного моделирования: разработка модели и ее анализ явно разделены. Разработка модели выполняется в среде редактора AnyLogic, а анализ модели происходит в среде исполнения. В каждой фазе существуют свои средства управления.

Переход из одной фазы в другую производится очень легко. Можно многократно использовать переход между фазами редактирования и исполнения модели при разработке модели.

2.2.2. Активные объекты, классы и экземпляры активных объектов

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.

В AnyLogic основным структурным блоком при создании моделей являются классы *активных объектов*. Использование активных объектов является естественным средством структуризации модели сложных систем: мир состоит из множества параллельно функционирующих и взаимодействующих между собой сущностей. Различные типы этих сущностей и представляют разные активные объекты.

Чтобы создать модель AnyLogic, нужно создать классы активных объектов (или использовать объекты библиотек AnyLogic). Определение активного объекта задает шаблон, и отдельные объекты, построенные в соответствии с этим шаблоном (экземпляры активного объекта), могут использоваться затем как элементы других активных объектов. Так в модели Balls в [2] определены два класса активных объектов: Ball и Root. В класс Root были включены два экземпляра класса Ball с именем *ball* и *ball1*, которые различались своими параметрами (начальными значениями координат x и y). Всегда один класс в модели является корневым. Для него в модели AnyLogic порождается один экземпляр с предопределенным именем *root*, он и запускается исполнительной системой AnyLogic на выполнение. Имя класса корневого активного объекта можно менять в окне его свойств.

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, он функционирует независимо от других объектов, взаимодействуя с ними и с внешней средой.

2.2.3. Объектно-ориентированный подход

AnyLogic использует объектно-ориентированный подход к представлению сложных систем. Этот подход позволяет простым и естественным образом организовать и представить структуру сложной системы с помощью иерархии абстракций. Например, на некотором уровне абстракции автомобиль можно считать неким единым объектом. Но более детально его можно представить как совокупность взаимодействующих подсистем: двигателя, рулевого управления, тормозной системы и т. п. Каждая из этих подсистем может быть представлена, если это необходимо, своей структурой взаимодействующих подсистем.

Именно такую иерархию абстракций позволяет создать AnyLogic при разработке моделей. В нашем примере *ball1* всю модель можно рассматривать как единый объект root. При более детальном рассмотрении видно, что этот объект содержит два экземпляра класса Ball с именами *ball* и *ball1*. Сам класс Ball может иметь свое сложное строение, которое будет скрыто в классе Root. Такая иерархия структуры может быть произвольной глубины.

2.2.4. Визуальная разработка модели

При построении модели Balls не использовано никаких других средств, кроме средств 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 парадигмой, принятой в AnyLogic при разработке моделей, является *визуальное проектирование* – построение с помощью графических объектов и пиктограмм иерархий структуры и поведения активных объектов.

2.2.5. Встроенный язык Java

AnyLogic является надстройкой над языком Java – одним из самых мощных и в то же время простых современных объектно-ориентированных языков. Все объекты, определенные пользователем при разработке модели на AnyLogic с помощью его графического редактора, компилируются в конструкции языка Java, а затем происходит компиляция всей собранной программы на Java, задающей модель, в исполняемый код.

Хотя при построении модели на AnyLogic разработчик использует конструкции языка Java в большей или меньшей степени, в действительности он никогда не разрабатывает полные программы, он не программирует, а лишь вставляет фрагменты кода в специально предусмотренные для этого поля окна **Код** и окон свойств объектов модели. Эти фрагменты выражают логику конкретных шагов или действий в модели. Например, событие касания мячом земли записано в модели Balls так:

$$y <= r \;\&\&\; vy < O$$

что является просто логическим выражением языка Java. Как и все другие включенные в модель программные фрагменты, это выражение должно быть

синтаксически правильной конструкцией Java, потому разработчик моделей AnyLogic должен иметь некоторое представление об этом языке.

2.2.6. Средства описания поведения объектов

Основным средством спецификации поведения объектов в AnyLogic являются переменные, таймеры и стейтчарты.

Переменные отражают изменяющиеся характеристики объекта.

Таймеры можно взводить на определенный интервал времени и по окончании этого интервала выполнять заданное действие.

Стейтчарты позволяют визуально представить поведение объекта во времени под воздействием событий или условий, они состоят из графического изображения состояний и переходов между ними.

Любая сложная логика поведения объектов модели в AnyLogic может быть выражена с помощью комбинации стейтчартов, дифференциальных и алгебраических уравнений, переменных, таймеров и программного кода на Java. Алгебраические и дифференциальные уравнения, как и логические выражения, записываются в AnyLogic аналитически.

2.2.7. Имитация нескольких параллельно протекающих процессов

Интерпретация любого числа параллельно протекающих процессов в модели AnyLogic скрыта от пользователя. Никаких календарей событий разработчик модели на AnyLogic не ведет, отслеживание событий во всех процессах, определенных в модели, выполняется системой автоматически. В модели Balls интерпретировалось поведение двух мячей – экземпляров активного объекта ball, каждый со своим поведением во времени. Никаких усилий разработчика для организации квазипараллелизма интерпретации при этом не требовалось.

2.2.8. 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 обращением к функции *getTime*.

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 одной и той же системе физических величин. Например, в модели

Balls все физические величины выражены в системе Си, в которой единица времени – секунда, а единица длины – метр.

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, конечно, единица физического 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. Этот режим используется для факторного анализа модели, набора статистики, оптимизации параметров модели и т. п. Поскольку анимация и другие окна наблюдения за поведением модели обычно существенно замедляют скорость интерпретации модели на компьютере, для повышения скорости выполнения модели эти окна нужно закрыть.

В *режиме реального времени* пользователь задает связь модельного времени с физическим временем, то есть устанавливается 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 в режиме реального времени можно понять на таком примере. При коэффициенте ускорения 4, если процессор успевает выполнить менее чем за 1 секунду все операции, которые в модели определены в течение четырех единиц модельного времени, он будет ждать до конца секунды. Если же процессор не успевает сделать это действие, то у него не будет интервала ожидания, и коэффициент ускорения будет меньше того, который установлен пользователем.

2.2.9. Анимация поведения модели

Удобные средства разработки анимационного представления модели в Any Logic позволяют представить функционирование моделируемой системы в живой форме динамической анимации, что позволяет увидеть поведение сложной системы. Средства анимации позволяют пользователю легко создать виртуальный мир (совокупность графических образов, ожившую мнемосхему и т. п.), управляемый динамическими параметрами модели по законам, определенным пользователем с помощью уравнений и логики моделируемых объектов. Визуальное представление поведения системы помогает пользователю проникнуть в суть процессов, происходящих в системе.

2.2.10. Интерактивный анализ модели

Многие системы моделирования позволяют менять параметры модели только до запуска модели на выполнение. Any Logic позволяет пользователю вмешиваться в работу модели, изменяя параметры модели в процессе ее функционирования. Поэтому окно анимации можно назвать «стендом» для проведения компьютерного эксперимента с моделью. Примером таких средств являются слайдеры модели Balls.

2.3. Дискретно-событийное моделирование в AnyLogic

Термин «дискретно-событийное моделирование» закрепился за моделированием систем обслуживания потоков объектов.

Базовые средства AnyLogic для построения моделей дискретно-событийных систем могут быть использованы в широком диапазоне совершенно различных приложений имитационного моделирования. Существует, однако, область приложений дискретно-событийного моделирования, в которой единая элегантная парадигма позволяет применить, фактически, одну и ту же методологию к решению множества важных практических проблем. Эта область – массовое обслуживание. Традиционным подходом к моделированию задач массового обслуживания является разработка библиотеки многократно используемых объектов, из которых как дома при крупнопанельном строительстве могут быть собраны совершенно различные модели, решающие разнообразные задачи этой области. В AnyLogic такой традиционный путь решения класса задач реализуется очень просто: создается библиотека типовых блоков, собирая которые в связанные структуры и настраивая их параметры можно значительно ускорить разработку моделей этого класса. Нужно помнить при этом, что все базовые средства AnyLogic могут быть использованы в случае, если функциональности библиотечных объектов недостаточно для решения специфических проблем.

