ИНФОРМАЦИОННОЕ ПРОСТРАНСТВО

Карпов Юрий Глебович

ИЗУЧЕНИЕ СОВРЕМЕННЫХ ПАРАДИГМ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ В СРЕДЕ *ANYLOGIC*

ВВЕДЕНИЕ

Хотя имитационное моделирование – чрезвычайно широкая сфера с большим количеством прикладных областей, в которых существует множество подходов и стилей моделирования, здесь можно выделить четыре основные парадигмы моделирования, то есть четыре системы взглядов, концепций и приемов, используемых в качестве «каркаса» при построении моделей:

– динамические системы,

– системная динамика,

– дискретно-событийное моделирование,

– мультиагентные модели.

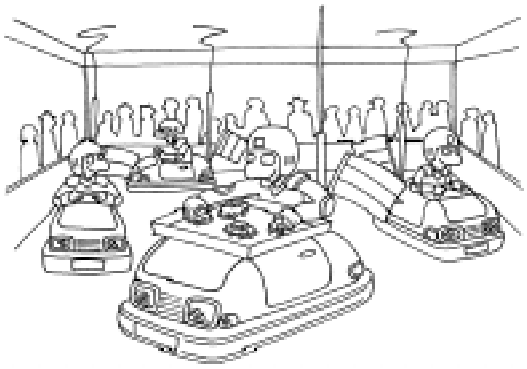
При обучении имитационному моделированию как прикладной дисциплине в узкой прикладной области обычно используют только один продукт, который позволяет строить модели только в рамках одной определенной идеологии. Например, в бизнес-школах такими продуктами являются *Vinsim* либо *Powersim*, поддерживающие разработку моделей в рамках системной динамики [Sterman]. В технических вузах обучают моделированию на базе инструментов GPSS или Arena, позволяющих строить модели дискретно-событийных систем [Рыжиков]. Математиков и физиков обучают использованию программного пакета *Simulink* для построения имитационных моделей динамических систем и их анализа.

С помощью нового отечественного программного инструмента *AnyLogic* [1; 2] изучение общих проблем и концепций имитационного моделирования может быть выполнено на основе только одного этого инструмента.

ANYLOGIC

AnyLogic основан на объектно-ориентированной концепции.

Другой базовой концепцией *AnyLogic* является представление модели как набора взаимодействующих параллельно функционирующих активных объектов. *Активный объект* в *AnyLogic* – это объект со своим собственным функционированием, взаимодействующий с окружением. Он может включать в себя любое количество экземпляров других активных объектов. Экземпляры ак-

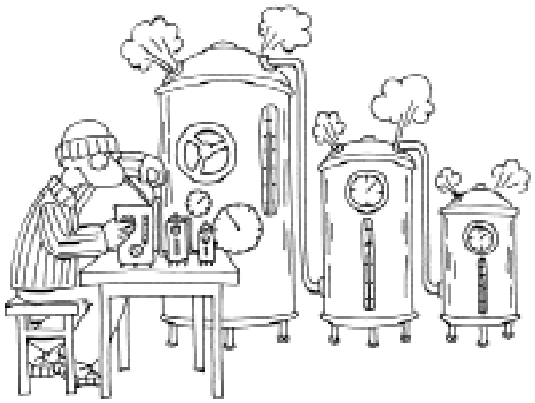


Активный объект в AnyLogic... может включать в себя любое количество экземпляров других активных объектов.

тивных объектов могут динамически порождаться и исчезать в соответствии с законами функционирования системы. Так могут моделироваться социальные группы, холдинги компаний, транспортные системы и т. п.

Графическая среда моделирования *AnyLogic* поддерживает проектирование, разработку, документирование модели, выполнение компьютерных экспериментов с моделью, включая различные виды анализа.

AnyLogic позволяет быстро создавать модели из базовых графических компонентов, работая в режиме *drag-and-drop*. Поэтому простые модели могут быть готовы уже через несколько минут после начала их разработки. Однако разработка реально полезных моделей всегда требует использования в той или иной степени программного кода. Необходимость программирования при разработке моделей является одной из основных проблем, затрудняющих освоение имитационного моделирования. Это требование является свойством всех инструментов моделирования: в любом из них для разработки серьезной модели требуется использование встроенного в этот симулятор языка программирования – либо специализированного языка, либо одного из универсальных языков программирования, с которым интегрирован симулятор. «Нормальной считается ситуация, когда после 10 минут манипуляции мышкой программист будет 10 часов писать и отлаживать тексты программ, которые придадут модели все желаемые им свойства» – замечает один из экспертов в области имитационного моделирования [3]. Некоторые авторы считают, что разработчику имитационных моделей абсолютно необходимо глубокое знание программирования: «работа в области прикладной математики без уверенного владения техникой программирования не только бессмысленна, но и опасна (и для общества, и для самого горе-специалиста). К сожалению, в последнее десятилетие увлечение «информационными технологиями» и демагогия о «непрограммирующих пользователях» заметно ухудшили программистскую подготовку будущих математиков и инженеров» [4]. Вопрос, конечно, заключается в том, сколько кода нужно писать при разработке модели, насколько сложной является интеграция программного кода в среду разработки, до какого уровня разработчик моделей, который обычно не является программистом, должен знать тонкости программирования на базовом языке симулятора и т. п. В *AnyLogic* базовым языком, совмещенным со средой разработки моделей, является *Java*, один из наиболее мощных и в то же время простых языков программирования. Включение программного кода на *Java* в модель *AnyLogic* является органичным и естественным, тексты языка интегрируются в модель чрезвычайно просто и элегантно. Если разработчик глубоко владеет приемами программирования, то он может с успехом использовать при построении модели всю мощь языка. Однако для построения даже сложных моделей на *AnyLogic* от разработчика требуются лишь самые общие, поверхностные знания о программировании.



