УЧРЕЖДЕНИЕ РОССИЙСКОЙ АКАДЕМИИ НАУК
ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР
ИМ. А.А. ДОРОДНИЦЫНА РАН

Ю. И. БРОДСКИЙ

РАСПРЕДЕЛЕННОЕ ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ СЛОЖНЫХ СИСТЕМ

ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР ИМ. А.А. ДОРОДНИЦЫНА
РОССИЙСКОЙ АКАДЕМИИ НАУК
МОСКВА 2010

УДК 519.876

Ответственный редактор
член-корр. РАН Ю.Н. Павловский

Делается попытка ввести формализованное описание моделей некоторого класса сложных систем. Ключевыми понятиями этой формализации являются понятия компонент, которые могут образовывать комплекс, и комплекса, состоящего из компонент, но на некотором более высоком уровне абстракции могущем восприниматься как единая компонента.

Предлагаемая концепция моделирования, во-первых, помогает построить синтез сложной многокомпонентной модели, что само по себе обычно является нетривиальной задачей, и во-вторых, позволяет создать в Интернете пиринговую сеть имитационного моделирования, в которой, с одной стороны, каждый участник сети может предоставлять во всеобщий доступ (опубликовывать) разработанные им методы, а с другой стороны, может использовать в разрабатываемых им моделях подходящие методы, разработанные и опубликованные в Интернете другими участниками сети моделирования.

Монография издается в авторской редакции.

Ключевые слова:
имитационное моделирование, распределенные вычисления, сложные системы, метакомпьютинг, GRID-технологии, распределенные вычисления в Интернете.

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований, грант 10-07-00176-а.

Рецензенты: Ю.А. Флеров,
Б.А. Суслаков

Научное издание

Предисловие

В книге делается попытка ввести формализованное описание моделей некоторого класса сложных систем, постепенно сложившееся в результате многолетнего опыта моделирования таких систем в отделе «Имитационные системы» ВЦ РАН, начиная с 80-х гг. прошлого века. Ключевыми понятиями этой формализации являются понятия компонент, которые могут образовывать комплекс и комплекса, состоящего из компонент, но на некотором более высоком уровне абстракции могущем восприниматься как единая компонента. В результате сложную систему можно рассматривать, начиная с одной компоненты на самом высоком уровне абстракции, и кончая фракталом компонент, на уровне максимально подробного моделирования. Степень подробности ограничивается лишь желанием разработчика модели.

Предлагаются методы анализа и синтеза многокомпонентных моделей, ориентированные на распределенные и параллельные вычисления. Устанавливается связь объектно-событийного моделирования и моделирования средствами классических динамических систем.

Предложенная концепция моделирования сложных многокомпонентных систем практически реализована в виде описываемого в книге макета инструментальной системы распределенного имитационного моделирования, основу которого составляет программное обеспечение рабочей станции пиринговой сети моделирования. С помощью рабочей станции можно публиковать в Интернете разработанные алгоритмы методов модели, описывать устройство и логику поведения моделей сложных систем и их компонент, а также собирать и запускать на имитационный эксперимент модели, состоящие как из компонент собственной разработки, так и из подходящих «чужих», опубликованных другими участниками сети.

Предлагаемое программное обеспечение позволяет развернуть в Интернете пиринговую сеть имитационного моделирования, которая способна (по крайней мере, теоретически) предоставить возможность различным научным коллективам обмениваться результатами своей деятельности в области моделирования сложных систем, а также интегрировать свои разработки в совместные распределенные модели.

Автор благодарен всем, без кого эта работа не могла бы состояться. Прежде всего – безвременно ушедшему Владимиру Юрьевичу Лебедеву, в плодотворном сотрудничестве и соавторстве с которым она начиналась.

Многолетнее постоянное общение с Юрием Николаевичем Павловским, обсуждение с ним различных подходов к построению имитационных моделей сложных систем, было определяющим в формировании предлагаемой в этой книге концепции моделирования.

Обсуждения проблем системного моделирования сложных процессов с Геннадием Ивановичем Савиным и Владимиром Федоровичем Огарышевым много лет назад, в период вхождения автора в тему моделирования сложных систем, были хотя и непродолжительными во времени, но весьма плодотворными и определившими для автора выбор направления его исследований на многие последующие годы.

Доброжелательное внимание и опека Александра Петровича Афанасьева и членов возглавляемого им коллектива позволили автору безболезненно войти в относительно новую для себя тему распределенных вычислений.

Автор благодарен своим коллегам по ВЦ РАН Николаю Вадимовичу Белотелову, Виктору Александровичу Горелику, Николаю Николаевичу Оленеву, Елене Ивановне Туголуковой за многолетнее плодотворное сотрудничество и всестороннюю поддержку.

Автор также благодарен своей семье, которая мужественно и стойко терпела его погруженность в эту работу.

Введение. Моделирование сложных систем: искусство, наука, технология

Большая Советская энциклопедия (статья Н.П. Бусленко) дает следующее определение сложной системы: «Сложная система – составной объект, части которого можно рассматривать как системы, закономерно объединенные в единое целое в соответствии с определенными принципами или связанные между собой заданными отношениями» [39].

Предметом данной книги будут такие сложные системы, про которые достаточно хорошо известно, из каких компонент они состоят, что эти компоненты умеют делать, по каким правилам они взаимодействуют друг с другом. Декомпозиция этих сложных систем на компоненты задается самой предметной областью моделирования, можно сказать, что «так они устроены». Проблемой при этом, и часто весьма нетривиальной, является воспроизведение поведения и оценка возможностей такой сложной системы в целом.

В настоящее время моделирование такого сорта чаще всего называют агентным, однако во времена, когда автор начинал заниматься моделированием сложных систем [32 – 34], этого термина еще не было, и мы называли свои модели объектными или объектно-событийными.

Зачем моделировать сложные системы

Давным-давно, за двести с лишним лет до нашей эры, первый китайский император Цинь Шихуанди захотел раз и навсегда решить проблему постоянных набегов северных кочевников на оседлых китайцев, и приказал построить Великую Китайскую Стену. Стену построили довольно быстро – за десяток с небольшим лет, однако это было сопряжено со столь значительными жестокостями властей и трудностями для народа, что до сих пор, через две с лишним тысячи лет, воспевается в китайских песнях и эпосе. Примерно понятно, какие

соображения в пользу проекта были у императора, также понятно, что опыта боевого применения подобных сооружений не было ни у кого – Великая Китайская Стена до сих пор уникальна. Не было тогда и средств научного анализа уникальных проектов таких, как например, имитационное моделирование. В результате, вскоре после смерти императора, жизнь показала полную неэффективность Стены. Северные кочевники форсировали ее, дошли до столицы и сменили правившую династию своей. С тех пор и доныне Великая Китайская Стена – туристический объект мирового значения и предмет воспевания народного эпоса Китая.

Сходный по масштабу проект возник сравнительно недавно, уже на памяти автора, в конце прошедшего XX века. В марте 1983 Р. Рейган выдвинул Стратегическую Оборонную Инициативу (СОИ) – долгосрочную программу создания системы противоракетной обороны (ПРО) с элементами космического базирования, позволяющую поражать стартующие баллистические ракеты а также наземные цели из космоса.

Политикам и военным 80-х годов прошлого столетия было важно знать, как будет функционировать и на что способна СОИ, Ответить на этот вопрос без имитационного моделирования оказалось невозможным. При этом экспертам в данной предметной области было вполне понятно, из чего она могла бы быть сделана – в основном, из того, что реально выпускалось промышленностью того времени, и в значительно меньшей степени из того, что могло бы быть разработано и выпущено специально для нее в ближайшие за началом проекта годы. А это уже вполне реальные объекты моделирования с достаточно хорошо известными заинтересованным лицам тактико-техническими характеристиками. Также достаточно хорошо было известно, как эти объекты должны взаимодействовать между собой, что позволяет осуществить их синтез в целостную модель. После этого остается только запустить имитационный эксперимент и наблюдать, что из этого получается.

Отмеченный выше интерес к возможностям СОИ был вовсе не данью праздному любопытству, – реализация этого проекта серьезно напрягла бы ресурсы целой нации на десятилетия, и было бы очень обидно получить в результате вторую Великую Китайскую Стену. Как известно, программа СОИ так и не была развернута, причем не в последнюю очередь из-за того, что имитационное моделирование показало ее неэффективность против одновременного массового запуска баллистических ракет.

Приведенный выше пример интересен еще и тем, что причина неэффективности СОИ кроется в законе сохранения момента количества движения – законе известном многим еще со школы. Однако непосредственно, без имитации, увидеть проявление этого закона в данной предметной области, насколько известно автору, не сумел никто из тогдашних ее исследователей. Имитационные же эксперименты достаточно быстро вывели на упомянутый закон – нужно было как-то объяснять поведение модели, достаточно странное и неожиданное для ее создателей. Например, почему решение задачи целераспределения удара по космической группировке противника всегда отдает подавляющее преимущество наземным, а не космическим ударным средствам. Или почему если все-таки принудительно сделать основным ударным средством космическую группировку (ведь столько было тогда говорено именно о «звездных войнах»), то нанесение удара по космическим объектам противника иногда начинается лишь через несколько часов(!), после отдания соответствующего приказа. Причина этих «странностей» в том что энерговооруженность ракет, стартующих с боевой станции космического базирования весьма невелика в сравнении с имеющимся у них моментом количества движения (станции движутся по орбите с первой космической скоростью). В результате эти ракеты не могут всерьез вырваться из плоскости орбиты станции и способны оперировать лишь в некоторой окрестности этой плоскости.

Если же нужно попасть в точку орбиты, лежащей совсем в другой плоскости, да еще тогда, когда в этой точке что-то есть, то с имеющейся небольшой энерговооруженностью эта точка не может быть произвольной, а должна быть окрестностью точки пересечения чужой орбиты и своей плоскости, и время подлета к ней тоже не произвольно – оно должно совпадать с появлением в ней интересующего объекта, кроме того интервал времени должен быть достаточным для маневра. Поэтому если необходимый маневр вообще возможен, иногда его начала приходится ждать несколько часов (даже если за это время может развернуться целый эпизод «звездных войн»).

Остается лишь порадоваться, что все ограничилось тогда объяснениями странностей поведения виртуальной системы – до разочарований возможностями реальной, в которую вложены бешеные силы и средства, дело так и не дошло!

Искусство, наука и технология моделирования сложных систем

> Если мы описываем увиденное и известное по опыту на языке логики — это наука; если же представляем в формах, внутренние взаимосвязи которых недоступны нашему сознанию, но которые интуитивно воспринимаются как осмысленные, — это искусство.
>
> А. Эйнштейн

Искусство и наука – два дополняющих друг друга способа познания мира, интуитивный и аналитический – гласит высказывание А. Эйнштейна, взятое эпиграфом. Автор склонен добавить к двум этим противоположным началам третье – технологию, являющуюся в определенной мере их порождением и синтезом. В нашем познании мира эти три начала участвуют вместе, взаимно влияя и обогащая друг друга. Можно назвать эти три начала и немного по-другому: интуиция – ло-

гический анализ – практическая деятельность, или интуитивное знание – формализованное знание – практическое знание.

Интуиция, время от времени приносит нам диковины, выловленные из океана непознанного, наука пытается построить к ним мост с берега познанного, тем самым все время расширяя его территорию (а океан впереди все равно попрежнему бесконечен), технология обустраивает берег прекрасными зданиями.

Нарисованная выше идиллическая картина на самом деле в жизни не столь гармонична. Каждое из упомянутых трех начал имеет свою собственную логику развития, отличную от двух других. Например, технология однажды будучи рожденной союзом искусства и науки, затем в своем развитии очень часто обращается к науке с возникающими в процессе этого развитии проблемами, однако не столь часто получает ответы на свои вопросы. Практическая деятельность чаще всего ставит перед наукой задачи неразрешимые на настоящем ее уровне, тем самым давая развитию последней мощный стимул. Часто в таких ситуациях выручает искусство-интуиция. Находятся эвристические решения, часто не имеющие обоснования своей общности или оптимальности или даже существования, но в конкретных применениях технологии работающие более-менее исправно. С точки зрения практики – всегда лучше иметь плохое решение проблемы, чем не иметь никакого. Очень часто такая ситуация, когда формально-логическое обоснование не успевает за связкой интуиция-практика наблюдается на прорывных участках нашего познания.

Например, одно из величайших достижений познания нового времени – изобретение математического анализа Ньютоном, Лейбницем и некоторыми их современниками было подвергнуто серьезной и справедливой критике со стороны философа и епископа Беркли за недостаточную логическую обоснованность. Как известно, безупречное формальное обоснование математического анализа последовало через сотню с

лишним лет, при этом пришлось пожертвовать первоначальной простотой и наглядностью некоторых его формулировок. А безупречное с математической точки зрения обоснование математического анализа в исходных формулировках, с использованием инфинитезимальных (неархимедовых) чисел, появилось вообще во второй половине XX века. Все это время математическим анализом активно пользовались – он стал основой естественных наук.

Возникает вопрос, а столь ли важно формально-логическое обоснование интуитивных постижений? Вряд ли Ньютон с Лейбницем понимали суть матанализа хуже, чем Коши с Вейерштрассом. По-видимому (см., например, [11]), если бы Ньютону с Лейбницем показали современный учебник матанализа, они бы сказали, что вообще-то и они все это прекрасно знали и понимали, и кроме того, знали и умели еще многое, что не вошло в этот учебник.

На взгляд автора, формально-логические обоснования и учебники нужны в первую очередь не адептам искусства и интуиции (хотя бывают полезны и им: интуиция всегда нуждается в проверке, чтобы Беркли потом не очень приставали), а следующим за ними поколениям студентов науки. Формализация – язык науки, благодаря чему передача научных знаний – технология, по времени занимающая семестры. У искусства и интуиции нет такого языка, поэтому передача искусства – таинство, и занимает десятилетия (если вообще происходит).

Моделирование сложных систем в настоящее время находится на той стадии своего развития, когда опережают технология и искусство, а наука пока отстает от них. Построено много успешно функционирующих моделей сложных систем, имеются и инструментальные системы моделирования, однако методы построения таких моделей и систем в основном эвристические, штучные для каждой такой реализации. Очень часто в подобной ситуации использование сложных и недостаточно формально обоснованных эвристических методов, ведет

к стремлению компенсировать их недостаточную обоснованность рассмотрением максимально широкого класса потенциально возможных случаев их применения. В результате рассматриваются в качестве возможных такие ситуации, которые на самом деле, как показывает последующий формальный анализ, возникнуть не могут. Анализируются и решаются проблемы, которых на самом деле нет. Такие пустые страхи автор привык называть «фобиями».

Еще одно следствие отставания формализации от связки практика-интуиция – появление в сложных проектах так называемых «архитектурных излишеств». Дело в том, что построить модель по-настоящему сложной системы – дело очень и очень непростое, а в начале проекта может казаться даже почти невозможным. Поэтому разработчики, как правило, не отказывают себе ни в каких средствах достижения цели. Чтобы хоть что-то получилось – будем применять все, что умеем. В результате в средствах построения моделей возникает определенная избыточность, многие вещи можно реализовывать несколькими способами, и неочевидно какой из них лучше.

Всякое излишество не дается даром. Во-первых, его надо реализовывать, а это дополнительный труд. Во-вторых, избыточную концепцию гораздо труднее изучать. Пользователю, особенно начинающему, гораздо проще знать единственный надежный путь к цели, чем выбирать из многих, различия между которыми он улавливает с трудом.

Цель данной работы – попробовать внести в тему моделирования сложных систем начала формализации и затем довести эти начала до практической реализации в виде инструментальной системы распределенного имитационного моделирования сложных систем.

Предлагается оттолкнуться от положений, признаваемых бесспорными большинством исследователей, работающих в области моделирования сложных систем, и попытаться формально вывести из них концепцию моделирования, избегая

при этом по возможности «фобий» и «архитектурных излишеств».

Итак, в свете сказанного выше, можно охарактеризовать содержание данной книги следующим образом: первая глава посвящена в основном искусству моделирования – обзору существующих ныне способов синтеза сложных систем. Обсуждаются различные проблемы, возникающие при таком синтезе. Поскольку эти способы синтеза сложных систем носят эвристический характер, они отнесены к искусству.

Во второй главе делается попытка построить языковую среду для формализации процесса синтеза модели сложной системы. Дается ряд определений, устанавливаются связи между получившими определения понятиями. В результате формального анализа удается решить ряд проблем, обсуждавшихся в первой главе, еще несколько проблем на самом деле оказываются псевдопроблемами. Предлагается в значительной мере формально обоснованная концепция моделирования сложных систем. Поэтому вторую главу можно отнести к науке моделирования.

Возможно, кому-то из экспертов в области моделирования предлагаемая «наука» покажется слишком простой и очевидной. В значительной мере так оно и есть – ведь это одна из первых подобных попыток. Тем не менее, как показывают приведенные примеры, эта попытка работает! Кроме того, как утверждалось выше, – формализация в первую очередь нужна не экспертам (им и так все или почти все ясно – на то они и эксперты!), а следующим за ними поколениям студентов, чтобы за относительно короткий срок обучения выйти на уровень владения предметом, приближающийся к уровню экспертов.

Наконец, третья глава посвящена технологии моделирования. Рассматривается компьютерная реализация концепции моделирования, описанной во второй главе – инструментальная система распределенного имитационного моделирования, предназначенная стать инструментом этой технологии.

Глава I. Распределенное моделирование сложных систем: проблемы и решения

1.1. Проблемы имитационного моделирования сложных систем

Коль скоро мы заявили, что предметом этой книги в первую очередь будут модели таких сложных систем, про которые достаточно хорошо известно, из каких компонент они состоят, что эти компоненты умеют делать, по каким правилам они взаимодействуют друг с другом, – здравый смысл сразу же подсказывает основой принцип организации вычислений в такой модели – дать возможность каждой из компонент проявить себя в полной мере, и наблюдать затем, что будет происходить при этом со всем многокомпонентным комплексом.

В самом деле, как мы убедимся далее, большинство систем моделирования основывается именно на этом принципе. Однако воплотить его в жизнь оказывается не так-то просто: при синтезе компонент в сложную систему возникает ряд проблем. Одна из таких важных и нетривиальных проблем – управление модельным временем.

1.1.1. Управление временем в имитационных системах

Простейший способ такого взаимодействия – дискретная модель с постоянным шагом моделирования, примеры таких моделей приводятся в первых главах. Однако постоянный шаг моделирования не всегда удобен, особенно при моделировании сложных, многокомпонентных явлений: различные компоненты модели имеют свои собственные характерные масштабы времени. Например, мы хотим моделировать взаимодействие двух военных блоков, находящихся на пороге войны и затем начинающих воевать. Тогда в мирное время, характер-

ным временем изменения экономических и военных характеристик сторон будет квартал, а то и год, если в странах проводится всеобщая мобилизация и перегруппировка войск – характерным временем будет неделя, если же начались боевые действия, счет пойдет на дни, если в боевых действиях участвует авиация – на часы, а если произведен пуск баллистических ракет и работают системы ПРО – уже на секунды, т.е, характерный масштаб измерения времени сложной модели в ходе имитационного эксперимента может изменяться на несколько порядков. Поэтому достаточно «серьезные» инструментальные системы имитации должны предоставлять средства для моделирования с переменным шагом времени.

Особенно серьезные проблемы с управлением модельным временем возникают, если параллельно выполняется несколько моделей, каждая со своим собственным управлением модельным временем, и при этом они обмениваются сообщениями, которые способны существенно изменять поведение получившей такое сообщение модели. Если брать предметную область моделей, которым посвящена эта книга – такое там происходит постоянно.

Возникает далеко не тривиальная проблема обеспечить правильное течение модельного времени всего комплекса. Нельзя, например, допустить, чтобы кто-то послал кому-то сообщение, жестко привязанное к определенному модельному времени, к примеру начинать в заданный модельный момент некую важную операцию, а получатель уже прожил в своей модели этот момент, и важное сообщение пришло в его прошлое.

Относительно этой и подобных связанных с управлением временем проблем существует немало решений. Мы не будем здесь их предварять, укажем лишь литературу, так или иначе посвященную этой теме: [3], [4], [6], [7], [8], [43], и в дальнейшем обратим внимание на то, как они решаются в приводимых ниже примерах.

1.1.2. Управление данными и ходом имитационных вычислений в системах моделирования

Кроме обмена сообщениями, параллельно протекающие процессы обычно обмениваются и данными (впрочем, сообщения – тоже данные). Здесь тоже может быть ряд серьезных проблем, связанных, например, с попытками нескольких параллельных процессов одновременно модифицировать одни и те же данные, или же попыткой процесса ознакомиться с данными в момент, когда их кто-то модифицирует, – тогда в его восприятии может возникнуть такой набор данных, который, например, в принципе невозможен по логике данной модели. С подобными проблемами почти всегда сталкиваются даже опытные программисты, которым раньше не приходилось писать параллельного кода, и вдруг почему-то пришлось.

Часто эту проблему решают введением дисциплины модификации данных: в любой момент времени любой набор данных имеет право менять лишь кто-то один в системе – все остальные должны ждать, пока это право не придет к ним. Однако эта дисциплина не дает универсального решения проблемы. Придуманы примеры, когда в ее результате А ждет В, В ждет С, а С ждет А, и стало быть, все они будут ждать вечно, если в ситуацию никто не вмешается извне.

Обычно организация имитационных вычислений приходится на долю инструментальных систем моделирования. Пользователи лишь программируют блоки, описывающие функциональности тех или иных составляющих модели в различных ситуациях, которые могут возникнуть для этих составляющих. Задача синтеза полноценной модели из таких описаний – тоже далеко не тривиальна.

Приведем лишь один пример. Организация вычислительного процесса имитации состоит в том, что инструментальная система, в соответствии с описаниями структуры и логики модели, сделанными пользователем, создает списки вызываемых программ, описывающих функциональности компонент моде-

ли, и также написанных пользователем на тех или иных языках программирования. Затем она организует выполнение этих программ – это и есть процесс имитационных вычислений. Заметим, что место в этих списках хотя и не случайно (скорее всего, оно зависит от того, кого первым описали, или создали, или активизировали), но скрыто от пользователя, недокументировано и поэтому неуправляемо им.

Тем не менее, программа организации имитационных вычислений всегда будет работать с такими списками по порядку. Важно, чтобы этот факт не вносил систематической ошибки в процесс имитационных вычислений.

Например, если в дуэли ковбоев «кто первым выстрелил – тот и прав» всегда будет побеждать тот, кого разработчик случайно описал первым, при создании этой модели – это будет ошибкой именно такого сорта.

Наконец, перейдем к обзору существующих решений этой и других сформулированных в первом параграфе проблем организации процесса моделирования сложных систем.

1.2. Решения. Примеры инструментальных средств моделирования

Никто не обнимет необъятного!

К. Прутков, «Плоды раздумья: мысли и афоризмы», №67.

В данном параграфе мы рассмотрим несколько реализаций средств моделирования. Автор заранее отгораживается известным афоризмом К. Пруткова от обвинений в неполноте предлагаемого обзора. В настоящее время и в самом деле немало средств моделирования, в том числе и сложных систем, в том числе и распределенных.

Из не вошедших в обзор этого параграфа, упомянем ряд открытых систем агентного моделирования: система с красивым названием MASON (Multi-Agent Simulator of Neighbor-

hoods – разработка Университета Джорджа Мейсона, Вирджиния), SeSAm (**Sh**ell for **S**imulated **A**gent Syste**m**s) – визуальная система мультиагентного моделирования с UML-подобным интерфейсом.

К сожалению не вошли в обзор «движки» компьютерных игр, особенно, конечно, сетевых. Вообще эта отрасль очень близка к поднятой в книге теме, конечно с учетом сильного перекоса от содержательной части моделирования в сторону визуализации результатов. К сожалению, игры – продукты коммерческие и нечасто открывают свои движки, особенно в том смысле, который бы интересовал разработчика моделей сложных систем, а не только создателя на его базе игры с несколько иным обликом.

Очень интересная разработка, на взгляд автора предвосхитившая некоторые идеи модных сейчас «облачных вычислений», много лет ведется в ВЦ РАН группой А.В. Воротынцева [42].

Однако, как учит К. Прутков, – нельзя объять необъятное!

Основная цель автора – не перечислить всех игроков данного поля, а проиллюстрировать на приведенных примерах, которые иногда представляются ему наиболее яркими, а иногда и просто, как в случае с системой MISS, более близкими, как на практике решаются обозначенные в предыдущем параграфе проблемы моделирования сложных систем. На взгляд, автора для иллюстрации наиболее распространенных решений в области построения моделей сложных систем, вполне достаточно рассматриваемых примеров.

Данный параграф по своему жанру является обзором. Поэтому, кроме пункта 1.2.2, оригинальным в нем может быть только отношение автора к описываемым программным продуктам, некоторые оценки и выводы. Описания инструментальных систем моделирования приводятся в основном в по-

рядке появления этих систем на свет, все они основаны на публикациях, ссылки на которые приводятся.

1.2.1. Система GPSS (General Purpose Simulation System)

Система имитационного моделирования общего назначения GPSS была разработана сотрудником фирмы IBM Джеффри Гордоном в 1961 году. За 10 лет, к 1971г были созданы 5 версий языка GPSS. В настоящее время, почти через полвека после создания первой версии, система GPSS – несомненно один из редко встречающихся действующих ветеранов в области инструментальных средств моделирования.

На нашу почву система GPSS была занесена в процессе реализации в СССР недоброй памяти проекта внедрения клонов компьютеров серии IBM 360/370, известных как ЕС ЭВМ. Среди программного обеспечения ЕС ЭВМ система получила название ПМДС (Пакет Моделирования Дискретных Систем).

Популярности системе GPSS в нашей стране несомненно добавила изданная в те годы обстоятельная монография Т. Дж. Шрайбера [61]. В ней была рассмотрена одна из ранних версий языка – GPSS/360, а также основные особенности более мощной V-й версии, поддерживаемой в то время компанией IBM. В середине 80-х компания IBM прекратила поддержку проекта GPSS.

В 1984 году появилась первая версия GPSS/PC для персональных компьютеров с операционной системой DOS, разработанная компанией Minuteman Software.

Наконец, в 1993 та же компания выпустила, по-видимому, наиболее популярную за все времена версию системы – GPSS World. За сравнительно небольшой период времени было выпущено несколько ее подверсий, при этом возможности системы в каждой из них все время расширялись.

Несмотря на более чем солидный возраст, система моделирования продолжает развиваться. Помимо основных версий,

постоянно появляются новые, например, учебная версия Micro-GPSS, разработанная в Швеции (Ингольф Сталл, Стокгольмская Школа Экономики), там же разработана WebGPSS, предназначенная для разработки учебных имитационных моделей в сети Интернет. На Украине разработан интересный проект – Object GPSS [45], где классический язык GPSS погружен в современную среду программирования, например, Delphi. В общем, проект GPSS продолжает активно жить, включается в вузовские курсы моделирования [55], имеет достаточно широкий круг приверженцев (см., например, http://www.gpss.ru).

Тем не менее, несмотря на слова про общее назначение в названии (впрочем, с названием все просто – в 60-х гг. практически все компьютерное моделирование, которое не занималось численным решением систем дифференциальных уравнений, – так или иначе занималось системами массового обслуживания), GPSS остается в основном инструментом моделирования процессов массового обслуживания. Концепция моделирования системы GPSS формулируется в терминах теории массового обслуживания: заявки (транзакты), генераторы заявок, очереди, одноканальные и многоканальные устройства и т. д., даются удобные средства реализации этой концепции.

Как всегда – концепция инструментальной системы это одновременно и ее сила, и ее слабость. К примеру, можно ли на инструментальной системе MISS, которая будет описана в следующем пункте, моделировать процессы массового обслуживания? Да, и автор это делал. Однако, придется программировать самому и генераторы заявок, и очереди, и обслуживающие устройства, и статистику тоже самому придется набирать и обрабатывать. Может оно и не слишком хитро, тем не менее, программируя даже самые простые вещи всегда есть где ошибиться. А в GPSS все это дается в готовом виде и главное – при этом все уже давно отлажено и правильно работает!

Можно не отвлекаться на технические проблемы и сосредоточиться на содержательной части модели.

А можно ли с помощью GPSS моделировать СОИ? Автор думает, что конечно да, хотя и не пробовал. В конце концов, СОИ задумывалась именно как колоссальная система массового обслуживания, так что никакого нарушения жанра здесь нет. Однако, если бы автор выполнил такую модель на GPSS тогда в 80-х, пожалуй его потом долго бы не оставляли сомнения: а не навязал ли он модели сложной системы готовую концепцию поведения, укладывая ее в достаточно жесткие рамки GPSS? А вот в настоящее время, представляя в общих чертах возможности СОИ, автор, случись такая необходимость, не побоялся бы моделировать ее средствами GPSS.

Это на самом деле – достаточно тонкий момент, и поэтому нуждается в пояснении. Например, в 1983г. Р. Рейган и его консультанты навязали в определенной мере свою концепцию всему мировому сообществу. Эту концепцию вкратце можно сформулировать так: «можно сбивать стартующие баллистические ракеты противника противоракетными средствами космического базирования». И ведь нельзя сказать, чтобы эта концепция уж совсем была бы неверна – их и в самом деле можно сбивать. Только вот для этого упомянутые противоракетные средства космического базирования должны быть в нужный момент в нужном месте и в достаточном количестве.

Проблема в том, что столько противоракет, сколько необходимо для полного подавления сотни ракет, стартующих в неизвестный заранее момент из одного и того же позиционного района, не вместят никакие небеса, и создать, а потом годами эксплуатировать такую группировку никому не под силу. Здесь можно видеть, как казалось бы небольшая неточность и небольшое недопонимание могут привести к очень и очень серьезным неприятным последствиям.