Средства AnyLogic, поддерживают моделирование дискретно-событийных систем. Эти средства включают сущности, порождающие события, и сущности, позволяющие изменять состояние системы как реакцию на наступившие события.

Основными средствами порождения событий в модели являются таймеры и стейтчарты (statecharts). Кроме базовых средств разработки дискретно-событийных систем, пользователю доступна библиотека Enterprise Library, в которой собраны высокоуровневые средства – блоки, позволяющие осуществлять моделирование широкого класса дискретно событийных систем массового обслуживания без использования программного кода на основе применения заранее определенных блоков библиотеки.

Библиотечные блоки значительно упрощают разработку указанных моделей в стиле drag and drop (перетащить и оставить).

Библиотечные элементы в моделях могут быть дополнены низкоуровневыми средствами, которые включают:

– стейтчарты (карты состояний), позволяющие графически описать сложное дискретное поведение;

– таймеры и события, позволяющие выразить изменение поведения объекта во времени или наступлении некоторого события;

– возможности определения своих собственных типов транзакций (сообщений) для взаимодействия активных объектов;

– механизм обмена активных объектов сообщениями через порты.

Стейтчарты – это средство визуального задания сложного поведения объектов, включающее иерархические состояния, разветвления (условные переходы), исторические состояния т. п. Дискретно-событийные системы меняют свое состояние мгновенно, под влиянием событий. Переходы могут быть активизированы полученными сообщениями, исчерпанием таймаута, возникновением событий и условиями. Сообщения могут быть произвольных типов, они посылаются и принимаются через порты.

В AnyLogic могут использоваться статические и динамические таймеры.

В дискретно-событийных моделях необходимо включение стохастики в модель. Для этого в AnyLogic предусмотрены более 35 различных генераторов распределенных случайных величин.

AnyLogic позволяет проведение экспериментов с моделью, выполнение анализа рисков, оптимизацию параметров модели.

Библиотека Enterprise Library содержит несколько десятков блоков с предопределенной функциональностью, полное описание которых содержится в справочных материалах [2, 10-13.]. Основные группы блоков:

– блоки управления потоком заявок;
– блоки классических ресурсов;
– блоки обработки;
– блоки транспортировки;
– блоки транспортировки по сети.

Блоки для управления потоком заявок имеют желтый цвет. К ним относятся: источник заявок (*Source*), сток (*Sink*), очередь заявок (*Queue*), разветвитель потоков (*Split*), сортировка заявок *SelectOutput* и др. Данная группа блоков удобна для создания моделей систем, относящихся к системам массового обслуживания.

Группа блоков классических ресурсов имеют зеленый цвет и включает в себя блоки *Resource*, *ProcessQ* и др. Объект *Resource* может создавать, хранить, выдавать и забирать ресурсы. Объект *ProcessQ* занимает ресурсы для заявки, задерживает заявку, а затем освобождает занятые ей ресурсы.

Группа блоков обработки имеют оранжевый цвет и включает два блока: задержку (*Delay*) и сервер (*Server*). Блок задержки удерживает каждую поступившую на его вход заявку на определенное время. Число возможных заявок, которые одновременно и независимо могут быть задержаны в одном блоке, определено параметром *capacity*.Время задержки может задаваться как реализация случайной величины, распределенной по некоторому закону. Блок задержки моделирует независимую обработку заявок, то есть время задержки каждой заявки в блоке не зависит от числа задержанных заявок, находящихся в блоке одновременно.

Сервер должен обеспечить такой режим, при котором каждая заявка получит определенное обслуживание. При этом на сервере на обслуживании могут находиться несколько заявок, поэтому время сервера делится между всеми заявками, одновременно находящимися в обработке.

Группа блоков транспортировки по сети предназначены для имитации заявок, время обслуживания которых в системе зависит от геометрических характеристик системы. Подобными особенностями обладают не только организации социального обслуживания населения, например, больницы, банки, но и производственные системы с конвейерами, складские системы и т. п.

В приложении представлены описания блоков, наиболее часто встречающихся при решении задач дискретно-событийного моделирования, приведенных в главе 3.

2.4. Заключение

Система моделирования AnyLogic обеспечивает поддержку всех этапов имитационного моделирования для различных типов динамических моделей – дискретных, непрерывных и гибридных, детерминированных и стохастических в любых их комбинациях в рамках одного инструмента.

Создание модели, ее выполнение, оптимизация параметров, анализ полученных результатов, верификация модели – все эти этапы удобно выполнять в среде AnyLogic. Данный инструмент обладает большим спектром разнообразных возможностей проведения как отдельных прямых экспериментов типа «*если-то»,* так и серий подобных экспериментов для решения всевозможных обратных задач, направленных на поиск параметров модели, оптимизирующих ее функционирование.

Удобный интерфейс и различные средства поддержки разработки в AnyLogic делают не только использование, но и создание компьютерных имитационных моделей в этой среде моделирования доступными даже для тех, кто в области вычислительной техники и программирования не является профессионалом.

Глава 3. ПРИМЕРЫ МОДЕЛЕЙ В СРЕДЕ ANYLOGIC

3.1. Модель дискретно-событийных систем

Рассмотрим типичную проблему массового обслуживания и то, как эта проблема может быть легко решена с использованием одной из библиотек, разработанных в AnyLogic [2, 10-13]. Системы массового обслуживания являются типичными системами дискретно-событийного типа. Среда AnyLogic открыта для разработки библиотек совершенно различного назначения.

Из-за однотипности, похожести задач, решаемых моделями систем массового обслуживания, удобно отдельные блоки (генераторы заявок, обслуживающие приборы, очереди и т. п.) реализовать как набор (библиотеку) объектов, из которых может собираться структура конкретной модели и параметры которых можно настраивать в зависимости от характеристик моделируемой системы.

Именно для этих целей в AnyLogic создана библиотека Enterprise Library. Она предоставляет высокоуровневый интерфейс для быстрого создания дискретно-событийных моделей с помощью блок-схем. Построим с помощью элементов библиотеки модель системы, предоставляющей сервисы – операционный зал банка.

Рассмотрим простую систему, предоставляющую сервисное обслуживание – операционный зал банка. В банке есть два менеджера, отвечающие за два различных типа операций: выдачу инвестиций и работу со счетом. К менеджерам в очереди стоят посетители. После обслуживания менеджером каждый клиент идет в кассу, получая либо сдавая деньги. Очередь в кассу является общей.

Цель моделирования такой системы может быть разной. Однако наиболее типичной целью исследования в подобных задачах массового обслуживания является *оценка эффективности системы,* т. е. нахождение числовых значений характеристик, описывающих качество обслуживания системой потока посетителей. Такими характеристиками является время, проведенное клиентом в банке, длина очереди, которую он отстоял, процент времени занятости обслуживающего персонала. Для поддержки принятия управленческих решений важно также уметь решать обратные задачи анализа: например, определять минимальное количество обслуживающего персонала при ограниченной средней длине очереди клиентов.

Клиенты приходят в банк обычно в случайные моменты времени. У каждого клиента свои вопросы по своему счету или по будущему кредиту, поэтому время обслуживания тоже случайно. Оплата в кассе занимает случайное время, поскольку один посетитель может приготовить точную сумму, а другому нужно дать сдачу. Операционный зал банка является типичной *системой массового обслуживания* (СМО). Такую систему можно представить моделью с небольшим числом типов абстрактных объектов: клиенты представляются заявками на обслуживание, а объекты, выполняющие обслуживание (менеджеры, кассиры), представляются приборами, обрабатывающими заявки. Объекты типа «*Очередь»* имитируют ожидание без обработки.

