КАК СТРОИТЬ ПРОСТЫЕ, КРАСИВЫЕ И ПОЛЕЗНЫЕ МОДЕЛИ СЛОЖНЫХ СИСТЕМ

А.В. Борщев (Санкт-Петербург)

Введение

Автор считает, что имитационные модели (а также компьютерные программы и многие другие творения человека), выглядящие неестественно, нестройно, неадекватно сложно, некрасиво, скорее всего, не являются полезными. Речь идет, естественно, не столько о красоте интерфейса пользователя и анимации, сколько о простоте и понятности конструкций, использованных разработчиком внутри.

Запутанная структура, необходимость длительных объяснений при передаче исходного кода другому разработчику, наличие «workarounds», то есть неестественных, нештатных приемов для обхода ограничений выбранной технологии, несоответствие сложности модели интуитивно понимаемой сложности задачи затрудняют разработку, использование, и поддержку моделей, ограничивают их жизненный цикл и являются источником трудно выявляемых ошибок.

Источником перечисленных «уродств» в моделях часто является не отсутствие у разработчика нужной квалификации, а *использование выразительных средств, не соответствующих решаемой задаче*. Это и есть основная тема данной статьи.

Необходимым условием создания внутренне стройных, минималистичных и, в конечном счете, полезных и долгоживущих имитационных моделей, является знание выразительных средств, то есть языков и стоящих за ними методов, и правильное их применение. Мы проведем краткую»инвентаризацию»существующих на настоящий момент языков описания динамики систем, то есть изменения систем во времени. Наша цель – показать разработчику моделей его настоящий арсенал и предложить взглянуть на вещи шире, возможно, отказавшись от привычного подхода (языка, инструмента) в пользу другого, дающего более естественный рисунок для конкретной задачи.

Что останется за рамками рассмотрения

Проблема выбора уровня абстракции решается разработчиком модели во всех без исключения проектах, и она очень тесно связана с выбором языка. Тем не менее эта проблема не является сейчас для нас центральной. Мы будем считать, что разработчик умеет упрощать реальный мир до нужного уровня, и дело остается за «малым»: выразить упрощенную динамику на одном или нескольких языках моделирования.

Мы будем концентрироваться на бизнес-приложениях имитационного моделирования, под которыми мы понимаем логистику, перевозки, цепочки поставок, производство, сферу обслуживания (включая здравоохранение), управление активами и парками, кадры, управление проектами и продуктами, рынок и конкуренцию, социальную динамику и динамику экосистем, политику и экономику, военные действия, сети, телекоммуникации, то есть, те сферы, где уровень абстракции и выбор языка неочевидны.

Задачи, для которых разработаны специальные методы, например, *микромоделирование* пешеходных или транспортных потоков не интересны, с точки зрения данной статьи, так как проблема выбора языка в них не стоит.

Инженерное и научное моделирование, то есть механика, химия, системы управления, теплообмен, гидродинамика не относятся к сфере компетенции автора, но проблема выбора языка там определенно существует, поскольку существуют сами языки: блок-схемы MATLABSimulink, различные типы дифференциальных уравнений, численное и аналитическое решение, метод конечных элементов.

Об источниках уродства в моделях. Пример

Для начала расскажем немного упрощенную (и измененную) историю из реальной жизни. Некий разработчик имитационной модели взялся моделировать небольшой участок производства, скажем, сталелитейного, где некие объекты, поступающие по трем входным линиям, должны были перемещаться двумя мостовыми кранами на две выходные линии (рис.1). Краны при этом используют одни и те же рельсы и могут конфликтовать.

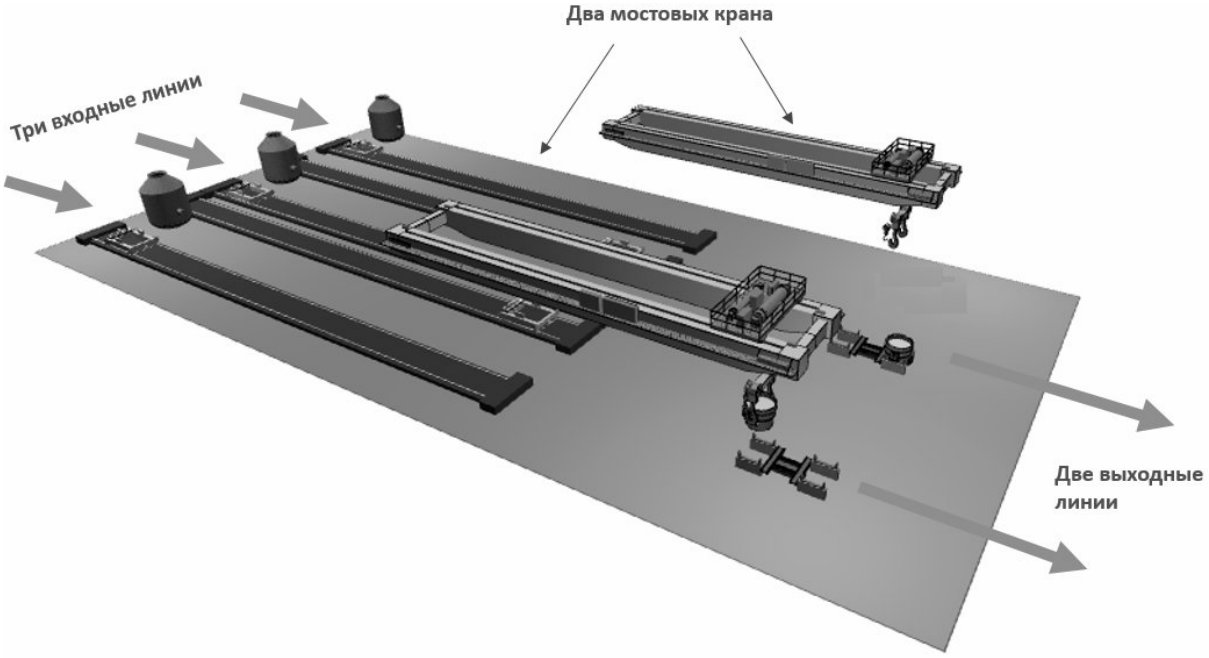

Рис. 1.Моделируемый объект

Выбрав в качестве первоначальной модели входных и выходных линий объект Conveyor, разработчик сконцентрировался на модели кранов. Помня, что нужно идти от простого к сложному, он начал с одного входа, одного выхода, а перемещение краном – представил объектом Delay (задержка) (рис. 2А). Затем он добавил второй вход и оставил пока один кран. Поскольку кран обслуживает две линии, задержка была заменена обслуживанием с использованием ресурса (рис. 2Б). Уточняя модель, разработчик разделил перемещение с грузом и обратно (рис. 2В). Затем ему захотелось, чтобы перемещение крана можно было визуализировать, для чего кран был представлен отдельной заявкой, постоянно циркулирующей в модели (рис. 2Г). Удовлетворившись такой моделью крана, разработчик решил перейти к заданным параметрам производства и добавил по одной входной и выходной линии и еще один кран, также представленные отдельной заявкой. Естественно, перед ним встала задача диспетчеризации кранов и разрешения конфликтов, которые он стал решать, вводя в диаграмму процесса кранов множество условий, ожиданий, дополнительных логических ресурсов. Результат (рис. 2Д) – абсолютно не читаемая диаграмма, целиком состоящая из искусственных конструкций. И естественно, расширение такого подхода на три крана (что потребовал заказчик через некоторое время) оказалось за гранью возможного.