Какой из этого можно сделать вывод? На взгляд автора, если мы не знаем, чего ожидать от сложной системы и целью

моделирования является именно знакомство с ее возможностями – желательно стараться дать компонентам системы максимальную возможность проявить себя, минимально связывая их какими-либо внешними по отношению к ним концепциями. Постараться максимально точно воспроизвести именно поведение данной системы, всячески избегая замены этого поведения своими или чьими-то еще предвзятыми представлениями о нем. При таком подходе меньше шансов совершить крупные качественные ошибки. Если же достаточно хорошо поняты качественные особенности поведения моделируемой системы – это поможет правильно уложить ее даже в весьма жесткую концепцию той или иной инструментальной системы имитационного моделирования, и затем с ее помощью прорабатывать различные сценарии эволюции модели.

В заключение отметим, что дискретно-событийный подход управления модельным временем, разработанный Джеффри Гордоном, с тех самых пор, в различных модификациях часто применяется в системах моделирования там, где возникает необходимость моделировать с переменным шагом времени.

1.2.2. Инструментальная система MISS (Multilingual Instrumental Simulation System)

Многоязыковая инструментальная система имитационного моделирования MISS была создана в конце 80-х гг. в ВЦ АН СССР [33]. С помощью этой системы в те годы было проведено несколько имитационных экспериментов по исследованию возможностей предложенной Р. Рейганом СОИ.

Несмотря на то, что с момента создания этой системы прошло более двух десятков лет, и особого распространения она не получила, принципы лежащие в ее основе, такие как объектно-событийный подход к представлению модели, наличие специального непроцедурного языка для описания структуры модели и характеристик ее компонентов, синтез модели

из составляющих ее компонентов через механизм событие-сигнал-сообщение, неоднократно находили применение в таких появившихся позднее, и получивших достаточное признание технологиях объектного анализа и объектно-событийного моделирования, как CORBA, UML, HLA.

Концепция моделирования лежащая в основе инструментальной системы MISS [32 – 34] базируется на объектном подходе, однако проработанном более детально. Инструментальная система гораздо более четко конкретизирует механизмы анализа и синтеза модели, и кроме того, дает набор средств, инструментов для их осуществления, поясним это на примере конкретизации основных понятий объектного анализа: объект – характеристики – методы – события.

При моделировании сложных систем часто бывает важным отразить определенную иерархию компонент системы (кто из кого сделан, или кто с кем организационно связан). Поэтому, одноранговое понятие объекта было детализировано следующим образом: вводится понятие **группы**, которая может состоять из групп и/или объектов. Объекты являются терминальными элементами этого рекурсивного определения иерархии. Таким образом, порождается дерево, отражающее иерархическое строение системы. Группа, являющаяся корнем этого дерева, называется головной группой. В концепции моделирования MISS головная группа может быть либо локальной моделью, либо частью распределенной модели, реализованной на одном компьютере. Далее, объекты сложных систем часто одновременно участвуют в нескольких процессах. Например, самолет летит, наблюдает, а может еще стрелять, бомбить, выбрасывать парашютистов и т.д. Для отражения участия объектов в различных процессах вводится понятие **прибора** – материального носителя этого процесса. Можно сказать, что объект оснащен рядом приборов, которые отражают его участие в тех или иных процессах, и все что ни делает объект, на самом деле выполняется одним из его приборов. Здесь

следует заметить, что не всегда у прибора может быть прототип в реальности. Например, Земля вращается вокруг Солнца в силу закона тяготения, а вокруг своей оси, – по инерции, в силу закона сохранения момента количества движения, и специальных двигательных установок ей для этого не нужно. Однако при моделировании этих процессов при помощи MISS, или же каким-либо другим способом, необходимо выполнять определенные действия, например, интегрировать уравнения Кеплера, чем и должен в концепции моделирования MISS заниматься соответствующий прибор. Таким образом, мы видим, что классическое понятие объект в концепции моделирования инструментальной системы MISS реализовано тройкой понятий: группа – объект – прибор. Иерархическое строение модели иллюстрируется следующим рисунком:

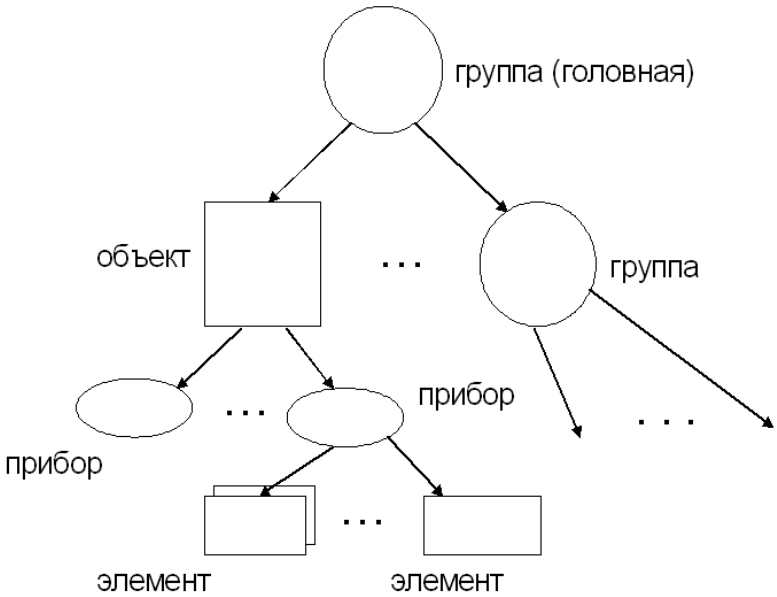

Рис.1. Схема иерархического строения модели.

Следующее понятие – характеристики объектов. Для эффективности хранения данных, в концепции моделирования

MISS, различаются **константы** – характеристики общие для всего типа групп, объектов или приборов (например, тактико-технические характеристики определенной модели самолетов), и **фазовые переменные** – характеристики экземпляров групп, объектов и приборов (например, координаты и скорость конкретного самолета). Здесь следует заметить, что «константы» MISS вовсе не должны оставаться все время постоянными, они постоянны (общие для всех экземпляров) лишь по отношению к экземплярам данного типа и их фазовым переменным, которые могут быть различны у разных экземпляров. Таким образом, тип или класс в MISS, помимо определения типа характеристик своих экземпляров, имеет экземпляр собственных характеристик с вполне конкретными их значениями (которые могут изменяться в ходе моделирования).

Следующее понятие объектного анализа – методы. В MISS все методы групп, объектов и приборов можно разделить на два класса: **служебных методов** и **элементов**, при этом, элементы могут быть только у приборов (у которых есть также и служебные методы), а с группами и объектами могут быть связаны только служебные методы. Служебные методы – это стандартные процедуры работы с характеристиками групп, объектов и приборов, планирования событий, работы с базой данных, отображения результатов моделирования. Служебные методы входят в состав MISS, и поэтому не требуют программирования, их просто нужно знать и применять. Имеется также возможность пользователю дописывать свои собственные служебные методы. Элементы – это алгоритмы функционирования приборов, алгоритмически неделимые элементы их деятельности. Функционирование прибора постулируется как чередование выполнения его элементов из некоторого их конечного заранее известного (обусловленного конструкцией прибора) набора. Задается некий начальный элемент, далее смена элементов определяется **автоматной функцией** прибора, входами которой являются завершившийся элемент и наступив-

шее событие. Алгоритмы элементов и, быть может, какие-то собственные служебные методы – вот все, что необходимо запрограммировать пользователю в рамках концепции моделирования MISS. В качестве возможных языков программирования разрешены C, C++ и МОДУЛА-2 (в принципе, в этот список мог быть включен практически любой язык программирования). Таким образом, с каждым типом или классом групп, объектов и приборов, во-первых, связан стандартный, встроенный в MISS набор служебных методов. Во-вторых, при описании типа или класса приборов, определяется набор его элементов, начальный элемент и автоматная функция переходов между элементами.

Следующая важная часть концепции моделирования принятой в инструментальной системе – способ обмена информации между объектами. Об одном способе обмена данными мы уже говорили – это служебные методы, предоставляющие различные виды доступа к характеристикам различных объектов. Однако область действия служебных методов – головная группа, т.е., часть распределенной модели, реализованная на одном компьютере, либо нераспределенная модель. С точки зрения современных информационных технологий, можно было бы расширить служебные методы и на удаленные объекты, воспользовавшись, например, технологией CORBA или же .Net Framework (функциональный аналог реализаций спецификации CORBA от фирмы Microsoft, которая первоначально не входила в комитет OMG), или быть может даже более простыми средствами RMI/ROA (Remote Method Invocation / Remote Object Activation – вызов удаленного метода / активизация удаленного объекта) современных языков программирования высокого уровня. Однако во времена создания MISS ничего из перечисленных выше средств еще не существовало. В результате был придуман достаточно универсальный единый способ обмена информацией между любыми объектами

как локальными, так и удаленными, через механизм обмена **сигналами/сообщениями.**

По-видимому, этот механизм так или иначе реализовывался во многих системах моделирования, например, о сигналах и сообщениях говорит Н.П. Бусленко в работах [39 – 40], весьма похожий механизм впоследствии вошел в спецификацию HLA [8]. По-видимому, это объясняется тем, что в жизни, при построении сложных технических систем часто поступают именно так, поэтому здравый смысл подсказывает разработчикам использовать этот механизм в своих моделях.

В основе описываемого механизма лежит постулируемая способность приборов принимать и посылать сигналы. Считается, что у каждого прибора может быть способность принимать несколько входящих сигналов и посылать несколько исходящих сигналов. Входящие и исходящие сигналы пронумерованы. Сам сигнал представляет булеву величину (либо он есть, либо его нет). Сигнал может сопровождаться сообщением, которое представляет собой список, состоящий из конечного числа записей одного типа.

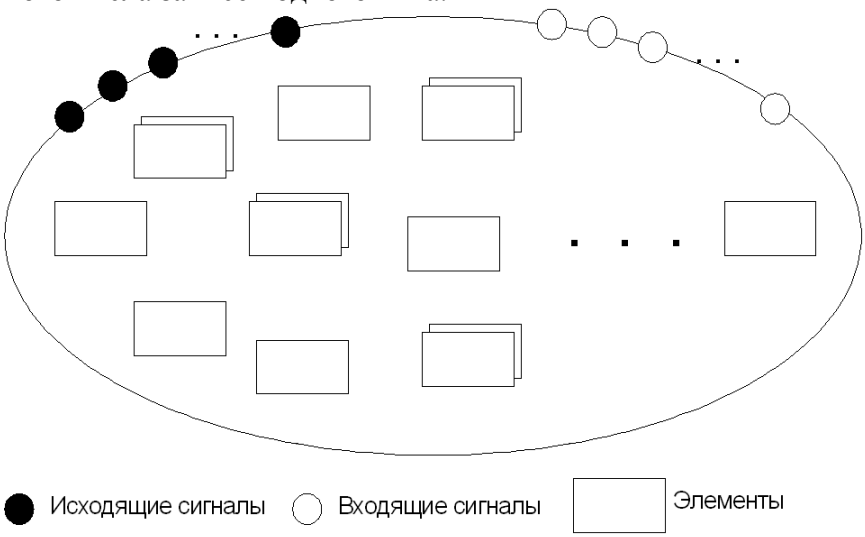

Рис.2. Схема структуры прибора.

Есть ли у прибора сигналы, сколько их, сопровождаются ли они сообщениями или нет, каких типов записи сообщений тех или иных сигналов – все это определяется в описании типа или класса прибора. Вопросы же откуда получает сигналы прибор, или куда он их посылает, на уровне описания типа/класса прибора остаются открытыми, кроме случая, когда прибор посылает сигнал сам себе. В последнем случае, в описании типа/класса прибора указывается коммутация соответствующих входного и выходного сигналов. При этом считается, что данный класс приборов так устроен (так коммутированы его сигналы), что исходящий сигнал номер такой-то приборов этого класса всегда попадает на их входящий сигнал номер этакий.

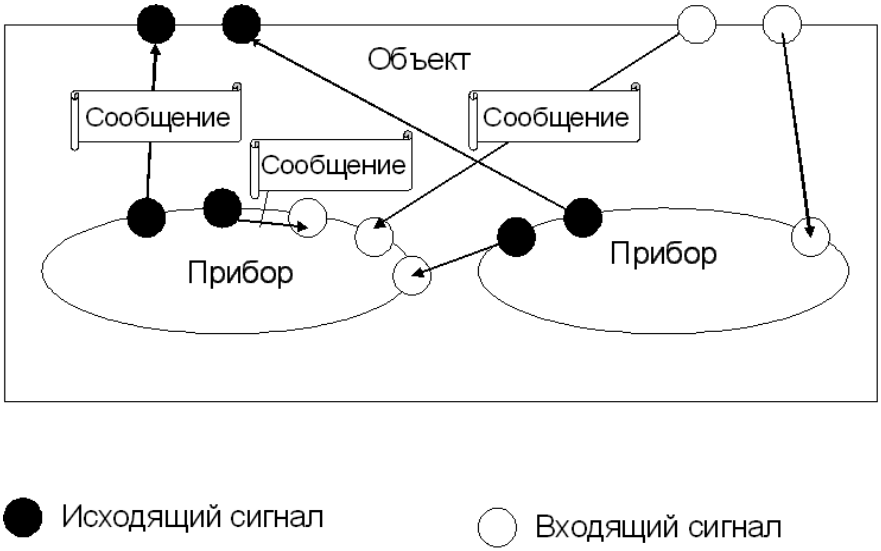

Рис.3. Схема коммутации сигналов на уровне объекта.

Про остальные сигналы можно лишь сказать, что они есть в конструкции прибора. Кому будут посылаться исходящие сигналы прибора и от кого будут приниматься входящие, зависит от того, в какие структуры далее будет включен этот прибор. Например, на уровне объекта (описание типа/класса

объекта) определяется, как это показано на рисунке ниже, какие выходные сигналы его приборов коммутируются с входными сигналами его же приборов, а какие идут дальше, на выходную шину объекта. Также определяется, каким приборам на вход попадают входные сигналы объекта. На рисунке некоторые сигналы показаны с сообщениями.

На уровне прибора также остается открытым вопрос, куда попадают его исходящие сигналы, и откуда берутся сигналы, входящие в него. Эти вопросы решаются на уровне охватывающей этот объект группы. Пример коммутации сигналов на уровне группы, содержащей в себе две входящих в нее группы, показан на рисунке. Отметим еще, что в MISS кроме коммутации сигналов «один к одному», возможна коммутация сигналов «один ко многим», т.е., один исходящий сигнал может поступать на несколько входящих (но не наоборот), что также отражено на рисунке ниже.

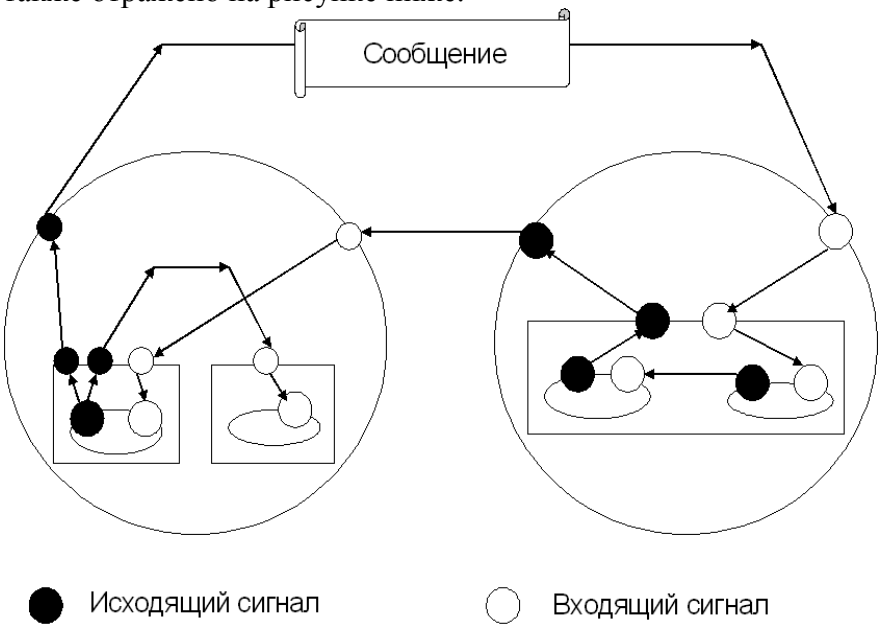

Рис.4. Схема коммутации сигналов на уровне группы.

Следует заметить, что каждый сигнал порождается обязательно каким-либо прибором и, хотя коммутация этого сигнала может быть весьма непростой и проходить через многие уровни иерархии модели, в конце концов, он тоже обязательно приходит на какой-либо прибор. Такова общая концепция моделирования – все, что делается в модели – делается элементами тех или иных приборов.

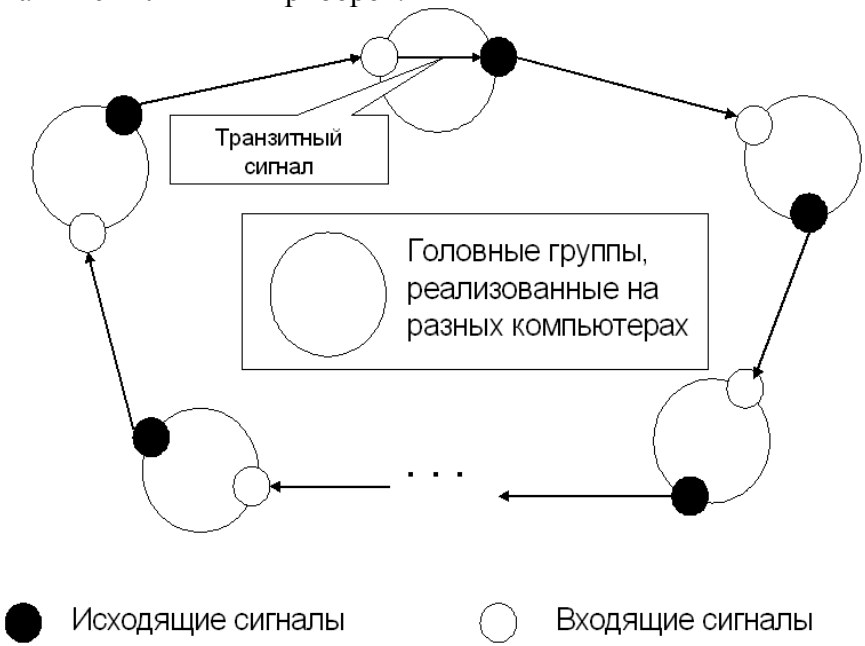

Рис.5. Структура распределенной модели.

Головная группа также может иметь входные и выходные сигналы. Факт наличия у головной группы сигналов однозначно свидетельствует, что эта группа описывает часть распределенной модели, реализованной на одном компьютере. Если же головная группа не имеет сигналов, то это локальная модель. Считается что в распределенной модели компьютеры, на которых реализованы распределенные компоненты модели, соединены кольцом, и сигналы с сообщениями ходят по этому

кольцу все время в одном направлении, например, по часовой стрелке. (Дело в том, что в 80-х сетевые технологии были развиты еще достаточно слабо, поэтому разработчикам MISS пришлось соорудить нечто подобное сети Token Ring, соединив несколько PC XT в кольцо через их последовательные порты.) В связи с чем у головной группы, помимо коммутации входных и выходных ее сигналов с соответствующими сигналами ее компонент, могут быть еще и «транзитные» сигналы, т.е., проходящие через эту группу «не задевая ее», дальше по кольцу распределенной модели к своим адресатам (т.к. другого пути к ним, кроме как через эту группу нет). Транзитный сигнал – это просто коммутация некоторого входящего в группу сигнала с неким исходящим, так как это может быть у прибора, посылающего сигнал самому себе, и так как никогда не может быть у обычных групп и объектов.

Можно критиковать данный способ коммутации сигналов как чрезмерно жесткий. Однако зато он позволяет, проектируя какую-либо компоненту модели полностью сосредоточиться лишь на свойствах самой этой компоненты и максимально отвлечься от свойств, как от охватывающих ее, так и вложенных в нее компонент. Это делает данную концепцию весьма технологичной при разработке сложной модели большим коллективом разработчиков. При этом одни и те же типы/классы компонент низших уровней могут входить составляющими в различные типы/классы компонент модели более высокого уровня.

В MISS, также как и в GPSS, моделирование осуществляется с переменным шагом модельного времени. Как уже говорилось выше, взаимодействие объектных составляющих модели (групп, объектов и приборов) осуществляется в возможно неизвестные заранее дискретные моменты времени – события, число которых конечно. Поэтому следующие важные понятия излагаемой концепции моделирования – это события и планирование событий.

Чтобы понять, как работает планирование событий, рассмотрим, как могут соотноситься продолжительности выполнения элементов с точностью моделирования по времени. Во-первых, отметим, что практически всегда, при любом шаге времени, найдутся такие элементы, которые по отношению к точности измерения времени в модели происходят мгновенно. Например, в приведенном выше примере «про войну» таким элементом будет взрыв боеголовки баллистической ракеты (мы ведь не собирались в этой модели имитировать сам процесс взрыва). Результат работы такого элементы также появляется сразу, мгновенно (бомба взорвалась – считайте радиусы поражения). Такие мгновенные элементы в MISS называются **сосредоточенными**. Какие элементы будут сосредоточенными, а какие нет, зависит в конечном итоге от того, какие процессы мы хотим имитировать в данной модели. Например, если мы хотим моделировать само явление ядерного взрыва, то в такой модели это будет продолжающийся во времени процесс, возможно, состоящий из нескольких продолжающихся во времени стадий. Тем не менее, опыт показывает, что если система достаточно сложна, при любом выборе масштаба времени, какие-то достаточно важные, чтобы быть отраженными в модели элементы оказываются сосредоточенными.

Противоположностью сосредоточенных элементов являются элементы **распределенные**. Выполнение таких элементов занимает некоторое время, не меньшее, во всяком случае, минимальной единицы времени, которую способна заметить данная модель. Такие элементы могут выполняться даже «вечно» (конечно, снова по отношению ко времени жизни модели), например, спутник летит вокруг Земли, или Земля вокруг Солнца. Второе свойство распределенных элементов, кроме продолжительности во времени – наличие определенного результата работы за любой промежуток времени, различимый моделью, возможно, зависящего от величины этого промежутка. Например, спутник на низкой орбите приблизительно за

полтора часа облетает Землю, за минуту пролетает полтысячи, а за секунду – восемь километров.

Третий тип элементов – **условно-распределенные**. Эти элементы продолжительны во времени как распределенные, но результат их действия наступает лишь по окончании выполнения всего элемента, как у сосредоточенных. Если же элемент не закончился, никакого результата его деятельности нет. Такими элементами, например, моделируются многие вычислительные процессы.

Распределенные элементы вместе с условно-распределенными называются медленными элементами, в отличие от быстрых сосредоточенных.

Проблема планирования событий состоит в том, что в ходе выполнения некоторого элемента может возникнуть событие, т.е. такая ситуация когда требуется синхронизировать действия с каким-либо другим объектом. Событие – это всегда потребность в синхронизации объектов, до этого действовавших независимо. Если событие вызвал сосредоточенный элемент, особых проблем не будет: такой элемент не занимает модельного времени, и после его окончания может быть легко установлена точка синхронизации. Сложнее если событие вызвал распределенный или условно-распределенный элемент, и при этом оно попало внутрь шага моделирования. Таким событием может быть, например, завершение медленного элемента, продолжительность которого несоизмерима с текущим шагом моделирования.

Для отслеживания возможных возникновений событий медленные элементы реализуются в виде пары методов – так называемого **таймера** и основного алгоритма элемента. Основной алгоритм элемента узнает, применяя соответствующий служебный метод, продолжительность текущего шага моделирования, и в соответствии с ней вычисляет действие этого элемента за это время, изменяя все что должно быть изменено, в соответствии с этим действием. Основной алгоритм условно-

распределенного элемента вычисляет действие только в последний раз, когда оказывается, что с окончанием текущего шага моделирования завершается и сам элемент. Если же элемент еще не завершается, то просто учитывается, что еще столько-то времени он выполнялся.

Таймеры медленных элементов не занимает модельного времени. Они должны быть вызваны до выполнения основных алгоритмов, чтобы проверить, не вызывает ли данный элемент на данном интервале времени, т.е., на данном шаге моделирования, каких-либо событий, и если оказалось, что вызывает, то запланировать с помощью соответствующих служебных методов точку синхронизации в момент ближайшего события. После того как отработают все таймеры, инструментальная система назначает следующий шаг моделирования, беря некий шаг по умолчанию, если никто ничего не назначил, или же минимальное время из назначенных таймерами, в противном случае. В вычислительном отношении понятно, что в самом худшем случае алгоритм таймера будет дублировать основной алгоритм. На практике же обычно он оказывается существенно проще. На приводившихся выше картинках таймеры элементов показаны рамочками, оттеняющими элемент.

Вызов таймеров перед каждым вызовом основного алгоритма не всегда эффективен. Достаточно часто встречаются элементы, про которые сразу можно сказать, что до самого своего завершения никаких событий они не вызывают. Для таких элементов существуют служебные методы, которые могут сразу жестко зафиксировать время их окончания, и до этого времени таймер больше вызываться не будет. В таких случаях будем говорить о блокировке таймера.

Строгое определение реализованного в MISS принципа синхронизации процессов состоит в следующем. Пусть начинается очередной такт имитации, в этот момент известно (почему – будет объяснено далее), какие элементы должны выполнять приборы, тогда:

1. для выполняемых распределенных и условно распределенных элементов вызываются незаблокированные таймеры, и если есть выполняемые сосредоточенные элементы, то шаг по времени устанавливается нулевым и вызываются их алгоритмы; если уже не установлен нулевой шаг, то в качестве момента окончания такта берется минимальное из времен, назначенных вызванными таймерами, и времен, которые заблокированные таймеры заказали ранее;
2. системные часы переводятся на время окончания такта;
3. вызываются основные алгоритмы тех выполняемых условно-распределенных элементов, чьи таймеры запланировали их завершение на момент, совпавший с найденным временем окончания такта; если шаг по времени оказался ненулевым, вызываются основные алгоритмы всех выполняемых распределенных элементов;
4. для каждого прибора, завершившего выполнение элемента (либо потому, что он – сосредоточенный, либо потому, что заказанное таймером время завершения совпало с моментом окончания такта), в соответствии с правилами его функционирования (см. ниже) определяется, к какому элементу он должен перейти в текущей ситуации; на этом такт имитации завершается.

Для того чтобы сформулированная схема приобрела четкость рабочего алгоритма, осталось уточнить, что подразумевается в пункте 4) под правилами функционирования приборов.

В MISS реализован подход, согласно которому прибор формирует свою последовательность элементов как конечный автомат, имеющий входами номер предыдущего элемента и полученные сигналы, а выходом – номер следующего элемента. Точнее говоря, когда прибор завершил выполнение очередного элемента, выбор следующего элемента определяется тем. какой именно элемент завершился и какие из совокупности

всех входных сигналов прибора поступили на момент принятия решения.

Данное положение формализуется введением автоматных функций приборов. Аргументов у каждой автоматной функции два: номер завершенного элемента и еще одно целое число, которое будем называть "номером старшего реализовавшегося события". Второй аргумент требует разъяснения. Суть в том, что для каждого прибора естественно разбить всевозможные комбинации значений булевых величин {есть сигнал, нет сигнала} на классы и считать, что при выборе очередного элемента роль играет не то, какая именно комбинация реализовалась, а то, к каким классам она принадлежит. Соответственно, для каждого элемента определяется свой набор алгебрологических функций от булевых индикаторов наличия входных сигналов, причем – такой набор, что хотя бы одна функция принимает значение "истина" при любой реализации значений аргументов. Эти функции, именуемые впредь событиями, ранжируются (нумеруются) и вторым аргументом автоматной функции служит номер старшего среди принявших значение "истина" событий.

Итак, выбор очередного элемента осуществляется по правилу:

1. по пришедшим на момент принятия решения сигналам вычисляются события из списка, отвечающего завершенному элементу;
2. просмотром в порядке убывания ранга (увеличения номера) определяется номер старшего реализовавшегося события;
3. номера завершившегося элемента и старшего реализовавшегося события подставляются в автоматную функцию прибора, и она дает номер следующего элемента.

Теперь для полной ясности способа организации вычислений осталось уточнить режим существования сигналов

(и связанных с ними сообщений) в модельном времени. Появляются, они как продукты деятельности приборов при выполнении ими своих элементов, а аннулируются автоматически. При завершении очередного такта имитации сразу после отработки основных алгоритмов, к приборам-получателям придут все те сигналы, которые были посланы приборами-отправителями на завершающемся такте. Эти сигналы (и сообщения) просуществуют в системе до аналогичного момента следующего такта имитации (будут доступны в вызываемых на этом такте таймерах и основных алгоритмах приборов-получателей).

Проиллюстрируем изложенный алгоритм синхронизации процессов диаграммой, показывающей последовательность вызовов таймеров и основных алгоритмов. Точки между ними подразумевают системные операции, в том числе - пересчет времени, обновление поля сигналов и определение переключений по автоматным функциям.

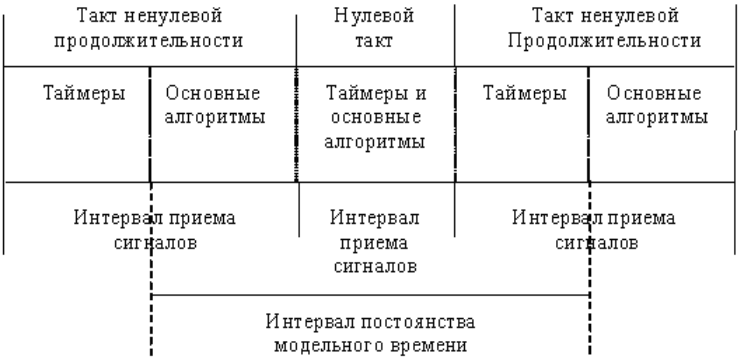

Рис.6. Принципы синхронизации процессов.

Для описания типов групп, объектов, приборов, их характеристик, состава, коммутации их сигналов между собой, используется специальный непроцедурный язык описания мо-

дели, соответствующая система программирования, состоящая из редактора, отладчика и сборщика модели включена в MISS.