Структура имитационной модели, которая даст ответ на поставленные вопросы, должна отражать структуру реальной системы массового обслуживания: заявки (клиенты банка) генерируются (входят в систему), становятся в очереди к обслуживающим приборам, а после полного обслуживания покидают систему. Характерной особенностью СМО является стохастическая природа описывающих эти системы характеристик. Структура модели операционного зала банка в данных терминах имеет вид рис. 3.1.

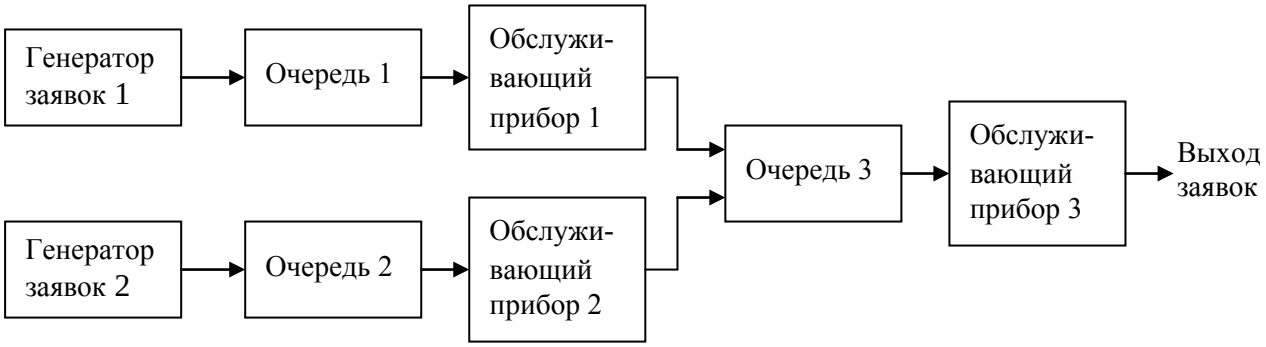

Рис. 3.1. Структура модели операционного зала банка

Рисунок 3.1 показывает структуру модели операционного зала банка на том уровне абстракции, который достаточен для ответа на подобные вопросы. Блоки, представленные на рис. 3.1, часто используются для моделирования систем массового обслуживания. Все такие блоки есть в библиотеке Enterprise Library.

Постройте модель операционного зала банка. Создайте новый проект, назовите его *QueuingModelLibrary*, откройте библиотеку Enterprise Library и в окно структуры корневого активного объекта методом drag-and-drop внесите из библиотеки следующие элементы, представленные на рис. 3.2. Она будет, фактически, повторять рис. 3.1. Эта структура будет собираться из элементов библиотеки AnyLogic, содержащей типовые блоки: генераторы заявок, очереди, задержки, блок стока (см. рис П1- рис. П4).

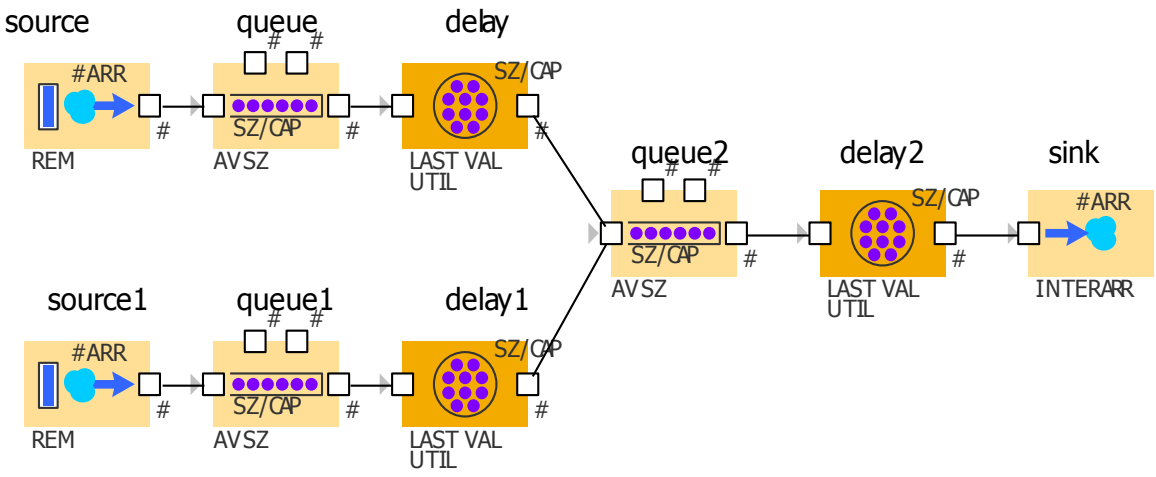

Рис. 3.2. Структура модели банка в среде AnyLogic

Обслуживающий прибор может быть представлен просто как блок задержки. Блоки *serverA, serverB и serverC* – это экземпляры блока Delay библиотеки. Соедините блоки нужным образом и запустите модель на выполнение. (Соединяются блоки библиотеки так же, как интерфейсные переменные или порты.)

Задача моделирования при наличии такой библиотеки сводится к набору структуры системы из стандартных блоков, настройке параметров блоков и запуску модели на выполнение, что требует для разработки описанной ранее модели нескольких минут. Никаких знаний о методах построения датчиков случайных чисел, об организации списка событий, программирования для разработки и анализа таких систем не нужно.

Настроим параметры включенных в модель блоков из библиотеки. Пусть в нашем примере в соответствии с измерениями (или предположениями) поток клиентов к первому менеджеру является случайным, интервалы времени между клиентами в нем распределены в соответствии с экспоненциальным законом распределения вероятности со средним 16,5 мин., поток ко второму менеджеру тоже подчиняется экспоненциальному закону со средним интервалом времени между приходами клиентов 12,5 мин.

Выделите блок *source*, и в окне свойств этого объекта в строчке параметра *interarrivalTime* запишите *exponential(1/16.5)*. Заметим, что параметром функции *exponential* (которая при обращении к ней генерирует реализацию случайной величины с экспоненциальным законом распределения вероятности) является интенсивность потока событий, обратная среднему времени между событиями. Остальные параметры этого объекта не изменяйте.

Аналогично измените значение параметра *interarrivalTime* у другого генератора потока заявок *source1*.

Пусть время обработки заявок обоими менеджерами и кассиром тоже случайно, и пусть оно распределено по экспоненциальному закону со средними значениями 15, 10 и 6,5. Введите соответствующие значения параметра *delayTime* у экземпляров *delay, delay1 и delay2*.

Запустите модель на выполнение. Целью моделирования подобной сервисной системы является, конечно, не просто имитация функционирования его служащих и поведения клиентов, а определение тех параметров, которые характеризуют качество сервиса и затраты на обеспечение этого сервиса при определенных характеристиках входного потока клиентов и структуры системы. В частности, средней длины очередей, занятость обслуживающего персонала и т. п.

В окне *root* модели банка можно наблюдать показатели работы. Например, после обработки 32 тысяч заявок получено, что 14032 заявки прошли через первого менеджера (*delay*), 18 364 заявки – через второго менеджера (*delay1*) и 32336 заявок обработано кассиром. Далее показано, что в данный момент времени до генерации очередной заявки объекту *source* осталось 10,635 мин., а объекту *source1* – 0,366 мин., в очереди к первому менеджеру стоит одна заявка, ко второму менеджеру – 7 заявок, а к кассиру – 60 заявок.