В чем причина неудачи? Разработчик хорошо владел дискретно-событийным моделированием и честно решал поставленную перед ним задачу в рамках данного конкретного подхода. Причина в том, что в этих рамках задача хорошо (красиво, «естественно») не решается. Нужно было подняться на уровень выше и подумать: какие еще средства я могу использовать для простого и удобного описания поведения моделируемого объекта.

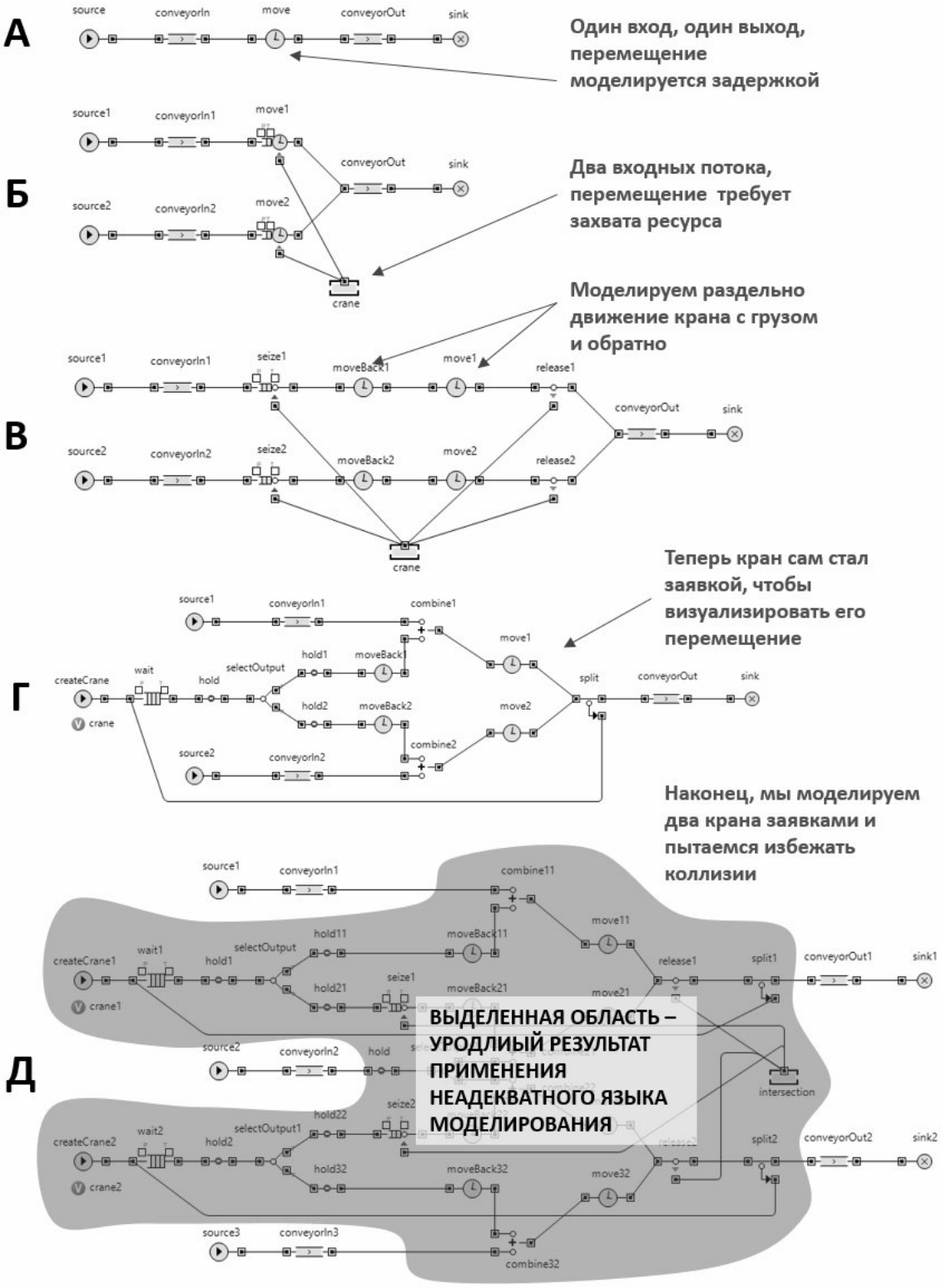

Рис. 2. История моделирования участка сталелитейного производства

Одно из возможных решений показано на рис. 3. Кран моделируется как самостоятельный объект (агент), его состояния, события и действия задаются при помощи стейтчарта (диаграммы состояний), фрагмент основного процесса «перемещение с помощью крана» выделяется в подпроцесс, который выступает в качестве интерфейса основного процесса и двух агентов-кранов.

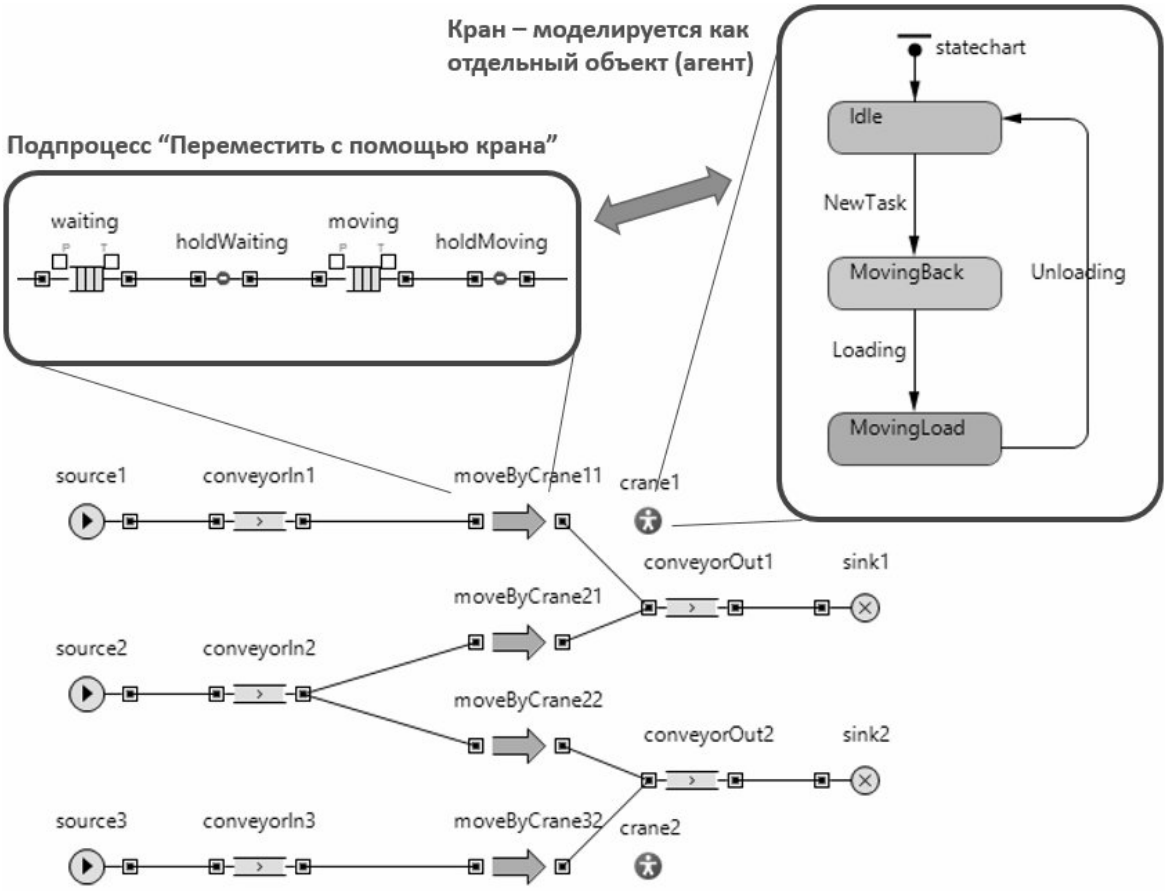

Рис. 3. Модель того же производства с краном в виде отдельного объекта (агента)