Мы видим, что описания компонент модели предельно независимы друг от друга. На каждом уровне иерархии модели упоминаются лишь составляющие данной компоненты, причем их устройство не конкретизируется, кроме декларирования у них определенных входных и исходящих сигналов. Это позволяет разрабатывать компоненты модели независимо друг от друга, различным коллективам разработчиков, при этом последним необходимо детально разбираться лишь в устройстве своей компоненты.

Результатом обработки системой программирования описаний групп, объектов и приборов является либо выявление в этом описании синтаксических или же логических ошибок и указание на них, либо, если ошибок нет, можно приступить к сборке модели. Входом сборщика является откомпилированное описание головной группы, выходом – некий костяк модели, локальной, если головная группа не имела сигналов в своем описании, или же компоненты распределенной модели, функционирующей на отдельном компьютере. Сборщик строит иерархическое дерево модели, находя по операторам описания компонент головной группы сами эти оттранслированные компоненты, затем ищет компоненты этих компонент, останавливая эту рекурсию на терминальных приборах. Если какие-либо компоненты модели отсутствуют в базе откомпилированных – выдается соответствующая диагностика. Тщательно проверяется коммутация сигналов: сигнал, выйдя из прибора, не имеет права «зависнуть», он обязательно должен, в конце концов, быть может, пройдя всю иерархию модели снизу вверх и обратно сверху вниз, прийти в какой-либо прибор (может быть в несколько приборов). Кроме того, если сигнал сопровождается сообщением, необходимо, чтобы типы сообщений, описанные в приборе, посылающем сигнал и приборе

его принимающем, совпали. В случае нарушения этих требований, сборщиком выдается соответствующая диагностика.

Сборщиком создается костяк модели, основу которого составляет база данных характеристик групп, объектов и приборов, входящих в модель, а также проложенные между приборами каналы коммутации сигналов. Этот костяк отличается от работоспособной модели тем, что, во-первых, в нем пока отсутствуют программные модули, реализующие функционирование элементов и во-вторых, база данных характеристик компонент модели пока пустая. Эта база данных может быть заполнена или вручную, или конструкторами объектных составляющих модели при ее запуске. Далее, этот костяк модели дополняется оттранслированными с допустимых инструментальной системой языков программирования программными модулями элементов (их просто нужно положить в каталог модели). Программные модули элементов присоединяются к модели путем вызова в их конструкторах соответствующих служебных методов, предоставляемых инструментальной системой. После этого модель готова к проведению имитационных экспериментов.

Обратим теперь внимание на то, как в системе MISS решались общие проблемы моделирования сложных систем, поставленные в первом параграфе этой главы.

Была успешно решена задача синтеза сложной многокомпонентной системы. Это позволило осуществить ряд имитационных экспериментов по исследованию возможностей системы СОИ. В качестве системы управления модельным временем выбрана консервативная схема с переменным шагом моделирования и системой предсказания наступающих событий. Естественно, возникали опасения, а не может ли зациклиться модель при такой схеме управления временем, из-за все время уменьшающегося шага моделирования. Однако разрешение такого рода вопросов было отдано на усмотрение разработчика модели. На усмотрение разработчика также была

отдана проблема доступа к характеристикам модели. В MISS кто угодно может получить доступ к чьим угодно фазовым переменным. Слишком жесткой оказалась принятая система коммутации сигналов. Например, она существенно осложняла порождение новых экземпляров во время имитации.

Судьба системы MISS поначалу складывалась весьма удачно. Так на рубеже 80-х – 90-х гг. было инициировано создание крупного центра моделирования, идеологической основой которого была выбрана концепция моделирования MISS. Дело всерьез шло к созданию стандарта распределенного, в том числе и полунатурного моделирования, каким в настоящее время мы знаем HLA. Была замечена система MISS и за рубежом, в 1990г. она получила первый приз в категории профессиональных программ в конкурсе программных продуктов, который проводила в СССР японская фирма ASCII Corporation – в то время один из крупных разработчиков компьютерных игр.[1]

Однако тогда реализации многих планов помешало вхождение в 1991г. нашей страны на десяток с лишним лет в «зону турбулентности». Авторам реализации MISS пришлось отложить свои разработки, одному на несколько лет, – другому, к сожалению, навсегда.

Второй подход автора к теме распределенного имитационного моделирования сложных систем произошел в середине

[1] ASCII Corporation, с 2008г. – ASCII Media Works – в настоящее время крупное японское издательство. Специализируется на публикации книг, манга (японских комиксов), журналов развлекательной и околокомпьютерной тематики, и распространении видеороманов и компьютерных игр в стиле аниме. Компания выделилась в 80-х гг. из японского отделения корпорации Microsoft, ради продвижения стандарта MSX для бытовых компьютеров на базе 8-разрядного процессора Zilog Z80. К концу 80-х стала крупным разработчиком компьютерных игр для этой платформы. На рубеже 90-х гг. планировала развернуть ряд зарубежных филиалов (США, СССР, Китай), ориентированных на разработку компьютерных игр. Реально лишь один из них, ASCII Entertainment, был создан в США в 1991г.

первого десятилетия уже нового века. Естественно, при этом хотелось учесть накопившийся в мире опыт в этой области, преодолеть недостатки предыдущей разработки и попробовать разобраться, а как вообще нужно делать такие системы. Насколько это удалось – должно показать содержание второй и третьей глав этой книги.

1.2.3. Спецификация HLA (High Level Architecture)

Следует сразу отметить, что в отличие от всех остальных инструментальных средств моделирования рассматриваемых в этом параграфе, высокоуровневая архитектура распределенного моделирования HLA является спецификацией, а не инструментальной системой. Это означает, что в ней четко прописаны правила, по которым должна создаваться и работать распределенная модель. Эта спецификация открыта, что означает, что всякий может взять ее за основу, или реализовать полностью в своей собственной инструментальной системе распределенного моделирования, или в отдельно взятой модели или в части модели. Точное следование спецификации должно обеспечить разработке совместимость с другими подобными разработками. При этом, любая реализация спецификации HLA, вообще говоря, может перестать быть открытой, став коммерческим продуктом – это зависит от намерений ее разработчика. В настоящее время имеется ряд инструментальных средств как свободно распространяемых, так и коммерческих, позволяющих разрабатывать HLA-совместимые приложения и обеспечивать функционирование HLA-федераций.

Сделаем еще одну оговорку. Спецификация HLA существенно объемнее и сложнее большинства остальных приводимых в этом параграфе примеров инструментальных средств моделирования. Поэтому здесь будут отражены лишь самые важные на взгляд автора ее характеристики. Более подробно познакомиться с этой архитектурой можно, например, в работах [8], [43]. Большая подборка различных материалов на эту

тему имеется на сайте Центра Грид-технологий и распределенных вычислений ИСА РАН по адресу: http://dcs.isa.ru.

Архитектура HLA для распределенного моделирования была разработана во второй половине 90-х, в интересах и при финансовой поддержке Министерства обороны США в целях обеспечения возможности взаимодействия всех типов моделей и поддержки их многоразового использования

Хотя создание HLA было инициировано Министерством обороны США, с самого начала и до сих пор эта архитектура является открытым стандартом, который развивается и поддерживается подразделением DMSO (Defence Modelling & Simulation Office) Министерства Обороны США.

HLA быстро стала стандартом «де факто» для тренажеров и симуляторов в военных приложениях, поскольку Министерство обороны США неукоснительно требовало их совместимости с HLA. В 2000 г. версия 2.3 HLA была принята в качестве стандарта IEEE 1516.

Модель в HLA рассматривается как набор подмоделей различного уровня агрегирования. Также как в MISS, выделяются три основных уровня иерархии. На нижнем уровне расположены объекты, из которых может состоять федерат – средний уровень иерархии – элементарная законченная и самостоятельная модель (в том числе, быть может, выполненная «в железе», или наоборот, – живой участник человеко-машинного эксперимента) в концепции моделирования HLA. Наконец, несколько федератов, в том числе и распределенных, объединенных некоторой общей задачей, могу образовывать федерацию – третий уровень иерархии.

Функционирование федерации обеспечивает специально разработанная инфраструктура времени выполнения RTI (Run-Time Infrastructure). В более простых инструментальных средствах моделирования, особенно в игровых, функционально аналогичную компоненту системы обычно называют «движком». В случае HLA, – по масштабу это скорее распределенная

операционная система, обеспечивающая обработку стандартных запросов компонент модели, в первую очередь – согласованное продвижение модельного времени федерации и всех федератов, а также поддержку информационных обменов в рамках федерации между федератами. RTI предоставляет федератам множество системных сервисов, которые разбиваются на шесть групп:

1. Управление федерацией.
2. Управление декларациями.
3. Управление объектами.
4. Управление владением.
5. Управление временем.
6. Управление распределением данных.

Как уже можно видеть, структура модели в HLA весьма непроста, и для успешной реализации нуждается в формальном описании. Для такого описания архитектурой предусмотрены специальные средства. Это шаблон объектных моделей OMT (Object Model Template), которому следуют описания объектов, федератов и федераций.

В соответствии с шаблоном объектных моделей описываются объектная модель федерации FOM (Federation Object Model) и объектная модель имитации – SOM (Simulation Object Model) – на уровне федератов. В SOM прописывается все взаимодействие федерата с федерацией.

В соответствии с описанием SOM, федерат может иметь право изменять атрибуты объектов (не обязательно своих). Через специальный сервис RTI это изменение передается другим федератам. В этом случае говорят, что первый федерат обновил атрибут, а получившие измененное значение – отобразили атрибут. Что касается чтения атрибутов объектов, в HLA это осуществляется снова через RTI, посредством механизма публикации/подписки, и опять же должно быть отражено в SOM.

Также в соответствии с описанием SOM федерат может посылать и принимать сообщения. Содержательно – это некоторые данные, генерируемые одним объектом, способные изменить состояния других объектов (аналогично, например сигналам/сообщениям MISS).

В HLA определены 10 правил, которым должны подчиняться федерации и федераты, по 5 правил для федераций и федератов.

Правила для федераций:

1. Каждая федерация должна иметь свою объектную модель FOM, задающую описание и взаимосвязи классов всех потенциально возможных в данной федерации объектов. FOM должна быть документирована в соответствии с эталоном OMT.
2. В каждой федерации все объекты должны находиться только в ее федератах, но не в RTI.
3. Во время «исполнения» федерации обмены данными напрямую между федератами запрещены; все обмены осуществляются только через RTI.
4. Федераты должны взаимодействовать с RTI только в соответствии со спецификацией интерфейсов HLA.
5. Во время работы федерации федераты могут изменять значения атрибутов различных (не обязательно своих) объектов модели. Право на такие изменения они получают от RTI. В любой заданный момент времени право изменения любого атрибута должен иметь только один федерат.

Правила для федератов:

1. Каждый федерат должен иметь свою объектную модель SOM, документированную в соответствии с OMT HLA.
2. Каждый федерат должен уметь обновлять и/или отображать значения атрибутов объектов, равно как посылать и/или получать сообщения, в соответствии с тем, как это сформулировано в его модели SOM.

3. Каждый федерат должен уметь динамически во время исполнения федерации получать и отдавать право изменения атрибутов объектов в соответствии с описанием, содержащимся в его модели SOM.
4. Федераты должны уметь изменять условия, при которых они обновляют атрибуты, находящиеся в их владении, в соответствии с моделями SOM.
5. Модельное время у каждого федерата в общем случае свое. Чтобы синхронизировать обмен данными с другими членами федерации, федераты должны иметь возможность управлять локальным временем своих моделей.

Обратим внимание на пятые пункты обоих наборов правил. Мы видим, что для того чтобы добиться однозначности вычислительного процесса, вводится достаточно сложный механизм динамической передачи прав на изменение атрибута. При этом, естественно, тот кто хочет изменить атрибут и не имеет соответствующего права – должен ждать его получения.

Что касается пятого пункта правил для федератов - оказывается, что единого модельного времени в федерации нет – каждый федерат живет по своему модельному времени. Тем не менее, федераты обмениваются данными и сообщениями, что требует синхронизации их модельных времен – недопустимо, например, чтобы сообщение пришло в чье-нибудь прошлое. Кроме того, в спецификацию HLA кроме обычной прогонки «так быстро как возможно», включены возможности прогонки в реальном и масштабируемом времени. В результате система управления временем в HLA, как утверждается, например в [43], уникальна тем, что по-видимому, объединяет в себе возможности всех когда-либо рассматривавшихся в моделировании сложных систем схем управления модельным временем. Поэтому автор даже не пытается дать здесь ее полное описание, тем более что это и не обязательно в контексте данной работы, а отсылает интересующихся к литературе на эту тему

[3], [4], [6], [7], [8], [43], или же к упоминавшемуся уже сайту Центра Грид-технологий и распределенных вычислений ИСА РАН: http://dcs.isa.ru, где приводится быть может не самое полное, зато достаточно краткое и понятное описание.

Подведем некоторые итоги. Несомненно, из всех известных автору средств моделирования – концепция HLA намного превосходит все остальные по универсальности и широте охвата всего, что только может встретиться в моделировании сложных систем и всего, что только может совместно работать в сети.

Однако, как всегда, недостатки – оборотная сторона достоинств. Концепция HLA на взгляд автора, слишком сложна и тяжеловесна. Чтобы ее изучить, нужно никак не меньше учебного семестра, а чтобы развернуть практикум и дать возможность попробовать сделать что-то хоть самое простое своими руками, а тем более овладеть ею как своим рабочим инструментом – боюсь, что намного больше.

Вспоминается детская сказка А.М. Волкова про Урфина Джюса. В ней Джюс захватив Изумрудный город придумал себе красивый, длинный и пышный титул, вот только его новые невольные подданные, дрожа от страха, ни разу не смогли его воспроизвести, не переврав.

Кроме того, автора не оставляет чувство, что на самом деле так много всего и не надо. Конечно, для того класса задач, которыми занимался автор, и которым посвящена эта книга, а вовсе не для всего на свете, что может встретиться в моделировании, и что может совместно работать в сети.

1.2.4. Инструментальная система моделирования AnyLogic

Инструментальная система имитационного моделирования AnyLogic – уникальный пример отечественной и притом коммерчески успешной разработки на рынке профессиональных средств моделирования [20], [48]. Система разработана

компанией XJ Technologies (http://www.xjtek.ru), первая ее версия появилась в 2000 г. Система AnyLogic изначально не позиционировалась, как распределенная, однако она открыта для создания различных надстроек, и известен опыт [60] создания надстройки, позволяющей с помощью AnyLogic осуществлять распределенное моделирование в стандарте HLA.

Система AnyLogic, представляет собой комплексный инструмент, охватывающий основные в настоящее время направления имитационного моделирования: системную динамику, системы массового обслуживания, агентное моделирование. Нас, в основном, будет интересовать последнее.

Система моделирования AnyLogic основана на объектно-ориентированной концепции. Эта концепция позволяет простым и естественным образом организовать и представить структуру сложной системы. Второй основной концепцией AnyLogic является представление модели как набора взаимодействующих параллельно функционирующих активностей.

Такой подход к моделированию интуитивно понятен и естественен во многих приложениях, поскольку мы с детства привыкли воспринимать окружающую действительность объектно. Активный объект AnyLogic – это объект со своим собственным функционированием, взаимодействующий с окружением. Он может включать в себя любое количество экземпляров других активных объектов. Активные объекты могут динамически порождаться и исчезать в соответствии с законами функционирования системы.

AnyLogic базируется на языке программирования Java – одном из самых мощных и в то же время простых для изучения современных объектно-ориентированных языков. Объекты, определенные пользователем при разработке модели на AnyLogic с помощью его графического интерфейса, транслируются в конструкции языка Java, которые затем компилируются в исполняемый код модели.

Хотя при построении модели на AnyLogic разработчик использует конструкции языка Java, он не разрабатывает полные программы, а лишь вставляет фрагменты кода в специально предусмотренные для этого поля окна «Код» и окон свойств объектов модели. Написанные разработчиком Java-фрагменты выражают логику тех или иных действий происходящих в модели. Эти фрагменты кода должны быть синтаксически правильными конструкциями Java, потому пользователь AnyLogic должен иметь определенное представление об этом языке.

Интеграция инструментальной системы с универсальным языком программирования – достаточно естественное решение при моделировании по-настоящему сложных систем. Такое же решение мы видим в HLA, и еще раньше в MISS.

Основным средством определения поведения объектов в AnyLogic являются переменные, таймеры и стейтчарты. Переменные отражают изменяющиеся характеристики объекта. Таймеры имеют примерно тот же смысл, что в MISS, их можно взводить на определенный интервал времени и по окончании этого интервала произойдет событие. Стейтчарты аналогичны автоматным функциям перехода MISS, они определяют смену состояний модели под воздействием тех или иных событий или условий. 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.

Так же как и в HLA и в MISS, в AnyLogic имеется специальный язык для описания модели, ее составляющих и различных связей между ними. Однако, в отличие от упомянутых ранее систем, в AnyLogic этот язык полностью графический, что создает пользователю, в особенности начинающему, дополнительные удобства, например, полностью избавляя от забот с каким-нибудь пропущенным разделителем или опиской в ключевом слове.

Система AnyLogic также имеет весьма богатые средства визуализации и анимации результатов имитационных экспериментов. Кроме того в ней имеются средства соотнесения модельного времени с реальным, например, можно запустить имитационный эксперимент в реальном масштабе времени, или, например, в 4 раза быстрее реального, ну и, конечно же в обычном режиме – «так быстро, как возможно».

В целом, по мнению автора, AnyLogic – очень качественный продукт, единственный существенный недостаток которого – высокая цена лицензии, в том числе и академической. Однако и это автор готов простить, понимая, что инструментальная система имитационного моделирования не может быть массовым продуктом, как например, операционная система, и поэтому должна отрабатывать затраты на разработку.

Компания XJ Technologies динамично развивается, в настоящее время имеет Европейское и Североамериканское отделения. Отечественная IT-отрасль вполне может ею гордиться, – пожелаем же ей дальнейших успехов!

1.2.5. Системы ABMS (Agent-Based Modeling and Simulation)

Системы ABMS мультиагентного моделирования начали появляться с конца 90-х, по-видимому, вначале испытав значительное влияние со стороны клеточных автоматов. В настоящее время несколько таких систем, в основном разработанных и поддерживаемых американскими университетами, стали популярными среди разработчиков биологических, социологических моделей, а также в обучении информатике.

Система Swarm (рой, толпа) – одна из первых по времени подобных систем моделирования, первоначально разработанная в Университете Санта-Фе, Нью-Мексико. В настоящее время эту разработку поддерживает группа разработчиков SDG (Swarm Development Group,

http://www.swarm.org).

Программное обеспечение Swarm представляет собой набор библиотек, обеспечивающих имитационные вычисления моделей, написанных на языках программирования C++ и Java. Такие библиотеки имеются для широкого набора компьютерных платформ. Ядро системы запускает код, написанный разработчиком на языке программирования высокого уровня (C++, Java).

Основа архитектуры Swarm – моделирование набора конкурентно взаимодействующих агентов. Планировщик времени Swarm позволяет моделировать только с постоянным шагом времени. Ведутся разработки по созданию распределенной версии Swarm.

Программное обеспечение Swarm распространяется свободно. Группа SDG позиционирует Swarm как экспериментальное программное обеспечение, т. е., могущее принести определенную пользу, однако находящееся в постоянном процессе разработки.

NetLogo – система моделирования, разработанная Ури Виленским из Северо-Западного университета, родившаяся из языка программирования Logo (язык графики черепашек), предназначенного для первого знакомства детей с началами программирования (http://ccl.northwestern.edu/netlogo/).

Система NetLogo – распределенная. Так как она часто применяется в учебном процессе, обычно распределенность в ней, это совместная работа учитель – класс.

В настоящее время доступна четвертая версия NetLogo, распространяемая свободно и действующая на различных платформах. Важной особенностью этой версии языка NetLogo является появление нового типа агентов. К черепашкам (turtles) и пятнышкам (patches) добавились связи (links). Агенты нового типа открывают новые возможности для моделирования сетевых отношений. Связь в NetLogo это – агент связывающий две черепашки или два узла.

Появление нового типа агентов позволяет рассматривать новые феномены и создавать новые модели, в которых большое значение играют связи между узлами сети.

Язык NetLogo достаточно прост и ученики, и учителя могут создавать в этой среде свои собственные авторские модели. В то же время это достаточно мощный язык и среда для построения не слишком сложных учебных моделей.

Repast (Recursive Porous Agent Simulation Toolkit) – система моделирования первоначально разработанная в Чикагском Университете, в настоящее время разработка поддерживается некоммерческой добровольной организацией ROAD (Repast Organization for Architecture and Development), http://repast.sourceforge.net.

Пожалуй это самая мощная многоплатформенная среда моделирования из перечисленных в этом пункте. Планировщик времени позволяет моделирование с переменным шагом модельного времени. Система включает богатую библиотеку моделей и развитые средства визуализации. Система моделирования Repast первоначально не позиционировалась распределенной, однако имеются ее версии способные взаимодействовать по стандарту HLA (проект HLA_RePast), с которым можно познакомиться по адресу: http://www.cs.bham.ac.uk/research/projects/hlarepast, а также интегрированный с этим проектом проект для Grid HLA_GRID_REPAST , информацию о котором можно найти по адресу: http://www.cs.bham.ac.uk/research/projects/dsgrid.

Основой концепции моделирования являются понятия агентов – действующих сущностей модели. Иерархия вложенных контекстов (contexts) является моделью окружения агентов. Контексты могут динамически изменяться, оказывая влияние на поведение агентов. С каждым контекстом связан набор типов взаимодействия агентов (projections), которые определяют возможные отношения между агентами. В результате поведение агента в этой системе чувствительно к

контексту (Context-Sensitive Behavior). К примеру, агент-солдат в «гражданском» контексте будет сидя говорить: «Привет!», всем входящим в комнату. В «военном» же контексте он при входе старшего по званию обязан встать и отдать честь.

Система моделирования Repast – свободно распространяемая.

Подводя итог примерам этого пункта, можно отметить, что свободно распространяемые системы мультиагентного моделирования динамично развиваются, иногда привнося очень интересные идеи в копилку концепций моделирования сложных систем. Некоторые из них набрали достаточную мощь, чтобы быть использованными при построении весьма серьезных моделей. Недостатком этих систем можно считать пожалуй лишь традиционные для свободного ПО проблемы с не слишком подробной документацией.

1.3. Некоторые выводы и открытые вопросы

Анализ приведенных примеров реализаций инструментальных средств моделирования показывает, что авторы этих разработок в разное время, и по всей видимости независимо друг от друга (во всяком случае автору почти не приходилось видеть ссылок, свидетельствовавших бы об обратном!), приходили к весьма сходным решениям относительно способов организации вычислительного процесса имитации.

Назовем основные из этих решений:

1. Объектная декомпозиция модели.
2. Моделирование с переменным шагом времени. Декомпозиция модельного времени на события и промежутки между ними.
3. Планирование или прогнозирование событий.

4. Декомпозиция активности модели на параллельные процессы-потоки активностей, каждый из которых есть череда сменяющихся элементарных алгоритмов.
5. Определение правил перехода между элементарными алгоритмами, как функции состояния и характеристик модели.
6. Взаимодействие компонент модели через механизм сигналов/сообщений.
7. Наличие специального непроцедурного языка для описания состава модели, связей между ее компонентами и некоторых правил их функционирования.
8. Интеграция инструментальной системы с одним или несколькими языками программирования высокого уровня.

Естественно, возникает вопрос, действительно ли это набор наилучших решений из всех возможных в данной области? Или же среди них есть случайно попавшие в список, без которых вполне можно было бы обойтись?

Кроме этого, нерешенным остается ряд проблем, связанных с моделированием сложных систем. Перечислим некоторые из них:

1. Все приведенные в качестве примеров инструментальные системы моделирования исходят из предположения, что моделирование интересующей разработчика предметной области в принципе возможно. Как человек отдавший немало времени и сил моделированию, автор не может не разделять оптимизма своих коллег. Однако сознавая, что крайняя степень такого оптимизма была бы в философском смысле лапласовским детерминизмом [47], и поэтому вряд ли может быть оправдана, хочет поднять вопрос: а все-таки когда возможно моделирование, в том смысле как его понимают, например, большинство разработчиков упомянутых в этой главе систем?

2. Все разработчики систем моделирования соглашаются, что сложные модели приходится моделировать с переменным шагом времени, уменьшая шаг моделирования в случае наступления на этом шаге некоторого события. Возникает вопрос: а не можем ли мы в результате все уменьшающегося шага моделирования получить сходящуюся последовательность, наподобие знаменитой апории Зенона про Ахиллеса и черепаху? И вообще, какой способ управления временем выбрать,– их много хороших и разных [3], [4], [6], [7], [43]?
3. Большинство разработчиков описанных выше систем соглашаются, что активность сложных моделей естественно декомпозировать на ряд параллельно выполняющихся процессов. Это было бы очень выгодно с точки зрения повышения эффективности вычислительного процесса – параллельность в настоящее время магистральный и далекий до исчерпания путь повышения эффективности вычислений. Как тогда быть с взаимодействием параллельно выполняющихся компонент? Разрешать ли всем желающим доступ ко всем характеристикам и организовывать очередность доступа к ним? Или лучше подписки? Или сигналы/сообщения? Или все вместе? Или что-то еще?
4. Большинство специалистов в области моделирования согласны с тем, что процесс имитационных вычислений должен быть детерминированным. Детерминированность процесса вычислений может, например, означать что создаются списки на выполнение каких-то действий с какими-то объектами. При этом объекты попадают в эти списки в случайном (в смысле отсутствия у разработчика модели явных намерений на это счет) порядке, например, становятся первыми в списке из-за того, что когда-то были первыми описаны или созданы. Возникает вопрос, как организовать имитационные вычисления

так, чтобы одни объекты моделирования не имели систематических преимуществ перед другими из-за случайно полученного места в том или ином списке обработки?

На поставленные выше вопросы автор попытается найти ответы в следующих главах.

Глава II. Концепция моделирования

В предыдущей главе был дан обзор современного состояния концепций и средств распределенного моделирования сложных систем. Из этого обзора можно сделать следующие выводы:

1. Задача синтеза сложной системы весьма нетривиальна, однако разрешима, причем может быть решена различными способами.
2. Область разработки схем синтеза сложных моделей в настоящее время находится в той стадии развития, когда разработчики удовлетворяются тем, что синтез, наконец, построен, не вдаваясь в рефлексии относительно того, почему выбрана именно эта, а не другая его схема, чем она лучше или хуже других и т.д., довольствуясь лишь тем, что выбранная схема работает для некоторого класса моделей (границы которого, впрочем, тоже редко становятся предметом специального исследования).
3. При построении упомянутого весьма сложного синтеза, разработчики не ограничивают себя в средствах его реализации какой-либо специальной дисциплиной их применения, по-видимому, полагая, что для решения столь сложной задачи избыточных средств не бывает. В результате, приходится решать задачу взаимодействия нескольких параллельно протекающих процессов в самом общем виде, откуда возникают такие непростые механизмы синхронизации времени, как TimeWrap, или оптимистический механизм. Отсюда же возникает забота о дисциплине доступа к фазовым переменным модели, и связанные с ней проблемы зацикливания, при ожидании доступа к фазе.

В данной работе мы будем исходить из того, что реализуемая модель удовлетворяет некоторому набору основных

гипотез (без выполнения которых имитационное моделирование сложных систем представляется нам невозможным). Приняв эти гипотезы, мы окажемся в принципиально иной ситуации – взаимодействовать между собой будут не произвольные параллельные процессы, а такие, которые являются составляющими модели, удовлетворяющей этим гипотезам. Казалось бы, разница невелика – что не может быть моделью сложной системы? Тем не менее, используя свойства модели, заявленные в основных гипотезах, удается построить достаточно простой синтез модели сложной системы из компонент, при этом решая за счет определенной дисциплины проектирования модели такую проблему, как, например, доступ к фазовым переменным.

2.1. Гипотеза о замкнутости

В основе предлагаемой концепции моделирования, ориентированной на широкий класс распределенных моделей сложных систем, лежит гипотеза о замкнутости моделей этого класса. Попробуем дать краткую аннотацию этой гипотезе. Ее основой является допущение о том, что моделирование изучаемого явления возможно в принципе, иначе просто нет предмета для дальнейшего разговора. Возможность моделирования означает возможность реализации детерминированного (даже для стохастических характеристик) и однозначного процесса вычислений последующих значений характеристик модели по их предыдущим значениям и значениям внешних переменных модели.

2.1.1. Гипотеза о замкнутости в первом приближении, внутренние переменные

Предположим, что мы изучаем некоторое явление, которое хотим моделировать. Шанс изучить это явление методами точных наук появляется, если удается измерить и численно выразить интересующие нас характеристики этого явления,

или как их еще называют, его внутренние переменные. В противном случае, упомянутое явление остается предметом изучения наук гуманитарных.

Предположим, что мы измеряем и изучаем численные характеристики $X_1, X_2, \ldots$, нашего явления. Наконец, на некотором X_n нам начинает казаться, что выбранных нами характеристик уже достаточно, чтобы описать данное явление полностью, или же с интересующей нас точностью, или хотя бы только то, что интересует нас в этом явлении. Например, в ставшей классической в области глобального моделирования работе [59], Дж. Форрестеру хватило пяти внутренних переменных, для описания в некотором устраивавшем его и его заказчиков приближении всей мировой динамики.

Утверждение о том, что выбранных характеристик $X_1, X_2, \ldots, X_n$ достаточно, чтобы полностью описать то, что интересует нас в изучаемом явлении, и есть гипотеза о замкнутости нашей модели. Из нее непосредственно следует, что если теперь нас интересует изменение выбранных нами внутренних переменных, например во времени, – то ему просто не от чего больше зависеть, кроме как от самих этих переменных (в виртуальном мире, образуемом нашей замкнутой моделью, больше просто ничего нет).