Блоки библиотеки Enterprise Library имеют и другие встроенные средства сбора статистической информации, однако по умолчанию сбор части статистики отключен для повышения производительности модели. В окне свойств блоков *queue, queue1* и *queue2,* а также *delay, delay1* и *delay2* в поле значений параметра *statsEnabled* вместо *false* вставьте из выпадающего меню значение *true*. Запустите систему на выполнение опять.

Если для экземпляров активных объектов *Queue* и *Delay* включен сбор статистики, то в системе собирается статистика по длинам очередей для *Queue* и статистика по коэффициенту использования (доля времени занятости) для блоков *Delay*, представляющих в нашей модели менеджеров. В системе обработано более 34 тысячи заявок. Видно, что средняя длина очереди *queue* равна 9,968, очереди *queuel* равна 3,963, а очереди *queue2* равна 12,437. Коэффициент использования двух менеджеров и кассира соответственно 0,914, 0,795 и 0,918.

На основе этого анализа можно с уверенностью сказать, что при указанных ранее входных потоках клиентов и производительности обслуживания зафиксированная в модели структура операционного зала банка неудовлетворительна. Первый менеджер (которого моделирует блок *delay*) и кассир (блок *delay2*) перегружены, средняя длина очередей к ним 10–12 человек, поэтому клиенты банка будут не удовлетворены.

3.2. Пример модели оптимизационного эксперимента

Имитационное моделирование часто используется для того, чтобы оценить качество возможных решений по управлению сложной системой и нахождению среди них оптимального решения.

В качестве блока регистрации значений выходных показателей и выбора очередного приближения при оптимизации пользователь AnyLogic может использовать любой внешний оптимизатор или же использовать оптимизатор OptQuest, встроенный в AnyLogic. Оптимизатор OptQuest разработан недавно на основе метаэвристик рассеянного поиска (scatter search) и поиска "табу" (tabu search) [2]. Этот оптимизатор является лучшим из предлагаемых сегодня на рынке профессиональных пакетов оптимизации для решения сложных проблем оптимизации.

OptQuest позволяет решить обратную задачу моделирования: он автоматически находит лучшие значения параметров имитационной модели, соответствующие максимуму или минимуму целевой функции как в условиях неопределенности, так и при наличии ограничений. Оптимизатор OptQuest запускается прямо из среды разработки модели. Рассмотрим использование этого оптимизатора на примере.

Рассмотрим систему предоставления услуг мобильной связи с помощью автоматической телефонной станции, на которую поступают вызовы. Поставщик сервиса выбирает оборудование автоматической телефонной станции, он оценивает параметры потока входных запросов (как часто люди будут звонить) и характеристики потока обслуживания (сколько времени требуется для получения консультации по телефону). Проблема состоит в определении числа каналов,

которые дадут ему максимальную прибыль. Введение дополнительных каналов связи приведет к росту доходов вследствие уменьшения числа необслуженных вызовов, но потребует дополнительных вложений. Какое число каналов выбрать поставщику сервиса?

Будем считать, что за каждый отвергнутый вызов платится штраф (усредненные потери от неудовлетворенности клиента), а обслуженные вызовы приносят доход. Имея стоимость поддерживаемых каналов связи, можно в качестве целевой функции в этой модели выбрать прибыль, определяемую как разность доходов за обслуживание телефонных вызовов и расходов на оборудование, обслуживание станции и штрафы. Прибыль будем подсчитывать в среднем за единицу времени, например за минуту.

Чтобы настроить оптимизацию в AnyLogic, нужно в общем случае выполнить следующие шаги:

1. Создать в разработанной модели оптимизационный эксперимент.
2. Задать оптимизационные параметры (компоненты изменяемого вектора исходных факторов x) 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, значение w которой отражает предпочтительность вектора исходных факторов x. Значение w должно быть доступно в конце каждого прогона модели, оно будет использоваться оптимизатором.
5. Задать ограничения, которые в конце каждого прогона определяют, допустимо ли значение вектора исходных факторов x. Ограничения можно не задавать (т. е. это опционально).
6. Задать условия прекращения оптимизации.

На рис. 3.3 представлена структура имитационной модели, с помощью которой решается эта проблема. Блок *"Генератор заявок"* имитирует приход вызовов. Блок *"Обслуживающий прибор"* в данном случае может обслужить вызов, только если есть свободный ресурс (канал соединения). Блок *"Ресурсы"* имитирует наличие ограниченного числа ресурсов, это число можно задать параметром. Блок *"Анализ"* направляет поступившие вызовы либо на блок *"О*бслуживающий прибор", если есть свободный ресурс (канал), либо на выход из системы при отсутствии свободного канала. Все блоки рис. 3.3 также есть в библиотеке AnyLogic, поэтому построение имитационной модели из таких блоков с настройкой их параметров займет у разработчика всего несколько минут.

Для того чтобы определить зависимость прибыли от числа N каналов, нужно подсчитать стоимость покупки станции и содержания каналов, при моделировании подсчитать доход за все обслуженные вызовы в течение некоторого времени и штраф за все необслуженные вызовы за это время.

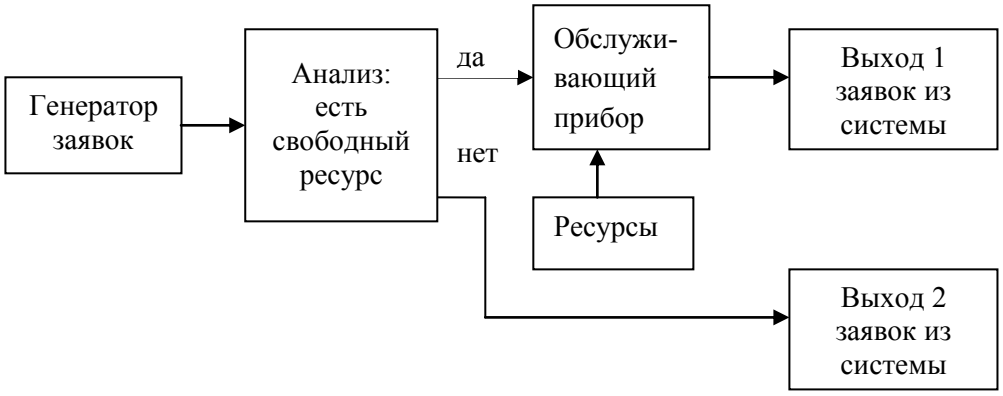

Рис. 3.3. Структура модели оценки прибыли при предоставлении сервиса мобильной связи

Изменяя параметр *N* имитационной модели, можно для каждого его значения получить возможную прибыль и выбрать наилучшее, оптимальное значение параметра. В AnyLogic встроен оптимизатор OptQuest – широко известный алгоритм поиска оптимума неаналитических функций.

Откройте новый проект, назовите его *MobileCommunication*. Из Enterprise Library с помощью мыши перенесите блоки структуры системы в поле редактора структуры модели и соедините их в соответствии с рис. 3.3. Нам нужен один источник заявок (блок *source*), блок, решающий, отказать или нет пришедшей заявке в обслуживании в соответствии с некоторым условием (*selectOutput*), блок обработки входных заявок с использованием ресурсов – в нашем случае каналов связи (*processQ*), блок ресурсов (*resource*) и два блока стока (*sink* и *sink1*) (рис. 3.4).