Пусть объектом управления является бойлер...

AnyLogic – это инструмент визуальной разработки моделей и визуального представления результатов моделирования. Использование визуализации при имитационном моделировании систем трудно переоценить. Наибольший эффект – вплоть до эффекта присутствия – дает анимированное представление поведения системы и ее частей в виде некоторой формы виртуальной реальности. В среде *AnyLogic* легко может быть создан виртуальный мир, подчиняющийся законам, которые разработчик вложил в модель. Имитационные модели с использованием анимации (анимированной ви-

зуализации поведения системы) являются основной частью тренажеров.

ДИНАМИЧЕСКИЕ СИСТЕМЫ

Динамические системы – это сложные физические объекты, поведение которых описывается системами алгебро-дифференциальных уравнений, а также событиями, меняющими либо среду, либо модель, либо даже саму структуру объекта. К этому классу относятся системы управления, физические и механические системы, объекты химической технологии и т. п. Инженерным подходом к моделированию таких объектов уже 40 лет назад была сборка схем из решающих блоков аналоговых компьютеров (интеграторов, усилителей и сумматоров), токи и напряжения в которых представляли переменные и параметры моделируемой системы. Этот подход и сейчас является основным в моделировании динамических систем, только решающие блоки являются не аппаратными, а программными. Он реализован, например, в инструментальных средах *Simulink*, *LabView*, *Easy5* и др.

Рассмотрим в качестве примера динамическую систему, состоящую из двух связанных подсистем: объекта управления и регулятора (рисунок 1).

Пусть объектом управления является бойлер, который нагревается до температуры T. Величину T необходимо поддерживать на заданном уровне T_z. Температура T зависит от мощности тока U, подаваемого на нагревательный элемент бойлера. Объект управления подвергается внешнему возмущению F, вследствие чего значение T может меняться. Поддержание значения T на заданном уровне T_z есть задача регулятора. Регулятор по разнице $T_z - T$ формирует величину входного параметра объекта управления (в нашем случае с помощью ПИ-регулятора регулируется мощность U, подаваемая на нагревательный элемент). Система алгебро-дифференциальных уравнений, описывающих поведение системы бойлер-регулятор, имеет вид:

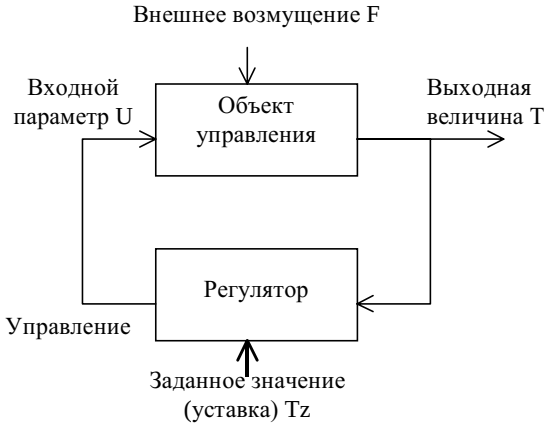


Рисунок 1. Структура системы регулирования.

$$d(T)/dt = 1/a\ *(\ k*U - F - T),$$
$$U = U_i + K_p*(T_z - T),$$
$$d(U_i)/dt = K_i*(T_z - T),$$

Для решения этой системы в редакторе пакета *Simulink* следует набрать блок-схему (рисунок 2), в которой связаны блоки интеграторов, сумматоров, усилителей, источников постоянных значений. В комментариях к подобной схеме в одном из руководств по системе *Simulink* [5] написано: «Такая схема может показаться очень запутанной. Но представьте себе сложность программирования этой системы на С или Фортране!» Конечно, по сравнению с программированием на алгоритмическом языке, использование таких блок-схем – это революционный шаг вперед. Однако современные информационные технологии позволяют использовать и другие, более удобные средства.

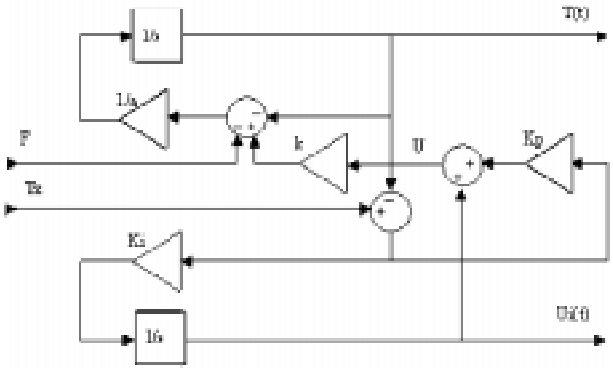


Рисунок 2. Блок-диаграмма для моделирования в Simulink.

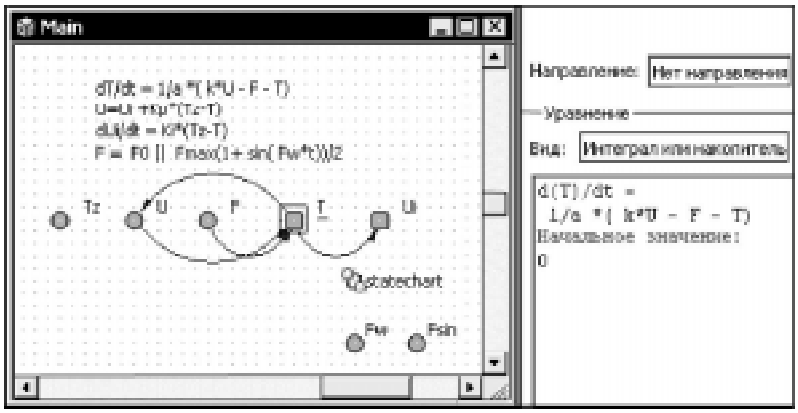


Рисунок 3. Определение переменной *T* в поле редактора *AnyLogic*.