Теперь, например, если есть веские основания считать зависимость внутренних переменных модели от времени абсолютно непрерывной, мы можем искать ее в виде обыкновенных дифференциальных уравнений:

$$\frac{dX_i}{dt} = F_i(X_1, \ldots, X_n), i = 1, \ldots, n. \qquad (2.1)$$

Если кроме изменения во времени нас интересует распределение одних внутренних переменных относительно других – получим систему дифференциальных уравнений в частных производных.

Дифференциальные уравнения, обыкновенные и в частных производных, являются основным языком точных наук со времен Ньютона и Лейбница. Однако, если предположение о непрерывности неестественно для нашей модели (например, зависимости некоторых переменных явно носят дискретный характер), гипотеза о замкнутости, тем не менее, провозглашает существование однозначной зависимости значений внутренних переменных в последующий момент времени, от их значений в предыдущий.

$$X_i(t+\Delta t) = F_i(X_1,...,X_n,\Delta t),\ i=1,...,n \qquad (2.2)$$

Гипотеза о замкнутости являет собой некий аналог неконструктивных теорем о существовании и единственности таких математических объектов, как например, решения дифференциальных уравнений. Провозглашается существование зависимостей вида (2.1)-(2.2), но ничего не говорится о том, как их найти – вид правых частей (2.1)-(2.2), вообще говоря, неизвестен.

Задача разработчика модели – найти эти правые части в виде некоторых аналитических зависимостей, или же реализовать их некоторым однозначным и детерминированным вычислительным алгоритмом.

2.1.2. Уточнение гипотезы о замкнутости, внешние переменные

Выдвинутое гипотезой о замкнутости предположение о том, что мы можем рассматривать наше явление изолировано от всего остального мира (изменения внутренних переменных зависят лишь от самих этих переменных), несомненно, является определенным упрощением: на самом деле все в этом мире взаимосвязано. С этими связями разработчик модели обычно сталкивается, когда начинает придумывать и реализовывать

неизвестные, но постулируемые гипотезой о замкнутости правые части уравнений (2.1)-(2.2).

Этот раздел назван уточнением гипотезы о замкнутости потому, что в процессе придумывания и построения правых частей уравнений (2.1)-(2.2) разработчику приходится уточнять первоначальную гипотезу, например, добавляя к первоначальным переменным новые, чтобы замкнуть модель. Так, например, в уже цитированной выше работе [59], Дж. Форрестеру пришлось к основным пяти переменным добавить около двух десятков вспомогательных, и примерно столько же связывающих их между собою функциональных зависимостей, для того, чтобы модель замкнулась.

Кроме того, всегда оказывается, что создаваемые правые части зависят помимо внутренних переменных $X_1, X_2, ..., X_n$, еще и от некоторого набора параметров $a_1, a_2, ..., a_m$, которые еще называют внешними переменными. Например, к внешним переменным относятся управления, если мы исследуем управляемую систему. В качестве еще одного примера можно привести следующий распространенный способ построения не слишком сложных моделей: неизвестная правая часть (2.1)-(2.2) представляется рядом по степеням аргументов, берется несколько первых членов этого разложения (обычно, один – два). Коэффициенты разложения и будут параметрами модели. Обычно в процессе идентификации модели их выбирают так, чтобы обратный прогноз характеристик был максимально близок набранной статистике – очевидно, одно из необходимых условий адекватности модели, но, к сожалению, не являющееся достаточным.

Параметры или внешние переменные модели отличаются от внутренних переменных тем, что они не являются предметом моделирования, а считаются заданными извне. Можно сказать, что внутренние переменные модели описывают состояние явления, в той мере, в какой это интересно или посильно разработчику модели, внешние же переменные описы-

вают все влияния внешнего по отношению к изучаемому явлению мира, которые учитываются при его моделировании.

Внутренние переменные зависят от внешних, но не наоборот, по крайней мере в рамках рассматриваемой модели. Иногда последнее утверждение называют **гипотезой об инвариантности** (например, [50]). В самом деле, если признать зависимость внешних переменных от внутренних – первые перестают чем-либо отличаться от последних – они также начинают входить в левые части соотношений (2.1)-(2.2) и становятся, тем самым, предметом моделирования, сама модель при этом становится несколько более подробной, за счет увеличения числа характеристик.

В свете сказанного выше, вернемся к началу предыдущего раздела. Мы измеряем и изучаем численные характеристики $X_1, X_2, \ldots$ исследуемого явления. Все больше характеристик становится нам доступно, однако, где-то нужно остановиться, если мы не собираемся построить модель «всего-всего на свете». Останавливаться будем в два этапа, во-первых, выберем внутренние переменные $X_1, X_2, \ldots, X_n$, которые на взгляд разработчика модели достаточно полно характеризуют изучаемое явление, именно их мы будем моделировать. Во-вторых, выберем внешние переменные $a_1, a_2, \ldots, a_m$. Про них можно сказать, что игнорировать их влияние на изучаемое явление в пределах точности разрабатываемой модели было бы нечестно, однако моделировать их мы не будем (суть гипотезы об инвариантности в том, что мы просто не берем внешние переменные внутрь модели). Теперь происходит окончательная остановка в выборе характеристик – кроме внутренних и внешних переменных в нашем виртуальном мире больше ничего нет.

Нежеланию моделировать внешние переменные может быть несколько причин:

1. Внешние переменные и в самом деле лежат вне данной модели, например, это внешние управления или важные

для изучаемого явления характеристики внешнего по отношению к нему мира.

2. В данной модели этого не требуется. Например, часто и охотно при построении моделей в качестве внешних переменных берут те, характерное время изменения которых, в пределах точности модели, на порядки превосходит характерное время изменения внутренних переменных. В этом случае при моделировании удобно считать их константами.
3. Мы просто не умеем их моделировать. Нередко оказывается, что пока нет идей, как содержательно моделировать некоторые характеристики явления, или же такие идеи есть, но они сопровождаются пониманием, что это моделирование будет на порядки сложнее, чем можно было бы себе позволить в рамках разрабатываемого проекта.
4. Нет смысла самим моделировать эти характеристики, так как их давно и успешно моделирует известный нам коллектив, результатами работ которого мы собираемся воспользоваться.

Теперь окончательно сформулируем гипотезу о замкнутости. Изменения внутренних переменных модели однозначно зависят от совокупности ее внутренних и внешних переменных. Уравнения (2.1)-(2.2) можно переписать в виде:

$$\frac{dX_i}{dt} = F_i(X_1,...,X_n,a_1,a_2,...,a_m),\ i=1,...,n; \qquad (2.3)$$

$$X_i(t+\Delta t) = F_i(X_1,...,X_n,a_1,a_2,...,a_m,\Delta t),\ i=1,...,n. \qquad (2.4)$$

При этом по-прежнему перед разработчиком стоит задача найти, придумать и практически реализовать зависимости, обозначенные в правой части буквой F.

О том, что разработчик способен решить эту далеко не тривиальную задачу, утверждает следующее предположение о

детерминированности и однозначности вычислительного процесса моделирования.

2.1.3. Детерминированность и однозначность вычислительного процесса моделирования

Детерминированность и однозначность вычислительного процесса моделирования соотносится с гипотезой о замкнутости примерно так же, как соотносятся конструктивные и неконструктивные теоремы о существовании и единственности. Фактически постулируется способность разработчика построить функционирующую модель изучаемого явления, то есть, реализовать детерминированный и однозначный процесс вычислений последующих характеристик явления, в зависимости от их предыдущих значений и предыдущих значений внешних переменных.

Если же такой способности нет – то пока и говорить дальше не о чем! Стало быть, время построения содержательной модели еще не настало, нужно продолжать измерять и изучать характеристики явления, например, методами информационного моделирования, строить временные ряды, искать корреляции и т.д., наконец, искать новые характеристики. До тех пор, пока не появится возможность принять гипотезу о замкнутости модели и о возможности построения детерминированного и однозначного процесса вычисления ее характеристик.

Заметим, что детерминированность вычислительного процесса моделирования вовсе не означает детерминированности самой модели. Модель может быть стохастической от начала и до конца, однако, ее разработчик должен четко знать, когда включить генератор случайных чисел и как однозначно вычислить величины, которые в данной модели считаются случайными.

Разработчик может заранее не знать, как поведет себя модель в тот или иной промежуток модельного времени (соб-

ственно, очень часто модель разрабатывается именно для того, чтобы наблюдать и прогнозировать с ее помощью поведение моделируемой системы в той или иной ситуации). Однако он всегда обязан знать, как об этом детерминировано и однозначно узнавать во время имитационного эксперимента, иначе никакой модели не получится.

Если детерминированность вычислительного процесса постулирует возможность вычисление любой из характеристик модели в любой из моментов модельного времени, то однозначность означает, что если в некоторый момент модельного времени почему-то появляется несколько значений одной и той же характеристики модели (например, ее вычисляют различными способами), то должен быть детерминированный механизм выбора из этого множества единственного значения. Этот выбор может осуществляться, например, посредством свертки имеющегося набора значений с некоторыми весами, а может носить и чисто вероятностный характер.

Итак, в дальнейшем мы будем рассматривать лишь те модели, для которых возможно постулировать:

1. Достаточность внутренних переменных для исчерпывающего описания состояния модели.
2. Достаточность внешних переменных для исчерпывающего описания взаимодействия модели с внешним по отношению к ней миром.
3. Зависимость в рамках модели внутренних переменных от внешних, но не наоборот (гипотеза об инвариантности).
4. Возможность детерминированного и однозначного вычислительного процесса имитации.

В заключение пункта сделаем два замечания относительно детерминированности и однозначности вычислительного процесса моделирования.

Во-первых, заметим, что высказанное выше требование детерминированности и однозначности вычислительного про-

цесса есть некий идеал, к которому следует стремиться при создании моделей. Реально в достаточно сложных системах, например, даже в известных операционных системах, эксплуатирующихся десятилетиями, – время от времени находятся уязвимости – с точки зрения моделирования, недокументированные цепочки внешних переменных, приводящие к неожиданному для разработчика результату. Например, к получению злоумышленником доступа к ресурсам компьютера. Такого рода уязвимости приходится отлаживать – выявлять и устранять. В упомянутом выше примере – десятилетиями.

Во-вторых, однозначность, вообще говоря, не означает единственности. То, что мы в любой ситуации знаем, как поступать дальше – несомненное благо с точки зрения организации вычислительного процесса, но это вовсе не означает, что нельзя было бы поступать каким-либо иным детерминированным образом, например, выбирая единственное значение характеристики, полученной несколькими способами.

2.2. Вопросы декомпозиции модели

Очень часто при моделировании сложной системы декомпозиция модели задается самой предметной областью моделирования: нередко мы изначально знаем, из каких компонент создана или могла бы быть создана сложная система, когда и по каким правилам эти компоненты взаимодействуют между собой, а целью исследования и моделирования как раз и является выяснение вопроса, как «все это вместе» будет функционировать в различных ситуациях. Именно так, например, обстояло дело в приводившемся выше примере про изучение возможностей СОИ.

Ситуация может быть и иной, например, если модель сложной системы может быть описана дифференциальными уравнениями. В системах дифференциальных уравнений, обыкновенных или в частных производных, в каждый момент времени каждая переменная зависит от всех остальных, и точ-

ная декомпозиция таких систем возможна крайне редко, в весьма экзотических случаях (например, [53]). Тем не менее, как будет показано далее, при определенных условиях накладываемых на правые части, возможна приближенная декомпозиция систем дифференциальных уравнений, причем при заданной точности приближения для любого конечного отрезка времени моделирования будет достаточно конечного числа моментов синхронизации компонент декомпозиции.

2.2.1. Дифференциальные уравнения и объектный анализ

Через всю историю науки проходят два способа изучения явлений – феноменологического, как описания взаимосвязи глобальных характеристик явления, и атомистического, как описания результата взаимодействия совокупности отдельных атомов, составляющих явление и имеющих определенные известные характеристики.

Описание дифференциальными уравнениями связей между глобальными характеристиками явления было основной методикой исследования в естественных науках, после создания Ньютоном и Лейбницем основ дифференциального исчисления.

Атомистический подход, тем не менее, тоже плодотворно применялся. Например, одним из крупнейших успехов физики конца XIX столетия было создание кинетической теории газов, позволившей дать атомистические интерпретации термодинамических потенциалов.

Волна интереса к атомистическому (объектному) представлению в информатике усилилась в 80-х годах XX века, как отклик на потребности с одной стороны моделирования сложных систем, а с другой – автоматизации проектирования таких систем. Появились объектно-ориентированные языки программирования, в которых класс объектов трактуется как:

- набор характеристик (данные, связанные с объектами этого класса);
- методы (функциональности объектов этого класса);
- события (некоторые сочетания характеристик, на которые объекты этого класса должны реагировать определенным образом, поскольку они «так устроены»).

Кроме того, в этих языках реализованы такие идеи как наследование объектами свойств и полиморфизм, облегчающие задачу построения новых объектов на основе готовых, а также инкапсуляция, открывающая пользователю функциональность объекта и скрывающая детали ее реализации.

Отметим, что объект в указанном выше смысле вполне может рассматриваться и как классическая математическая конструкция. Действительно, определяемое в объектном анализе понятие класса объектов, является частным случаем понятия «рода структуры» в бурбаковском формализме [38], [53].

2.2.2. Связь между динамическими и объектными моделями

В системе дифференциальных уравнений в каждый момент времени каждая переменная зависит от каждой.

Рассмотрим задачу Коши для динамической системы

$$\dot{\vec{x}} = f(\vec{x}, t),\ \vec{x}(0) = \vec{x}_0,\ t \in [0, T] \qquad (2.5)$$

Классическая теория декомпозиции [53] изучает факторобъекты и подъобъекты математических объектов, в том числе и систем дифференциальных уравнений. Сложность в том, что как упоминалось ранее, точная декомпозиция в указанном смысле системы вида (2.5) – редкое явление. Приведем два примера. Пусть функция f гладкая, и существует диффеоморфизм $\vec{z} = I(\vec{x}, t)$, $\vec{z} = (\vec{z}^1, \vec{z}^2)$, приводящий систему (2.5) к виду:

$$\dot{\vec{z}}^1 = G^1(\vec{z}^1, t), \tag{2.6}$$

$$\dot{\vec{z}}^2 = G^2(\vec{z}^1, \vec{z}^2, t). \tag{2.7}$$

Это означает, что $\vec{z}^2$ зависит от $\vec{z}^1$, но не наоборот, – $\vec{z}^1$ совершенно не зависит от $\vec{z}^2$. Таким образом, выполнена гипотеза об инвариантности, и $\vec{z}^1$ можно считать внешними переменными модели (2.7). Модель же (2.6) никак не зависит от (2.7). В данном случае (2.6) есть фактор-система системы (2.5). Можно, например, считать, что у некоторого управляющего органа есть агрегированная модель (2.6) явления (2.5), и на основании этой модели он вырабатывает управления $\vec{z}^1$, которыми воздействует на управляемую систему (2.7). При этом характеристики $\vec{z}^2$ системы (2.7) вне поля зрения этого управляющего органа.

Еще более экзотический случай – когда система (2.5) распадается на два подъобъекта, т.е. (2.7) не зависит от $\vec{z}^1$:

$$\dot{\vec{z}}^2 = G^2(\vec{z}^2, t). \tag{2.8}$$

В этом случае исходная модель (2.5) распадается на две ничем не связанные между собой модели (2.6) и (2.8). Такое тоже встречается и в жизни и в моделировании (например, [28]), но крайне редко.

В свете сказанного выше, мы будем рассматривать более общий случай. Будем называть разбиением $\{A\}$ вектора $\vec{x}$ некоторое произвольное разбиение его компонент на два подвектора $\vec{x}^1$ и $\vec{x}^2$. В соответствии с этим разбиением, представим нашу систему (2.5) в виде:

$$\dot{\vec{x}}^1 = f^1(\vec{x}^1, \vec{x}^2, t), \tag{2.9}$$

$$\dot{\vec{x}}^2 = f^2(\vec{x}^1, \vec{x}^2, t). \qquad (2.10)$$

Здесь $\vec{x}^2$ – внешние переменные в (2.9), а $\vec{x}^1$ – внешние переменные в (2.10). Можно сказать, что мы, не производя над исходной моделью (2.5) никаких преобразований (вроде диффеоморфизмов предыдущих примеров), тем не менее, разбили ее на две замкнутые подмодели (2.9) и (2.10). Проблема только в том, что эти подмодели слишком тесно связаны между собой: они влияют друг на друга в каждый момент времени.

На уровне гуманитарного понимания проблемы, здесь уместно заметить, что, по-видимому, в основе объектного восприятия нами окружающего мира, лежит наша воспитываемая с раннего детства способность выделять из всей доступной нам совокупности его характеристик $\vec{x}$, в качестве отдельных объектов те наборы характеристик $\vec{x}^1$, $\vec{x}^2$, …, чьи компоненты взаимодействуют между собой внутри набора гораздо интенсивней, чем с характеристиками других наборов. Поэтому большинство известных нам объектов взаимодействуют между собой лишь иногда, а все остальное время вполне независимы друг от друга.

Вернемся к точным формулировкам. По-прежнему будем считать, что разбиение $\{A\}$ вектора $\vec{x}$ на два подвектора $\vec{x}^1$ и $\vec{x}^2$ произвольно. Перейдем к приближениям системы (2.5). Это тем более уместно, что если она сколько-нибудь сложна – все равно ее придется решать численно. Покажем, что при определенных условиях, с любой степенью точности систему (2.5) можно приблизить совокупностью объектов, взаимодействующих друг с другом лишь в конечное число моментов времени – событий, а в промежутках между событиями независимых друг от друга. Пусть функция f в правой части достаточно «хорошая», например, удовлетворяет условию Липшица по совокупности своих переменных. Будем называть системой

событий $\{t_n\}$ разбиение отрезка $[0,T]$ точками $0 = t_0 < t_1 <,...,< t_n = T$. Через $\vec{x}_0(t)$ будем обозначать решение задачи Коши (2.5). Через $\vec{x}_{\{A\},\{t_n\}}(t)$ будем обозначать склеенное решение следующих задач Коши на отрезках $[t_i, t_{i+1}]$, $i = 0,..., n-1$:

$$\begin{aligned} \dot{\vec{x}}^1 &= f^1(\vec{x}^1, \vec{x}^2 \equiv \vec{x}^2(t_i), t),\ \vec{x}^1(t_i); \\ \dot{\vec{x}}^2 &= f^2(\vec{x}^1 \equiv \vec{x}^1(t_i), \vec{x}^2, t),\ \vec{x}^2(t_i). \end{aligned} \qquad (2.11)$$

Очевидно, $\vec{x}_{\{A\},\{t_n\}}(t)$ – непрерывная функция, так как ее производная кусочно-непрерывна. Таким образом, мы разбили исходную систему (1), где все переменные всегда зависят от всех, на два объекта $\vec{x}^1$ и $\vec{x}^2$.

С точки зрения приводившихся выше рассуждений о моделях и моделировании, компоненты $\vec{x}^2$ являются теперь внешними переменными для модели $\vec{x}^1$ и наоборот, компоненты $\vec{x}^1$ являются внешними переменными модели $\vec{x}^2$. Внешние переменные наших моделей изменяются дискретно в конечное число моментов времени – событий: t_i, $i = 0,..., n$. Модели $\vec{x}^1$ и $\vec{x}^2$ взаимодействуют между собой лишь в моменты событий, поставляя друг другу значения внешних переменных. В промежутках же между событиями они полностью независимы друг от друга.

Теорема 2.1. (О приближенной декомпозиции системы (2.5) на два подобъекта [51]).

Пусть функция f в правой части (2.5) удовлетворяет условию Липшица по совокупности своих переменных. Тогда для любого разбиения $\{A\}$ компонент вектора $\vec{x}$ на два подвектора $\vec{x}^1$ и $\vec{x}^2$, и для любого числа $\varepsilon > 0$, найдется конеч-

ная система событий $\{t_n\}$, такая что если $\vec{x}_0(t)$ - решение исходной задачи Коши (2.5), а $\vec{x}_{\{A\},\{t_n\}}(t)$ - решение задач Коши системы (2.11), то для этих решений будет справедлива оценка:

$$\max_{t\in[0,T]}\left|\vec{x}_0(t)-\vec{x}_{\{A\},\{t_n\}}(t)\right|\le\varepsilon.$$

Доказательство.

Возьмем последовательность разбиений отрезка $[0,T]$ на n равных частей. Пусть этим разбиениям соответствуют решения $\left\{\vec{x}_n^1(t),\vec{x}_n^2(t)\right\}$ системы (2.11). Из выполнения условий Липшица для функции f

$$\left|f(\vec{x},t)-f(\vec{x}_0,0)\right|\le const\left|\vec{x}\right|+constT$$

следует оценка:

$$-const-conste^{constT}\le\left\{\vec{x}_n^1(t),\vec{x}_n^2(t)\right\}\le const+conste^{constT},$$

или окончательно, $\left|\left\{\vec{x}_n^1(t),\vec{x}_n^2(t)\right\}\right|\le const,\ \forall n$, т.е., семейство функций $\left\{\vec{x}_n^1(t),\vec{x}_n^2(t)\right\}$ равномерно ограничено на $[0,T]$. Отсюда, и из условия Липшица следует еще одна оценка:

$$\left|f(\left\{\vec{x}_n^1(t),\vec{x}_n^2(t)\right\},t)\right|\le const,\forall n \qquad (2.12)$$

Далее, пусть $t,\tau\in[0,T]$ и $t<\tau$, оценим разность

$$\left|\left\{\vec{x}_n^1(\tau),\vec{x}_n^2(\tau)\right\}-\left\{\vec{x}_n^1(t),\vec{x}_n^2(t)\right\}\right|.$$

Предположим, что на интервал (t,τ) попало $k \geq 0$ точек из нашего разбиения отрезка $[0,T]$ на n равных частей, $t_1,...,t_k$. Обозначим $t_0 = t$ и $t_{k+1} = \tau$, тогда

$$\left\{\vec{x}_n^1(\tau), \vec{x}_n^2(\tau)\right\} - \left\{\vec{x}_n^1(t), \vec{x}_n^2(t)\right\} =$$

$$= \sum_{i=0}^{k} \left(\left\{\vec{x}_n^1(t_{i+1}), \vec{x}_n^2(t_{i+1})\right\} - \left\{\vec{x}_n^1(t_i), \vec{x}_n^2(t_i)\right\}\right).$$

На каждом из отрезков $[t_i, t_{i+1}], i = 0,...,k$, функция $\left\{\vec{x}_n^1(t), \vec{x}_n^2(t)\right\}$ дифференцируема, поэтому из (2.12) следует

$$\left|\left\{\vec{x}_n^1(t_{i+1}), \vec{x}_n^2(t_{i+1})\right\} - \left\{\vec{x}_n^1(t_i), \vec{x}_n^2(t_i)\right\}\right| \leq const(t_{i+1} - t_i),$$

суммируя это неравенство по i от 0 до k, получаем

$$\left|\left\{\vec{x}_n^1(\tau), \vec{x}_n^2(\tau)\right\} - \left\{\vec{x}_n^1(t), \vec{x}_n^2(t)\right\}\right| \leq const(\tau - t), \forall n.$$

Последняя оценка означает равностепенную непрерывность семейства функций $\left\{\vec{x}_n^1(t), \vec{x}_n^2(t)\right\}$ на $[0,T]$. Следовательно, по теореме Арцела [44], из этого семейства можно выделить равномерно сходящуюся подпоследовательность, которую также будем обозначать $\left\{\vec{x}_n^1(t), \vec{x}_n^2(t)\right\}$. Пусть ее предел – функция $\left\{\vec{x}_*^1(t), \vec{x}_*^2(t)\right\}$.

Рассмотрим оператор

$$\Phi(x) = x(t) - \int_0^t f(x(\tau), \tau) d\tau,$$

и попробуем оценить величину $\left\|\Phi(\left\{\vec{x}_*^1(t), \vec{x}_*^2(t)\right\})\right\|_{C[0,T]}$. Построим аналогичный оператор и для декомпозированной системы (2.11).

Обозначим $t^*(t) = \max\left\{t_j : t_j \in \{t_n\}, t_j \le t\right\}$,

$J(t) = j : 0 \le j < n;\ t_j = t^*(t)$.

Положим

$$\Psi(\{x^1, x^2\}) = \{x^1(t), x^2(t)\} -$$

$$- \int_{t^*(t)}^{t} \{f^1(x^1(\tau), x^2(t^*(t)), \tau),\ f^2(x^1(t^*(t)), x^2(\tau), \tau)\} d\tau -$$

$$- \sum_{j=1}^{J(t)} \int_{t_{j-1}}^{t_j} \{f^1(x^1(\tau), x^2(t_{j-1}), \tau),\ f^2(x^1(t_{j-1}), x^2(\tau), \tau)\} d\tau .$$

Так как $\left\{\vec{x}_n^1(t), \vec{x}_n^2(t)\right\}$ – решение системы (2.11), то $\Psi(\left\{\vec{x}_n^1(t), \vec{x}_n^2(t)\right\}) = 0$, поэтому справедлива следующая оценка:

$$\left\|\Phi(\left\{\vec{x}_*^1(t), \vec{x}_*^2(t)\right\})\right\| = \left\|\Phi(\left\{\vec{x}_*^1(t), \vec{x}_*^2(t)\right\}) - \Psi(\left\{\vec{x}_n^1(t), \vec{x}_n^2(t)\right\})\right\| \le$$

$$\le \left\|\left\{\vec{x}_*^1(t), \vec{x}_*^2(t)\right\} - \left\{\vec{x}_n^1(t), \vec{x}_n^2(t)\right\}\right\| +$$

$$+ \sum_{i=0}^{n} \int_{t_i}^{t_{i+1}} \Big| f(\left\{\vec{x}_*^1(\tau), \vec{x}_*^2(\tau)\right\}, \tau) -$$

$$- \left\{f^1(\left\{x_n^1(\tau), x_n^2(t_i)\right\}, \tau),\ f^2(\left\{x_n^1(t_i), x_n^2(\tau)\right\}, \tau)\right\}\Big| d\tau .$$

Отсюда, и из равностепенной непрерывности семейства функций $\left\{\vec{x}_n^1(t), \vec{x}_n^2(t)\right\}$ и условия Липшица, следует оценка:

$$\left\|\Phi\left(\left\{\vec{x}_*^1(t), \vec{x}_*^2(t)\right\}\right)\right\| \le \max_{t\in[0,T]}\left|\left\{\vec{x}_*^1(t), \vec{x}_*^2(t)\right\} - \left\{\vec{x}_n^1(t), \vec{x}_n^2(t)\right\}\right| +$$

$$+ const \max_{t\in[0,T]}\left|\left\{\vec{x}_*^1(t), \vec{x}_*^2(t)\right\} - \left\{\vec{x}_n^1(t), \vec{x}_n^2(t)\right\}\right|.$$

Так как эта оценка верна для всех n, а $\left\{\vec{x}_n^1(t), \vec{x}_n^2(t)\right\}$ равномерно сходится к $\left\{\vec{x}_*^1(t), \vec{x}_*^2(t)\right\}$, заключаем что $\Phi\left(\left\{\vec{x}_*^1(t), \vec{x}_*^2(t)\right\}\right) = 0$, т.е., $\left\{\vec{x}_*^1(t), \vec{x}_*^2(t)\right\}$ - решение исходной системы (1). Но тогда, в силу единственности решения системы (1), справедливо: $\left\{\vec{x}_*^1(t), \vec{x}_*^2(t)\right\} = \vec{x}_0(t)$. Следовательно, последовательность $\left\{\vec{x}_n^1(t), \vec{x}_n^2(t)\right\}$ равномерно сходится к $\vec{x}_0(t)$, и, стало быть, начиная с некоторого номера N, при всех $n \ge N$ будет справедливо:

$$\max_{t\in[0,T]}\left|\vec{x}_0(t) - \left\{\vec{x}_n^1(t), \vec{x}_n^2(t)\right\}\right| \le \varepsilon,$$

что и завершает доказательство теоремы. ■

Отметим, что данная теорема не является оригинальной. Подобного сорта факты хорошо известны всем изучавшим или занимавшимся теорией и методами приближенных вычислений. Здесь она приводится полностью, потому что, на взгляд автора, многое проясняет в контексте данной работы.

Еще отметим связь выполнения условия Липшица для правой части (2.5) с существованием конечного числа точек синхронизации для любого конечного отрезка времени моделирования. В пункте 2.2.4 подробно рассматривается простейший пример модели, где правая часть разрывна, и в результате этого, у точек синхронизации компонент модели появляется

точка накопления, что может привести к «зацикливанию» модели при приближении к этой точке.

2.2.3. Лапласовские модели

> Современные события имеют с событиями предшествующими связь, основанную на очевидном принципе, что никакой предмет не может начать быть без причины, которая его произвела...
>
> П.С. Лаплас

В предыдущем разделе мы в достаточной степени обсудили проблемы, связанные с замкнутостью модели и детерминированностью вычислительного процесса моделирования, чтобы, наконец, перейти к точным формулировкам.

Будем рассматривать моделирование некоторого явления на конечном отрезке времени $[0,T]$. Предположим, что нам известны значения внешних переменных нашей модели $\vec{a}(t)$ в любой точке $t \in [0,T]$.

Определение 2.1.

Будем называть модель замкнутой в точке $t \in [0,T)$, если найдется число $\Delta t > 0$, $t + \Delta t \in (0,T]$, которое будем называть отрезком прогноза модели для точки t, такое что для любого $\tau \in (t, t + \Delta t]$:

1. Известен алгоритм, позволяющий детерминировано, однозначно и с устраивающей разработчика модели точностью вычислять по значениям внутренних $\vec{x}(t)$ и внешних $\vec{a}(t)$ переменных в момент модельного времени t, значения внутренних переменных $\vec{x}(\tau)$ в момент модельного времени τ.
2. Упомянутый алгоритм обладает следующим свойством аддитивности: если выполняется предыдущий пункт и

$\tau \in (t, t+\Delta t)$, то алгоритм позволяет получить прогноз $\vec{x}(\theta)$, исходя из начальных значений $\vec{x}(\tau)$ и $\vec{a}(\tau)$, по крайней мере на полуинтервале $\theta \in (\tau, t+\Delta t]$, и этот прогноз с устраивающей разработчика точностью совпадает на отрезке $[\tau, t+\Delta t]$, с прогнозом, полученным на всем отрезке $[t, t+\Delta t]$, исходя из начальных значений характеристик $\vec{x}(t)$ и $\vec{a}(t)$.