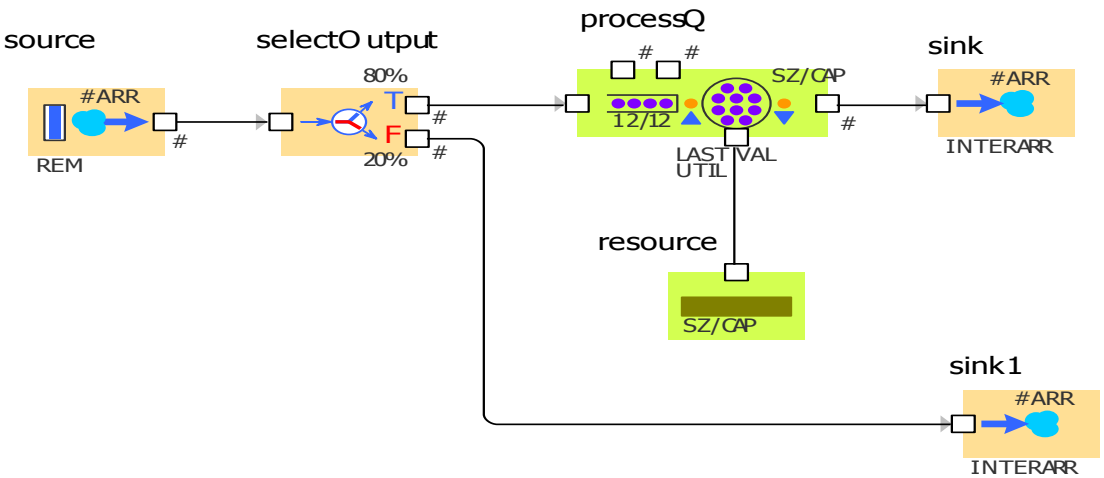

Рис. 3.4. Структура модели для решения проблемы мобильной связи

Очевидно, что поток поступающих заявок и поток обслуживания имеют вероятностную природу. Для простоты будем считать, что они имеют экспоненциальный закон распределения. Введем в модель три параметра: целый *N*

(число линий связи) и два вещественных *lambda* (интенсивность поступления заявок) и *mu* (интенсивность их обслуживания каждым каналом). Установим начальные значения: *N*=3, *lambda*=l,5, *mu*=0,5.

Для определения характеристик эффективности системы установим следующие параметры для блоков системы.

Блок *source***.** У параметра *interarrivaiTime* оставим экспоненциальное распределение, но установим у этого распределения параметр *lambda*.

Блок *selectOutput* посылает принятые заявки на свой выход при условии истинности параметра *selectCondition*. В нашей модели это условие должно быть истинно только в том случае, когда процессор *processQ* имеет свободные линии. Число обрабатываемых в этом блоке заявок можно получить, обратившись к функции *sizeDelay( )*. Итак, в поле параметра *selectCondition* блока *selectOutput* нужно установить *processQ.sizeDelay()<N*.

Блок *processQ*. Здесь параметр *delayTime* установите как *exponential* (mu). Остальные параметры оставьте неизмененными. В частности, параметр *quantity* определяет число ресурсов на одну входную заявку, его значение уже установлено равным 1, параметр *queueCapacity* определяет длину очереди в процессоре, но у нас длина очереди не будет влиять на работу блока (мы определили, что в блок приходит заявка, если только для нее есть ресурс).

Блок *resource* требует установки только одного параметра *capacity*, установите для него значение *N*.

Запустите модель в режиме виртуального времени. Через несколько секунд некоторые из требуемых характеристик уже появятся в анимированном окне структуры.

Эксперимент с моделью *ErlangProblem1* из папки Model ExampIes\Part III [2] показывает, что в данном примере всего пришло 39425 вызовов, 65 % из них обслужено, остальным отказано. Среднее время между обслуженными заявками 1,026. Таким образом:

– абсолютная пропускная способность системы, т. е. среднее число заявок, обслуженных в единицу времени равно 1/1,026 = 0,975;

– относительная пропускная способность системы, т. е. средняя доля пришедших заявок, обслуживаемых системой получилось 65 %;

– вероятность отказа заявке в обслуживании будет 34 %.

Однако статистические характеристики системы играют лишь вспомогательную роль при анализе систем. Действительной целью моделирования является анализ возможной прибыли при различных значениях вектора входных параметров системы. В нашей задаче единственным таким параметром является N – число поддерживаемых каналов связи.

Прибыль, полученную владельцем АТС, можно вычислить как доход за обслуживание вызовов минус штраф за отвергнутые вызовы и расходы на оборудование:

прибыль = доход – штраф – стоимостьОборудования.

Увеличение числа каналов, обеспечивающих связь, уменьшает штраф и повышает доход, но при этом растут и расходы на оборудование. Варьируя число

каналов при конкретных значениях остальных параметров системы, можно найти то оптимальное число каналов, которое принесет максимальную прибыль. Именно значение прибыли в этой задаче играет роль значения w функции полезности.

Ясно, что аналитически эту проблему не решить, так как все функции здесь неаналитические. Поскольку доход и штрафы можно подсчитать с помощью имитационной модели, то вся проблема является типичной проблемой имитационного моделирования. Для нахождения оптимального числа каналов N, максимальную прибыль, можно перебрать все значения N от 1 и так далее, но можно использовать эксперимент с оптимизацией.

Прибегнем к оптимизации, использующей модель *ErlangProblem2* в папке Model Examples\Part III [2].

Сделайте этот эксперимент текущим, выбрав соответствующую команду в контекстном меню объекта (его название в этом случае будет изображаться жирным шрифтом). Запустите модель. Оптимизационный эксперимент найдет наилучшее число каналов оптимизации $N = 31$, максимальная прибыль поставщика сервиса в этом случае будет 0,376 у.е. в минуту.

3.3. Модель динамических систем

Динамические системы – это сложные объекты, поведение которых описывается системами алгебраических и дифференциальных уравнений, а также событиями, меняющими либо среду, либо модель, либо даже саму структуру системы. К этому классу относятся системы управления, системы обработки сигналов, а также физические объекты, объекты химической технологии и т. п.

Рассмотрим в качестве примера классическую динамическую систему, состоящую из двух связанных подсистем: объекта управления и регулятора (рис. 3.5).

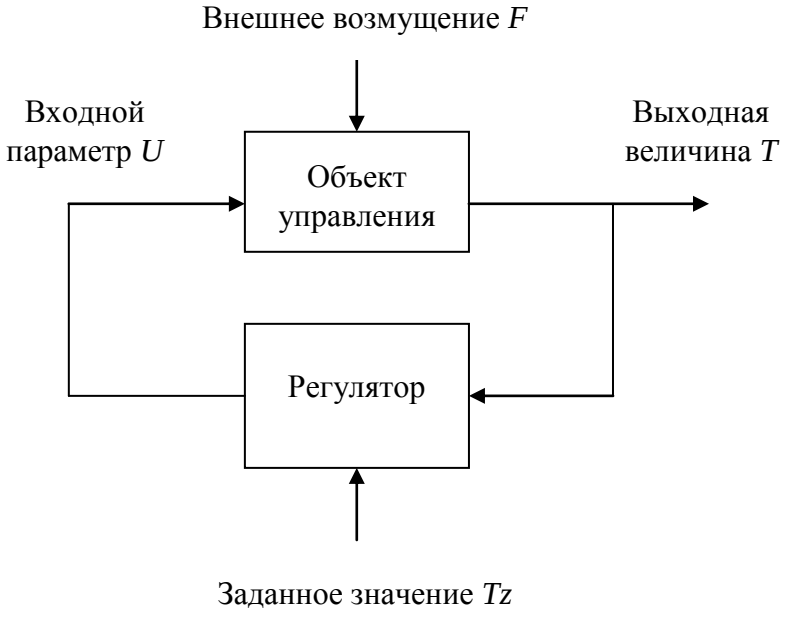

Рис. 3.5. Структура системы регулирования

Пусть *объектом управления* является бойлер, который нагревается до температуры *T*. Величину температуры *T* необходимо поддерживать на заданном уровне *Tz*. Температура бойлера зависит от входного параметра – в нашем случае от мощности тока *U*, подаваемого на нагревательный элемент бойлера. Объект управления подвергается внешнему возмущению *F* (*F* может характеризовать внешнюю температуру или теплоизоляцию), вследствие чего значение выходного параметра *T* может меняться. Поддержание значения *T* на заданном уровне *Tz* есть задача регулятора. *Регулятор* по разнице заданного и текущего значений выходного параметра (*Tz* – *T*) формирует величину входного параметра объекта управления – в нашем случае регулируется мощность *U*, подаваемая на нагревательный элемент.