В *AnyLogic* алгебро-дифференциальные уравнения, описывающие поведение динамической системы, записываются как обычные аналитические выражения (рисунок 3). Построение такой модели требует от разработчика минимума времени.

AnyLogic имеет библиотеку современных численных методов, из которой при запуске модели автоматически выбирается метод, наиболее подходящий для решения данной системы алгебро-дифференциальных уравнений. Удобные средства отображения изменения переменных в зависимости от времени с помощью графиков и диаграмм позволяют наблюдать и анализировать изменение системных переменных в зависимости от времени. В *AnyLogic* можно также построить анимированное представление системы, в котором динамика ее поведения будет наглядно отображаться (рисунок 4).

В программных продуктах, поддерживающих разработку моделей динамических систем, обычно трудно выражать дискретные события, а также строить модели, в которых присутствуют как непрерывные, так и дискретные переменные. В *AnyLogic* спецификация моделей таких гибридных систем удобно выполняется с помощью так называемых *стейтчартов* (карт состояний) (см. рисунок 5). На этом рисунке представлен результат моделирования движения мяча, скачущего по ступенькам. В отдельном окне представлен стейтчарт, который содержит одно состояние с переходами, которые отслеживают отскоки мяча от ступеньки и от угла ступеньки. Графики отображают зависимость вертикальной координаты мяча от времени и от другой координаты при его движении. Анимация представляет движение мяча в двумерном пространстве.

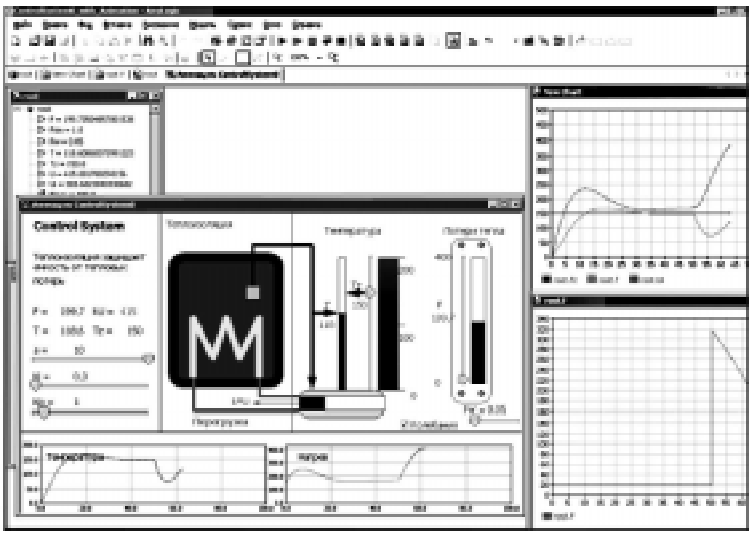
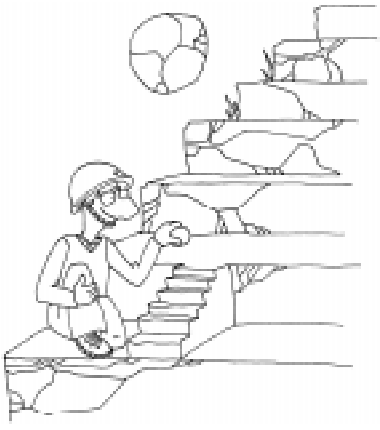


Рисунок 4. Выполнение имитационной модели регулятора в *AnyLogic*.



...результат моделирования движения мяча, скачущего по ступенькам

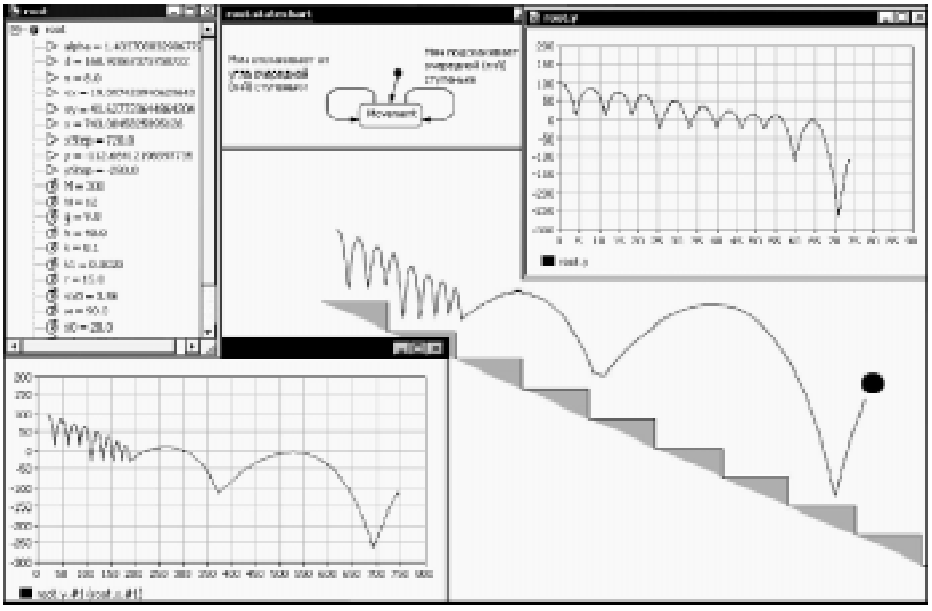


Рисунок 5. Эксперимент с мячом, прыгающим по ступенькам.