Обсуждение определения замкнутости.

Данное выше определение может показаться недостаточно строгим, действительно, не очень понятно, что такое «алгоритм» в данном контексте, и тем более, что такое «устраивающая разработчика точность». Попробуем прояснить эти понятия.

Пусть $P \subset R_n$ – замкнутое ограниченное множество допустимых значений внутренних переменных модели, а $Q \subset R_m$ – замкнутое ограниченное множество допустимых значений ее внешних переменных. Будем называть алгоритмом отображение $\vec{x} = F(\vec{x}, \vec{a}, \Delta t)$, желательно сюръективное (что соответствовало бы управляемости модели внешними характеристиками), из $P \times Q \times [0, T]$ в P. Вообще говоря, отображение F не предполагается непрерывным, так как некоторые характеристики модели могут быть дискретными.

«Устраивающей разработчика точностью» можно назвать топологию на P, которую разработчик согласится считать таковой, т. е. оценивать близость в P, в смысле этой топологии. Близость в указанной топологии будем считать эквивалентностью. В этом есть определенная неточность, так как для настоящей транзитивности потребовалась бы свойство инфинитезимальности (неархимедовости) отклонений. Однако, в приближенных вычислениях часто подобное допущение принимается, так как любое конечное число «транзакций» всегда можно обеспечить.

Топология «устраивающей разработчика точности» вполне может оказаться слабее топологии задаваемой нормой R_n, и притом настолько, что в смысле этой топологии отображение F уже может оказаться непрерывным относительно Δt. Если эта непрерывность кроме того будет равномерной и равностепенной относительно $\vec{x}, \vec{a} \in P \times Q$, то алгоритм, задаваемый отображением F, удовлетворит условиям определения 2.1. Приведем два примера.

Пример 1.

Пусть модель задается системой обыкновенных дифференциальных уравнений

$$\dot{\vec{x}} = f(\vec{x}, \vec{a}, t),$$

где правая часть удовлетворяет условию Липшица по совокупности переменных. Тогда отображение, задающее алгоритм приближенного интегрирования системы имеет вид:

$$\vec{x}(t + \Delta t) = \vec{x}(t) + \Delta t f(\vec{x}(t), \vec{a}(t), t).$$

Можно убедиться, что так заданное отображение удовлетворяет определению 2.1.

Пример 2.

Часто при моделировании сложных систем, среди внутренних переменных модели есть и непрерывные, есть и дискретные, а упомянутая выше ослабленная топология «устраивающей разработчика точности» состоит в том, что в обычной топологии соответствующего подпространства R_n оценивается лишь часть непрерывных характеристик модели, а остальные характеристики просто не принимаются во внимание. По существу мы факторизуем фазовое пространство модели по части «неинтересных для разработчика» характеристик и пользуемся топологией фактор-пространства. Такой выбор можно объяснить тем, что как уже упоминалось выше, часто моделирование начинается с выбора ряда «основных» характеристик,

ради изучения которых и создается модель. Остальные же характеристики могут появиться в результате попыток замкнуть модель, и их близость может и не представлять для разработчика самостоятельного интереса. С непрерывными же характеристиками, описываемыми дифференциальными уравнениями, все получается также как и в предыдущем примере.

Из свойства аддитивности алгоритма данного выше определения замкнутости в точке следует

Лемма 2.1.

Замкнутая в точке модель, замкнута и по крайней мере во всех внутренних точках отрезка прогноза для этой точки. ■

Определение 2.2.

Будем называть модель локально замкнутой на отрезке $[0,T]$, если она замкнута в каждой точке $t \in [0,T)$.

Определение 2.3.

Будем называть модель прогнозируемой или лапласовской на отрезке $[0,T]$, если существует конечное разбиение этого отрезка точками $0 = t_0 < t_1 <,...,< t_n = T$, такое что модель замкнута в каждой из точек t_{i-1}, $i = 1,...,n$, и каждый из полуинтервалов $(t_{i-1}, t_i]$, $i = 1,...,n$, принадлежит отрезку прогноза для своего левого конца.

Лемма 2.2.

Если модель прогнозируема на отрезках $[t_0, t]$ и $[t, t_1]$, то она будет прогнозируемой и на отрезке $[t_0, t_1]$. ■

Определение 2.4.

Будем называть модель прогнозируемой в точке $t \in (0,T]$, если найдется точка $\tau_t \in [0,t)$, которую будем называть базовой для точки t, такая что:

1. Модель замкнута в точке τ_t.
2. Точка t принадлежит отрезку прогноза модели в точке τ_t: $t \in (\tau_t, \tau_t + \Delta\tau_t]$.

Определение 2.5.

Будем называть модель локально прогнозируемой на отрезке $[0,T]$, если она прогнозируема в каждой точке $t \in (0,T]$.

Теорема 2.2. (О прогнозируемости модели на отрезке).

Для прогнозируемости модели на отрезке модельного времени $[0,T]$, необходима и достаточна ее локальная замкнутость, и локальная прогнозируемость на этом отрезке.

Доказательство.

Необходимость. Пусть модель прогнозируема на отрезке $[0,T]$. Замкнутость модели в левых концах отрезков $[t_{i-1},t_i]$, $i=1,...,n$ следует из определения прогнозируемости. Следовательно, в силу леммы 2.1 она замкнута в любой из точек $t \in (t_{i-1},t_i)$, $i=1,...,n$, а стало быть и локально замкнута.

Любая точка $t \in (0,T]$ принадлежит одному из полуинтервалов $(t_{i-1},t_i]$, $i=1,...,n$ из определения прогнозируемости на отрезке. Стало быть, левый конец этого полуинтервала, точка t_{i-1}, удовлетворяет требованиям, предъявляемым определением 2.4 к базовой точке для t. Поэтому наша модель прогнозируема в любой точке $t \in (0,T]$, и, стало быть, локально прогнозируема.

Достаточность. Пусть наша модель локально замкнута и локально прогнозируема на отрезке $[0,T]$. Точку 0 покроем отрезком прогноза для нее $[0,\Delta_0]$, из определения замкнутости модели в этой точке. Точку T покроем отрезком $[\tau_T,T]$, где τ_T – базовая точка для T, из определения прогнозируемости в точке. Каждую точку $t \in (0,T)$ покроем отрезком $[\tau_t,t+\Delta t]$, где τ_t – базовая точка, а Δt – отрезок прогноза для t. Получаем, что каждая точка $t \in [0,T]$ покрыта отрезком, в который входит вместе с некоторой своей окрестностью: $[0,\Delta_0)$ для точки 0, $(\tau_T,T]$ для точки T, и $(\tau_t,t+\Delta t)$ для остальных

$t \in (0,T)$. Из способа построения отрезков покрытия следует, что на каждом из них модель прогнозируема смысле определения 2.3 прогнозируемости модели на отрезке.

В силу компактности отрезка $[0,T]$, из построенного покрытия можно выделить конечное покрытие. Выбросим из этого конечного покрытия все отрезки, целиком принадлежащие другим отрезкам. После этого любую пару пересекающихся отрезков $[a_1,b_1]$ и $[a_2,b_2]$ таких что $a_1 < a_2$, но $b_1 > a_2$, заменяем парой $[a_1,a_2]$ и $[a_2,b_2]$ и снова выбрасываем отрезки, целиком принадлежащие другим. Таким образом, получаем конечное покрытие из отрезков, пересекающихся не более чем в одной точке. По способу построения каждый отрезок либо остался исходным, либо уменьшен справа. Упорядочим концы отрезков, получаем:

$$0 = t_0 < t_1 <,...,< t_n = T .$$

На каждом из отрезков $[t_{i-1},t_i]$, $i = 1,...,n$, модель прогнозируема по построению этих отрезков. Следовательно, по лемме, 2.2 она является прогнозируемой на отрезке $[0,T]$, что и требовалось доказать. ■

Эта теорема представляет некий аналог известных теорем существования – гарантирует существование глобального прогноза на отрезке модельного времени $[0,T]$, при возможности локальных прогнозов для каждой точки.

Что касается единственности – ее, вообще говоря, нет, но вместо нее есть факторизация по не вполне строгому отношению эквивалентности, связанному с не до конца формализованной характеристикой нашего моделирования – выбором топологии «устраивающей разработчика модели точности вычислений». Все прогнозы на отрезке $[0,T]$, полученные в силу теоремы 2.2, наследуют характеристику близости в этой топологии, и, стало быть, в некотором смысле эквивалентны – от-

личаются друг от друга лишь в пределах устраивающей разработчика модели точности имитационных вычислений.

2.2.4. Пример модели локально замкнутой, но не лапласовской (не прогнозируемой в одной из точек отрезка)

Как следует из теоремы 2.2, локальная прогнозируемость плюс локальная замкнутость на отрезке эквивалентна прогнозируемости модели на этом отрезке времени. Из одной локальной замкнутости, вообще говоря, прогнозируемость на отрезке не следует, как показывает следующий пример.

Рассмотрим задачу, встречающуюся в курсе математики начальной школы. Два пешехода идут навстречу друг другу с постоянной скоростью. Между ними летает муха с постоянной по абсолютной величине скоростью, большей, чем скорость любого из пешеходов. Как только муха долетает до одного из пешеходов, она тут же разворачивается и летит к другому. В школьной задаче обычно спрашивается, какое расстояние пролетит муха за время от начала движения, до момента встречи пешеходов. Нас этот вопрос школьной задачи не будет интересовать совсем, наш интерес – воспроизведение в модельном времени описанного в условиях задачи процесса движения пешеходов и мухи.

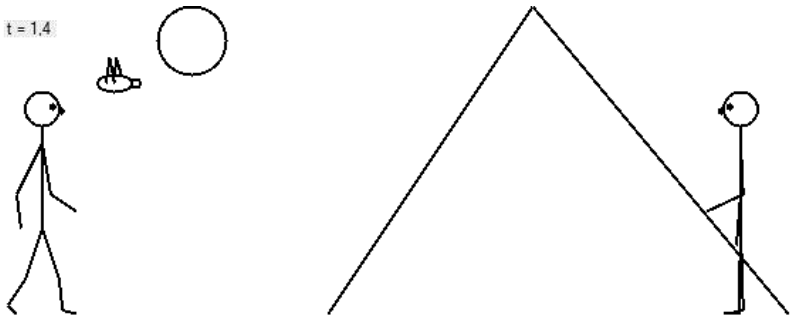

Рис.7. Пешеходы и муха.

Обычно автор, рассказывая эту модель студентам, спрашивает, а что будет происходить с ней после того, как пешеходы встретятся? На некоторое время в аудитории воцаряется молчание. Дело в том, что на языке приведенных выше определений, эта модель не прогнозируема в точке встречи пешеходов. Действительно, невозможно вычислить скорость мухи в этой точке – скорость мухи разрывна (стало быть, условию Липшица и теореме 2.1 наша модель не удовлетворяет), и в момент встречи пешеходов имеет два предельных значения: $v_{мухи}$ и $-v_{мухи}$. Поэтому в точке встречи пешеходов старая история приключений мухи полностью заканчивается, а новая не может начаться без дополнительных предположений, относительно ее скорости в этот момент. Всей предыстории модели недостаточно для определения скорости мухи в момент встречи пешеходов, это и означает, что модель не прогнозируема в этой точке.

Во все остальные моменты времени модель прогнозируема, и во все, включая момент встречи пешеходов – замкнута, и, стало быть, является локально замкнутой. Прогнозируемой же на любом отрезке времени, включающем точку встречи пешеходов, она не является, так как не прогнозируема в этой точке. В результате, для построения этой модели недостаточно конечного числа моментов синхронизации характеристик модели, как это требуется в определении прогнозируемой на отрезке модели. Действительно, каждый из моментов, когда муха долетает до любого из пешеходов, должен быть включен в моменты синхронизации модели, так как в эти моменты меняется направление скорости мухи, и если такой момент пропустить – муха пролетит мимо пешехода и будет нарушен сценарий модели. Моментов, когда муха долетает до одного из пешеходов, – бесконечное количество, и момент встречи пешеходов – их предельная точка.

В некотором смысле «спасти» данную модель может переход от точных вычислений к приближенным, использование

того самого не до конца формализуемого понятия «устраивающей разработчика модели точности вычислений». Действительно, при малых расстояниях между пешеходами, например, при расстояниях меньших характерного размера мухи, связь данной модели с какой-либо содержательной предметной областью становится очень слабой – «так в жизни быть не может, и никогда не бывает». Поэтому вполне разумным может быть предложение выбрать устраивающую разработчика точность вычислений, порядка характерных размеров мухи, и как только расстояние между пешеходами становится меньше этой точности, считать, что пешеходы уже встретились. Это предположение позволяет избежать дурной бесконечности и сохранить разумную связь модели с предметной областью моделирования, правда идентифицировать скорость мухи в точке встречи оно все равно не помогает.

2.2.5. Декомпозиция элементарной сложной модели

Итак, выше мы показали, что объектное представление не вполне чуждо даже для феноменологических моделей, заданных системами дифференциальных уравнений (особенно, если предполагается решать эти системы численными методами). Поэтому в дальнейшем объектное представление модели будет для нас основным.

Далее, в соответствии с теоремой 2.2, естественно потребовать от модели локальной прогнозируемости (иначе пришлось бы заботиться о ее идентификации в непрогнозируемых точках, и, возможно, бороться с бесконечным количеством точек синхронизации). После этого, упомянутая теорема обеспечивает нам конечное число точек синхронизации, и существование алгоритмов, вычисляющих характеристики модели в промежутках между ними.

Вообще говоря, после этого можно было бы приступать к реализации упомянутых алгоритмов. Однако, помимо принципиальных вопросов моделирования, рассмотренных выше,

имеются также важные технологические вопросы. Например, если модель по-настоящему сложная, и реализовывать ее предполагается коллективом разработчиков, технологично декомпозировать все алгоритмы на элементарные в вычислительном отношении так, чтобы разработчикам можно было максимально концентрировать внимание и усилия на реализации этих элементарных алгоритмов, и минимально – на организации внешних по отношению к ним вычислительных процессов. Также декомпозиция подобного рода полезна при организации распределенных вычислений, с возможностью для разработчиков моделей обмена по сети результатами своих разработок в области моделирования. Наконец, очень часто в предметной области моделирования ряд процессов развивается параллельно друг другу. Очень хотелось бы сохранить эту параллельность и в модели – распараллеливание вычислений в настоящее время один из магистральных путей развития информатики, причем распараллеливание произвольных алгоритмом зачастую дается с большим трудом. Тем более ценно сохранить эту параллельность там, где она досталась, что называется «от природы».

Попробуем декомпозировать имитационные вычисления на элементарные алгоритмы, каждый из которых реализуется соответствующей подпрограммой-методом.

Выделим имитационные вычисления, которые могут протекать параллельно друг другу. Такие вычисления будем называть процессами. Если в нашей модели параллельно ничего не происходит – значит, в ней имеется только один процесс.

Процесс – это то, что разворачивается в модельном времени последовательно. Алгоритмически различные этапы процесса будем называть элементами. Если алгоритмически различных этапов нет – процесс состоит из единственного элемента. Считается, что количество элементов процесса конечно и известно заранее. Элемент – это элементарный алгоритм, реализуемый одной подпрограммой-методом. Процесс реали-

зуется чередованием своих элементов. Чередование методов-элементов, в силу гипотезы о замкнутости модели, определяется состоянием внутренних и внешних переменных модели. Подробнее на правилах чередования элементов мы остановимся далее.

Одной из особенностей моделирования сложных систем можно считать необходимость учета разномасштабных по времени явлений, происходящих в них. Более подробное обсуждение связанных с этим проблем можно найти в [51], здесь же мы просто проведем классификацию элементов модели по отношению к модельному времени:

1. Сосредоточенные или быстрые элементы. Это элементы, которые происходят мгновенно по отношению к модельному времени. Сосредоточенные элементы – один из способов вычисления заведомо дискретных характеристик модели.
2. Распределенные или медленные элементы. Эти элементы имеют ненулевую продолжительность в модельном времени, т.е., занимают время не меньшее, характерного для данной модели интервала осреднения по времени. Кроме того, для таких элементов всякому разумному интервалу времени можно сопоставить результат выполнения элемента за этот интервал времени. Распределенные элементы – наиболее естественный способ вычисления непрерывных характеристик модели.
3. Условно-распределенные элементы. Такие элементы достаточно характерны для моделирования, например, информационных процессов. Они, как и распределенные, занимают ненулевое модельное время. Однако, в отличие от распределенных элементов, результат их деятельности возникает лишь в конце выполнения. Никаких промежуточных результатов за меньшее время быть не может, и если по каким-либо причинам работа такого элемента прерывается до его штатного оконча-

ния - никаких результатов она не приносит. Вообще говоря, условно-распределенные элементы можно не рассматривать отдельно, моделируя каждый такой элемент последовательностью пустого распределенного элемента и выдающего результат сосредоточенного.

Заметим, что наличие в модели сосредоточенных элементов, скорее всего, ставит крест на выполнении условия Липшица из теоремы 2.1. Действительно, сосредоточенный элемент, как и всякий другой, нужен чтобы изменять некоторые характеристики модели. Поскольку он не занимает модельного времени – изменяемые им характеристики как функции модельного времени меняются скачком, т.е., непрерывными быть не могут. Сказанное не означает, что при наличии сосредоточенных элементов, у моментов синхронизации модели обязательно появятся точки накопления. Модель может и при наличии дискретных характеристик оказаться локально прогнозируемой, и тогда с моментами синхронизации все будет в порядке. Тем не менее, наличие сосредоточенных элементов побуждает повысить бдительность по отношению к подобным проявлениям.

2.2.6. События. Диспетчеризация вычислительного процесса элементарной сложной модели

Теперь на основании декомпозиции вычислительного процесса и принятых предположений о его детерминированности и однозначности, попробуем построить синтез имитационных вычислений во времени.

Как мы видели выше, в теоремах 2.1-2.2, при различных исходных предположениях этих теорем, естественным образом возникают точки синхронизации имитационных вычислений, в дальнейшем мы будем называть эти точки событиями. Посмотрим, как развиваются в модельном времени выделенные нами параллельные процессы, и что будет являться событиями.

В первом приближении можно сказать, что все модельное время занимает выполнение распределенных процессов с некоторым шагом моделирования Δt, который будем называть стандартным.

Если мы предполагаем прогнозируемость нашей модели (а без такого предположения не очень понятно как мы сможем ее построить), очевидно, стандартный шаг моделирования не должен быть больше минимального отрезка прогноза модели. На самом деле это означает, что величина стандартного шага моделирования согласуется с масштабом осреднения характеристик модели: шаг моделирования Δt не может быть слишком велик, за время Δt непрерывные характеристики модели должны изменяться не слишком сильно, иначе в модели может возникнуть явление динамического хаоса [51].

Во-вторых, может оказаться, что продолжительность выполнения некоторых элементов в модельном времени несоизмерима с величиной стандартного шага Δt. Несмотря на то, что непрерывные характеристики модели, как мы предположили, мало изменяются на интервале Δt, мы считаем неверным откладывать окончание выполняемого элемента до окончания шага моделирования. Действительно, у сложной модели могут быть и разрывные, и даже дискретные характеристики. Кроме того, окончание какого-либо элемента в принципе может вызвать целый каскад сосредоточенных элементов, часть которого или даже весь можно и упустить, отложив момент синхронизации. В предлагаемой концепции моделирования, в случаях несоизмеримости момента окончания элемента со стандартным шагом моделирования, предлагается сокращать продолжительность шага моделирования до момента окончания заканчивающегося элемента.

В-третьих, из принятого предположения о прогнозируемости модели и о детерминированности и однозначности вычислительного процесса следует, что в начале каждого шага

моделирования возможно прогнозирование окончания любого из выполняемых на этом шаге элементов.

События – точки синхронизации процессов модели. Событие – всегда окончание шага моделирования, оно может быть обусловлено:

1. Окончанием стандартного шага моделирования.
2. Окончанием выполнения текущего элемента (смена элементов).
3. Вычислением характеристик модели, важных в данный момент модельного времени для других процессов. Важных в данном контексте – значит таких, с учетом которых в каком-то другом процессе может произойти смена элементов.

В первом и третьем случаях смена выполняющегося элемента необязательна (но может и совпасть). Больше никаких событий в системе декомпозированной выбранным нами способом произойти не может.

Из гипотезы о замкнутости модели следует, что наступление любого события в некий момент модельного времени может зависеть лишь от значений внутренних и внешних переменных модели в этот момент (ничего другого в нашем виртуальном мире просто нет). Из предположения о локальной прогнозируемости модели следует, что находясь в левом конце очередного шага моделирования t_i, разработчик модели имеет возможность однозначно и с устраивающей его точностью определить, когда и какие события наступят на интервале модельного времени $[t_i, t_i + \Delta t]$. Отметим, что никакой необходимости в применении схем управления временем с правого конца шага моделирования, типа «оптимистической» или Time-Wrap здесь не возникает. На уровне гуманитарного (а стало быть, и субъективного) понимания автором ситуации, можно заметить, что, по-видимому, все, что было существенного для моделирования сложных систем в этих схемах, некоторым об-

разом вошло в определение 2.4 понятия прогнозируемости модели в точке.

Из сказанного выше можно вывести следующую схему алгоритмизации вычисления событий. Событиями называются методы, вычисляющие неотрицательное вещественное число, которое трактуется затем как прогноз наступления соответствующего события, т. е., как интервал модельного времени от текущего момента до момента наступления события. Если вычисленное число равно нулю – событие наступило.

Таким образом, если в некотором процессе возможен переход от элемента A к элементу B, то обязан существовать метод-событие $E_{\{A,B\}}$, прогнозирующее этот переход.

Из перечисленных выше трех типов точек синхронизации модели, следует также возможность существования событий вида $E_{\{A,A\}}$. Это означает, что элемент A не собирается заканчиваться, но обращает внимание всех выполняющихся в текущее время элементов на полученные им характеристики модели. Возможно, они вызовут смену элементов в каких-то параллельных процессах.

Отметим, что из принятия требования детерминированности и однозначности вычислительного процесса моделирования, следует невозможность одновременного наступления событий $E_{\{A,B\}}$ и $E_{\{A,C\}}$, относящихся к одному и тому же процессу. Если нечто подобное происходит – это свидетельствует лишь о недостаточной проработанности модели.

Теперь мы готовы привести схему имитационных вычислений элементарной модели сложной системы. Во-первых, задается стандартный шаг моделирования Δt. Во-вторых, считается, что в начале шага моделирования известны текущие элементы всех процессов и все внутренние и внешние характеристики модели. Далее,

1. Вычисляются события связанные с текущими элементами процессов. Если есть наступившие события, про-

веряется нет ли переходов к быстрым (сосредоточенным) элементам, если они есть – выполняются быстрые элементы (они становятся текущими), затем возврат к началу п.1; если нет переходов к быстрым элементам – совершаются переходы к новым медленным (распределенным) элементам, затем возврат к началу п.1.

2. Если нет наступивших событий – из всех прогнозов событий выбирается ближайший $\Delta\tau$.
3. Если стандартный шаг моделирования не превосходит прогнозируемого времени до ближайшего события, $\Delta t \le \Delta\tau$ – моделируем текущие распределенные элементы со стандартным шагом Δt. В противном случае – моделируем их с шагом времени до ближайшего спрогнозированного события $\Delta\tau$.
4. Возвращаемся к началу п.1.

Может возникнуть вопрос, а не окажется ли у множества событий точек накопления, при предлагаемой системе выбора шага моделирования? На него отвечают теоремы 2.1 и 2.2. Например, если модель локально прогнозируема на отрезке моделирования, или все элементы описаны дифференциальными уравнениями, все правые части которых удовлетворяют условию Липшица – существует возможность организовать имитационные вычисления без точек накопления.

2.2.7. Компонента - элементарная сложная модель

Перейдем теперь к формальному описанию основной конструкции предлагаемой концепции моделирования. Назовем ее компонентой. Компонента – это в некотором смысле «элементарная» сложная система. Основой конструкции компоненты будет объект объектного анализа, в том смысле, что описываемая ниже компонента есть некий тип или класс моделей, а заполняться данными и запускаться на счет будут экземпляры этого класса. Опишем «устройство» компоненты.

1. Характеристики. Компонента, как и объект, имеет характеристики. Как и в случае динамических систем, эти характеристики мы будем разбивать на внутренние и внешние. Внутренние характеристики – это то, что компонента моделирует, внешние – это то, что она знает о внешнем мире. Обычно мы будем предполагать локальную прогнозируемость компоненты.

2. Процессы. Функциональность компоненты удобно структурировать следующим образом: Считается, что компонента реализует один или несколько параллельно выполняющихся процессов. Процесс состоит в чередовании элементов – алгоритмически элементарных методов.

3. Элементы. Элементарные, алгоритмически однородные методы, реализующие функциональности компоненты (т. е. то, что компонента умеет делать, получение на основании значений некоторых внутренних и внешних характеристик компоненты, новых значений некоторых ее внутренних характеристик).

По отношению к модельному времени некоторые элементы выполняются мгновенно, это сосредоточенные или быстрые элементы. Быстрыми элементами можно моделировать дискретные характеристики системы.

Выполнение других элементов занимает определенное время. Если при этом элемент для любого промежутка времени $\Delta\tau$, не превосходящего стандартный шаг моделирования Δt выдает некий осмысленный результат, такой элемент называется распределенным или медленным. Распределенные элементы – естественное средство вычисления непрерывных характеристик модели.

Может оказаться и так, что выполнение элемента занимает определенное модельное время, но результат его действия наступает лишь в конце, после полного выполнения элемента, т. е. никаких промежуточных результатов за время меньшее полного времени выполнения нет. Такие элементы

называются условно-распределенными. Вообще говоря, условно-распределенные элементы можно не рассматривать как отдельный класс, а моделировать парой: пустой распределенный элемент, за которым идет сосредоточенный, выдающий результат.

Частью предлагаемой концепции является жесткая дисциплина работы методов с фазовыми переменными модели: каждый метод имеет право изменять только «свои» переменные. Эта дисциплина основана на принятии предположения о детерминированности и однозначности имитационных вычислений. В рамках предлагаемой концепции, конфликт доступа возникающий, когда методы A и B вычисляют одну и ту же характеристику $x = x_A$ и $x = x_B$, может быть разрешен, например, введением метода C, который получая на входе в качестве параметров x_A и x_B, вычисляет на их основании искомую характеристику x, устраняя тем самым не только конфликт доступа к ней, но и очевидно, имевшую место неоднозначность вычисления упомянутой характеристики. Принятая дисциплина доступа к характеристикам позволяет вызывать параллельно те методы, которые в модельном времени выполняются одновременно.

Наконец, последнее, что следует заметить относительно элементов. Как и всякий метод, каждый элемент имеет входные и возвращаемые параметры. Концепция моделирования предполагает возможность распределенного вычисления элементов, т. е. элемент, вообще говоря, может быть найден разработчиком компоненты на просторах Интернета, поэтому его параметры таковы, какими их сделал его автор, и скорее всего, никак не согласованы с характеристиками компоненты, которые придумал ее разработчик. Поэтому, при описании компоненты обязательно должно быть уделено внимание описанию коммутаций входных параметров элементов с внутренними и внешними характеристиками компоненты и выходных параметров элементов – с ее внутренними характеристиками.

4. События. Содержательно, события – это то, что нельзя пропустить при моделировании динамики системы – точки синхронизации различных ее функциональностей, представляемых процессами. Точки, когда получены такие значения характеристик модели и внешних переменных, на которые обязаны отреагировать некоторые процессы компоненты.

Формально событие – функция значений внутренних и внешних переменных в начале шага моделирования. С точки зрения организации имитационных вычислений, событие – это метод, входными параметрами которого является подмножество внутренних и внешних характеристик компоненты, а выходной параметр один – прогнозируемое время до наступления этого события. Если это прогнозируемое время равно нулю – значит, событие уже наступило. События управляют чередованием элементов в процессе.

Для каждой упорядоченной пары элементов процесса $\{A, B\}$, если между ними возможен переход, то ему обязан соответствовать метод-событие $E_{\{A,B\}}$, прогнозирующий время этого перехода. Возможны также события вида $E_{\{A,A\}}$, прерывающее выполнение элемента A, например, если его еще не нужно заканчивать, но он вычислил характеристики компоненты, которые могут повлечь смену элементов других процессов. Процесс перехода должен быть однозначным. Одновременное наступление событий $E_{\{A,B\}}$ и $E_{\{A,C\}}$ говорит лишь о том, что разработчик модели при ее проектировании упустил из своего рассмотрения этот случай, которому, быть может, должен соответствовать переход $\{A, D\}$.

5. Выполнение компоненты. Компонента – элементарная, но, тем не менее, полноправная модель сложной системы. Поэтому ее можно запустить на выполнение имитационного эксперимента. Конечно, если имеются начальные данные и способ нахождения внешних характеристик компоненты в моменты событий. Алгоритм функционирования компоненты

после ее запуска на счет описан в виде четырех правил в конце параграфа 2.2.6.

2.2.8. Комплексы компонент и комплекс как компонента

Компоненты могут объединяться в комплекс, при этом (необязательно) может оказаться, что некоторые компоненты явно моделируют внешние переменные некоторых других компонент. Для того, чтобы полностью описать комплекс, достаточно указать:

1. Какие компоненты и в каком количестве экземпляров в него входят.
2. Коммутацию компонент внутри комплекса, если она имеет место, т. е., какие внутренние переменные каких компонент являются какими внешними переменными и каких именно компонент комплекса.