Выходная величина *T* простого объекта управления описывается уравнением:

$$dT/dt = (kU - F - T)/a,$$

где *a* – коэффициент пропорциональности, определяемый параметрами объекта управления (выберем в нашем примере его значение равным 10);

k – коэффициент усиления объекта по управлению (установим его значение равным 1);

F – изменяющееся внешнее воздействие (температура окружающей среды или теплоизоляция).

Будем использовать в нашем примере так называемый регулятор с пропорционально-интегральным законом управления (ПИ-регулятор). Выходную величину (управление) регулятора определим так:

$$U = Ui + Kp(Tz - T),$$

где *U* – управление, которое состоит из интегральной и пропорциональной части;

Ui – интегральная составляющая управления;

Kp – коэффициент при пропорциональной составляющей управления (установим его значение равным 1).

Интегральная составляющая управления может быть задана так:

$$dUi/dt = Ki(Tz - T),$$

где *Ki* – коэффициент при интегральной составляющей управления (установим его значение 0,3).

Поставим задачу исследовать с помощью модели качество регулирования температуры бойлера в зависимости от соотношений параметров объекта управления и регулятора.

В среде AnyLogic может быть использовано несколько различных подходов для реализации динамических систем. Алгебро-дифференциальные уравнения, описывающие поведение сложной динамической системы, представляются в форме Коши и записываются непосредственно как выражения в соответствующем поле окна свойств переменных.

Для моделирования системы управления можно в поле редактора определить все переменные и параметры этой системы, затем для каждой переменной в соответствующем поле окна свойств этой переменной записать

правую часть определяющего ее уравнения. Такая модель *ControlSystem0* представлена в папке Model Examples\Part IV [2]. Переменные *T* и *Ui* определяются как вещественные вида *Интеграл*, а правая часть дифференциальных уравнений записывается в соответствующем поле в виде выражения. Например, для переменной *T* это выражение имеет вид рис. 3.6 (на желтом фоне в поле редактора здесь представлен комментарий – перечисление функциональных соотношений переменных).

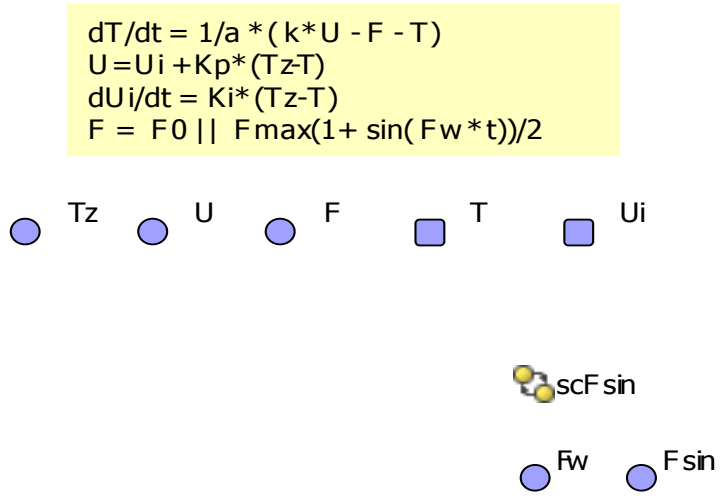

Рис. 3.6. Главное окно редактора

В модели *ControlSystem0* используются два режима для задания различного поведения внешней среды, определяемой значением параметра *F*. Переключение режимов в моделях динамических систем легко осуществляется с помощью стейтчарта. В нашей модели стейтчарт с именем *statechart* имеет два состояния. В одном состоянии никаких соотношений для *F* не определено, и, следовательно, эта переменная определена своим постоянным (начальным или текущим) значением. В другом состоянии *F* определено уравнением $F=Fmax(1+sin(Fw \cdot t))/2$. Переход в это состояние осуществляется, как только логическая переменная *Fsin* станет истинной (в результате внешнего воздействия).

Использование стейтчартов для переключения моделей позволяет легко строить сложные гибридные модели динамических систем, т. е. такие модели, в которых непрерывное поведение прерывается дискретными событиями, вызывающими изменение поведений.

С моделью *ControlSystem0* можно выполнять различные эксперименты, меняя ее параметры в окне дерева объектов модели. Так изменив после запуска модели *ControlSystem0* значение параметра Fsin с 0 на 1 (со значения *false* на значение *true*) получим переключение режима внешней среды.

Другой вариант моделирования – это построение модели с иерархической структурой. В нашем примере система управления рис. 3.5 состоит из двух

подсистем: бойлера и регулятора (контроллера). При создании модели более удобно ее структурировать, отдельно задав описание бойлера и контроллера как классов активных объектов, а затем поместить в корневой активный объект по одному их экземпляру, соединив интерфейсные переменные.

Модель *ControlSystem1* из [2] представляет эту систему управления, построенную из соответствующих классов активных объектов. Для удобства переменные *Tz*, *U*, *F* и *T* продублированы также в корневом активном объекте, они соединены с соответствующими интерфейсными переменными экземпляров классов *controller* и *boiler*.

Для моделирования динамических систем уже с 50-х годов XX века стал использоваться метод построения структурных схем из решающих блоков аналоговых компьютеров: источников тока, интеграторов, усилителей, сумматоров, умножителей. Токи и напряжения в блоках представляли константы, переменные и параметры моделируемой системы, а сами блоки реализовывали те преобразования, которые удобно выполнить аппаратно с электрическими токами: интегрирование, сложение и умножение.

Начиная с тех давних пор и по сегодняшний день блочный подход является основным в моделировании динамических систем, только сейчас решающие блоки являются не аппаратными, а реализуются программно [2, 7-9]. Подход, при котором блочная структура системы собирается из стандартных блоков библиотеки, по этой структуре автоматически строится система уравнений, которая затем решается с помощью численных методов на компьютере, реализован, например, в инструментальной системе моделирования Simulink. Подобные блок-диаграммы удобны для быстрого построения небольших структур, но наглядности и удобства в таких схемах нет. Большие трудности в построение таких блок-схем вносит необходимость переключения режимов, например в нашем примере для исследования поведения системы с постоянной и меняющейся характеристикой *F* внешней среды необходимо дополнить схему.

При разработке моделей динамических систем в AnyLogic можно использовать блоки соответствующей библиотеки (Dynamic System Library) точно так же, как это делается в пакете Simulink. Тогда модель системы управления будет реализована в виде диаграммы из блоков этой библиотеки. Эта модель называется *ControlDynSystem* и находится в папке Model Examples [2]. Все разработанные модели системы управления совместимы. Это означает, в частности, что при разработке моделей в AnyLogic можно применять различные стили, повторно использовать уже разработанные модели подсистем.

ЗАКЛЮЧЕНИЕ

Многочисленные примеры, приведенные в ряде источников [2, 10-13], показывают, что при разработке модели с помощью Any Logic от пользователя фактически не требуется знаний, не относящихся непосредственно к моделированию. Ему достаточно иметь только первоначальные знания о программировании. Для реализации и анализа стохастических имитационных моделей разработчик должен иметь лишь общие познания в статистике, но, например, ему не нужно самому реализовывать генераторы случайных чисел, так как они уже реализованы в инструменте моделирования. Проведение эксперимента с моделью легко выполняется на основе визуализации ее поведения и удобной интерпретации результатов.