Инвентаризация языков имитационного моделирования

Диаграммы процессов (Processflowcharts, Discreteeventsimulation)

Основная идея представления моделируемой системы в виде процесса следующая: попытайтесь выделить в системе объекты (entities, в русской терминологии обычно *заявки*) и последовательность операций, через которую эти объекты проходят. Операции могут включать задержки, обработку, захват и освобождение ресурсов, перемещение и т.п. Задержки, конкуренция за ресурсы и ограниченная пропускная способность обрабатывающих устройств порождают очереди. В стандартный набор блоков в процессном моделировании входят также принятие решения (ветвление процесса), разделение заявки на несколько, слияние, группировка (batch) и разгруппировка (unbatch). Процесс задается графически в виде диаграммы, которая обычно начинается с блока-источника заявок и заканчивается блоком, удаляющим заявки из системы (рис. 4).

Заявки могут представлять клиентов, пациентов, телефонные звонки, документы, детали, транзакции, пакеты (в сети), паллеты, коробки, а также, например, проекты, идеи. Ресурсы – это обслуживающий персонал, доктора, транспортные средства, оборудование, станки, краны, серверы, процессоры. Как заявки, так и ресурсы могут иметь индивидуальные характеристики, от которых зависит протекание процесса.

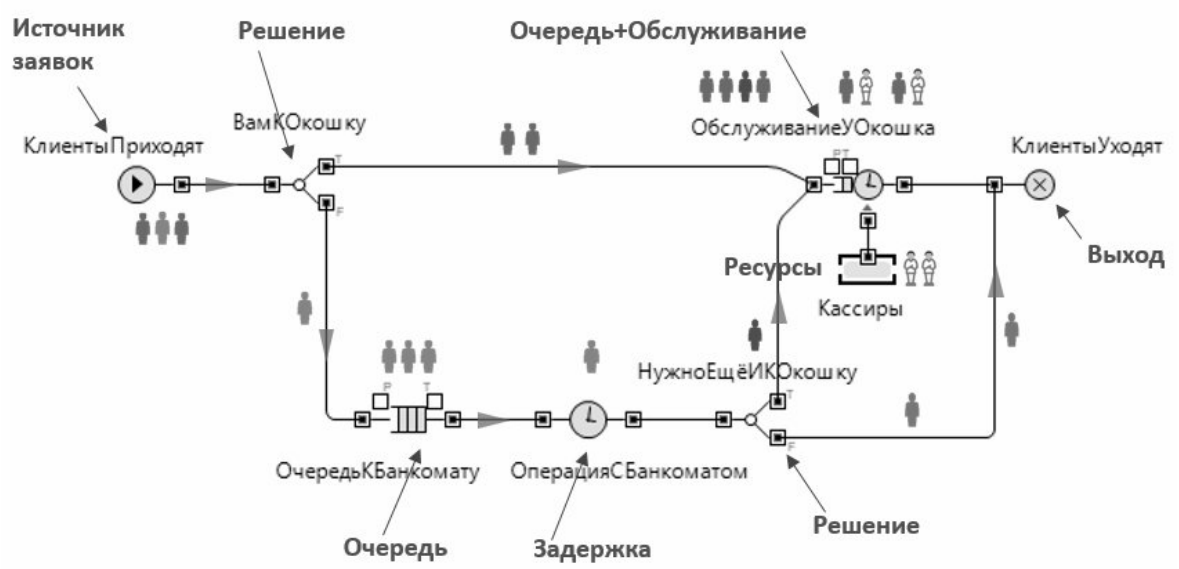

Рис. 4. Процесс обслуживания клиентов в банке

Имитация процессов выполняется в дискретном времени: время прыгает от одного события к другому через неравномерные промежутки, при этом событиями являются появление заявки, начало и окончание обслуживания, захват ресурса и т.п. Поэтому подобный тип моделирования получил название *дискретно-событийное моделирование*, discreteeventsimulation/ modeling (что на сегодняшний момент не является удачным, так как в большинстве агентных моделей время течет точно так же). Времена обслуживания, моменты появления заявок, выход из строя ресурсов обычно задаются через вероятностные распределения, так что абсолютное большинство дискретно-событийных моделей – стохастические.

Естественно, диаграммы процессов более всего приспособлены для описания различных систем обслуживания, в особенности тех, где процесс четко зафиксирован, например, колл-центров, некоторых типов бизнес-процессов. Системы с менее четкими процессами, например, функционирование госпиталя, бывает трудно описать стандартными средствами дискретно-событийного моделирования из-за наличия сложной системы приоритетов, прерываний, прямого взаимодействия между ресурсами. В моделировании производства, как мы видим из примера в предыдущей части, а также в логистике и цепочках поставок процесс также не всегда адекватное средство описания поведения.

Линейные «полудискретные» потоки (DiscreteRates/Flows)

Предположим, система, которую вы моделируете, действительно хорошо представима в виде процесса, но вот только вместо дискретных объектов через процесс протекает жидкость. Или же объекты дискретные, но их очень много, скажем, 100 000 абсолютно одинаковых бутылок в час, а вам нужно моделировать два месяца производства. Для эффективного описания таких систем некоторые инструменты дискретно-событийного моделирования (например, ExtendSim [1], рис.5) предлагают специальный набор блоков, включающий емкость (Tank), вентиль (Valve), смешение (Mixили Merge), разделение (Divergeили Split), источник, сток, а также блоки-конвертеры потоков в дискретные заявки, и наоборот.

В моделях, собранных из таких блоков, значения потоков меняются только по особым событиям (например, заполнение или опустошение емкости, или изменение состояния вентиля), а в промежутках между ними остаются постоянными. При таких *кусочно-постоянных* потоках количество жидкости в емкостях будет *кусочно-линейным*. Имитация процесса не требует численного решения с шагом по времени, и состояние системы в момент следующего события может быть мгновенно и точно вычислено заранее. Поэтому подобное простое расширение дискретно-событийного моделирования в сторону непрерывной динамики является очень естественным. Нелинейная динамика, однако, такой технологией описана быть не может.