ДИСКРЕТНО-СОБЫТИЙНЫЕ СИСТЕМЫ

Рассмотрим пример модели сервиса, предоставляемого автоматической телефонной станцией. Поставщик сервиса для мобильной связи выбирает оборудование телефонной станции и задается вопросом, сколько потребуется телефонных каналов для получения максимальной прибыли?

Пусть правила Тарифного плана таковы, что каждый обслуженный телефонный вызов приносит поставщику сервиса некоторый доход, а за каждый отклоненный вызов он должен заплатить штраф. Покупка оборудования АТС и содержание каждого канала связи обходятся в некоторую сумму, зависящую от числа каналов АТС. Варьируя число каналов, можно найти то оптимальное их число, которое принесет максимальный доход.

Структура имитационной модели, с помощью которой решается эта проблема, представлена на рисунке 6. Генератор заявок имитирует приход телефонных вызовов. Вызов может быть принят обслуживающим прибором, только если есть свободный канал соединения. Блок «Ресурсы» имитирует наличие ограниченного числа ресурсов (каналов), это число можно задать параметром. Блок анализа направляет посту-

...пример модели сервиса, предоставляемого автоматической телефонной станцией.

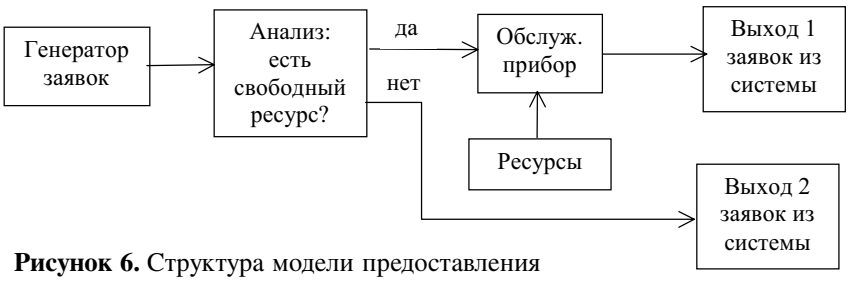


Рисунок 6. Структура модели предоставления сервиса мобильной связи.

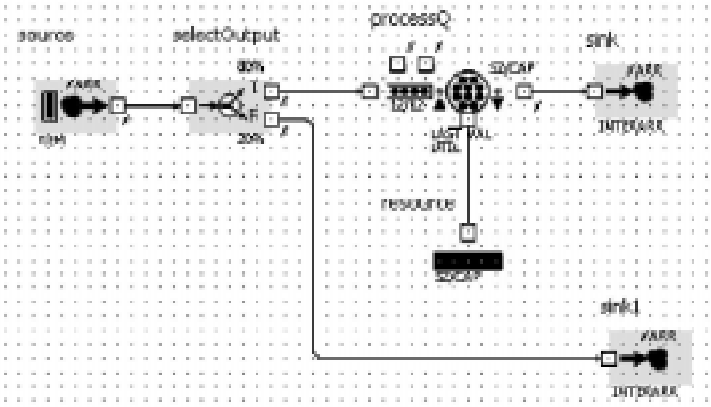


Рисунок 7. Модель в *AnyLogic* для оптимизации сервиса мобильной связи.

пившие вызовы либо на обслуживание, если есть свободный канал, либо на выход из системы при отсутствии свободного канала.

В библиотеке *Enterprise Library* системы *AnyLogic* включены все блоки, присутствующие на рисунке 6, поэтому построение имитационной модели из таких блоков с настройкой их параметров займет у разработчика всего несколько минут. Эта модель показана на рисунке 7.

Рассмотрим другой пример: модель широко известного алгоритма распределенного завершения. Проблема распределенного завершения относится к классу тех проблем, которые при очень простой формулировке имеют весьма нетривиальное решение. В то же время, большое число прикладных задач в области распределенных вычислений может быть сведено к этой проблеме.

Рассматривается *N* машин, связанных в произвольную компьютерную сеть. Все машины вместе выполняют некоторый распределенный алгоритм вычислений, который мы будем называть базовым алгоритмом. Каждая машина может быть либо в активном состоянии, выполняя базовые вычисления, либо в пассивном состоянии, если она завершила свою часть распределенного вычисления. Активная машина, выполняющая базовые вычисления, может посылать сообщения с просьбой о помощи в вычислениях каким-либо другим машинам в сети, передавая им часть работы. Сообщения получаются адресатом после некоторой задержки в сети без потерь. Если пассивная машина получает сообщение, она становится активной, активная машина, получившая сообщение, остается активной и продолжает вычисления.

Рассматривается N машин, связанных в произвольную компьютерную сеть

Из активного состояния переход машины в пассивное состояние может произойти по завершении ее части общих вычислений в любой момент. Прием сообщения является единственным событием, которое переключает машины из пассивного в активное состояние. Отсюда следует, что состояние всей системы, в котором все машины пассивны, а в сети нет задержанных сообщений, посланных ранее какой-нибудь из активных машин, является стабильным: распределенное вычисление завершено.

Цель алгоритма распределенного завершения состоит в том, чтобы некоторая машина, например 0-я, обнаружила, что система пришла в это стабильное состояние. Желательно, чтобы это было обнаружено как можно скорее после того, как стабильное состояние наступило. Для этого выделенная машина должна использовать информацию от остальных машин об их состояниях. Ясно, что алгоритм определения распределенного завершения не может быть просто проверкой текущего состояния всех машин в сети и принятия решения о завершении вычислений, если все машины отрапортовали о том, что они пассивны. Во-первых, мгновенно такую проверку выполнить

в распределенной системе невозможно, и, во-вторых, в канале могут находиться задержанные сообщения базового алгоритма, которые могут активизировать перешедшую в пассивное состояние машину, которая уже послала рапорт о своем пассивном состоянии. Поэтому алгоритм обнаружения завершения вычислений в распределенной сети машин является нетривиальным.