При объединении компонент в комплекс, следует иметь в виду, что если мы обычно предполагаем локальную прогнозируемость компонент, что предполагает однозначность их вычислительных процессов, то при объединении в комплекс даже локально-прогнозируемых компонент, однозначность вычислительного процесса может быть потеряна. Так может произойти, если по каким-то причинам, несколько компонент вычисляют одну и ту же в предметной области, характеристику моделируемого явления. В связи с вышесказанным, дадим следующее определение:

Определение 2.6.

Комплекс называется однозначным, если любые две внутренние характеристики любых его различных компонент (в том числе и различных экземпляров одного типа) имеют в предметной области моделирования различающийся смысл.

Лемма 2.3.

Любой комплекс можно привести к однозначности, увеличив число его компонент.

Доказательство.

Например, пусть n компонент A_1, …, A_n, имеют среди своих характеристик такие x_1, …, x_n, которые моделируют одну и ту же величину x в предметной области. Добавим в комплекс новую компоненту A_0, которая будет однозначно и детерминировано моделировать упомянутую величину x своей единственной внутренней переменной x_0, на основании своих внешних переменных x_1, …, x_n (например, путем свертки или случайного выбора), которые она получает от компонент A_1, …, A_n. В полученном комплексе величину x предметной области моделирует единственная характеристика x_0, а характеристики x_1, …, x_n моделируют уже не x, а значения различных внешних переменных компоненты A_0, и поэтому, являются вполне различимыми между собой в предметной области моделирования.

Пользуясь последним утверждением, в дальнейшем мы будем рассматривать только однозначные комплексы.

Однозначный комплекс, состоящий из многих компонент, вовне может проявляться в качестве единой компоненты.

Введем следующую операцию объединения компонент однозначного комплекса:

1. Внутренними переменными комплекса объявляется объединение внутренних переменных всех его компонент.
2. Процессами комплекса объявляется объединение всех процессов его компонент.
3. Методами комплекса объявляется объединение всех методов его компонент.
4. Событиями комплекса объявляется объединение всех событий его компонент.
5. Внешними переменными комплекса объявляется объединение всех внешних переменных его компонент, из

которого исключаются все те переменные, которые моделируются явно какими-либо компонентами.

Операция объединения превращает комплекс в компоненту, причем объединенный комплекс, рассматриваемый как компонента, наследует от своих компонент такое важное свойство, как прогнозируемость на отрезке.

Теорема 2.3.

Если комплекс однозначный, а входящие в него компоненты локально прогнозируемы на отрезке модельного времени $[0,T]$, то и полученная в результате применения к комплексу операции объединения компонента будет локально прогнозируемой на отрезке $[0,T]$.

Доказательство.

Пусть $t \in (0,T]$. Пусть составляющие однозначный комплекс компоненты A_1, …, A_n локально прогнозируемые, стало быть, все они прогнозируемы в точке t. Каждой компоненте A_i поставим в соответствие базовую для t точку τ_t^i (определение 2.4). Пусть $\tau_t^* = \max\limits_{1 \le i \le n} \tau_t^i$. Очевидно, $\tau_t^* < t$, так как для любого $1 \le i \le n$, справедливо $\tau_t^i < t$. Точка τ_t^*, в силу свойства аддитивности имитационных алгоритмов компонент, является базовой для t, для любой из компонент A_1, …, A_n, следовательно, наш комплекс прогнозируем в точке t. Поскольку точка t была произвольной, комплекс прогнозируем в любой точке $t \in (0,T]$, и, следовательно, является локально прогнозируемым на отрезке $[0,T]$. ■

В результате описанной операции объединения и с учетом теоремы 2.3, комплекс становится компонентой, наследующей свойство прогнозируемости от своих составляющих, со всеми вытекающими отсюда последствиями. Например, комплекс как компоненту можно запустить на выполнение, по сформулированным в конце пункта 2.2.6 правилам.

Этот факт позволяет строить модель как фрактальную конструкцию, сложность которой (и соответственно подробность моделирования) ограничивается лишь желанием разработчика.

Заметим, что из способа синтеза компонент в комплекс (коммутация внутренних переменных одних компонент с внешними других) и принятой для каждой компоненты гипотезы об инвариантности, следует, что на уровне различных компонент конфликтов по поводу доступа к характеристикам комплекса быть не может. На уровне отдельной компоненты, как уже упоминалось выше, методы проектируются таким образом, что между ними тоже не может быть конфликтов доступа, в силу требования однозначности имитационных вычислений.

2.3. Подведение итогов

2.3.1. Прогнозируемость и замкнутость модели

На уровне гуманитарного понимания, под возможностью построения имитационной модели некоторого явления на отрезке времени $[0,T]$, по-видимому, обычно понимается нечто близкое к данному выше в пункте 2.2.3 определению 2.3 прогнозируемости модели на отрезке времени.

Отметим, что утверждение о возможности моделирования в этом смысле любого явления, вполне в духе детерминизма Лапласа [47], и поэтому наряду с признанием таких проявлений спонтанности как чудо, акт творчества, проявление свободной воли или просто радиоактивный распад, является скорее предметом религиозно-философских убеждений, нежели научным фактом.

Оказывается, что для детерминизма моделирования в смысле определения 2.3, недостаточно лишь классической замкнутости модели в смысле определения 2.1, – возможности распространить имеющееся в момент t знание о состоянии

системы и окружающего мира на знание о состоянии системы на небольшом последующем интервале времени $[t, t+\Delta t]$. Кроме этого, еще необходима обусловленность каждого текущего состояния системы некоторой его предысторией в смысле определения 2.4 прогнозируемости модели в точке. Без такой обусловленности, эволюция модели разбивается непрогнозируемыми моментами на ряд эпизодов, не вполне связанных между собою причинно-следственной связью.

Между прочим, у Лапласа имеется следующее широко цитируемое высказывание, взятое эпиграфом пункта 2.2.3: «Современные события имеют с событиями предшествующими связь, основанную на очевидном принципе, что никакой предмет не может начать быть без причины, которая его произвела...» [47], которое, по-видимому (Лаплас обусловливает современные события их предысторией), свидетельствует о понимании им важности именно прогнозируемости в смысле определения 2.4, для возможности построения детерминированной модели изучаемого явления.

Требование замкнутости модели в начальной точке и прогнозируемости ее в остальных точках отрезка моделирования гарантирует (теорема 2.2) существование прогноза поведения модели на отрезке в смысле определения 2.3, однако, вовсе не гарантируют, что любые усилия разработчика в области построения модели завершатся именно таким прогнозом. Например, вспомним знаменитую апорию Зенона Элейского про Ахиллеса и черепаху. По существу (в отличие, например, от «пешеходов и мухи»), – это и замкнутая и прогнозируемая во всех точках модель. Однако ее разработчик Зенон так выбрал системные события (моменты, когда Ахиллес добегает до того места, где в начале шага находилась черепаха), что у них оказалась точка накопления – момент, когда Ахиллес поравняется с черепахой. С точки зрения развиваемой здесь теории, так получилось из-за того, что выбранные Зеноном точки синхронизации произвольны и неудачны в том смысле, что в

предметной области модели за ними не стоят какие-либо реальные события, из-за которых стоило бы прерывать моделирование с постоянным шагом. Однако, выбери Зенон иные события – и знаменитой апории не получилось бы. Таким образом, успех разработчика в построении модели будет зависеть от владения им искусством моделирования, так как процесс такого построения в настоящее время не формализован. Наука же моделирования посредством теоремы 2.2 может лишь пообещать разработчику, что его усилия не бессмысленны, так как в принципе такое построение возможно.

2.3.2. Однозначность и параллельные вычисления

Попытка со стороны двух или более параллельно выполняющихся в модельном времени элементов изменить одну и ту же характеристику модели, всегда означает неоднозначность имитационных вычислений указанной характеристики. Поэтому последовательное проведение в жизнь требования однозначности имитационных вычислений на стадии проектирования модели должно обеспечить безопасный параллельный вызов одновременно в модельном времени выполняющихся элементов.

Еще на стадии проектирования модели все ее характеристики должны быть распределены между ее элементами, которые могут выполняться одновременно в модельном времени так, чтобы ни в коем случае никакой элемент не имел права изменять «чужие» характеристики. Каждый элемент имеет право лишь на изменение «своих», выделенных именно ему, характеристик модели. Дисциплина изменения каждым элементом только своих характеристик должна быть столь же жесткой, как например, требование обходиться без оператора goto в структурном программировании. Точно также как и в предыдущем пункте, можно сказать, что осуществление этой дисциплины в значительной мере относится к искусству моделирования. В значительной, а не полной, – потому что напри-

мер, такой формальный способ разрешения неоднозначности, уже упоминавшийся выше, как введение дополнительных характеристик и дополнительных элементов, выбирающих единственную характеристику из серии неоднозначных – работает всегда, не заменяя при этом, конечно, искусство моделирования.

Существование формального способа разрешения неоднозначностей должно убеждать разработчика в том, что требование жесткой дисциплины доступа к характеристикам модели вполне осуществимо. Примерно такую же роль в обосновании жестких требований структурного программирования играла известная теорема Дейкстры, доказывающая принципиальную возможность обходиться в программах без оператора безусловного перехода.

Награда за исполнение жесткой дисциплины доступа к характеристикам модели – возможность параллельного запуска на счет элементов, выполняющихся в модельном времени одновременно.

2.3.3. Декомпозиция модели и организация вычислительного процесса имитации

Организация вычислительного процесса имитационного эксперимента как простейшей, однокомпонентной, так и более сложной модели, также в конце концов приведенной к однокомпонентному виду, происходит согласно четырем правилам, сформулированным в конце пункта 2.2.6.

Так как эти правила одинаковы для любой модели сложной системы, вполне естественно возложить упомянутую организацию вычислительного процесса имитации на специальное программное обеспечение для подобных задач – на инструментальную систему распределенного имитационного моделирования сложных систем, о которой пойдет речь в следующей главе.

Для реализации модели на компьютере, от разработчика потребуется лишь

1. Описать как устроена модель, а именно, для каждой компоненты описать какие процессы в нее входят, из каких элементов они состоят, какие элементы могут переключаться на какие, какие события вызывают эти переключения, наконец, указать коммутацию входящих и выходящих параметров элементов с характеристиками модели. На этапе трансляции таких описаний может быть первичный контроль за однозначностью вычислительного процесса. Затем можно строить иерархию комплексов из имеющихся компонент, рассматривая каждый комплекс как новую компоненту. Все упомянутые описания можно делать на специальном непроцедурном языке описаний компонент и комплексов, как это будет описано в следующей главе, а можно, как это модно делать в последние годы, придумать для этого специальный графический интерфейс.
2. Написать на языке программирования все методы-элементы и методы-события. При этом, сосредоточенные элементы – это достаточно обычные методы, получающие входящие параметры и возвращающие выходящие. Распределенные же, кроме этого, еще всегда получают продолжительность текущего шага моделирования Δt, и должны уметь выдавать свой результат в соответствии с этой продолжительностью.

В следующей главе будет показано, как описанная выше концепция реализована в программном обеспечении макета рабочей станции пиринговой сети имитационного моделирования, а также приведен пример, как на этой рабочей станции реализуется простейшая распределенная имитационная модель – уже упоминавшаяся в этой главе модель «пешеходы и муха».

Глава III. Инструментальная система распределенного имитационного моделирования

Инструментальные средства имитационного моделирования сложных систем относятся к инструментам технологии моделирования. Всякая инструментальная система помогает сделать что-то полезное в своей предметной области лишь в рамках реализованной ею концепции и лишь настолько, насколько пользователь системы овладел этой концепцией.

Настоящая глава посвящена описанию практической реализации концепции распределенного имитационного моделирования сложных систем, изложенной в предыдущей главе. Следует заметить, что в настоящее время реализован лишь макет инструментальной системы, основные цели которого – практическая проверка концепции моделирования и применение в учебном процессе. Некоторые вещи в макете реализованы пока упрощенно, например, взаимодействие рабочих станций в сети: в макете пока не реализовано запланированное применение технологии IARnet [5], [9] для организации распределенных вычислений, хотя хороший задел в этом направлении имеется [16], [30], [31], [52]. Также пока не реализована служба поиска и информационная служба, которые должны быть реализованы на специальных серверах. Тем не менее, реализованы базовые функции сети распределенного моделирования (быть может, с некоторыми упрощениями), и реализованы первые отладочные модели.

3.1. Пиринговая сеть распределенного моделирования в Интернете

Предлагаемая инструментальная система во-первых, помогает построить синтез сложной многокомпонентной модели, что само по себе обычно является нетривиальной задачей, и, во-вторых, позволяет создать в Интернете пиринговую сеть моделирования, в которой с одной стороны, каждый участник

сети может предоставлять во всеобщий доступ (опубликовать) разработанные им методы, а с другой стороны, может использовать в разрабатываемых им моделях подходящие методы, разработанные другими участниками и опубликованные ими в Интернете. При этом от публикуемых методов требуется лишь, чтобы они по своему текущему состоянию (набору внутренних характеристик), набору внешних характеристик (параметров) и интервалу времени моделирования, могли вычислять внутренние переменные в конце этого интервала. Как правило, так или иначе именно этим и занимается большинство компьютерных программ. Публикация метода, помимо описания его назначения и интерфейса на естественном языке, предполагает в дальнейшем возможность его удаленных вызовов. В основе инструментальной системы лежит описанная в предыдущей главе концепция анализа и синтеза сложных систем, ориентированная на параллельное выполнение методов, одновременно протекающих в модельном времени.

Основой сети распределенного моделирования является программное обеспечение пиринговой рабочей станции, оно состоит из клиентской и серверной частей. Клиентская часть ответственна за создание моделей. Она содержит средства, позволяющие по описаниям на специальном формальном непроцедурном языке ЯОКК [36], [37] (языке описания комплексов и компонент) методов, событий, компонент и комплексов строить информационную структуру модели, в первую очередь ее базу данных, а также средства поддержки выполнения модели и наблюдения за результатами моделирования. При этом, методы, обеспечивающие функциональность модели могут быть как локальными, так и удаленными. Серверная часть пирингового клиента предоставляет в первую очередь сервис удаленных вызовов опубликованных на этом хосте методов, а также сервис просмотра каталога и описаний этих методов.

3.1.1. Архитектура модели

Основой формализации модели является понятие **компоненты**. Компонента имеет внутренние и внешние характеристики. Деятельность компоненты осуществляют один или более параллельно протекающих **процессов**. Процесс состоит в чередовании конечного числа заранее известных **элементов** — элементарных с алгоритмической точки зрения методов. Чередование элементов определяется наступлением **событий**. Событие — метод, прогнозирующий интервал времени до наступления соответствующего системного события, заключающегося в смене элементов процесса. Если этот интервал равен нулю — считается что событие наступило, и происходит соответствующая ему смена элементов процесса. Все методы компоненты, т.е. элементы и события, могут либо быть написаны разработчиком модели на одном из допустимых языков программирования, либо же найдены в готовом виде в пиринговой сети моделирования — это те части модели, которые могут быть распределены в сети. Поскольку часть методов компоненты может быть выполнена сторонними разработчиками, после публикации в Интернете своих методов, не контролирующими процесс их включения в те или иные компоненты создаваемых в сети моделей, неестественно считать множества характеристик компоненты и множества получаемых и возвращаемых методом параметров, совпадающими. При этом вполне естественно считать множества входящих и выходящих параметров метода подмножествами множества характеристик компоненты. В связи с последним замечанием, при формальном описании компоненты, возникает необходимость описывать коммутацию характеристик компоненты с входящими и исходящими параметрами ее методов.

Компоненты могут объединяться в **комплекс**, при этом (необязательно) может оказаться, что некоторые компоненты явно моделируют внешние переменные некоторых других компонент. Здесь разрешается одной компоненте моделиро-

вать характеристики, являющиеся внешними переменными многих компонент, но не разрешается неоднозначность, когда одна чья-то внешняя переменная моделируется многими компонентами. Такая неоднозначность, впрочем, может быть легко преодолена, введением новой компоненты, получающей в качестве внешних переменных значения упомянутой характеристики, вычисленные различными компонентами, и моделирующей в качестве своей внутренней переменной уже единственное «окончательное» значение этой характеристики.

Таким образом, при формальном описании комплекса, кроме перечисления входящих в него компонент, возможно, придется указывать их коммутацию (т.е., какие внутренние переменные одних компонент моделируют какие внешние переменные других). Определенный таким образом комплекс, после описанной в операции объединения его компонент с исключением из этого объединения явно моделируемых внешних переменных, сам становится компонентой, т.е., начинает удовлетворять формальному определению таковой. Этот факт позволяет строить модель как фрактальную конструкцию, сложность которой (и соответственно подробность моделирования) ограничивается лишь желанием разработчика.

Остановимся на архитектуре распределенной модели. Одну и ту же концепцию моделирования, в том числе и описанную в предыдущей главе, можно воплотить на практике разными способами. В настоящее время архитектура программного обеспечения рабочей станции такова, что вся модель от начала и до конца всегда собирается на одной рабочей станции. При этом, вообще говоря, ни один из методов модели, будь то элемент или же событие, может не быть реализован на этой рабочей станции. Любые методы модели могут быть «чужими» и выполняться удаленно на других компьютерах сети распределенного моделирования. В этом контексте слова о том, что вся модель находится на рабочей станции, означают, что на ней по исходным описаниям создается база данных

модели, которая затем заполняется начальными данными, а впоследствии содержит все характеристики модели, как внутренние так и внешние, на всех отработавших шагах моделирования. Распределено вызываются только методы, которым в момент вызова передаются их входные параметры, а после вызова принимаются их возвращаемые параметры.

Заметим, что такая архитектура модели существенно отличается, например, от спецификации HLA [8], где распределено выполняются, вообще говоря, вполне равноправные федераты. Остановимся на том, почему выбрана именно такая архитектура. Основой для нее послужил следующий принцип системной интеграции: «если почти до нуля минимизировать требования системного интегратора к остальным участникам проекта – появится небольшой шанс, что они в конце концов, быть может, выполнят эти требования».

Гораздо проще подготовить к опубликованию в сети метод – обычную процедуру с входящими и возвращаемыми параметрами, чем, например, компоненту. В последнем случае, например, пришлось бы достаточно четко представить себе, что такое эта самая компонента. Кроме того, это резко утяжелило бы соответствующий сервис – пришлось бы заботиться о поддержании баз данных всех экземпляров удаленно вызванных компонент, решать проблемы управления временем - делать все то, что так утяжелило HLA. В принципе, конечно, это возможно реализовать при дальнейшем развитии проекта, однако у автора есть осознание трудностей, но нет полной уверенности в необходимости и даже полезности подобной реализации.

В настоящее время, если у разработчика модели возникает желание использовать не только какой-либо метод, а всю «чужую» компоненту целиком – он должен импортировать описатель этой компоненты и откомпилировать его на своей рабочей станции. После этого можно включать ее в свои модели – на рабочей станции будут построены базы данных для ее

экземпляров, а все методы будут вызываться удаленно. Если возникает желание использовать чей-то комплекс – необходимо импортировать его описатель и описатели всех его компонент, откомпилировать их и в дальнейшем работать с комплексом как с компонентой.

Для формального описания компонент и комплексов модели, служит специальный непроцедурный язык, описываемый далее.

3.1.2. Язык описания комплексов и компонент (ЯОКК)

В данном разделе разобран специализированный непроцедурный язык, на котором должны составляться описания классов компонент и комплексов модели. По поводу языка описаний стоит сделать несколько замечаний.

Во-первых, почему язык. Если бы автор создавал коммерческий продукт, пожалуй вместо языка уместнее было бы разработать графический интерфейс пользователя (GUI), где мышкой на рабочее поле вытаскивались бы нужные сущности в виде соответствующих пиктограмм, и мышкой же проводились бы необходимые связи между ними. Однако основной целью автора была иллюстрация работоспособности предлагаемой в первой главе концепции моделирования, притом с сильной ориентацией на учебный процесс, а этим целям, на взгляд автора, лучше соответствует именно язык. В концепции моделирования существуют определенные понятия, например, комплекс, компонента – в языке описаний им соответствуют одноименные описатели. В концепции моделирования имеются понятия коммутации компонент в комплексе или коммутации элементов в компоненте – в языке ЯОКК им соответствуют описатели коммутации и т. д. Отметим, что все сказанное выше, вовсе не исключает создание в будущем надстройки над системой, которая переводила бы графические манипуляции с пиктограммами в соответствующие конструкции ЯОКК, –

можно считать, что это задача следующего этапа реализации системы.

Во-вторых, почему именно этот язык. Потребность в непроцедурных языках, на которых описывается не что нужно выполнить, а то, кто как устроен, и как и с кем связан внутри и вне компоненты, возникла достаточно давно. Ниже перечислены несколько языков подобной направленности, с указанием их разработчиков, и иногда систем, где они применялись:

- SQL (IBM, ANSI) — 1986г.
- Язык MISS (ВЦ АН СССР) — 1986-1990гг.
- Язык IDL (CORBA, OMG) — 1991г.
- Язык Slice (ICE, ZeroC) — 2003г.
- XML (W3C, SOAP, Microsoft) — 1996-2005гг.
- Язык UML (OMG, UML Partners) — 1997-2005гг.

По функциональности выразительных средств, для наших целей подошли бы кроме второго, пожалуй два последних. Однако из них UML, несмотря на определенную популярность в среде разработчиков моделей, слишком избыточен для заявленных в данной работе целей, а потому был бы неоправданно сложен в реализации. Язык XML достаточно гибок, чтобы можно было организовать компиляцию необходимого для данного проекта его подмножества. Однако, в силу специфики его синтаксиса, описания на нем, на взгляд автора, были бы недостаточно наглядны для целей обучения. Поэтому была выбрана модификация проверенного, хотя и не получившего широкого распространения решения, реализованного ранее автором совместно с В.Ю. Лебедевым [33], – измененный в сторону упрощения (так как с тех пор заметно упростилась концепция моделирования) язык MISS, который в данной системе моделирования получил и новое название – язык описания комплексов и компонент (ЯОКК).

Отметим, что хотя аккуратнее употреблять термин «описание класса компонент», мы ради краткости будем пользоваться словосочетанием «описание компоненты». Лингвисти-

ческие формулы записываются ниже в стандартной нотации, согласно которой служебные слова выделяются жирным шрифтом, разделители – кавычками, а необязательные включения квадратными или фигурными скобками, причем фигурные скобки указывают на возможность повторения.

3.1.2.1. Общий синтаксис ЯОКК

Собирая модель, инструментальная система автоматически генерирует ее базу данных, руководствуясь при этом содержимым специальных описателей ЯОКК. Таких описателей бывает четыре типа:

1. Описатель типа данных.
2. Описатель комплекса.
3. Описатель метода.
4. Описатель компоненты.

Все описатели, кроме описателя типа данных состоят из заголовка и нескольких параграфов. Описатель типа данных представляет из себя единственный параграф.

Параграфы начинаются заголовком, после которого через точку с запятой идут операторы. Если параграф не последний – он заканчивается началом заголовка следующего параграфа, если последний – ключевым словом **END;** которое заканчивает и весь описатель.

Порядок параграфов в описателе свободный, если между ними нет зависимости. Если такая зависимость есть – зависящие параграфы должны появляться в тексте описателя позже тех, от которых зависят.

В описателях допустимы комментарии C++ подобного синтаксиса, т.е. игнорируются строки начинающиеся с «//», и любой текст, находящийся между символами «/*» и «*/».

Перейдем к перечисленным выше описателям.

3.1.2.2. Описатель типа данных

Заголовок описателя типа данных начинается ключевым словом **TYPE**, за которым следует идентификатор – имя типа.

Заканчивается заголовок – «**;**» – точкой с запятой. После заголовка, разделяемые символами «**;**» – точкой с запятой идут операторы описания данных. Синтаксис оператора описания данных следующий:

<имя типа> <описание данных>{«**,**»<описание данных >}«**;**»

Имя типа отделяется пробелом от одного или нескольких разделенных запятой описаний данных, заканчивается оператор точкой с запятой. Здесь имя типа – либо одно из встроенных имен типов, приведенных в таблице ниже, либо имя типа, чей описатель типа данных расположен выше данного в охватывающем описателе, либо это каталогизированный тип данных. Каталогизированным тип данных становится после успешной компиляции его описателя.

Встроенные типы определяются следующей таблицей:

Тип в инструментальной системе	Тип в языке C#	Тип в .Net	Длина в байтах
int	int	System.Int32	4
double	double	System.Double	8
long	long	System.Int64	8
char	char	System.Char	2
byte	byte	System.Byte	1
bool	bool	System.Boolean	1
uint	uint	System.UInt32	4
short	short	System.Int16	2
ulong	ulong	System.UInt64	8
address	void*	System.Void*	4[1]
decimal	decimal	System.Decimal	16
float	float	System.Single	4

[1] Для 32-разрядных систем. Для 64-разрядных – 8.

sbyte	sbyte	System.SByte	1
ushort	ushort	System.UInt16	2

Описание данных – это либо идентификатор, либо массив. Массив – это идентификатор, в квадратных скобках справа от которого, положительными целыми числами через запятую указаны размерности массива.

Пример описателя типа данных:

```
TYPE test;
    double X, Y[2,3,4], Z;
    bool a, b[4];
    int i;
END;
```

Тогда если где-то описано поле vrbl:

test vrbl;

то имеет смысл поле vrbl.X[0,1,3] типа **double**, а также булевские поля vrbl.a и vrbl.b[2].

3.1.2.3. Описатель комплекса

Описатель комплекса открывается заголовком, который состоит из ключевого слова **COMPLEX**,

За заголовком идут параграфы:

1. Параграф компонент.
2. [Параграф коммутации компонент.]

Обязательный параграф компонент открывается ключевым словом **COMPONENTS**, после которого через запятую идут имена компонент за которыми в круглых скобках стоит целое число – количество экземпляров компоненты, определяемого ее именем типа, входящих в комплекс. Если число экземпляров – 1, описатель количества экземпляров (1) разрешается опустить. Заканчивается параграф символом «**;**» – точкой с запятой. Для успешной компиляции комплекса

необходимо, чтобы описатели входящих в него компонент были откомпилированы ранее.

Необязательный параграф коммутации компонент описывает, какие внутренние переменные каких компонент вычисляют какие внешние переменные каких компонент. Параграф открывается ключевым словом **COMMUTATION**, за которым следуют операторы коммутации. Операторы коммутации имеют следующий вид:

<имя компоненты>«(»<экземпляр компоненты>«)**.**»
<поле параметра компоненты> «**=**»
<имя компоненты>«(»<экземпляр компоненты >«)**.**»
<поле внутренней переменной компоненты>«**;**»

До нужного элемента данных, возможно придется добираться через несколько квалификаторов, поэтому поля параметра компоненты и поля внутренней переменой компоненты в операторах коммутации могут иметь вид:

{<идентификатор>**.**}<поле>

где поле – либо идентификатор, либо элемент массива.

Результат компиляции комплекса – текстовый описатель этого комплекса, как компоненты.

3.1.2.4. Описатель метода

Методы – это или элементы процессов, или же методы вычисляющие наступление событий. Это те части модели, которые могут вызываться как локально, так и распределено. Считается, что методу передается некий набор параметров, тип которого должен быть описан, и некий набор параметров возвращается методом. Так как тип этих наборов параметров, в особенности для удаленных методов, определяется разработчиком метода, а не разработчиком модели, возникает вопрос о коммутации полей параметров метода с полями фазовых переменных и/или параметров компоненты. Кроме того, методы различаются на события и методы реализующие элементы

процессов, а последние, еще и по отношению с модельному времени на:

- сосредоточенные – происходящие мгновенно в модельном времени,
- распределенные – занимающие не менее одного модельного такта и дающие определенный результат своего выполнения в виде изменений внутренних переменных модели в конце каждого такта.

Описатель метода начинается заголовком – ключевым словом **METHOD** за которым следует идентификатор – имя метода, за которым следуют необязательные [«**:**» <тип метода>], наконец, завершает заголовок «**;**» – точка с запятой.

Тип метода – одно из ключевых слов **FAST**, **SLOW** или **EVENT,** соответствующее сосредоточенным, или распределенным элементам, или же событиям. По умолчанию тип метода, реализующего элемент процесса, считается **SLOW,** и в этом случае явное его указание разрешается опустить.

Далее следует необязательный параграф адреса метода. Он имеет вид:

ADDRESS «**:**» <адрес хоста> «**;**»

Адрес хоста, это IP-адрес, или DNS-адрес, или же ключевое слово **local**. По умолчанию адрес метода – **local**, в этом случае параграф адреса можно опустить.

Далее следуют параграфы описаний входящих и возвращаемых параметров метода. По синтаксису они отличаются от описателя типа лишь другими ключевыми словам заголовка – **INPUT** и **OUTPUT** соответственно, и тем, что имя типа после ключевых слов необязательно. Если, тем не менее, оно присутствует – каталогизированный тип данных с таким именем появится в базе данных рабочей станции в результате успешной компиляции описателя метода.

Если возвращаемые параметры полностью совпадают с входными – параграф возвращаемых параметров можно опустить. У событий также нет параграфа возвращаемых парамет-

ров, так как известно, что они всегда возвращают значение типа **double** – прогнозируемое время до наступления соответствующего события.

3.1.2.5. Описатель компоненты

Описатель компоненты начинается заголовком – ключевым словом **COMPONENT,** за которым следует идентификатор – имя компоненты, за которым следует «**;**» – точка с запятой, завершающая заголовок.

За заголовком следуют параграфы описателя:

1. [Параграфы типов.]
2. Параграф внутренних переменных.
3. [Параграф параметров.]
4. Параграф методов.
5. Параграф коммутации.
6. [Параграф переключателей.]

Порядок параграфов в описателе несущественен, кроме параграфов типов, которым естественно быть в начале, так как всякий тип должен быть описан до его использования. Каждый параграф начинается соответствующим ему ключевым словом и заканчивается ключевым словом следующего параграфа. Последний параграф заканчивается ключевым словом **END** и «**;**» – точкой с запятой.