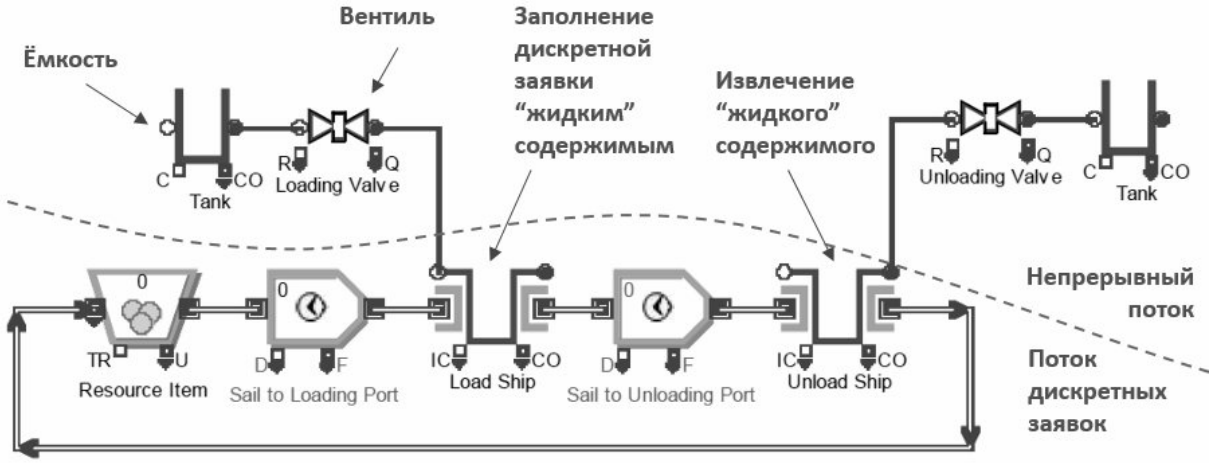

Рис. 5. Простая модель перевозки нефти танкером в ExtendSim

Диаграммы потоков инакопителей (StockandFlow, SystemDynamics)

Вот один из фундаментальных методологических принципов системной динамики: «Смотрите на вещи издалека. Рассматривайте дискретные события и решения как «пену на волнах настоящего поведения системы». Примите парадигму непрерывных изменений, где индивидуальные события и решения смазаны, неотчетливы» [2].

Это призыв радикально повысить уровень абстракции, агрегировать индивидуальные объекты и события и подумать, какие законы могут управлять их количествами. Системная динамика предлагает два основных графических языка: диаграммы причинно-следственных связей (causalloopdiagrams, рис. 6 вверху), не имеющие семантики, достаточной для имитации, и диаграммы потоков и накопителей (stockandflowdiagrams, рис. 6 внизу) – фактически, визуальную форму систем алгебраических и дифференциальных уравнений первого порядка. Таким способом могут быть описаны системы с нелинейной динамикой, а выполнение модели соответственно требует работы численных методов.

Интересно, что, помимо высокоуровневых моделей общества, экономики, финансов, рынков, графический язык системной динамики успешно применяется и для моделирования физических объектов (например, водных ресурсов), а в последнее время – и на микроуровне для моделирования принятия решений отдельным человеком или организацией *внутри агентной модели*, рис. 8.

К преимуществам системной динамики стоит отнести легкость введения в модель новых понятий и связей, а также вычислительные затраты на имитацию, не зависящие от порядка цифр в модели. Расплата за это – стирание индивидуальности объектов (perfectmixassumption), и повсеместно подразумеваемые экспоненциальные распределения.

Системная динамика иногда пытается преодолеть свою непрерывную природу и делает **шаги в сторону дискретности**. Например, понимая, что моделируемые объекты все-таки могут существенно различаться своим поведением, специалисты по системной динамике ввели массивы (индексы, subscripts) и разделили накопители и потоки на ячейки: скажем, мужчины и женщины, этнические группы, регионы. Соответственно, параметры, определяющие поведение, также индексируются. Введение возрастных цепочек (agingchains) – не что иное как разделение людей на дискретные группы по возрасту, имеющие различное поведение. Введение «дискретных» конструкций типа Oven, Conveyor, Queuев инструменте iThink/STELLA [3] – попытка уйти от экспоненциально распределенных задержек. Функции типа sterili pulse – попытка моделировать единичные дискретные события.

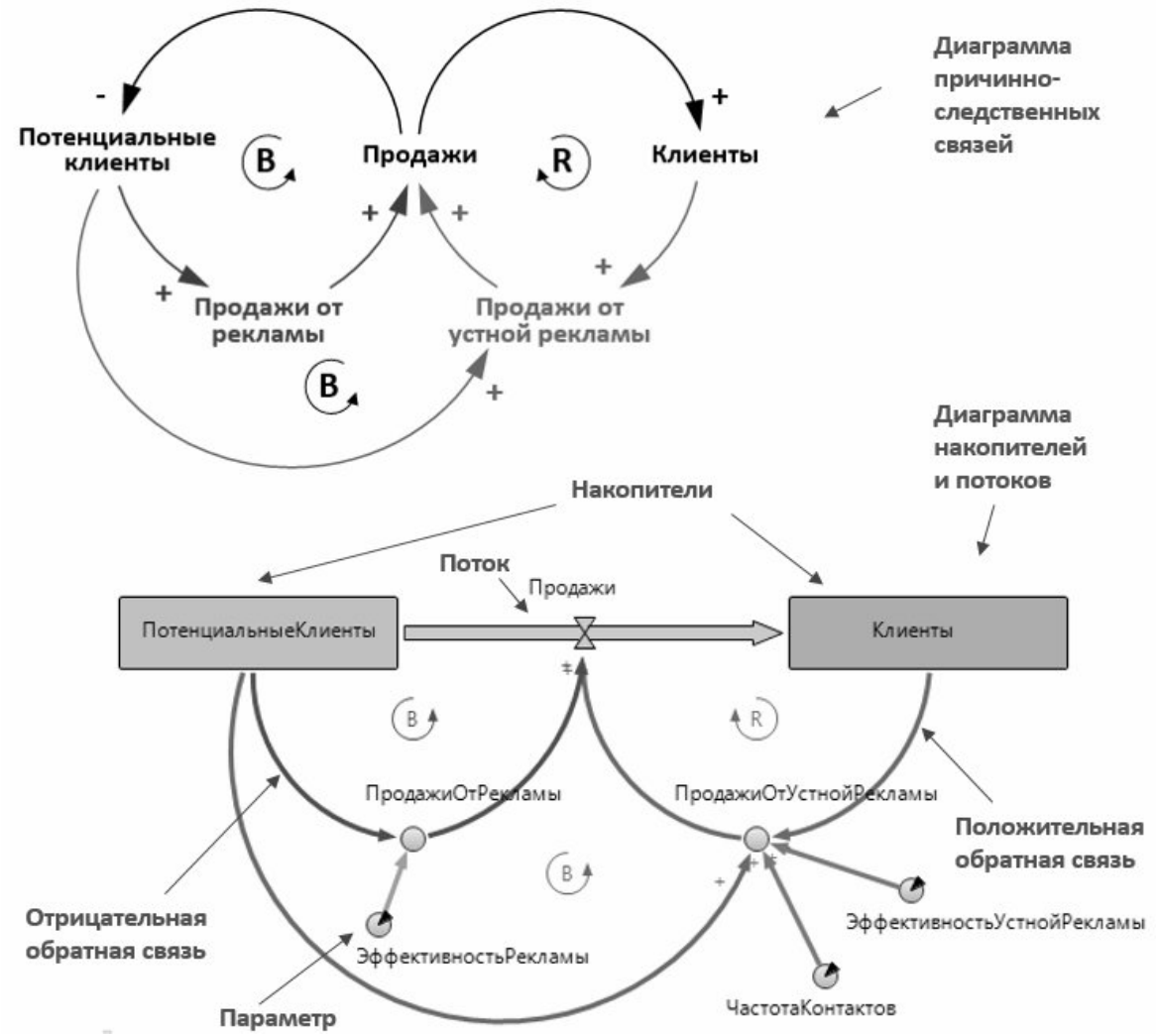

Рис. 6. Диаграммы, использующиеся в системной динамике на примере модели [15]