Поэтому можно надеяться, что внедрение в практику разработки имитационных моделей программного инструмента AnyLogic положит начало революционным изменениям в этой области, к изменению понимания самого процесса моделирования. Эти изменения заключаются в переносе основной тяжести при разработке моделей с программирования на *создание моделей*, предоставляя разработчику модели удобный графический визуальный язык для выражения нужных абстракций и анализа модели, для управления процессом разработки модели. Все это в конечном счете должно привести к существенному уменьшению трудоемкости создания моделей.

AnyLogic позволяет интегрировать разработанную модель с офисным и корпоративным программным обеспечением, включая электронные таблицы, базы данных, ERP и CRM системы. Опыт использования AnyLogic для разработки моделей из разных областей (маркетинг, логистика, социальная динамика и др.) показал, что данный инструментарий имеет существенное преимущество перед традиционными инструментами моделирования именно в тех проектах, разработка которых требует выхода за границы одной единственной парадигмы моделирования, например, в части поддержки различных уровней абстракции объектов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Габрин, К.Э. Основы имитационного моделирования в экономике и управлении: учебное пособие / К.Э. Габрин, Е.А. Козлова. – Челябинск: Изд-во ЮУрГУ, 2004. – 108 с.
2. Карпов, Ю.Г Имитационное моделирование систем: введение в моделирование с помощью AnyLogic 5 (+ CD) / Ю.Г. Карпов. – СПб.: БХВ-Петербург, 2005. – 400 с.
3. Коровин, А.М. Новые информационные технологии в административном и корпоративном управлении: учебное пособие / А.М. Коровин. – Челябинск: Изд-во ЮУрГУ, 2003. – 113 с.
4. Логиновский, О.В. Математическое моделирование в социальных, экономических и технических системах: учебное пособие / О.В. Логиновский, А.М. Коровин. – Челябинск: Изд-во ЮУрГУ, 1998. – 114 с.
5. Логиновский, О.В. Моделирование: учебное пособие / О.В. Логиновский, И.В. Емельянова. – Челябинск: Изд-во ЮУрГУ, 2001. – 115 с.
6. Логиновский, О.В. Управление и стратегии: учебное пособие / О.В. Логиновский. – Челябинск: Изд-во ОГУ, ЮУрГУ, 2001. – 704 с.
7. Озеров, Л.А. Моделирование систем управления: учебное пособие к лабораторным работам / Л.А. Озеров. – Челябинск: Издательский центр ЮУрГУ, 2009. – 52 с.
8. Сирота, А.А. Компьютерное моделирование и оценка эффективности сложных систем: учебное пособие / А.А. Сирота. – М.: Техносфера, 2006, – 280 с.
9. Советов, Б.Я. Моделирование систем: практикум / Б.Я. Советов, С.А. Яковлев. – М.: Высшая школа, 2003. – 294 с.
10. AnyLogic. Руководство пользователя. – http://www.xjtek.com/products/anylogic5/usersmanual.pdf.
11. AnyLogic. Учебное пособие по Enterprise Library. – http://www.xjtek.com/products/anylogic5/enterpriselibrarytutorial.pdf.
12. AnyLogic. Справочное руководство по Enterprise Library. – http://www.xjtek.com/products/anylogic5/usersguide.pdf.
13. http://www.xjtek.ru.

ПРИЛОЖЕНИЕ
ОПИСАНИЕ БЛОКОВ БИБЛИОТЕКИ ENTERPRISE LIBRARY

Source

Источник заявок. Обычно используется в качестве начальной точки потока заявок, или как генератор ресурсов, транспортеров, и т. д. Генерирует заявки любых подклассов базового класса Entity через случайные промежутки времени. Время генерации может как подчиняться закону распределения, так и определяться заданным Вами расписанием. Может быть задано как максимально допустимое число генераций, так и число заявок, создаваемых за каждый раз. Если создаваемые заявки не могут покинуть объект, то они хранятся в буфере выходного порта. Если создание заявок подчиняется закону распределения, то время до создания следующей заявки вычисляется при создании заявки; следовательно, оно может быть сделано вероятностным, детерминированным, зависящим от каких-то дополнительных данных, и т. д.

Рис. П1. Объект Source

Параметры:

Имя	Тип
onExit	code (динамический)
newEntity	Class
generationType	integer
firstArrivalTime	real
interarrivalTime	real (динамический)
entitiesPerArrival	integer (динамический)
arrivalList	LookupTable
period	real
arrivalsMax	integer
canWaitAtOutput	boolean

Delay

Задерживает заявки на заданное время. Одновременно могут быть задержаны сразу несколько заявок (не более заданной вместимости объекта *capacity*). В отличие от объекта Server, заявки задерживаются независимо друг от друга – время задержки вычисляется отдельно для каждой заявки. Как только время

задержки истекает, заявка тут же покидает объект. Если объект Delay заполнен полностью, то новую заявку он не примет.

Вместимость объектов Delay может изменяться с помощью объекта Schedule. Объект Schedule автоматически управляет вместимостью в соответствии с заданными значениями времен до следующей поломки и до починки (TTF и TTR) и рабочим расписанием.

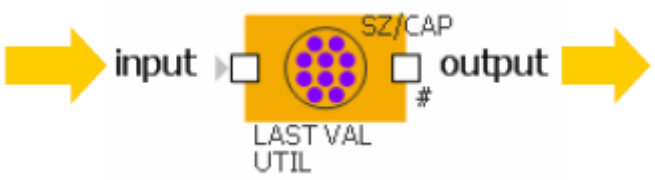

Рис. П2. Объект Delay

Когда выполняется код параметра *onEnter*, значение времени, на которое должна быть задержана заявка, доступна как *delayTimeValue*. Время задержки может быть стохастическим (как, например, значение по умолчанию), детерминированным, может зависеть от заявки или любой другой информации.
Параметры:

Имя	Тип
onEnter	code (динамический)
onExit	code (динамический)
delayTime	real (динамический)
scale	real
capacity	integer
statsEnabled	boolean
animationShape	ShapeBase
animationType	integer
animationForward	boolean
schedule	Schedule (динамический)

Queue

Объект Queue моделирует очередь, он хранит поступающие заявки в определенном порядке: FIFO (заявки помещаются в очередь в порядке поступления), LIFO (заявки помещаются в порядке, обратном поступлению), RANDOM (заявки помещаются в произвольные места очереди) или PRIORITY (заявки помещаются в очередь в соответствии со значением своих полей priority). Заявка может покинуть объект Queue различными способами:

– «обычным способом» через порт *output*, когда объект, следующий в блок-схеме за этим объектом, готов принять заявку;

– через порт *outputTimeout*, после того, как заявка проведет в очереди заданное количество времени (если включен режим таймаута);

– через порт *outputPreempted*, будучи вытесненной другой поступившей заявкой при заполненной очереди (если включен режим вытеснения);

– «вручную», путем вызова функции remove(int i).

В первом случае объект Queue покидает заявка, находящаяся в самом начале очереди (в нулевой позиции). Если заявка направлена в порт *outputTimeout* или *outputPreempted*, то она должна покинуть объект мгновенно. Если включена опция вытеснения *preemption*, то объект Queue всегда готов принять новую заявку, в противном случае при заполненной очереди заявка принята не будет.

Вместимость очереди может быть установлена бесконечной (Infinity). Если Вы хотите отображать на анимации не все, а только некоторые заявки, то используйте параметр *entitiesToAnimate*. Этот параметр задает, сколько заявок от начала очереди должно быть отображено на анимации. Аниматор должен иметь возможность показывать заданное количество заявок. Например, Вы можете использовать аниматор типа SET с ломаной линией, выбранной в качестве анимационной фигуры и значением параметра *entitiesToAnimate*: number of points in polyline.