Назовем алгоритм обнаружения распределенного завершения зондированием. Зондирование может быть инициировано выделенной машиной в любом состоянии всей сети, при любом распределении активностей в ней. Поэтому запущенный алгоритм зондирования может дать отрицательный результат, если сеть машин не находится в стабильном состоянии. Обычно алгоритм зондирования запускается выделенной машиной периодически: если очередной раунд оказался неудачным, инициируется следующий раунд.

Существует несколько решений проблемы распределенного завершения. Алгоритм, модель которого представлена здесь, предложен Э. Дейкстрой [8], работает на произвольном числе машин, пронумерованных от 0 до N, связанных в виртуальное кольцо. Алгоритм состоит из повторения циклов зондирования сети посылкой зонда (токена) по кольцу машин.

Данный распределенный алгоритм состоит из следующих локальных правил, которые, как доказал Дейкстра, гарантируют решение проблемы:

1. Каждая машина в сети поддерживает счетчик c. Каждая посылка сообщения базового алгоритма увеличивает c на 1; прием базового сообщения уменьшает c на 1. Таким образом, сумма всех счетчиков в сети машин в некоторый момент времени равна числу сообщений, задержанных в сети. Если сумма значений счетчиков 0, то никаких базовых сообщений в сети нет.

2. Машина 0 инициирует шаг зондирования в любой момент и в любом своем состоянии, посылая токен со значением 0 машине $N - 1$. Каждая машина i задерживает полученный токен до тех пор, пока не станет пассивной. После этого значение токена увеличивается на c, и токен передается машине $i - 1$.

3. И машины, и токен имеют цвет. Вначале все машины и токен белые. Машина, получившая базовое сообщение, становится черной. Когда машина передает токен, она становится белой. Если черная машина передает токен, токен становится черным, в противном случае токен сохраняет свой цвет.

4. Когда машина 0 получает свой токен, прошедший по кольцу всех машин, очередной цикл зондирования закончен. Машина 0, получив токен ждет, пока она не перейдет в пассивное состояние. После этого она делает заключение о завершении базовых вычислений в сети машин при следующих условиях:

– сама машина 0 белая;

– полученный токен белый;

– сумма значений токена и счетчика с машины 0 равна 0.

В противном случае завершения на этом цикле не обнаружено, машина 0 запускает новый цикл зондирования.

На рисунке 8 представлена анимация модели на *AnyLogic*, реализующей алгоритм распределенного завершения. Каждый прямоугольник представляет одну машину. Прямоугольники разделены на две части. Одна часть отражает состояние машины при выполнении базового алгоритма: зеленый цвет представляет пассивную машину, красный – активную машину. Стрелка от машины i к машине k показывает, что в канале

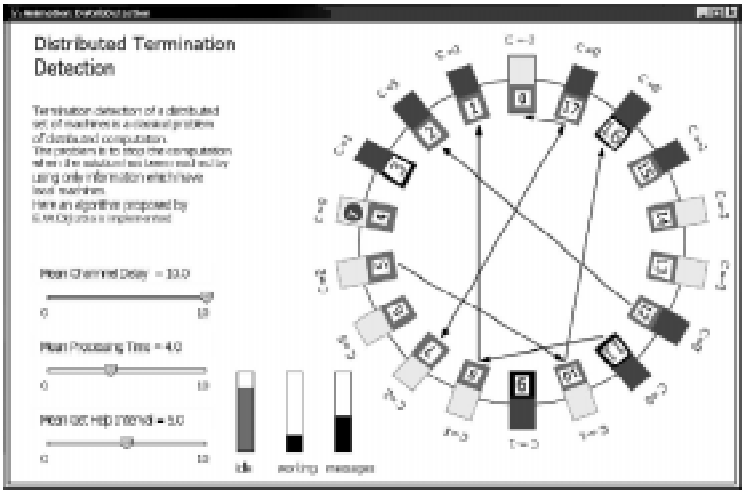


Рисунок 8. Анимация работы алгоритма распределенного завершения.

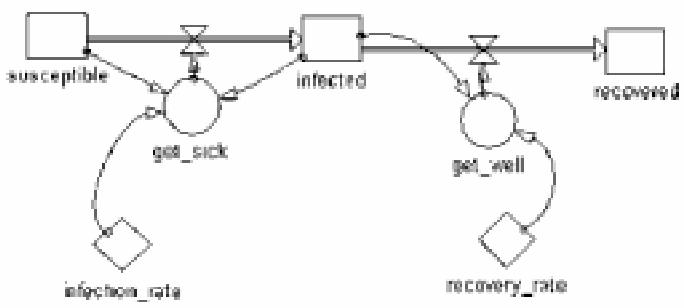


Рисунок 9. Модель *AnyLogic* развития эпидемии в *Powersim*.

находится сообщение базового алгоритма с передачей части работы *k*-й машине от *i*-й. Другая часть прямоугольников характеризует состояние алгоритма зондирования. Например, на рисунке 8 машина 4 является белой, она имеет токен. Этот токен окрашен в черный цвет, при своем движении по кольцу машин он подсчитал сумму (равную 7) всех значений параметра с пройденных машин.