Если ваша системно-динамическая модель начинает переполняться такими «псевдодискретными» элементами, значит вам пора подумать об использовании процессного (дискретно-событийного) моделирования для всей или для части модели. Любопытно, что некоторые разработчики, пытаясь сделать свою системно-динамическую модель как можно более детальной, создают массивы с таким набором размерностей, что общее количество ячеек начинает превышать реальное количество моделируемых объектов. Скажем, если в модели потребительского рынка Лос-Анджелеса создать 25 этнических групп * 130 микрорайонов * 5 возрастных групп * 2 пола * 5 уровней дохода * 4 уровня образования * 20 видов потребительских предпочтений, то получится 13 000 000 ячеек – больше, чем все население Лос-Анджелеса. Если такой уровень детализации необходим, то, конечно, нужно отказываться от системной динамики в пользу агентного моделирования, при котором вы создадите ровно столько агентов с индивидуальными свойствами, сколько требуется.

Автоматы и их расширения (Statecharts)

Рассмотренные выше методы и визуальные языки разработаны для представления системы в целом (system-levelview). Сейчас мы будем говорить о языке описания индивидуального поведения одного объекта. Необходимость в подобных языках возникла с ростом популярности агентного моделирования [8], которое предлагает ровно противоположный взгляд на вещи (individual-centricview): мы можем не знать законы функциониро-

вания системы в целом, но можем иметь представление о том, как ведут себя ее составляющие – индивидуальные объекты: люди в различных ролях, транспортные средства, станки, продукты, проекты, идеи, организации, активы, инвестиции, квадратные километры земли. Тогда мы задаем поведение этих объектов, объединяем их в систему, а поведение системного уровня выясняется в результате выполнения модели.

Автоматы хорошо подходят для описания объектов, в жизни которых играют роль внешние и внутренние события (в том числе, получение сообщений, изменение условий, задержки и таймауты) и происходит *смена состояний*, то есть набор возможных событий и реакция на них меняются. В каждый момент времени автомат находится *ровно в одном из своих состояний*, и это состояние определяет его реакции и варианты будущего. (В этом отличие автоматов от диаграмм процессов или потоков/накопителей: там состояние системы «размазано» по диаграмме). Если у объекта набор событий и реакций не меняется, автомат будет вырожден в одно состояние, и, возможно, нет смысла его использовать.

В практическом моделировании используются не классические конечные автоматы с «плоской структурой», а их расширение, основой которого послужила работа Дэвида Харела [4], и которое затем было включено в состав UML [5], рис. 7. В этом расширении, которое лучше назвать диаграммой состояний, или стейтчартом (statechart), добавлены вложенные состояния, позволяющие описывать события и прерывание, общие для групп состояний, ветвление переходов, возвращение в прежнее состояние. Интересно, что наиболее «продвинутое» расширение – «ортогональные» состояния, позволяющие распараллеливать (единый для автомата) поток управления на несколько и собирать его обратно, – не получили широкого распространения из-за сложности понимания, что же на самом деле будет происходить.

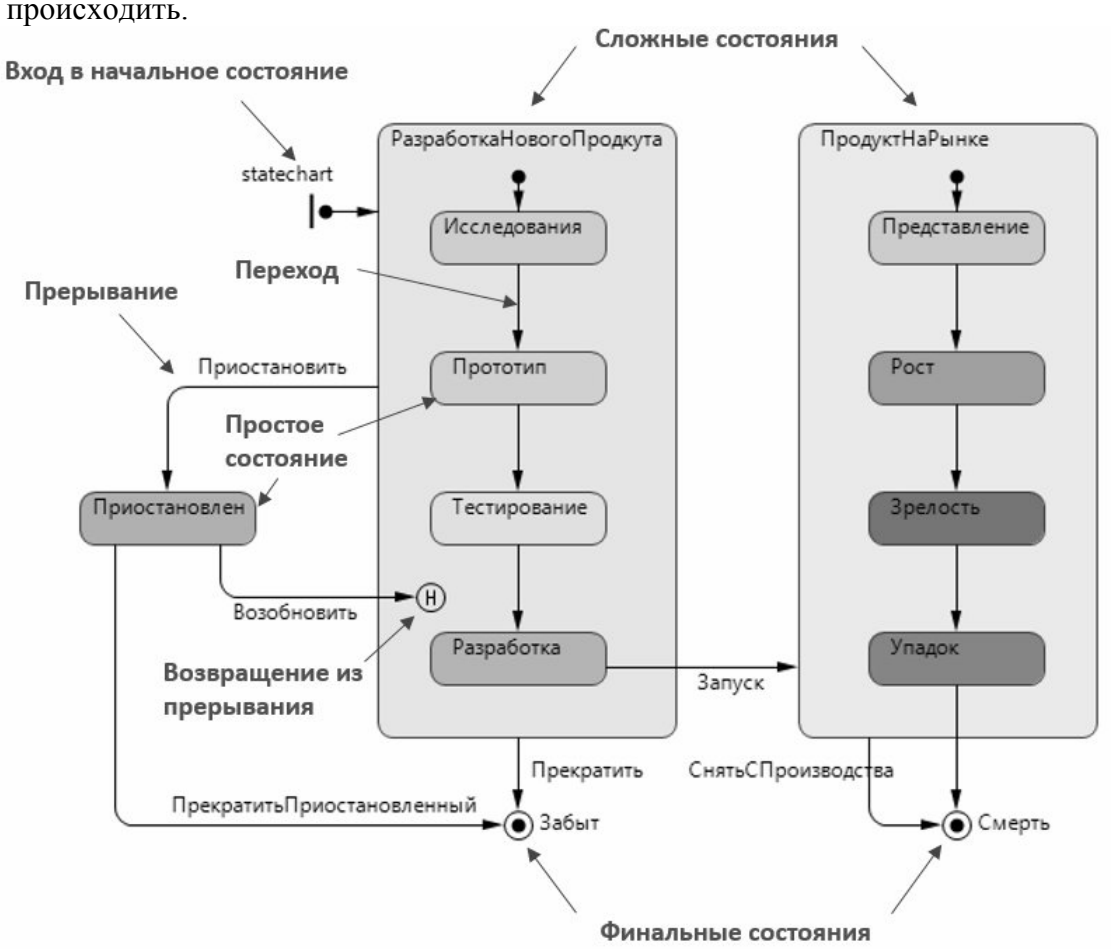

Рис. 7. Диаграмма состояний (стейтчарт) жизненного цикла продукта

.