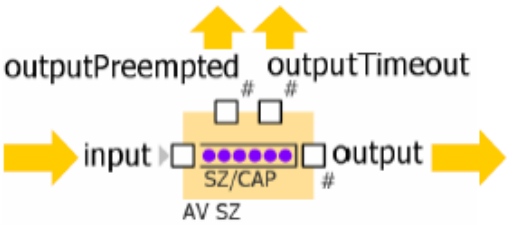

Рис. П3. Объект Queue

Параметры:

Имя	Тип
onEnter	code (динамический)
onExitPreempted	code (динамический)
onExitTimeout	code (динамический)
onAtExit	code (динамический)
onExit	code (динамический)
queueType	integer
capacity	integer
entitiesToAnimate	integer
preemption	boolean
timeout	boolean
timeoutTime	double (динамический)
statsEnabled	boolean
animationShape	ShapeBase

animationType	integer
animationForward	boolean

Server

Обслуживает заявки. Задерживает заявку, пока она не получит требуемое время обслуживания. Несколько заявок могут обслуживаться одновременно, разделяя объект Server, т. е., время обслуживания, предоставленное заявке в единицу модельного времени, обратно пропорционально числу параллельно обслуживаемых в данный момент заявок (как в мультизадачном процессоре). Следовательно, в отличие от объекта Delay, заявки, обслуживаемые одновременно, влияют друг на друга. Требуемое время обслуживания вычисляется для каждой заявки отдельно. Как только заявка получает требуемый объем обслуживания, она мгновенно покидает объект. Если вместимость объекта будет достигнута, то новую заявку он не примет.

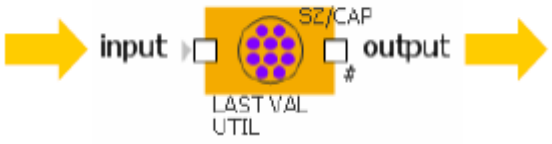

Рис. П4. Объект Server

Параметры:

Имя	Тип
onEnter	code (динамический)
onExit	code (динамический)
serviceTime	double (динамический)
capacity	integer
statsEnabled	boolean
performance	double
animationShape	ShapeBase
animationType	int

Sink

Уничтожает поступившие заявки. Обычно используется в качестве конечной точки потока заявок. Объект Sink автоматически подсчитывает входящие заявки *(getCount())* и высчитывает среднюю интенсивность входящего потока *(getAverageRate()).*

Рис. П5. Объект Sink

Параметры:

Имя	Тип
onEnter	Code (динамический)

SelectOutput

Принимает заявку, и затем, в зависимости от заданного условия, передает ее на один из двух выходных портов. Условие может зависеть от самой заявки или от какой-то другой информации. Поступившая заявка покидает объект в тот же момент времени. Объект SelectOutput обычно используется для сортировки заявок в зависимости от их типов.

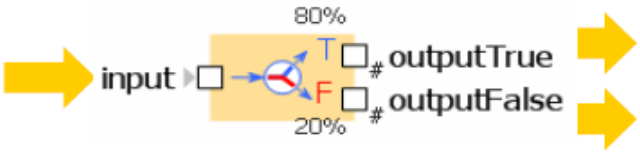

Рис. П6. Объект SelectOutput

Параметры:

Имя	Тип
onEnter	code (динамический)
onExitTrue	Code (динамический)
onExitFalse	Code (динамический)
selectCondition	boolean (динамический)

ProcessQ

Занимает ресурсы для заявки, задерживает заявку, а затем освобождает занятые ей ресурсы. К порту *access* объекта ProcessQ должны быть подсоединены один или несколько объектов Resource. Объект содержит последовательную комбинацию объектов SeizeQ, Delay и Release. Функциональность и интерфейс этих трех объектов наследуются объектом ProcessQ (за исключением объекта Release, который настроен так, чтобы освобождать именно те ресурсы, которые были ранее заняты объектом SeizeQ).

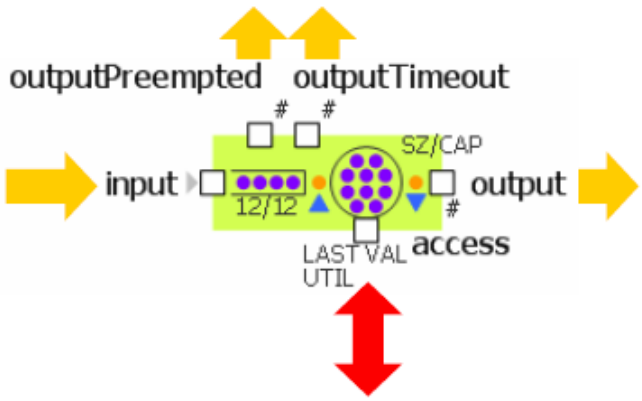

Рис. П7. Объект ProcessQ

Параметры:

Имя	Тип
onEnter	code (динамический)
onExitPreempted	code (динамический)
onExitTimeout	code (динамический)
onSeizeUnit	code (динамический)
onEnterDelay	code (динамический)
onReleaseUnit	code (динамический)
onExit	code (динамический)
quantity	integer (динамический)
selectResource	Resource (динамический)
priority	integer
queueType	integer
queueCapacity	integer
entitiesToAnimateQ	integer
preemption	boolean
timeout	boolean
timeoutTime	double (динамический)
delayTime	double (динамический)
delayCapacity	int
statsEnabled	boolean
animationShapeQ	ShapeBase
animationTypeQ	integer
animationForwardQ	boolean
animationShapeDelay	ShapeBase
animationTypeDelay	integer
animationForwardDelay	boolean

Resource

Предоставляет ресурсы, которые могут быть заняты и освобождены заявками с помощью объектов SeizeQ, ProcessQ и Release. Ресурсы – это объекты класса Entity или созданного Вами подкласса этого класса. Объект Resource может создавать, хранить, выдавать и забирать ресурсы. Количество ресурсов может изменяться динамически. Объект Resource лучше всего подходит для моделирования относительно небольших количеств индивидуально неповторимых предметов, таких, как операторы, машины, устройства, критические секции, и т. п.

В любой момент времени ресурсом может владеть только одна заявка; следовательно, заявки конкурируют за право обладания ресурсами. В случае одновременного получения нескольких запросов, Resource удовлетворяет их вначале в соответствии с их приоритетами, а затем – согласно их временным меткам. Приоритетом в данном случае является значение параметра *priority* объекта SeizeQ, который пытается занять ресурс для заявки (не путайте с полем priority заявки). Временная метка хранит значение времени, с которого заявка ожидает получения ресурса (время, когда заявка попала в очередь объекта SeizeQ). Более старые запросы обслуживаются в первую очередь. В том случае, если запрос не может быть обслужен из-за нехватки свободных ресурсов, он пропускается, а обслуживаются запросы с более низкими приоритетами или с более поздней временной меткой.

Объект Resource имеет порт *access*, который должен быть подсоединен к портам *access* объектов SeizeQ, Release или ProcessQ. Один объект Resource может быть подсоединен сразу к нескольким таким объектам, и наоборот, один объект SeizeQ или Release может быть подсоединен сразу к нескольким объектам Resource.

Рис. П8. Объект Resource

Параметры:

Имя	Тип
onSeizeUnit	code (динамический)
onReleaseUnit	code (динамический)
onGenerate	code (динамический)
capacity	integer
newUnit	Class
statsEnabled	boolean

animationShape	ShapeBase
animationType	integer
schedule	Schedule (динамический)

Split

Объект Split создает заданное число копий каждой поступающей заявки и пересылает их дальше через порт *outputCopy*. Класс новых заявок задается пользователем. Копирование занимает нулевое время – как только заявка входит в объект Split, она тут же покидает его вместе с копиями.

Рис. П9. Объект Split

Параметры:

Имя	Тип
onEnter	code (динамический)
onExit	code (динамический)
onExitCopy	code (динамический)
newCopy	Class
numberOfCopies	int (динамический)

ОГЛАВЛЕНИЕ