СИСТЕМНАЯ ДИНАМИКА

Системная динамика является еще одной парадигмой имитационного моделирования. Эта парадигма вводит идею описания сложных систем на высоком уровне абстракции, на котором исследователь абстрагируется от индивидуальных объектов системы (людей, машин, документов, товаров) и рассматривает только агрегированные количественные их значения, а также взаимные зависимости потоков объектов. Эта идея была предложена Дж. Форрестером почти 50 лет назад [9]. Он показал, что динамика функционирования сложных систем (в первую очередь, производственных и социальных систем) существенно зависит от структуры связей и временных задержек в действиях и принятии решений, которые существуют в системе. В соответствии с этой парадигмой, для системы строится графическая диаграмма причинных связей и влияния во времени одних параметров системы на другие, а затем модель, представленная такой диаграммой, имитируется на компьютере.

Системная динамика выработала свою графическую нотацию для построения структур потоковых диаграмм (*stock-and-flow* диаграмм), представляющих причинно-следственные связи в сложной системе. Эта нотация включает простые графические элементы с ясной семантикой, которая позволяет по построенной графической схеме взаимных зависимостей переменных и параметров системы упростить построение уравнений ее динамики и проигрывать их во времени.

Рассмотрим пример. На рисунке 9 изображена структура классической модели распространения эпидемии, построенная в графическом редакторе инструмента моделирования *Powersim*. В этой модели исследуется зависимость динамики числа заболевших (например, гриппом) и выздоровевших после болезни.

Здесь обозначены:

susceptible – общее число людей, восприимчивых к инфекции;

infected – общее число заболевших людей;

recovered – число людей, выздоровевших после инфицирования;

infection_rate – доля заболевающих в единицу времени;

recovery_rate – доля выздоравливающих в единицу времени;

get_sick – число людей, заболевающих в единицу времени;

get_well – число людей, выздоравливающих в единицу времени.

В этой модели исследуется зависимость динамики числа заболевших ... и выздоровевших...

Модель на рисунке 9, на которой представлены переменные, потоки и функциональные зависимости, отражает следующие соотношения:

d(susceptible)/dt = - get_sick;

d(infected)/dt = get_sick - get_well;

d(recovered)/dt = get_well;

get_sick = f(infected, susceptible, infection_rate);

get_well = f(infected, recovery_rate).

В модели для переменных *get_sick* и *get_well* необходимо задать конкретный вид функциональных зависимостей.

AnyLogic включает средства, позволяющие разработчику моделей мыслить (и строить модели) в терминах графических элементов системной динамики. На рисунке 10 изображена структура той же модели распространения эпидемии, построенная в *AnyLogic*.

После запуска модели можно анализировать зависимость динамики системы от параметров (рисунок 11).

Системная динамика оказалась очень эффективной парадигмой исследования организационных систем (например, решения таких сложных проблем организационных систем, как анализ рынка, управление проектами, управление цепочками поставок).

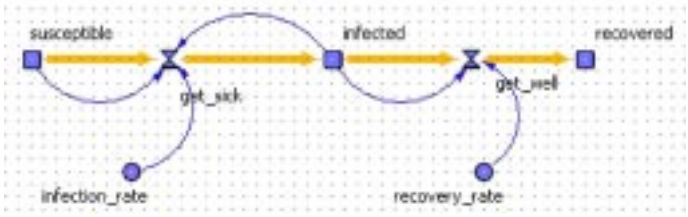


Рисунок 10. Модель *AnyLogic* развития эпидемии.

Рассмотрим еще одну модель, которая была построена Эдвардом Лоренцем в [10] для очень упрощенного описания процессов, происходящих в атмосфере и определяющих погоду. В модели переменная X задает интенсивность конвекции, Y – это разница температур нисходящего и восходящего потоков, а Z – изменение со временем вертикальной температуры в некоторой точке.

С формальной точки зрения модель не представляет ничего сложного: это просто три нелинейных дифференциальных уравнения. Однако в ней присутствуют три различных петли обратных связей зависимостей переменных, что делает эту систему трудной для понимания и анализа (рисунок 12).

Несмотря на кажущуюся простоту системы, аналитического решения у нее нет: система уравнений существенно нелинейна. Сложные взаимные связи переменных делают динамику модели весьма интересной. Во-первых, при некоторых начальных условиях модель описывает так называемый

модель... очень упрощенного описания процессов, происходящих в атмосфере...

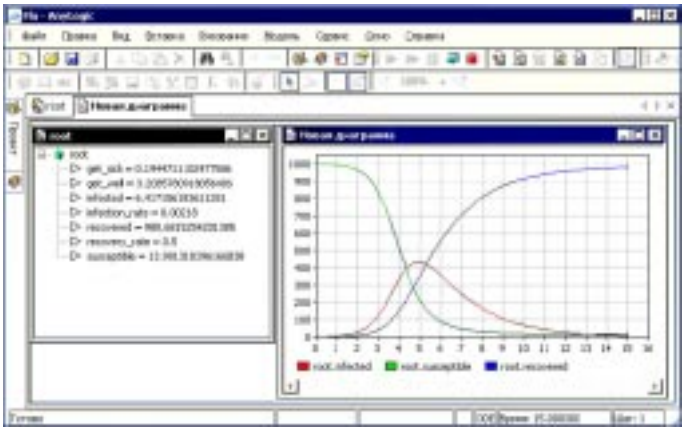


Рисунок 11. Результат выполнения модели динамики развития эпидемии.

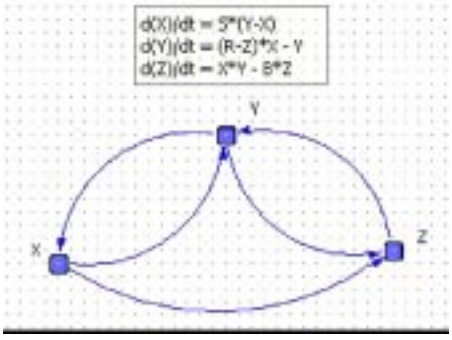


Рисунок 12. Модель Лоренца в *AnyLogic*.