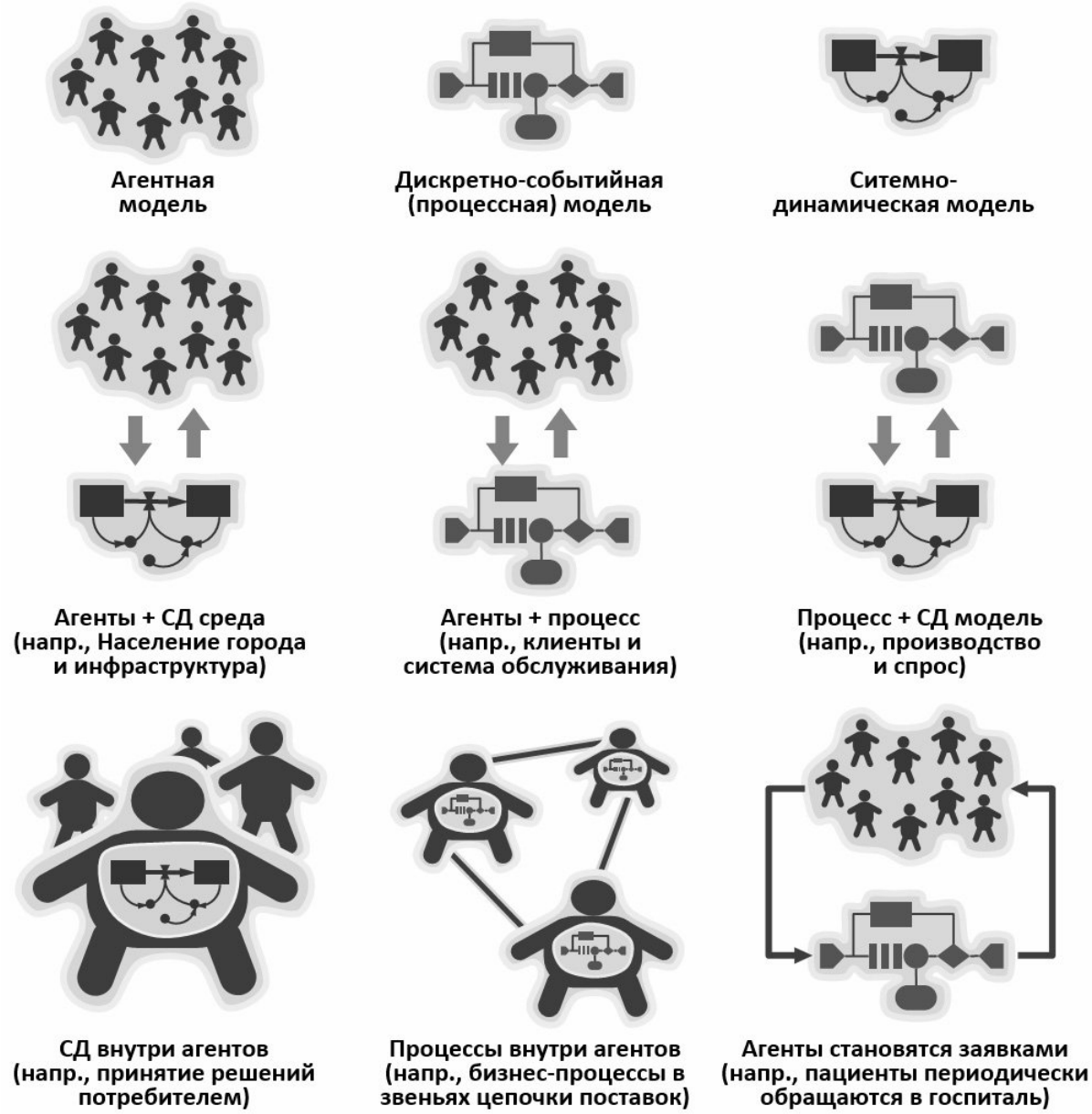

Рис. 8. Наиболее распространенные многоподходные архитектуры моделей

Автоматы хорошо комбинируются с дискретно-событийными моделями, описывая, например, элементы оборудования со сложным поведением. Пример такого взаимодействия – «исправленная» модель производства на рис. 3.

Архитектуры, использующие несколько подходов (Multi-MethodModeling)

Бывает, что решаемая проблема не вписывается полностью в какой-то один подход к созданию моделей. Как мы видим из примера в начале статьи, попытки «стоять до конца» в своем любимом методе, языке, инструменте в таких случаях либо приводит к громоздким и неуклюжим конструкциям, либо разработчик просто оставляет часть проблемы за рамками модели, сужая ее границы.

Проблема границ модели (modelboundary) вообще важная и интересная. Предположим, вы строите модель цепочки поставок, IT-инфраструктуры, или колл-центра. Вас будет интересовать интенсивность поступления заказов, транзакций или звонков, и, скорее всего, вы добьетесь от заказчика каких-то данных, значений, повторяющихся паттернов, тенденций, и вы будете трактовать их как входные данные модели, приходящие извне. На

самом деле, однако, эти величины являются выходами других динамических систем, например, потребительского рынка, или пользователей вашего сервиса. Более того, эти системы находятся во взаимодействии с той, которую вы моделируете, и последняя может сама влиять на них. Например, время цикла в цепочке поставок будет влиять на удовлетворенность клиентов, которая, в свою очередь, влияет на повторные заказы, и, через общение клиентов друг с другом, на рост клиентской базы.

Единственная методология, которая явно ставит проблему выбора границ модели – это системная динамика [6]. Однако очень высокий уровень абстракции, предлагаемый системной динамикой, ограничивает ее применение. Комбинируя различные методы и языки, вы сможете подобрать естественный способ представления для каждого компонента моделируемой системы. Количество таких «многоподходных» архитектур бесконечно, наиболее часто используемые показаны на рис. 8. В статье [7] подробно описано, как эти архитектуры могут быть реализованы в рамках одного инструмента имитационного моделирования AnyLogic [9].

Мертвые языки

Бывает, что язык уже, кажется, убедительно доказал свою нежизнеспособность, а все равно, нет-нет, да и появится статья типа «Как здорово я применил <давно мертвый> язык для моделирования того-то и того-то». Ярким примером такого языка в имитационном моделировании являются сети Петри [10]. Возможно, в обучении теории параллельных вычислений сети Петри и могут быть полезны; также, кому-то может быть забавно играть с ними как с математическим объектом – это сейчас не важно, мы говорим о практическиполезных средствах описания динамики систем. Сетям Петри в 1980х годах прочили в этом качестве большое будущее. Были созданы десятки, если не сотни инструментов моделирования, основанных на сетях Петри [11]. Были придуманы различные расширения сетей Петри: в них добавили время, раскрасили фишки. Но увы: сообщество практиков попробовало и отказалось от этой формы описания. На сегодняшний день использование сетей Петри маргинально и сконцентрировано в нескольких европейских университетах.

Причины фиаско очевидны. «Чистые» сети Петри являются красивой математической конструкцией, для которой разработаны методы анализа, позволяющие проверять свойства сетей *без имитации* их поведения, что, естественно, очень ценно. Однако в таких сетях отсутствуют средства, необходимые для описания реальных систем (скажем, тех же бизнес-процессов) на приемлемом уровне абстракции, а именно: время, задержки, очереди, индивидуальные свойства фишек, условия, приоритеты и т.д. А сети Петри, расширенные в сторону практического моделирования, исключают возможность их неимитационного анализа и начинают превращаться в неудачную, визуально неинтуитивную версию диаграмм процессов.