Параграфы типов необязательны, они нужны лишь если определенные в них типы используются в параграфах внутренних переменных и параметров. Синтаксис их был описан выше в описателе типов.

Синтаксис параграфов внутренних переменных и параметров такой же как и параграфов типов – разница лишь в ключевых словах параграфов: **PHASE** и **PARAMETERS** соответственно. Кроме того, имя типа после ключевого слова необязательно. Если оно присутствует – тип данных с таким именем появится в базе данных рабочей станции в результате успешной компиляции описателя. Если нет, то чтобы иденти-

фицировать поля характеристик компоненты достаточно либо имени компоненты, либо используются специальные ключевые слова **phase** и **param** (см. далее).

Разбиение характеристик компоненты на внутренние и внешние в описателях компонент сделано исключительно ради более строгого воплощения концепции моделирования. Компиляция переводит описания внутренних и внешних характеристик в единую базу данных характеристик.

Параграф методов открывается ключевым словом **METHODS**. В этом параграфе через «**;**» – точку с запятой перечислены процессы компоненты в смысле пункта 2.2.5. Каждый процесс в этом перечислении – это перечисленные через запятую имена образующих его элементов. Порядок элементов в этом перечислении произволен, за исключением того, что первым в списке идет так называемый «корневой» элемент, который будет считаться текущим при первом запуске модели на выполнение. Имена элементов должны быть уникальными в пределах описания: нельзя одинаково назвать два разнотипных элементов одной компоненты, но использование одного имени элемента в описаниях разных компонент не возбраняется. Приведем пример параграфа методов:

METHODS
Man_0_move;
Man_1_move;
Fly_0_move, Fly_0_Uturn;
END;

Параграф коммутации начинается ключевым словом **COMMUTATION**. За ключевым словом следуют операторы коммутации, они бывают двух видов:

1. Коммутация входящих параметров методов, ее синтаксис:
   <имя метода>«**.**»<имя входящего параметра> «**=**»
   [**phase** | **param** «**.**»]<имя поля>«**;**»
   Входящий параметр метода связывается с внутренней

(ключевое слово **phase**) или внешней (ключевое слово **param**) переменной компоненты. Если в описателе компоненты нет параграфа параметров, квалификатор «**phase.**» можно опустить, оставив в операторе коммутации лишь имя поля.

2. <имя поля внутренней переменной> «**=**»
   <имя метода>«**.**»<имя возвращаемого параметра>«**;**»
   Возвращаемый параметр метода связывается с полем внутренней переменной компоненты.

До нужного элемента данных, возможно придется добираться через несколько квалификаторов, поэтому все имена в операторах коммутации вполне могут иметь вид:

{<идентификатор>**.**}<поле>

где поле – либо идентификатор, либо элемент массива.

Для успешной компиляции параграфа коммутации необходимо, чтобы описатели методов, присутствующих в операторах коммутации, были откомпилированы до этого. При компиляции операторов коммутации проверяется соответствие типов коммутируемых полей.

Параграф переключателей начинается ключевым словом **SWITCHES**. За ним следуют операторы переключений, которые имеют вид:

<имя текущего элемента> «**:**»

[<имя события>«**,**»] <имя следующего элемента >«**;**»

Если переход безусловный, т. е., происходит всегда, что достаточно типично для быстрых элементов, – имя события не указывается.

3.2. Макет рабочей станции сети распределенного моделирования

Основой сети распределенного моделирования является программное обеспечение пиринговой рабочей станции сети

распределенного моделирования. Оно состоит из клиентской и серверной частей.

Серверная часть рабочей станции обеспечивает сервис удаленных вызовов опубликованных на этом хосте методов и ведение журнала таких вызовов, где запоминается имя метода, время вызова и IP-адрес вызвавшей метод рабочей станции.

Клиентская часть ответственна за создание и хранение моделей, и проведение с ними имитационных экспериментов. Функционально она состоит из трех частей:

1. Средства работы с описателями ЯОКК.
2. База данных и средства работы с ней.
3. Средства поддержки выполнения моделей.

Дадим краткие описания этих составляющих клиентской части рабочей станции.

Средства работы с описателями ЯОКК – это текстовый редактор с интегрированным компилятором. В редакторе можно набрать текст описателя, или же импортировать его из текстового файла и редактировать, а затем запустить его компиляцию.

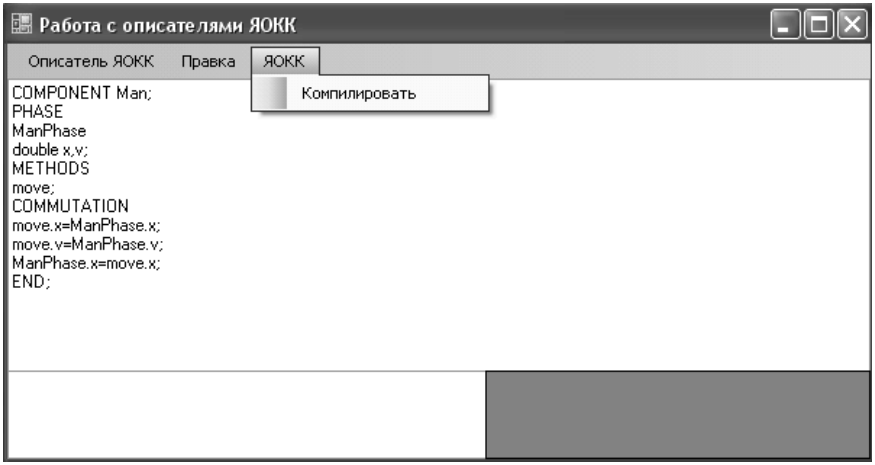

Рис.8. Работа с описателями ЯОКК.

Результатом компиляции может быть, например, указание на синтаксические ошибки в описателе. Если же синтаксических ошибок в описателе нет – выдается диагностика об успешной компиляции, и в базе данных рабочей станции добавляется ряд записей, соответствующих появлению в распоряжении разработчика нового класса, который описывался откомпилированным описателем.

Описатель комплекса компилируется в описатель комплекса как компоненты, при этом если составляющие его компоненты не известны системе (еще не были откомпилированы), то компиляция завершится аварийно.

Также проверяется соответствие типов при коммутации, если такого соответствия нет – выдается соответствующее сообщение об ошибке компиляции.

Для откомпилированного класса компонент можно построить экземпляр модели, для последующего его запуска на выполнение. Экземпляр компоненты это база данных, состоящая из 7 таблиц:

1. Первая из таблиц – собственно и есть база данных модели, здесь хранятся по порядку ссылки на внутренние и внешние характеристики модели. Каждая запись таблицы соответствует точке синхронизации модели. Самую первую из записей (и единственную до выполнения компоненты) необходимо заполнить до запуска модели на выполнение – это процесс идентификации модели. Именно эта таблица создается для экземпляра компоненты по описаниям типов данных класса. Остальные шесть таблиц достаются экземпляру готовыми, от откомпилированного описателя класса компоненты.
2. Таблица реализаций методов, содержит 4 поля. Первое поле «НомерРеализации» – числовой ключ. Второе поле текстовое «Метод» – имя метода. Третье поле текстовое «Сборка» – имя сборки, содержащей метод. Наконец, четвертое текстовое поле «Адрес» – IP или DNS

адрес хоста, где находится указанная сборка, или слово «local», если это собственная сборка, находящаяся на самой рабочей станции.

3. Таблица процессов компоненты, содержит 2 поля. Первое – ключ «НомерПроцесса», второе текстовое – «Процесс», содержит имя процесса, полученное в результате компиляции соответствующего описателя.
4. Таблица методов компоненты, содержит 6 полей. Первое – ключ «НомерЭлемента». Второе – текстовое «Элемент», содержит имя элемента, полученное из описателя компоненты. Третье числовое «Тип», может содержать 3 значения: 1 – распределенный элемент; 2 – сосредоточенный элемент; 3 – событие. Четвертое поле булевское «Текущий» имеет значение true, если в данный момент элемент является текущим (выполняемым) элементом своего процесса. В процессе идентификации модели необходимо каждому из процессов назначить единственный текущий элемент. Пятое поле – числовое «Процесс» – это номер соответствующего ключа из таблицы процессов. Наконец, шестое числовое поле «Реализация» – номер соответствующего ключа из таблицы реализаций.
5. Следующая таблица – таблица переходов между элементами, в ней 5 полей. Первое – ключ «НомерПерехода». Второе числовое «Процесс» – номер ключевого поля в таблице процессов. Третье, четвертое и пятое поля числовые «ТекущийЭлемент», «СледующийЭлемент» и «Событие» – все это номера ключевого поля в таблице методов. Если переход безусловный (такое может быть у сосредоточенных элементов), в поле «Событие» должен стоять 0 (ключи таблицы методов начинаются с 1).
6. Далее следует таблица коммутации входящих параметров методов. Считается, что соответствие типов при

коммутации уже было проверено на этапе компиляции компоненты. Таблица содержит 4 поля. Первое, как всегда ключ «НомерСвязи». Второе числовое «Элемент» – номер ключевого поля в таблице методов. Третье поле числовое «НомерВхода» – порядковый номер входного параметра в списке входных параметров метода. Наконец четвертое – тоже числовое «НомерФазы» – порядковый номер соответствующей характеристики в базе данных характеристик компоненты (в первой из таблиц).

7. Наконец, последняя таблица – коммутации возвращаемых параметров методов. Она содержит 4 поля и очень похожа на предыдущую. Первое поле – ключ. Второе – «Элемент» – номер ключевого поля в таблице методов. Третье числовое «НомерВыхода» – порядковый номер возвращаемого параметра. Наконец, четвертое, числовое «НомерФазы» – номер соответствующей характеристики в первой таблице. У событий возвращаемый параметр – время до его наступления, которое всегда обрабатывается автоматически поэтому последние два поля у событий всегда 1 и 0.

В настоящее время, в период отладки ПО рабочей станции, для облегчения процесса отладки пользователю открыты все семь таблиц. В дальнейшем непосредственный доступ к первой из них будет заменен специальной утилитой работы с базой данных – второй из перечисленных выше основных компонент клиентской части ПО рабочей станции, способной обрабатывать поля данных произвольных типов в соответствии с их описаниями на ЯОКК. В настоящее время такая утилита в полной мере реализована лишь в первой версии системы [31]. Доступ к шести остальным таблицам будет открыт лишь для пользователей, обладающих специальными полномочиями.

Имея в распоряжении перечисленные выше семь таблиц, средство поддержки выполнения компонент – третья из основ-

ных составляющих клиентской части ПО рабочей станции – организует вычислительный процесс выполнения компоненты, вычисляя события, составляя списки элементов на выполнение и выполняя элементы в соответствии с этими списками. Списки на выполнение проверяются по таблице коммутации возвращаемых параметров на то, чтобы два элемента не возвращали одну и ту же характеристику. Если все в порядке – элементы списка выполняются параллельно, если нет – последовательно, последнее, однако, является признаком плохо спроектированной компоненты, как об этом неоднократно упоминалось выше.

ные компоненты Запуск

RecordID	FlyPhase_0_x	FlyPhase_0_v	ManPhase_0_x	ManPhase_0_v	ManPhase_1_x	ManPhase_1_v	DT
1	0	3	0	1	50	-1,5	1
2	3	3	1	1	48,5	-1,5	1
3	6	3	2	1	47	-1,5	1
4	9	3	3	1	45,5	-1,5	1
5	12	3	4	1	44	-1,5	1
6	15	3	5	1	42,5	-1,5	1
7	18	3	6	1	41	-1,5	1
8	21	3	7	1	39,5	-1,5	1
9	24	3	8	1	38	-1,5	1
10	27	3	9	1	36,5	-1,5	1
11	30	3	10	1	35	-1,5	1
12	33	3	11	1	33,5	-1,5	1
13	33,33333333333...	3	11,11111111111...	1	33,33333333333...	-1,5	0,11
14	33,33333333333...	-3	11,11111111111...	1	33,33333333333...	-1,5	0
15	30,33333333333...	-3	12,11111111111...	1	31,83333333333...	-1,5	1
16	27,33333333333...	-3	13,11111111111...	1	30,33333333333...	-1,5	1
17	24,33333333333...	-3	14,11111111111...	1	28,83333333333...	-1,5	1
18	21,33333333333...	-3	15,11111111111...	1	27,33333333333...	-1,5	1
19	18,33333333333...	-3	16,11111111111...	1	25,83333333333...	-1,5	1
20	16,66666666666...	-3	16,66666666666...	1	25	-1,5	0,55
21	16,66666666666...	3	16,66666666666...	1	25	-1,5	0
22	19,66666666666...	3	17,66666666666...	1	23,5	-1,5	1
23	22,22222222222...	3	18,51851851851...	1	22,22222222222...	-1,5	0,85
24	22,22222222222...	-3	18,51851851851...	1	22,22222222222...	-1,5	0

Рис.9. Изменение характеристик модели во время выполнения.

Кроме описаний на ЯОКК компонент, комплексов, типов данных, элементов и событий, разработчик модели должен

написать программы, реализующие все элементы и события модели (тогда в соответствующем столбце таблицы реализаций будет стоять слово «local»), или же найти их реализации в Интернете (тогда в соответствующем столбце таблицы реализаций будет стоять IP или DNS адреса хостов, опубликовавших эти методы).

Кроме элементов и событий, разработчик модели может (но не обязан) запрограммировать так называемый «сервисный модуль» – сборку, которая содержит конструктор модели – метод с зарезервированным именем «Prepare», метод-деструктор с зарезервированным именем «Finish», метод, вызываемый после такта моделирования ненулевой длительности – «AfterSlow», после такта нулевой длительности – «AfterFast». Метод-конструктор может быть полезен, например, для первоначального заполнения базы данных (идентификации модели), методы «AfterSlow» и «AfterFast» – для визуализации процесса моделирования. Параметры этих методов – массив объектов (object[]), в котором передаются все характеристики модели.

Далее, на примере неоднократно упоминавшейся в этой книге модели «Пешеходы и муха», приводится пример реализации распределенной модели в описываемой инструментальной системе моделирования.

3.3. Пример реализации распределенной модели «Пешеходы и муха»

Модель «Пешеходы и муха» – неоднократно упоминавшаяся в этой книге простейшая непрогнозируемая модель.

Два пешехода идут навстречу друг другу с постоянной скоростью. Между ними летает муха с постоянной по абсолютной величине скоростью, большей чем скорость любого из пешеходов. Как только муха долетает до одного из пешеходов, она тут же разворачивается и летит к другому.

С нашей точки зрения модель интересна тем, что при крайней простоте алгоритмов компонент, она предъявляет достаточно серьезные требования к организации вычислительного процесса, например, приходится моделировать с переменным шагом модельного времени. Кроме того, эта задача интересна тем, что непрогнозируема в момент встречи пешеходов: скорость мухи разрывна в любой окрестности точки встречи и, следовательно, не имеет в этой точке предела.

Модель реализована как тестовый пример, необходимый для отладки программного обеспечения рабочей станции распределенного моделирования, а также как учебный пример, иллюстрирующий правила создания моделей и их компонент в предлагаемой среде распределенной имитации.

Следует заметить, что реализация столь простых моделей средствами инструментальной системы распределенного моделирования, конечно же есть пальба из пушки по воробьям. Как можно видеть из приведенных ниже листингов – гораздо проще и быстрее было бы просто сесть и написать это все на том же C# или C++ от начала и до конца. Однако на более сложных моделях, как например, моделирование СОИ, применение инструментальных средств конечно же оправдывается с лихвой. Каждый из коллективов разработчиков, создающих свою компоненту, может полностью сосредоточиться на ее содержательной части, и не задумываться о проблемах интеграции компонент в моделируемый комплекс.

3.3.1. Неформальное описание модели

Модель представляет из себя комплекс, состоящий из двух экземпляров компоненты «пешеход» и одного экземпляра компоненты «муха». Компонента «пешеход» устроена совсем просто: она реализует единственный процесс, состоящий из единственного элемента-метода «движение». «Движение» – распределенный элемент, по текущим значениям внутренней переменной X – координаты пешехода и внешней переменной

V – его скорости, а также по величине текущего шага модельного времени DT, этот метод вычисляет значение внутренней переменной X в конце шага модельного времени по формуле $X(t+DT)=X(t)+V*DT$. Никаких других методов, а следовательно переходов и связанных с ними событий, единственный процесс компоненты «пешеход» не имеет.

Компонента «муха» устроена сложнее. Она также участвует в единственном процессе, но этот процесс состоит из двух элементов:

- «Движение» – тот же самый алгоритм, что и у пешехода, поэтому может быть реализован тем же самым методом.
- «Разворот» – у компоненты «муха» скорость является внутренней переменной, а не параметром, как у «пешехода». «Разворот» – мгновенный метод, он не занимает модельного времени, а действие его состоит в том, что скорость мухи меняет знак: из V становится $-V$.

Элемент «разворот» всегда переходит в элемент «движение». Элемент «движение» переходит в элемент «разворот» по наступлению события «долет до пешехода». Таким образом, с компонентой «муха» связано событие «долет до пешехода». Алгоритм его вычисления – наиболее сложный в данной модели. На входе он получает координаты и скорости всех компонент модели, на выходе же дает время до ближайшей встречи с пешеходом. Для этого сперва определяется, к какому из пешеходов летит муха (знаки скоростей у мухи и пешехода должны быть различны), затем расстояние между ними делится на сумму абсолютных величин скоростей. Если расстояние нулевое – муха уже долетела, и, соответственно, событие уже наступило.

Предметом распределения вычислений в этой модели являются методы и событие. Каждый из них может находиться на отдельном компьютере в сети, при наличии соответствующего сервиса, обеспечиваемого рабочей станцией распределенного моделирования. (Например, они предоставляются

станцией, расположенной на хосте simul.ccas.ru). Модель спроектирована так, что все методы и события одного шага моделирования вызываются асинхронно.

3.3.2. Описание модели на языке ЯОКК

Описание комплекса

```
COMPLEX menANDfly;
COMPONENTS
Man(2), Fly(1);
COMMUTATION
Fly(0).man0Phase.x=Man(0).x;
Fly(0).man0Phase.v=Man(0).v;
Fly(0).man1Phase.x=Man(1).x;
Fly(0).man1Phase.v=Man(1).v;
END;
```

Описание компоненты "пешеход"

```
COMPONENT Man;
PHASE ManPhase;
double x,v;
METHODS
move;
COMMUTATION
move.x=ManPhase.x;
move.v=ManPhase.v;
ManPhase.x=move.x;
END;
```

Описание компоненты "муха"

```
COMPONENT Fly;
PHASE FlyPhase;
double x,v;
PARAMETERS FlyParam;
FlyPhase man0Phase, man1Phase;
METHODS
```

move, Uturn;
SWITCHES
move: reaching, Uturn;
Uturn: move;
COMMUTATION
move.x=FlyPhase.x;
move.v=FlyPhase.v;
move.dt=MODEL.dt;
FlyPhase.x=move.x;
Uturn.v=FlyPhase.v;
FlyPhase.v=Uturn.v;
reaching.x=FlyPhase.x;
reaching.v=FlyPhase.v;
reaching.m0x=FlyParam.man0Phase.x;
reaching.m0v=FlyParam.man0Phase.v;
reaching.m1x=FlyParam.man1Phase.x;
reaching.m1v=FlyParam.man1Phase.v;
END;

Описание метода "движение"

METHOD move;
// по умолчанию подразумевается **SLOW**
ADDRESS : 192.168.137.1;
INPUT
double x, v;
// всем медленным по умолчанию всегда передается dt
// после объявленных, и всем всегда t - в самом конце
OUTPUT
double x;
END;

Описание метода "разворот"

METHOD Uturn : **FAST**;
// по умолчанию **ADDRESS** : **local**;
INPUT

double v;
/****** Поскольку про **OUTPUT** ничего не сказано – он такой же, т.е.
OUTPUT
double v;
**********/
END;

Описание события "долет до пешехода"

METHOD reaching : **EVENT**;
ADDRESS : simul.ccas.ru;
INPUT
double x, v, m0x, m0v, m1x, m1v;
// У всех событий выход всегда - **double** dt; - прогноз его
// наступления
END;

Генерируемое автоматически описание комплекса "пешеходы и муха" как компоненты

COMPONENT menANDflyAScomp;
PHASE menANDflyPhase;
double FlyPhase_0_x, FlyPhase_0_v, ManPhase_0_x, ManPhase_0_v, ManPhase_1_x, ManPhase_1_v;
METHODS
Man_0_move;
Man_1_move;
Fly_0_move, Fly_0_Uturn;
SWITCHES
Fly_0_move: Fly_0_reaching, Fly_0_Uturn;
Fly_0_Uturn: Fly_0_move;
COMMUTATION
Man_0_move.x=ManPhase_0_x;
Man_0_move.v=ManPhase_0_v;
ManPhase_0_x=Man_0_move.x;
Man_1_move.x=ManPhase_1_x;

```
Man_1_move.v=ManPhase_1_v;
ManPhase_1_x=Man_1_move.x;
Fly_0_move.x=FlyPhase_0_x;
Fly_0_move.v=FlyPhase_0_v;
FlyPhase_0_x=Fly_0_move.x;
Fly_0_Uturn.v=FlyPhase_0_v;
FlyPhase_0_v=Fly_0_Uturn.v;
Fly_0_reaching.x=FlyPhase_0_x;
Fly_0_reaching.v=FlyPhase_0_v;
Fly_0_reaching.m0x=ManPhase_0_x;
Fly_0_reaching.m0v=ManPhase_0_v;
Fly_0_reaching.m1x=ManPhase_1_x;
Fly_0_reaching.m1v=ManPhase_1_v;
END;
```

3.3.3. Реализация методов модели на языке C#

Абстрактные классы "пешехода" и "мухи"

Вообще говоря, подобные абстрактные классы необязательны. Здесь они играют роль заголовков С или модулей определений Модулы-2. Их можно передавать на удаленные станции с целью прояснения интерфейсов методов (которые вообще говоря, должны быть описаны на языке ЯОКК).

Описание интерфейса "пешехода"

```
using System;
using System.Collections.Generic;
using System.IO;
namespace DefMan
{
public abstract class Man
{
public struct ManPhase
{
public double x;
```

```
public double v;
}
public abstract void move(MemoryStream inStream,
MemoryStream outStream);
}
}
```

Описание интерфейса "мухи"

```
using System.Text;
using System.IO;
namespace DefFly
{
public abstract class Fly
{
public struct FlyPhase
{
public double x;
public double v;
}
public struct FlyParam
{
public FlyPhase man0Phase;
public FlyPhase man1Phase;
}
public abstract void reaching(MemoryStream inStream,
MemoryStream outStream);
public abstract void Uturn(MemoryStream inStream,
MemoryStream outStream);
// Обратим внимание, что метода "движение" здесь нет.
//Сразу планируем воспользоваться
// соответствующим методом компоненты "пешеход".
}
}
```

Реализация метода "пешехода"

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
namespace DefMan
{
public class ImpMan : Man
{
public ImpMan()
{
}
public override void move(MemoryStream inStream,
MemoryStream outStream)
{
ManPhase phase = new ManPhase();
double dt;
BinaryReader inp = new BinaryReader(inStream);
phase.x = inp.ReadDouble();
phase.v = inp.ReadDouble();
// Прочли фазу компоненты из входного потока
dt = inp.ReadDouble();
// для распределенного элемента за фазой и параметрами
//еще интервал времени
inp.Close();
inStream.Flush();
inStream.Close();
BinaryWriter outp = new BinaryWriter(outStream);
// Реализация основного алгоритма движения
phase.x += phase.v * dt;
outp.Write(phase.x);
outp.Close();
outStream.Flush();
```

```
// Возвращаем только координату
}
}
}
```

Реализация метода и события "мухи"

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
namespace DefFly
{
public class ImpFly: Fly
{
public ImpFly()
{
}
private void finish(double t, MemoryStream outStream)
{
// Запись времени долета мухи до пешехода в выходной
//поток.
BinaryWriter outp = new BinaryWriter(outStream);
outp.Write(t);
outp.Close();
outStream.Flush();
}
public override void reaching(MemoryStream inStream,
MemoryStream outStream)
{
// Метод - событие. Вычисляет ближайшее время долета
//мухи до пешехода.
FlyPhase phase = new FlyPhase();
FlyParam param = new FlyParam();
BinaryReader inp = new BinaryReader(inStream);
```

```
phase.x = inp.ReadDouble();
phase.v = inp.ReadDouble();
param.man0Phase.x = inp.ReadDouble();
param.man0Phase.v = inp.ReadDouble();
param.man1Phase.x = inp.ReadDouble();
param.man1Phase.v = inp.ReadDouble();
// Прочли фазу и параметры компоненты "муха" из вход-
//ного потока.
inp.Close();
inStream.Flush();
inStream.Close();
// Закрыли входной поток, больше он не нужен.
// Событие "долет мухи до пешехода" - это когда у них
//уже равные координаты, а скорости еще противопо-
//ложного знака.
// После разворота мухи равенство координат - это уже не
//событие.
if (phase.v < 0)
{
// Если скорость мухи отрицательна
if (param.man0Phase.v > 0)
{
// Если скорость 1-го пешехода положительна
finish((phase.x - param.man0Phase.x) / (param.man0Phase.v
- phase.v), outStream);
return;
}
// Стало быть, это скорость 2-го пешехода положительна
finish((phase.x - param.man1Phase.x) / (param.man1Phase.v
- phase.v), outStream);
return;
}
// Стало быть, скорость мухи положительна
if (param.man0Phase.v < 0)
```

```
{
// Если скорость 1-го пешехода отрицательна
finish((param.man0Phase.x - phase.x) / (phase.v -
param.man0Phase.v), outStream);
return;
}
// Стало быть, это скорость 2-го пешехода отрицательна
finish((param.man1Phase.x - phase.x) / (phase.v -
param.man1Phase.v), outStream);
}
public override void Uturn(MemoryStream inStream,
MemoryStream outStream)
{
// Сосредоточенный метод разворота мухи
FlyPhase phase = new FlyPhase();
BinaryReader inp = new BinaryReader(inStream);
phase.v = inp.ReadDouble();
inp.Close();
inStream.Flush();
inStream.Close();
BinaryWriter outp = new BinaryWriter(outStream);
phase.v = -phase.v;
outp.Write(phase.v);
outp.Close();
outStream.Flush();
}
}
}
```

3.3.4. Сервисный модуль

Кроме непосредственно модели, которая в данном случае полностью описана выше, имеется возможность присоединить к модели необязательный так называемый сервисный модуль, с методами типа конструктора, деструктора и методами вызы-

вающимися после шагов медленных и быстрых элементов. Эти методы могут быть полезны для отладки модели и визуализации результатов моделирования. Сервисный модуль может быть только локальным! Метод-конструктор должен иметь имя "Prepare", метод-деструктор - имя "Finish", метод, вызываемый после такта моделирования ненулевой длительности - "AfterSlow", после такта нулевой длительности - "AfterFast". Параметры этих методов - массив объектов (object[]), в котором передаются все фазовые переменные модели.