«детерминированный хаос» – детерминированную модель, поведение переменных которой чрезвычайно хаотично (другое название модели – аттрактор Лоренца). Во-вторых, эта модель демонстрирует высокую чувствительность к начальным условиям. Например, при значениях параметров ($X = Z = S = 10$, $Y = 0$, $R = 28$, $B = 8/3$) незначительные изменения параметров существенно меняют траектории переменных.

Поставим эксперимент. На рисунке 13 представлены результаты одновременного моделирования двух независимых идентичных процессов, один с переменными (X, Y, Z), а другой – его копия с переменными ($X1$, $Y1$, $Z1$). В обеих моделях для всех переменных и параметров установлены одинаковые значения, за исключением начального значения $Z1$, которое отличается от $Z = 10$ в десятом знаке, то есть $Z1 = 10.00000001$.

Эксперимент на рисунке 13 показывает, что через некоторое время после старта значения X начинают существенно отличаться от значений $X1$. Таким образом, в этой модели объективно существующий предел точности измерений, определяющих начальное состояние любой реальной системы, не дает возможности точно рассчитать будущее состояние системы на основе известных начальных условий. Эта модель описывает процессы в атмосфере, поэтому она демонстрирует внутреннюю ограниченность предсказаний погоды, которую можно интерпретировать так: даже взмах крыла бабочки где-нибудь в лесах Бразилии может через месяц вызвать ураган в Техасе. Лоренц назвал этот эффект *«эффектом Бабочки»*.

Модели системной динамики, построенные для анализа реальных экономических и бизнес-процессов, процессов урбанизации, динамики народонаселения и т. п., тоже являются нелинейными, они обычно значительно сложнее модели Лоренца. Перекрестное взаимное влияние процессов в них может вызвать совершенно неожиданные эффекты, иногда похожие на эффект «детерминированного хаоса», описываемый моделью Лоренца. Джей Форрестер в работе [11] пишет о том, что большая корпорация, город, экономика, правительство – все это примеры сложных систем, поведение которых в корне отличается от того, что мы обычно предполагаем из опыта наблюдения простых систем. Модель Лоренца демонстрирует возможность возникновения хаотического поведения в подобных системах и трудности прогноза в них. Задачи моделирования – предсказать такие ситуации и ввести управления, предотвращающие хаотическое развитие процессов и направляющие их в желаемое русло. В глубокой моногра-

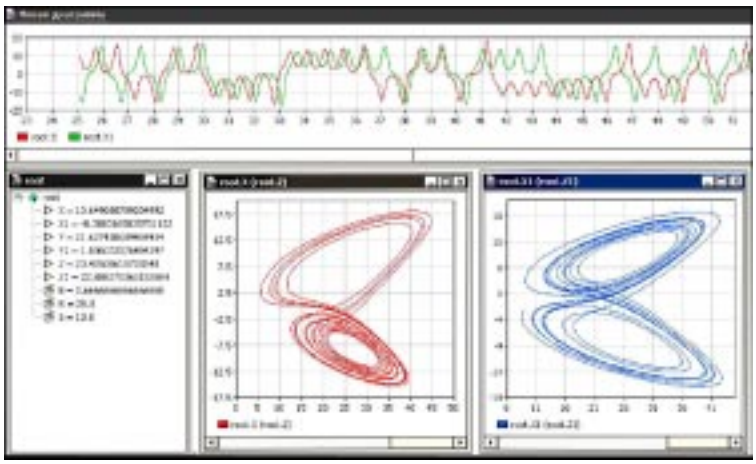


Рисунок 13. Эксперимент, демонстрирующий влияние на погоду «взмаха крыла бабочки».

фии [12] Дж. Штерман, профессор Слоановской бизнес-школы MIT, приводит огромное число моделей системной динамики, используемых в экономике и бизнесе.

МНОГОАГЕНТНЫЕ МОДЕЛИ

Парадигма многоагентных моделей возникла относительно недавно. *Агент* – это некоторая сущность, которая обладает активностью, автономным поведением, может принимать решения в соответствии с некоторым набором правил, может взаимодействовать с окружением и другими агентами, а также может сама изменяться (эволюционировать). Многоагентные (или просто агентные) модели используются для анализа децентрализованных систем, динамика функционирования которых не определяется глобальными правилами и законами. Цель агентных моделей: получить представление об общем поведении системы, о возникающих в системе глобальных правилах, исходя из предположений об индивидуальном поведении ее отдельных активных объектов.

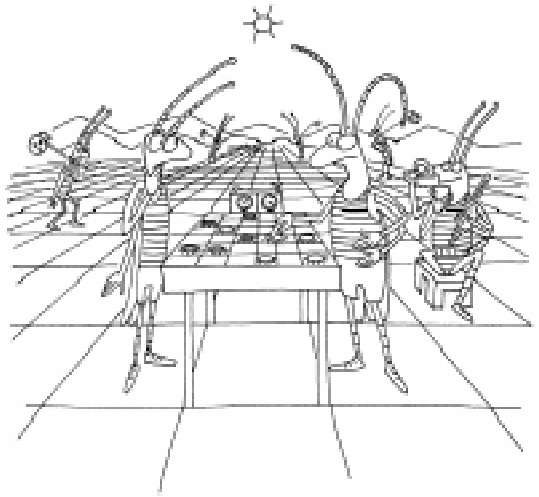
Моделирование агентов и многоагентных систем не представляет сложностей в *AnyLogic* ни в концептуальном, ни в техническом аспекте: все указанные выше свойства агентов легко реализуются в этой среде. Основной концепцией *AnyLogic* является та, что модель состоит из активных объектов, имеющих каждый свои правила поведения и взаимодействующих через явно определенные интерфейсы. Поэтому агентный подход к построению моделей является в *AnyLogic* совершенно естественным: в этой среде можно быстро создавать модели с агентами, взаимодействующими как друг с другом, так и со средой. *AnyLogic* является единственным профессиональным продуктом, поддерживающим агентное моделирование [13]. В *AnyLogic* возможна реализация моделей, содержащих десятки и даже сотни тысяч активных агентов. На рынке ПО кроме *AnyLogic* не существует других профессиональных инструментов, позволяющих эффективно строить агентные модели (кроме академических систем типа *Swarm* и *Repast*).