Вспомогательные средства – код для описания данных и алгоритмов

В любых неигрушечных моделях наряду с графическими конструкциями вы почти всегда встретите текст – куски программного кода. В качестве языка может использоваться специализированный скрипт (например, Siman, SimScript, NetLogoscript), скрипт общего назначения (Python, JavaScript), или язык программирования высокого уровня (Java, C++).

Как дополнение к визуальным формализмам, код бывает уместен и полезен для описания структур данных и работы с ними, то есть условий, алгоритмов, логики принятия решений и т.п. Кроме того, код может быть использован для связи элементов модели тогда, когда это невозможно графически. С точки зрения модельного времени, все «кодовые» вычисления происходят, как правило, мгновенно. Иногда, впрочем, в код включают операторы работы с временем и событиями, такие как wait, waitfor, delay, moveto с ожиданием прибытия и т.д. Мнение автора: вся динамика модели должна быть вынесена в визуальные формы описания, если таковые вообще поддерживаются, а код должен быть оставлен для вычислений, не имеющих временно*й* семантики. Текст, с его последовательной структурой, не приспособлен для описания прерываний, ожиданий с таймаутами, ожида-

ний с вариантами событий. Посмотрите, насколько менее наглядно текстовое описание поведения агента в модели распространения эпидемии, выполненное на языке frabjous (специализированный скрипт, [12]), чем то же самое поведение в виде стейтчарта: рис. 9.

```
discrete model sir
  on startup send "infect" to anyone
  network connections of people by vonneumann
  diagram flu starting with susceptible
    state susceptible displays yellow
      on receive "infect" switch to infectious
    state infectious displays red
      on timeout 10 switch to recovered
      on rate 1 send "infect" to neighbor connections
    state recovered displays blue
  agent people with
  flu
  Population 100
```

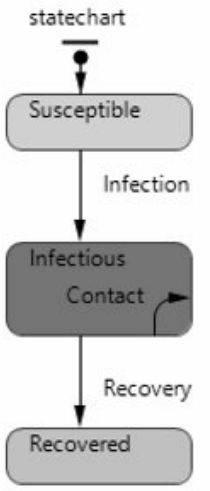

Рис. 9. Текстовое и графическое описание поведения агента в модели эпидемии[16]

Любопытен и обратный пример. Инструмент ExtendSim поддерживал (по крайней мере, в старых версиях) графическую арифметику – вычисления выражений можно было задавать, компонуя в графическом редакторе объекты типа сложения и умножения, которые соседствовали и соединялись с объектами типа задержка или очередь. Такая смесь разнородных, по сути, элементов вряд ли добавляет модели читаемости и является примером неуместного использования графики вместо текста.

Вспомогательные средства – структурирование модели

Помимо собственно средств описания динамики, с увеличением объема модели появляется нужда в средствах ее структурирования и описания связей между элементами. Из средств структурирования модели мы выделим следующие:

Простое деление большой плоской модели на «страницы» или «секторы» – самый примитивный способ структурирования, который применяется, в частности, в некоторых инструментах системной динамики (например, в Vensim [13] и iThink [3]). В тот момент, когда диаграмма потоков, накопителей, вспомогательных переменных и их связей перестает помещаться на экран, разработчик режет ее на части (естественно, пытаясь придать этому разделению смысл). При этом, чтобы линии связей не тянулись от одной части к другой, создаются «тени» переменных (shadowvariables), представляющие переменную из одной части в другой. С точки зрения современных представлений о структурном дизайне, такой способ борьбы со сложностью является чудовищным: связи компонентов оказываются спрятанными, абсолютно непонятно, какие из переменных данного сектора используются в других и т.д. Тем не менее он до сих пор используется.

Простая иерархия (вынесение части плоской модели в отдельный объект).Этот способ применялся в ExtendSim и некоторых других инструментах дискретно-событийного моделирования на ранних стадиях их развития. Часть большого процесса, имеющая какой-то законченный смысл, перемещалась в «подпроцесс», при этом она получала собственный интерфейс – точки входа и выхода заявок. Наличие интерфейса гарантирует возможность модификации внутренностей подпроцесса без опасения нарушить что-то в основном процессе, что является, конечно, большим шагом вперед по сравнению с секторами, описанными выше.

Простая объектная архитектура (объекты параметризуются и могут быть многократно использованы). Это и есть основной способ структурирования, использующийся на сегодняшний день в имитационном моделировании и поддерживаемый большинством современных инструментов. Модель разрабатывается как множество объектов (на самом деле, это уже классы, но мы пока прибережем это слово), каждый из которых имеет четко определенный интерфейс (порты входа и выхода заявок, параметры, возможно, «публичные» функции или переменные). Объекты могут быть использованы в других объектах (то есть поддерживается иерархия произвольной глубины), при этом есть один объект верхнего уровня. Но самое главное, возможно создание произвольного количества экземпляров одного и того же объекта с разными параметрами в разных частях модели. Все современные большие модели используют подобное структурирование.

Объектная архитектура с наследованием.Такой важнейший прорыв в борьбе со сложностью, как механизм наследования, не получил пока большого распространения среди разработчиков имитационных моделей. Даже в агентном моделировании, где он был бы наиболее уместен, люди предпочитают обходиться другими средствами. Например, при создании модели динамики населения было бы вполне естественно иметь базовый класс Person со свойствами, общими для всех людей, и его подклассы Malи Female, добавляющие свойства, специфические для мужчин и женщин. На практике, однако, в большинстве случаев будет создан один класс Person, в нем – параметр sex, а на уровне поведения будут развилки по условию типа sex == MALE. Очевидно, для эффективного использования наследования необходимо «объектно-ориентированное мышление», отсутствие которого, впрочем, не препятствует созданию хороших моделей. Более того, по мнению автора, лучшие разработчики моделей получаются отнюдь не из программистов, а из экспертов в конечных областях применения, которые не любят писать много кодов.

Для описания **связей между объектами модели** помимо собственно ручного рисования их в графическом редакторе может использоваться программное создание и объектов, и связей с чтением структуры из внешнего источника, например, файла Excel, базы данных, или специализированного средства описания сетей, такого как Pajek [14]. Программное создание структур используется в основном в агентных моделях, но не только в них. Иногда так создается диаграмма процесса. Например, в производстве полупроводников процесс может состоять из тысяч операций, последовательность которых будет меняться в зависимости от типа продукта, которых также может быть много. Нарисовать диаграмму такого процесса вручную практически не возможно, поэтому элементы процесса и их связи создаются программно, а описание читается, скажем, из Excel. Естественно, инструмент моделирования должен поддерживать программное создание и динамическое связывание объектов.

Кодирование моделей «с нуля»

Бывает, что имитационные модели пишут «с нуля» на обычном языке программирования, таком как C++ илиJava. В 90% случаев это время и деньги, потраченные зря. Типичные случаи такие.

В государственных организациях и больших компаниях. Программисты содержатся в штате, им нужно за что-то платить зарплату, они, естественно, тоже хотят, чтобы было за что, и любыми способами стараются поставить организацию в зависимость от себя. Изучению новых технологий препятствуют *лень* и *страх*: «Я эксперт в C++, а кто я буду в новом незнакомом инструменте имитационного моделирования?» Поэтому аргументация программистов такая: зачем нам тратить 15-20 тысяч долларов на какой-то софт, который еще нужно осваивать, да еще непонятно, подойдет ли он для нашей задачи? Мы сами все напишем с нуля, сразу учитывая специфику нашего бизнеса, все будет работать быстро, и выйдет дешевле. На самом деле выйдет дороже и работать не будет. Типичный результат – месяцы кодирования, тысячи строк кода, убогий интерфейс, баги, долгие болезненные мо-

дификации, и все это выбрасывается, как только разработчик покидает компанию. Никакое программирование на коленке не может конкурировать с профессиональными графическими средами создания моделей, быстрыми, отлаженными в течение лет «движками» и средствами визуализации, а самое главное – с коммерческой поддержкой продуктов.