Сервисный модуль модели "пешеходы и муха"

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;
using System.Threading;
namespace MYXA
{
public partial class MYXA : Form
{
public MYXA()
{
InitializeComponent();
}
object[] para = new object[9];
private bool execute=true;
private readonly Pen blackPen = new Pen(Color.Black, 2);
private void DrawAman(int step, Graphics dc, float x,
double v)
{
```

```
float kX = this.Width / 50;
float kY = this.Height / 20;
x -=0.6f;
// Коррекция координаты
if (v < 0) ++step;
// Чтобы шли не в ногу ;-)
RectangleF headArea = new RectangleF(x * kX, 5 * kY, 2 *
kX, 2 * kY);
RectangleF noseArea = v > 0 ? new RectangleF((x + 2) * kX,
6 * kY, kX / 4, kY / 4) : new RectangleF(x * kX - kX / 4, 6 *
kY, kX / 4, kY / 4);
RectangleF eyeArea = v > 0 ? new RectangleF((x + 2) * kX –
kX / 2, 6 * kY - kY / 4, kX / 4, kY / 4) : new RectangleF(x *
kX + kX / 4, 6 * kY - kY / 4, kX / 4, kY / 4);
dc.DrawEllipse(blackPen, headArea);
dc.DrawEllipse(blackPen, noseArea);
dc.DrawEllipse(blackPen, eyeArea);
if (step % 2 > 0)
{
dc.DrawLine(blackPen, (x + 1) * kX, 7 * kY, (x + 1) * kX,
13 * kY);
dc.DrawLine(blackPen, (x + 1) * kX, 13 * kY, (v>0)?(x + 2)
* kX : x*kX, 16 * kY);
dc.DrawLine(blackPen, (x + 1) * kX, 13 * kY, (v>0)? x *
kX: (x + 2) * kX, 16 * kY);
dc.DrawLine(blackPen, (v > 0) ? (x + 2) * kX : x * kX, 16 *
kY, (v > 0) ? (x + 2.2f) * kX : (x-0.2f)*kX, 17.8f * kY);
dc.DrawLine(blackPen, (v > 0) ? (x + 2.2f) * kX : (x - 0.2f) *
kX, 17.8f * kY, (v > 0) ?(x + 3) * kX : (x-1)*kX, 18 * kY);
dc.DrawLine(blackPen, (v > 0) ? x * kX : (x + 2) * kX,
16 * kY, (v > 0) ? (x - 1) * kX : (x+3)*kX, 17.5f * kY);
dc.DrawLine(blackPen, (v > 0) ? (x - 1) * kX : (x + 3) * kX,
17.5f * kY, (v > 0) ? x * kX - kX / 2 : (x + 2.5f) * kX,
18 * kY);
```

```
dc.DrawLine(blackPen, (x + 1) * kX, 8 * kY, (v > 0) ?
(x + 2.5f) * kX : (x+0.5f)*kX, 11 * kY);
dc.DrawLine(blackPen, (v > 0) ? (x + 2.5f) * kX : (x + 0.5f) *
kX, 11 * kY, (v > 0) ? (x + 3f) * kX : (x-1)*kX, 12 * kY);
dc.DrawLine(blackPen, (x + 1) * kX, 8 * kY, (v > 0) ?
(x - 0.5f) * kX : (x+2.5f)*kX, 11 * kY);
dc.DrawLine(blackPen, (v > 0) ? (x - 0.5f) * kX : (x + 2.5f) *
kX, 11 * kY, (v > 0) ? (x - 0.25f) * kX : (x + 2.25f) * kX,
13 * kY);
}
else
{
dc.DrawLine(blackPen, (x + 1) * kX, 7 * kY, (x + 1) * kX,
13 * kY);
dc.DrawLine(blackPen, (x + 1) * kX, 12.99f * kY, (v > 0) ?
(x + 2.1f) * kX : (x + 0.9f) * kX, 18 * kY);
dc.DrawLine(blackPen, (v > 0) ? (x + 2.1f) * kX : (x + 0.9f) *
kX, 18 * kY, (v > 0) ? (x + 2) * kX : x * kX, 18 * kY);
dc.DrawLine(blackPen, (x + 1) * kX, 12.99f * kY, (v > 0) ?
(x + 0.85f) * kX : (x + 2.15f) * kX, 17.8f * kY);
dc.DrawLine(blackPen, (v > 0) ? (x + 0.85f) * kX :
(x + 2.15f) * kX, 17.8f * kY, (v > 0) ? (x + 2.85f) * kX :
(x + 0.15f) * kX, 18 * kY);
dc.DrawLine(blackPen, (x + 1) * kX, 8 * kY, (v > 0) ?
(x + 0.85f) * kX : (x + 2.15f) * kX, 11 * kY);
dc.DrawLine(blackPen, (v > 0) ? (x + 0.85f) * kX :
(x + 2.15f) * kX, 11 * kY, (v > 0) ? (x + 3f) * kX : (x - 1) *
kX, 12 * kY);
dc.DrawLine(blackPen, (x + 1) * kX, 8 * kY, (x +1) * kX,
11 * kY);
dc.DrawLine(blackPen, (x + 1) * kX , 11 * kY, (v > 0) ?
(x + 2.25f) * kX : (x + 0.75f) * kX, 14 * kY);
}
}
```

```
private void DrawAfly(int step, Graphics dc, float x,
double v)
{
float kX = this.Width / 50;
float kY = this.Height / 20;
x += 0.45f;
// Коррекция координаты
if (v < 0) ++step;
// Чтобы махи чередовались аккуратно
RectangleF bodyArea = new RectangleF((x-1) * kX, 4 * kY,
2 * kX, kY);
RectangleF headArea = v > 0 ? new RectangleF((x + 1) * kX,
4.25f * kY, kX / 2, kY / 2) : new RectangleF((x-2.5f) * kX,
4.25f * kY, kX / 2, kY / 2);
dc.DrawEllipse(blackPen, bodyArea);
dc.DrawEllipse(blackPen, headArea);
if ((step % 2 > 0)!=(v < 0))
{
dc.DrawLine(blackPen, x * kX, 4.5f * kY, (x - 0.3f) * kX,
3 * kY);
dc.DrawLine(blackPen, (x - 0.3f) * kX, 3 * kY, (x - 0.5f) *
kX, 4.5f * kY);
dc.DrawLine(blackPen, (x+0.4f) * kX, 4 * kY, (x + 0.1f) *
kX, 3 * kY);
dc.DrawLine(blackPen, (x + 0.1f) * kX, 3 * kY, (x ) * kX,
4 * kY);
}
else
{
dc.DrawLine(blackPen, x * kX, 4.5f * kY, (x - 0.3f) * kX,
6f * kY);
dc.DrawLine(blackPen, (x - 0.3f) * kX, 6 * kY, (x - 0.5f) *
kX, 4.5f * kY);
dc.DrawLine(blackPen, (x+0.4f) * kX, 5 * kY, (x + 0.1f) *
```

```
kX, 6 * kY);
dc.DrawLine(blackPen, (x + 0.1f) * kX, 6 * kY, (x ) * kX,
5 * kY);
}
}
private void transfer(object[] param)
{
for (int i = 0; i < 9; i++)
{
para[i] = param[i];
}
}
private void DrawAll(Graphics dc)
{
lock (this)
{
int step = (int)para[0];
double flyX = (double)para[1];
double flyV = (double)para[2];
double man0X = (double)para[3];
double man0V = (double)para[4];
double man1X = (double)para[5];
double man1V = (double)para[6];
float kX = this.Width / 50;
float kY = this.Height / 20;
dc.Clear(Color.White);
dc.DrawLine(blackPen, 20 * kX, 18 * kY, 32 * kX, 0);
// Фон гора
dc.DrawLine(blackPen, 32 * kX, 0, 47 * kX, 18 * kY);
// Фон гора
RectangleF sunBox = new RectangleF(10 * kX, 0, 4 * kX,
4 * kY);
dc.DrawEllipse(blackPen, sunBox); // Солнце
DrawAman(step, dc, (float)man0X, man0V);
```

```
DrawAman(step, dc, (float)man1X, man1V);
DrawAfly(step, dc, (float)flyX, flyV);
string time=((double)para[8]).ToString();
if (time.Length > 5)
time = time.Substring(0, 5);
label2.Text = "t = " + time;
}
}
public bool Prepare(object[] param)
{
transfer(param);
Invalidate();
// this.Refresh();
Update();

return (execute);
}
public bool AfterFast(object[] param)
{
lock (this)
{
transfer(param);
// this.Refresh();
Invalidate();
Update();
}
return (execute);
}
public bool AfterSlow(object[] param)
{
lock (this)
{
double x0 = (double)param[3];
double x1 = (double)param[5];
```

```
transfer(param);
Invalidate();
Update();
// this.Refresh();
execute = execute && (x1 - x0 > 0.1);
/*
if (!execute)
MessageBox.Show("Вот оно!!!");
*/
}
return (execute);
}
public bool Finish(object[] param)
{
return (true);
}

protected override void OnPaint(PaintEventArgs e)
{
base.OnPaint(e);
Graphics dc = e.Graphics;
DrawAll(dc);
/********* если хочется сохранить картинки ****
Bitmap bm = new Bitmap(Width, Height);
Rectangle rec = new Rectangle(0, 0, Width, Height);
Rectangle rec1 = new Rectangle(8, 30, Width-16, Height-38);
DrawToBitmap(bm, rec);
Bitmap bmf = bm.Clone(rec1,
System.Drawing.Imaging.PixelFormat.DontCare);
bmf.Save("fly" + ((int)para[0]).ToString() + ".bmp");
**********************************************/
}
}
}
```

Заключение

В данной работе автору хотелось, оттолкнувшись от некоторых общепризнанных в кругах разработчиков имитационных моделей положений, таких как требование замкнутости модели, однозначности и детерминированности имитационных вычислений, попробовать дедуктивно вывести из этих предположений некоторые свойства получаемых моделей.

Полученные результаты можно считать начальными элементами теории моделирования сложных систем. Несмотря на всю свою элементарность, построенная теория работает. Одним из нетривиальных выводов этой теории является заключение о недостаточности замкнутости в традиционном ее понимании для построения модели. В связи с этим, вводится понятие прогнозируемости модели. Оно отличается от понятия замкнутости направлением взгляда на отрезок моделирования. Замкнутость смотрит на этот отрезок с его левого конца, прогнозируемость – с правого. Взгляд справа отражает цитата из П.С. Лапласа, взятая эпиграфом к пункту 2.2.3, взгляд слева – оставшееся у автора от прослушанных в студенческие и аспирантские годы курсов философии представление о Лапласовском детерминизме, которое несколько утрируя можно выразить следующим образом: «Господь Бог (если таковой был), однажды сотворив мир, полностью удалился от дел, и с тех пор мир развивается по законам динамических систем, нужно только уметь находить соответствующие уравнения, подставлять в них начальные условия, решать и получать научный прогноз». Между прочим, уже упоминавшаяся работа Лапласа, так и называется: «Изложение системы мира» [47].

По поводу подобного детерминизма, автор, ощущая себя тоже немного творцом, хотя бы виртуального мира пешеходов и мухи (приходилось несколько раз реализовывать этот мир на компьютере), может со всей ответственностью заявить, что опочить можно себе позволить лишь до момента встречи пешеходов. К этому моменту творцу пора вновь браться за свое

творение и срочно творить самое что ни на есть настоящее чудо – давать мухе новую скорость. Сам виртуальный мир пешеходов и мухи, исходя из естественных, лапласовских законов своего развития, с этой проблемой справиться не в силах. Можно, конечно, возразить, что не бывает на свете таких мух, и такое возражение, по-видимому, справедливо. Однако что-то подсказывает, что в нашем реальном мире могут встретиться вещи и похитрее упомянутой мухи.

Если кроме локальной замкнутости потребовать еще и локальной прогнозируемости – можно получить лапласовскую или прогнозируемую модель (теорема 2.2). Возможность получения лапласовской модели становится конструктивной лишь в простейшем случае, когда модель задается системой обыкновенных дифференциальных уравнений с правой частью, удовлетворяющей условию Липшица по совокупности своих переменных (теорема 2.1). При этом, как показывает пример с апорией «Ахиллес и черепаха», у разработчика всегда остается возможность «испортить» плохой реализацией даже самую хорошую (лапласовскую и даже липшицеву) модель.

Далее, если наша модель лапласовская – из определения 2.3 следует, что спор о том, какой способ диспетчеризации времени пессимистический или оптимистический лучше, становится бессмысленным. Упомянутое определение утверждает полное право разработчика получать все характеристики модели исходя из их значений и значений внешних переменных на левом конце шага моделирования. В самом деле, если функция, вычисляющая некую характеристику модели «хорошая» (например, равномерно непрерывная по t на шаге моделирования), то при правильно выбранном шаге моделирования, безразлично с какого его конца вычислять ее значения. Если же она «совсем плохая» (например, дискретная да еще и стохастическая) – снова это безразлично, так как все равно необходимо привлекать генератор случайных чисел, который,

по-видимому, правильнее привязывать к продолжительности шага моделирования, как к множеству положительной меры, а не к его концам.

На взгляд автора, как уже говорилось об этом во введении, использование сложных и недостаточно формально обоснованных эвристических методов, ведет к стремлению компенсировать их недостаточную обоснованность рассмотрением максимально широкого класса потенциально возможных случаев их применения. Такое стремление часто приводит к анализу избыточно широкого класса возможных применений эвристических методов. Часто если впоследствии удается провести формальный анализ этой области, оказывается, что некоторые случаи, анализу которых уделялись значительное внимание и силы, на самом деле никогда не реализуются. Подобные пустые хлопоты автор называет «фобиями». По-видимому к «фобиям» можно отнести и слишком замысловатые схемы диспетчеризации времени. Хотя следует признать, что в их основе, по-видимому, лежит верное интуитивное представление о различной роли правых и левых концов отрезков моделирования, формализованное в данной работе понятиями локальной замкнутости и локальной прогнозируемости.

Автор и сам долгое время страдал одной из таких «фобий» – опасением, что алгоритм организации имитационных вычислений компоненты с переменным шагом времени, описанный в конце пункта 2.2.6, будет зацикливаться в связи с выбором все меньшего и меньшего шага, как это происходит в моделях «пешеходы и муха» или «Ахиллес и черепаха». Настолько, что очень удивлялся, почему в работах по моделированию сложных систем, кроме него никто не трубит об этой страшной опасности. По мере осознания такого свойства моделей, как прогнозируемость в точке, и, в особенности, после доказательства теоремы 2.2, эти опасения в значительной мере уменьшились. Действительно, точки непрогнозируемости – все же достаточно специфическое явление, поэтому есть наде-

жда выявить их на стадии предварительного анализа модели. Далее же – «предупрежден, значит вооружен» – можно избегать дурной бесконечности, ограничиваясь разумной точностью вычислений в их окрестности, как об этом говорилось в пункте 2.2.4. Если же точек непрогнозируемости нет и модель лапласовская – ее, конечно, можно испортить, как в случае с Ахиллесом и черепахой, однако теорема 2.2 утверждает, что решение есть, модель можно построить. Как – это в настоящее время вопрос искусства моделирования, – у автора есть формализованный ответ на этот вопрос лишь для моделей, удовлетворяющих условиям теоремы 2.1 и состоит он просто в выборе достаточно малого шага моделирования.

Еще одна распространенная «фобия» в области моделирования сложных систем – опасения попыток одновременного изменения характеристик модели параллельно выполняющимися алгоритмами, а также связанных с этим зацикливаний системы при попытке установить очередность доступа к характеристикам. Если мы однажды признали однозначность вычислительного процесса имитации – всякая попытка одновременно изменить какую-либо характеристику модели несколькими алгоритмами есть преступление против этой однозначности. При этом всегда имеется легкий способ избежать этого преступления – ввести новую компоненту, задачей которой будет выбирать «из многих – одно». Предложение автора по этому поводу – распределение характеристик компоненты между ее элементами на этапе проектирования компоненты с последующим жестким запретом изменять «чужие» характеристики. В макете рабочей станции распределенного моделирования, описанном в третьей главе, реализовано две проверки исполнения этой дисциплины, на этапе компиляции описаний и на этапе выполнения модели.

Жесткая дисциплина «каждый меняет только свои характеристики» разрешает и еще одну достаточно распространенную при обсуждении проблем моделирования «фобию». Про-

иллюстрируем ее на примере «дуэли ковбоев». Как известно, в мире ковбоев прав тот, кто выстрелил первым. С другой стороны, детерминированность процесса имитационных вычислений предполагает составление списков элементов на выполнение и в этих списках всегда кто-то будет первым, кто-то вторым, а кто-то и последним. Место в списке скорее всего будет зависеть от порядка компиляции тех или иных описателей. Возникает вопрос, не получат ли те элементы, которые окажутся в начале этого списка систематических преимуществ перед теми, кто попадет в конец? В качестве решения этой проблемы, автору приходилось, например, слышать предложения на каждом шаге моделирования стохастически переупорядочивать списки элементов на выполнение. На самом деле проблему можно решить дисциплиной доступа к характеристикам на этапе проектирования модели. Действительно, если каждый из ковбоев «выключает» другого, например, записывая в его характеристики признак «убит», тогда и в самом деле тот, кто оказался раньше в списке имеет преимущество – пока очередь дойдет до второго – он окажется уже выключенным. Если же проектировать модель в соответствии с концепцией моделирования, изложенной в этой книге, – каждый из ковбоев должен записать во внешние переменные другого, что в того летит пуля с определенной скоростью, из определенного места. А уж решать вопрос о том умирать ли ему от этой пули, или, быть может, он успеет убежать, будет каждый ковбой сам, исходя из значений своих внутренних и внешних переменных. При таком подходе безразлично кто и какое место занял в списке элементов на выполнение.

Кроме названных выше «фобий», предложенные во второй главе теоретические основы моделирования позволяют устранить из концепции построения сложных моделей такой способ синхронизации выполнения их компонент, как обмен ими сигналами и сообщениями. Этот способ с точки зрения здравого смысла кажется вполне приемлемым [39, 40, 8]: так

часто происходит в жизни, так часто поступают при конструировании технических систем. Немалую дань отдал этому способу синхронизации и автор в ранних работах [32 – 34, 1, 2] (см., например, раздел посвященный инструментальной системе имитационного моделирования MISS в первой главе). Однако способ синхронизации компонент через сигналы и сообщения, хотя и работает, но с точки зрения развитой во второй главе теории, оказывается избыточным. Для синхронизации компонент модели, исходя из гипотезы о ее замкнутости, вполне достаточно гораздо более фундаментальных механизмов связи, таких как связь внутренних характеристик одних, с внешними характеристиками других компонент. Всякое же излишество лишь «утяжеляет» и «удорожает» любую конструкцию, делает концепцию моделирования труднее как для восприятия, так и для реализации.

Из приведенного перечня решенных проблем и псевдопроблем-«фобий» следует, что предложенные во второй главе начала формализации моделирования сложных систем, несмотря на всю свою элементарность, тем не менее, вполне успешно работают.

Разработанная концепция моделирования сложных систем реализована на практике, в виде программного обеспечения макета рабочей станции пиринговой системы распределенного имитационного моделирования.

На рабочей станции во-первых, можно полностью создавать модели сложных систем из элементов собственной разработки и элементов, опубликованных другими участниками пиринговой сети. Во-вторых, можно публиковать в сети реализованные на этой рабочей станции элементы, которые после этого становятся доступными для включения их в модели, разрабатываемые на других станциях.

В дальнейшем предполагается сделать пиринговую сеть моделирования частично-централизованной, разработав информационные серверы, которые содержали бы каталоги дос-

тупных в Интернете ресурсов в области моделирования, их описаний, руководств по использованию и т. д.

С современным состоянием проекта разработки пиринговой сети распределенного имитационного моделирования можно познакомиться на сайте отдела «Имитационные системы и исследование операций» ВЦ РАН в Интернете, по адресу: http://simul.ccas.ru/Distr.

Литература

1. *Brodskii Y. I.* Simulation Software //Encyclopedia of Life Support Systems (EOLSS), Oxford, EOLSS Publishers Co. Ltd., 2002.
2. *Brodskii Y. I., Tokarev V. V.* Fundamentals of simulation for complex systems. //Encyclopedia of Life Support Systems (EOLSS), Oxford, EOLSS Publishers Co. Ltd., 2002.
3. *Chandy, K. M. and Misra J.* Distributed Simulation: A Case Study in Design and Verification of Distributed Programs //IEEE Transactions on Software Engineering SE-5(5), 1978: 440-452.
4. *Chandy, K. M. and Misra J.* Asynchronous Distributed Simulation via a Sequence of Parallel Computations //Communications of the ACM 24(4), 1981: 198-205.
5. *Emelyanov S.V., Afanasiev A.P., Grinberg Y.R., Krivtsov V.E., Peltsverger B.V., Sukhoroslov O.V., Taylor R.G., Voloshinov V.V.* Distributed Computing and Its Applications. Felicity Press, Bristol, USA, 2005, 298 p.
6. *Fujimoto, R. M.* Parallel and Distributed Simulation Systems A Wiley-interscience publication, New York, Chichester, Weinheim, Brisbane, Singapore, Toronto, 2000, 300 p.
7. *Fujimoto, R. M.* Distributed Simulation Systems //Proceedings of the 2003 Winter Simulation Conference (WSC-2003), December, 2003, pp. 124-134.
8. *Kuhl F., Weatherly R., Dahmann J.* Creating Computer Simulation Systems: An Introduction to the High Level Architecture NY: Prentice Hall PTR, 1999. – 212 p.
9. *Афанасьев А.П., Волошинов В.В., Гринберг Я.Р., Емельянов С.В., Кривцов В.Е., Сухорослов О.В.* Реализация GRID-вычислений в среде IARnet // Информационные технологии и вычислительные системы, №2, 2005, С. 61-76.
10. *Арнольд В.И.* Жесткие и мягкие математические модели М.:

МЦНМО, 2000, 32 с.
11. *Арнольд В.И.* Гюйгенс и Барроу, Ньютон и Гук: первые шаги математического анализа и теории катастроф, от эвольвент до квазикристаллов М.: Наука, 1989, 96 с.
12. *Белотелов Н.В., Бродский Ю.И., Винокуров С.Ф., Высоцкий М.Н., Коровко М.А., Кручина Е.Б., Миносьянц С.С., Мягков, А.Н., Оленев Н.Н., Павловский Ю.Н., Тарасова Н.П.* Лабораторный практикум по математическому моделированию. М.: РХТУ им. Д.И. Менделеева. 2009. 63 с.
13. *Белотелов Н.В., Бродский Ю.И., Кручина Е.Б., Оленев Н.Н., Павловский Ю.Н.* Имитационная игра на основе Экологическо – Демографическо – Экономической Модели (ЭДЭМ): описание и инструкция пользователю (учебное пособие) М.: РХТУ им. Д.И. Менделеева, 2003. 84 с.
14. *Белотелов Н.В., Бродский Ю.И., Оленев Н.Н., Павловский Ю.Н.* Эколого-социально-экономическая модель: гуманитарный и информационный аспекты //Информационное общество. № 6. 2001. С. 43-51.
15. *Белотелов Н.В., Бродский, Ю.И. Оленев Н.Н., Павловский Ю.Н., Тарасова Н.П.* Проблема устойчивого развития: естественно-научный и гуманитарный анализ. М.: Фазис. 2004. 108 с.
16. *Белотелов Н.В., Бродский Ю.И., Павловский Ю.Н.* Компьютерное моделирование демографических, миграционных, эколого-экономических процессов средствами распределенных вычислений. М.: ВЦ РАН, 2008. 123 с.
17. *Белотелов Н.В., Бродский Ю.И., Павловский Ю.Н.* Сложность. Математическое моделирование. Гуманитарный анализ. М.: Книжный дом «ЛИБРОКОМ», 2009, 320 с.

18. *Белотелов Н.В., Бродский Ю.И., Павловский Ю.Н.* Глава 16 «Имитационное математическре моделирование и прогноз глобальных процессов» //В монографии Прогноз и моделирование кризисов и мировой динамики под ред. А.А. Акаева, А.В. Коротаева, Г.Г. Малинецкого, М.: Изд-во ЛКИ, 2009, 352 с.
19. *Белотелов Н.В., Бродский Ю.И., Павловский Ю.Н.* Разработка инструментальной системы распределенного моделирования. //IV Всероссийская научная конференция «Математическое моделирование развивающейся экономики и экологии», ЭКОМОД-2009, /Сборник трудов. Киров, изд-во. ВятГУ, 2009, С. 61-73.
20. *Боев В. Д., Кирик Д. И., Сыпченко Р. П.* Компьютерное моделирование: Пособие для курсового и дипломного проектирования. — СПб.: ВАС, 2011. — 348 с.
21. *Бродский Ю.И.* Эколого-социально-экономическая имитационная модель: технология реализации. //Моделирование, декомпозиция и оптимизация сложных динамических процессов. М.: ВЦ РАН, 2001. С. 89-107.
22. *Бродский Ю.И.* Проблемы создания центра имитационного моделирования в ИНТЕРНЕТ. //Моделирование, декомпозиция и оптимизация сложных динамических процессов. М.: ВЦ РАН, 1998. С. 29-35.
23. *Бродский Ю.И.* Разработка инструментальных средств имитационного моделирования сложных технических комплексов для параллельных вычислительных систем. //Моделирование, декомпозиция и оптимизация сложных динамических процессов. М.: ВЦ РАН, 1999. С. 33-92.
24. *Бродский Ю.И.* Декомпозиционные методы в инструментальных средствах имитации. Тез. докл. 1-й Московской конференции «Декомпозиционные методы в

математическом моделировании». М.: ВЦ РАН, 2001. С. 21-22.
25. *Бродский Ю.И.* О переходе от макроописания математической модели к ее объектно-событийному представлению //«Моделирование, декомпозиция и оптимизация сложных динамических процессов», М.: ВЦ РАН. 2004. С. 22-28.
26. *Бродский Ю.И.* К разработке концепции построения инструментальной системы распределенного моделирования. //«Моделирование, декомпозиция и оптимизация сложных динамических процессов», М.: ВЦ РАН. 2007. С. 14-34.
27. *Бродский Ю.И.* Описание, компоновка и работа модели в инструментальной системе распределенного моделирования. //«Моделирование, декомпозиция и оптимизация сложных динамических процессов», М.: ВЦ РАН, 2008. С. 24-46.
28. *Бродский Ю.И.* Толерантность и нетерпимость с точки зрения системной динамики и исследования операций М.: ВЦ РАН, 2008, 53 с.
29. *Бродский Ю.И.* Некоторые вопросы синтеза сложных распределенных имитационных моделей //«Моделирование, декомпозиция и оптимизация сложных динамических процессов», М.: ВЦ РАН, 2010, С. 25-40.
30. *Бродский Ю.И., Гринберг Я.Р., Павловский Ю.Н.* Имитационное моделирование в распределенной информационно-вычислительной среде // Проблемы вычислений в распределенной среде Труды ИСА РАН. - М.: РОХОС, 2003, С. 121-140.
31. *Бродский Ю.И., Гринберг Я.Р., Павловский Ю.Н.* Имитационное моделирование в распределенной информационно-вычислительной среде // Проблемы вычислений в распределенной среде: прикладные задачи. Ч.2, Труды ИСА РАН. - М.: РОХОС, 2004, С. 95-110.

32. *Бродский Ю.И., Лебедев В.Ю.* Инструментальная система для построения имитационных моделей хорошо структурированных организационно-технических комплексов //Вопросы кибернетики. Проблемы математического моделирования и экспертные системы, М.: Научный совет АН СССР по комплексной проблеме «Кибернетика», 1990, С. 49-64.
33. *Бродский Ю.И., Лебедев В.Ю.* Инструментальная система имитации MISS М.: ВЦ АН СССР, 1991, 180 с.
34. *Бродский Ю.И., Лебедев В.Ю., Огарышев В.Ф., Павловский Ю.Н., Савин Г.И.* Общие проблемы моделирования сложных организационно-технических систем //Вопросы кибернетики. Проблемы математического моделирования и экспертные системы, М.: Научный совет АН СССР по комплексной проблеме «Кибернетика», 1990, С. 42-48.
35. *Бродский Ю.И., Новицкий В.И., Павловский Ю.Н.* Об алгоритме, формирующем иерархическую систему инвариантов изоморфизмов отображений конечных множеств в себя //Дискретный анализ и исследование операций, Серия 2, Том 13, №2, 2006, С. 21-30.
36. *Бродский Ю.И., Павловский Ю.Н.* Разработка инструментальной системы распределенного имитационного моделирования //Прикладные проблемы управления макросистемами. Под ред. Попкова Ю.С., Путилова В.А. Т.39, М.: ИСА РАН, 2008. С. 79-99.
37. *Бродский Ю.И., Павловский Ю.Н.* Разработка инструментальной системы распределенного имитационного моделирования. //Информационные технологии и вычислительные системы, №4, 2009, С. 9-21.
38. *Бурбаки Н.* Теория множеств. М.: Мир. 1965. 456 с.
39. *Бусленко Н.П.* Сложная система //Статья в Большой Советской Энциклопедии, 3-е изд., М.: Советская энциклопедия, 1969-1978.

40. *Бусленко Н.П.* Моделирование сложных систем М.: Наука, 1978, 400 с.
41. *В.В.Воеводин., Вл.В.Воеводин.* Параллельные вычисления. - СПб.: БХВ-Петербург, 2002. 608 с.
42. *Воротынцев А.В.* Концепция сетевых информационно-вычислительных библиотек М.: ВЦ РАН, 2009, 108 с.
43. *Замятина Е.Б.* Современные теории имитационного моделирования (Специальный курс для магистров второго курса) Пермь: ПГУ, 2007, 119 с.
44. *Колмогоров А.Н., Фомин С.Г.* Элементы теории функций и функционального анализа. М.: «Наука», 1972, 496с.
45. *Королев А.Г.* Моделирование систем средствами Object GPSS. Практический подход в примерах и задачах. Учебное пособие. Луганск: Изд-во Восточно-Украинского нац. ун-та, 2005, 137с.
46. *Краснощеков П.С., Петров А.А.* Принципы построения моделей. Изд. 2-е, пересмотр. и дополнен., М.: Фазис. 2000. 412 с.
47. *Лаплас П.С.* Изложение системы мира М.: Наука, 1982, 676 с.
48. *Осоргин А.Е.* AnyLogic 6. Лабораторный практикум Самара: ПГК, 2011, 100 с.
49. *Павловский Ю.Н.* Имитационные системы и модели М.: Знание, 1990, 46 с.
50. *Павловский Ю.Н.* Имитационные модели и системы М.: Фазис, 2000, 131 с.
51. *Павловский Ю.Н., Белотелов Н.В., Бродский Ю.И.* Имитационное моделирование: учеб. пособие для студ. высш. учеб. заведений М.: Изд. центр «Академия», 2008, 236 с.
52. *Павловский Ю.Н., Белотелов Н.В., Бродский, Ю.И. Оленев Н.Н.* Опыт имитационного моделирования при анализе социально-экономических явлений М.: МЗ Пресс, 2005, 137 с.

53. *Павловский Ю.Н., Смирнова Т.Г.* Введение в геометрическую теорию декомпозиции М.: Фазис, 2006, 169 с.
54. *Савин Г.И.* Системное моделирование сложных процессов. М.: Фазис: ВЦ РАН, 2000, 276 с.
55. *Советов Б.Я., Яковлев С.А.* Моделирование систем: Учеб. для вузов – 3-е изд., перераб. и доп. – М.: Высш. шк., 2001, 343 с.
56. *Таненбаум Э.* Распределенные системы: принципы и парадигмы. - СПб: Питер, 2003, 877 с.
57. *Тарасова Н.П., Павловский Ю.Н., Кручина Е.Б., Белотелов Н.В., Бродский Ю.И., Оленев Н.Н.* Методика применения эколого-демографо-экономической модели виртуального государства для оценки качества человеческого потенциала //Менеджмент в России и за рубежом, №4, 2006, С.38-45.
58. *Турчин В.Ф.* Феномен науки: Кибернетический подход к эволюции Изд. 2-е, М.: ЭТС, 2000, 368 с.
59. *Форрестер Дж.* Мировая динамика М.: Физматгиз. 1978. 168 с.
60. *Харитонов В.В.* Распределенное моделирование гибридных систем. //Материалы межвузовской научной конференции, СПб.: СПбГТУ, 2002, С. 128-129.
61. *Шрайбер Т. Дж.* Моделирование на GPSS М.: Машиностроение, 1980, 592 с.

Содержание

Бродский Юрий Игоревич

РАСПРЕДЕЛЕННОЕ ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ СЛОЖНЫХ СИСТЕМ

Подписано в печать 21.12.2010
Формат бумаги 60×84 1/16
Уч.-изд.л. 6,5. Усл.-печ.л. 9,75
Тираж 120 экз. Заказ 56

Отпечатано на ротапринтах в Вычислительном центре
им. А.А. Дородницына Российской академии наук
119991, Москва, ул. Вавилова, 40