Рассмотрим в качестве примера простую модель коллективного поведения, названную *Heat Bugs* (тепловые жуки), впервые разработанную в пакете *Swarm*. В дискретной среде, разбитой на клетки, двигаются «жуки», которые выделяют тепло. Тепло распространяется в среде. Каждый жук имеет свою собственную «идеальную» температуру среды, в которой он предпочитает находиться, и имеет сенсор, с помощью которого он может определить, в каком направлении температура среды ближе к его «идеальной» температуре. Это позволяет жуку найти то направление, в котором он должен двигаться, чтобы достигнуть клетки с устраивающей его температурой. Среда имеет следующие характеристики:

(1) тепло распространяется равномерно во всех направлениях со скоростью, пропорциональной разнице температур в соседних клетках;

(2) тепло «испаряется» (уменьшается) в каждой клетке пропорционально количеству тепла, которым обладает клетка.

«Мир жуков» является динамическим и трудно предсказуемым. Например, даже находясь в клетке с «идеальной» для него температурой, жук нагревает ее, выделяя тепло, поэтому со временем он может направиться в более комфортную для него клетку, поскольку состояние среды изменится. Кроме того, жук может поступать и



«Мир жуков» является динамическим и трудно предсказуемым.

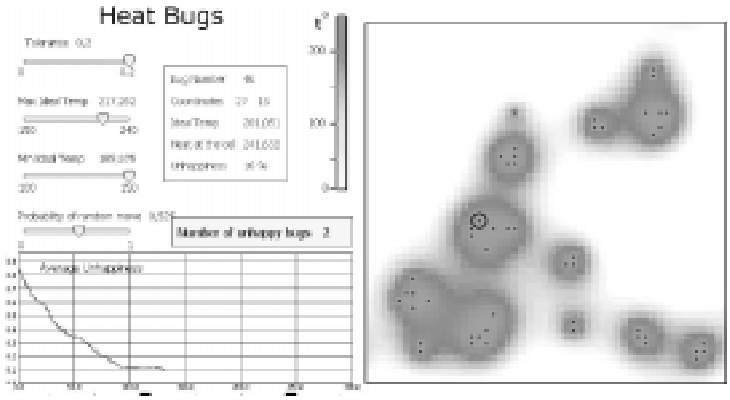


Рисунок 14. Анимационное представление динамики мира «тепловых жуков».

нерационально: в частности, к области, наиболее близкой к его «идеальной» температуре, жук может направиться не напрямую.

Анимация модели «мира тепловых жуков» представлена на рисунке 14. В окне анимации значение температуры в каждой клетке представлено цветом. Температурная шкала выведена рядом с полем, представляющим среду. На рисунке видно, что жуки собираются в группы, согревая друг друга. Выделив мышкой в поле анимации конкретного жука, можно видеть его параметры – его координаты, идеальную температуру, его коэффициент «некомфортности» (относительную разницу его идеальной температуры и реальной температуры среды в той клетке, в которой он находится). С этой моделью могут быть проведены эксперименты, показывающие, как изменение параметров повлияет на глобальное поведение всего коллектива.

Литература

1. www.anylogic.ru

2. *Карпов Ю.Г.* Имитационное моделирование систем. Введение в моделирование на AnyLogic 5 // БХВ_Петербург, СПб., 2005.

3. *Толуев Ю.* Записки симуляциониста, любящего и уважающего GPSS // http://www.gpss.ru/index-h.html, 2002.

4. *Рыжиков Ю.И.* Имитационное моделирование. Теория и технология // СПб.: КОРОНА принт, 2004. 384 с.

5. *G.D.Buckner.* Simulink: A Graphical Tool for Dynamic System Simulation // Technical Report Dept of Mechanical and Aerospace Engineering North Carolina State University, 2002.

6. *Лоу А.М., Кельтон В.Д.* Имитационное моделирование. 3-е издание // СПб: Питер, Киев: BHV, 2004. 847 с.

7. *Banks J., ed.* Handbook of Simulation, Principles, Methodology, Advances, Applications, and Practice // J.Wiley & Sons, Inc., 1998. 849 p.

8. *E.W.Dijkstra, W.H.J.Feijen and A.van Gasteren.* Derivation of a termination detection algorithm for distributed computations. // Information Processing Letters. Vol. 16, p. 217–219, 1983.

9. *Jay W.Forrester.* Industrial Dynamics: a Major Breakthrough for Decision Makers // Harvard Business Rev., 1958.

10. *E.N.Lorenz.* Deterministic nonperiodic flow // J.Atmosf. Sci., 20, 130–131, 1963.

11. *J.Forrester.* Urban Dynamics // Productivity Press, 1969.

12. *John D.* Sterman. Bysiness Dynamics. System Thinking and Modeling for a Complex World // Mc.Graw Hill, 2000. 982 p.

13. *Борщев А.В.* Практическое агентное моделирование и его место в арсенале аналитика // Exponenta Pro, № 3–4, 2004. (См. также http://www.gpss.ru/index-h.html).

Карпов Юрий Глебович, профессор, доктор технических наук, заведующий кафедрой Распределительных вычислений и компьютерных сетей Санкт-Петербургского Политехнического Университета.