В университетах. Во-первых, та же лень разбираться в чужой технологии, подкрепленная волнующим предвкушением начать что-нибудь с чистого листа. Кроме того, *амбиции*: в академической среде очень часто цель – не решить проблему, а создать видимость, что ты умнее других. В этом смысле фраза «Мы создали новый уникальный подход в ИМ» звучит круче, чем «Мы взяли MATLAB или VenSim и создали модель, которая позволила выбрать/понять/сэкономить…» Кроме того, если взяться за практическую задачу, то кто его знает, получится или нет. А «новый уникальный подход» все равно никто смотреть не будет. Отсюда такое количество бессмысленных наукообразных разработок студентов, руководимых бессовестными преподавателями.

Автор настаивает на том, что имитационное моделирование – область сугубо прикладная, и любые разработки новых подходов, языков, «теорий» должны быть обоснованы *неприменимостью существующих подходов к стоящим перед нами практическим задачам*. Поэтому, когда мы слышим о какой-нибудь очередной «полифонии мультимодельных конгломератов», то можем быть уверены в том, что деятельность ее создателя не сэкономила никому ни рубля, а самому ему, конечно же, принесла пару публикаций (которых нужно бы стыдиться) и незаслуженную ученую степень.

Есть ли случаи, когда кодирование модели «с нуля» оправданно? Безусловно. Бывает, что задача стоит так, что ни один из существующих языков не способен эффективно описать моделируемую систему. Вот тогда и нужно пробовать создавать специальный инструмент. Если задача при этом окажется достаточно общей, это, возможно, приведет к созданию нового направления в моделировании. В свое время в эпидемиологии, исследовании потребительских рынков, социальных науках возникла потребность моделировать поведение каждого человека индивидуально, с учетом его уникальных параметров, во взаимодействии с другими людьми и средой. По этому запросу постепенно сложился ряд технологий, которые теперь в совокупности называются агентным моделированием.

Заключение

Как следует из содержания статьи, автор считает эстетические качества исходного кода модели (или программы) необходимым условием ее полезности. Важнейшими из них являются естественность и минимализм (отсутствие лишних деталей). Принцип бритвы Оккама «не следует привлекать новые сущности без крайней на то необходимости» – вообще фундаментальный методологический принцип всего моделирования. Под естественностью автор понимает следующее: если то, что у вас получилось, кажется вам не вашим хитроумным изобретением, а конструкцией, «существовавшей всегда», которую вам наконец удалось найти, то это конструкция естественная.»Программировать надо не *на* языке, а *для* языка» [17] – это высказывание также на 100% применимо к созданию имитационных моделей.

Тем не менее насколько абсолютен принцип «все гениальное – просто»? В частной беседе этот вопрос был задан автором профессиональному математику. На что был получен ответ: «Не всегда!» Математик привел примеры фундаментальных теорем, доказательства которых многостраничны, изобилуют рассмотрением множества частных случаев и, в общем, не удовлетворяют эстетическому критерию, описанному выше. То же и в моделировании. Бывают случаи, когда особенности моделируемой системы и поставленной задачи не позволяют создавать абсолютно стройные и прозрачные модели. Например, в одной из моделей железнодорожных перевозок, выполненных нашей компанией, пространство, остающееся в вагоне после загрузки его основным видом груза, в редких случаях при выполнении определенных условий могло заполняться другим грузом. Эта особенность повлекла за собой существенное усложне-

ние алгоритма оптимизации и рост общего объема исходного кода модели, непропорциональный проценту случаев, когда это происходило в реальном мире.

Подобные исключения неизбежны. Но каждый раз, когда то, что вы создаете начинает расти в объеме и сложности, вы обязаны задаться вопросом: а нельзя ли это упростить? Не могу ли я отказаться от части переменных, блоков, состояний? Чаще всего на текущем этапе разработки выкинуть что-либо бывает непросто, и тогда вы должны вернуться («откатиться») назад, и, возможно, пересмотреть архитектурные решения, принятые ранее. Выбросить часть диаграммы процесса и заменить его на диаграмму состояний или, скажем, поменять системно-динамическую модель на агентную будет страшно: вы задумаетесь об усилиях потраченных зря и о времени, которое предстоит потратить на пересоздание модели или ее части. Но это единственный (и самый эффективный!)путь построения полезных моделей. Не бойтесь итераций с выбрасыванием части вашего кода и неизбежным «рефакторингом» оставшейся части. Подобные итерации не удлиняют, а сокращают время выполнения проекта. И – имейте в своем арсенале больше выразительных средств (языков).

Литература

1. **Krahl D.** 2009. ExtendSim Advanced Technology: Discrete Rate Simulation. In: Proceedings of the 2009 Winter Simulation Conference M.D. Rossetti, R.R. Hill, B. Johansson, A. Dunkin and R.G. Ingalls, eds.
2. System Dynamics Society. The Field of System Dynamics. www.systemdynamics.org/what-is-s/isee systems – Vendor of iThink and STELLA simulation software. www.iseesystems.com
3. **Harel, D.**1987. *Statecharts: A Visual Formalism for Complex Systems*.Science of Computer Programming, Volume 8, Issue 3, June 1987.
4. *UML State Machine – статья в Википедии*. en.wikipedia.org/wiki/UML_state_machine
5. **Sterman, J.** 2000. Business Dynamics: Systems Thinking and Modeling for a Complex World. Irwin/McGraw-Hill.
6. **Borshchev, A.** 2013. Multi-Method Modeling. In: Proceedings of the SIMEX'2013.
7. **Borshchev, A, and A. Filippov.** 2004. *From System Dynamics and Discrete Event to Practical Agent Based Modeling: Reasons, Techniques, Tools*. In: Proceedings of the 22nd International Conference of the System Dynamics Society, Oxford, England.
8. AnyLogic – Multi-Method Simulation Software. anylogic.com
9. *Сети Петри – статья в Википедии*. ru.wikipedia.org/wiki/Сети_Петри
10. **Petri Nets Tools Database Quick Overview**.
www.informatik.uni-hamburg.de/ TGI/PetriNets/tools/quick.html
11. **Schneider, O., Dutchyn, C. & Osgood, N.** 2012, *Towardsfrabjous: a two-level system for functional reactive agent-based epidemic simulation*. In: Gang Luo; Jiming Liu & Christopher C. Yang, ed., IHI '12 Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium.
12. Vensim – Simulation software. vensim.com
13. Pajek – Program for Large Network Analysis. pajek.imfm.si
14. **Bass, F.** 1969. A new product growth model for consumer durables. Management Science 15 (5).
15. *Compartmental models in epidemiology – статьявВикипедии*. en.wikipedia. org/wiki/Compartmental_models_in_epidemiology
16. К сожалению, автору не удалось найти источник этой цитаты, она принадлежит одному из теоретиков программирования 80-х годов XX века.