На правах рукописи

Бродский Юрий Игоревич

ПРОБЛЕМА ОПИСАНИЯ И СИНТЕЗА РАСПРЕДЕЛЕННЫХ ИМИТАЦИОННЫХ МОДЕЛЕЙ СЛОЖНЫХ МНОГОКОМПОНЕНТНЫХ СИСТЕМ

Специальность: 05.13.17 Теоретические основы информатики

диссертация на соискание ученой степени

доктора физико-математических наук

Научный консультант д.ф.-м.н., член-корр. РАН,

Павловский Юрий Николаевич.

Москва – 2015

Оглавление

Введение

Предметом данной работы будут проблемы имитационного моделирования сложных систем. При этом, таких сложных систем, которые в предметной области имеют явно выраженную составную структуру, причем их компоненты сами могут быть сложными системами. Тем не менее, продвигаясь вниз в анализе устройства таких компонент, однажды мы доходим до «атомов», про которые известно все: как они устроены, что умеют делать, по каким правилам взаимодействуют между собой.

Предлагается новый подход к проектированию и компьютерной реализации имитационных моделей сложных многокомпонентных систем, отличающийся от господствующего в последние десятилетия объектного подхода. Он назван в работе модельным синтезом, а его программная реализация – модельно-ориентированным программированием. Под модельным синтезом, в первую очередь, понимается подход к решению сформулированной выше проблемы построения модели сложной системы на основе моделирования составляющих ее компонент. Кроме того, модельный синтез – это еще и альтернатива объектному анализу – основной парадигме современного проектирования и программирования больших программных комплексов, если речь идет о создании программных комплексов, имеющих выраженное «атомистическое» строение.

Центральным понятием предлагаемого подхода и в то же время элементарным кирпичиком для построения любых более сложных конструк-

ций является понятие модели-компоненты. Модель-компонента наделена более сложной структурой, чем, например, объект объектного анализа. Эта структура обеспечивает модели-компоненте поведение – способность стандартным образом отвечать на стандартные запросы ее внутренней и внешней среды.

Организация имитационных вычислений модели-компоненты оказывается инвариантной относительно операции объединения моделей-компонент в модель-комплекс, поскольку модель-комплекс сохраняет структуру модели-компоненты. Это позволяет, во-первых, строить фрактальные модели любой сложности и, во-вторых, реализовывать вычислительный процесс даже очень сложных моделей единообразно – единой программой. Кроме того, предлагаемый подход к компьютерной реализации многокомпонентных имитационных моделей полностью исключает императивное программирование и позволяет производить программный код высокой степени параллельности.

Проблемами имитационного моделирования, притом весьма непростыми, являются:

1. Зафиксировать это наше знание в виде полного и достаточно формального описания устройства сложной системы.
2. На основе такого описания попытаться воспроизвести поведение сложной системы (построить ее имитационную модель), например, с целью изучения и оценки возможностей такой системы в целом.

В плане неформального понимания того, что такое сложная многокомпонентная система, автору очень близки следующие три высказывания видного специалиста в области сложных систем и их имитационного моделирования, член.-корр. АН СССР Н.П. Бусленко, почерпнутые из работ [34, 35]:

«Сложная система – составной объект, части которого можно рассматривать как системы, закономерно объединённые в единое целое в соответствии с определенными принципами или связанные между собой заданными отношениями».

«В каждый момент времени элемент сложной системы находится в одном из возможных состояний; из одного состояния в другое он переходит под действием внешних и внутренних факторов».

«Для построения синтеза поведения сложной системы необходимо дать ее компонентам возможность в полной мере проявить себя».

В некотором смысле все, что предложено в данной работе, является одной из возможных реализаций приведенных выше высказываний.

Со времен античности и до наших дней существует два основных подхода к моделированию сложных систем:

1. Один из них, называвшийся в разное время и в различных ситуациях «феноменологическим», «волновым», «системно-динамическим» состоит в том, что у изучаемой системы выделяются поддающиеся измерению характеристики, описывающие эту систему в целом. По-

стулируется достаточность выделенных характеристик для описания системы (гипотеза о замкнутости) и ищутся связи между этими характеристиками. В рамках данного подхода работали Гераклит и милетские философы, Гюйгенс в споре с Ньютоном о природе света, создатели классической термодинамики.

2. Второй подход, называвшийся «агентным», «объектным», «атомистическим», «корпускулярным», предполагает выводить свойства сложной системы из свойств и способов взаимодействия «атомов» – неких простейших объектов, эту систему составляющих. Этот подход развивали античные атомисты Левкипп и Демокрит, в новое время – Ньютон, в упоминавшемся выше споре с Гюйгенсом о природе света, создатели статистической физики.

В настоящее время общепринятой точкой зрения по данному вопросу является признание равноценности и взаимной дополняемости «корпускулярного» и «волнового» описания реальности. Некоторые явления удобнее описывать «феноменологически», другие – «атомистически», а наиболее полное знание мы имеем в тех областях, где оба эти описания проникают друг в друга, как, например, истолкование классических термодинамических потенциалов методами статистической физики.

В данной работе будет рассматриваться исключительно «атомистический» подход. Не потому, что автор считает его «важнее» системно-динамического, а потому, что он оказывается примерно в половине случа-

ев удобнее для моделирования реальных систем, и, следовательно, вполне заслуживает отдельного внимательного изучения (впрочем, как и системно-динамический).

Развивать атомистический подход к созданию сложных систем также можно двумя способами:

1. Дедуктивный способ. Предположим, что нужно создать сложную систему с заданной функциональностью (например, операционную систему или компилятор), а в выборе «атомов», реализующих различные функциональности мы свободны – разработка ведется с чистого листа. Тогда естественно поступить по-платоновски – проектировать систему сверху вниз, облекая основные идеи ее построения в соответствующие им формы, и воплощать эти формы в программный код. Идеалисты утверждают, что именно так и был создан этот мир. У программистов, для осуществления дедуктивного проектирования системы есть мощное средство – объектно-ориентированное программирование (ООП), чье наследование с возможностью переопределения методов, позволяет построить иерархию классов вполне в духе Платона.
2. Индуктивный способ. Предположим, что мы не создаем новые миры, а моделируем уже созданные. Тогда «атомы» системы жестко заданы предметной областью. Они определены и не могут быть иными, их свойства и способы взаимодействия также хорошо известны. Задачей

является воспроизведение и изучение поведения всей сложной системы, состоящей из этих «атомов». Заметим, что в данном случае обычно иерархическая структура классов ООП мало чем может помочь – нужно уметь работать с теми «атомами», которые есть в системе, а они далеко не всегда представляют собой строгую иерархию. Подробно эта проблема будет освещена далее.

Заметим, что в жизни дедуктивный и индуктивный способы построения сложных систем связаны своего рода диалектикой. Допустим, мы далеко продвинулись по дедуктивному пути, например, построили .NET – иерархическую систему из десятков тысяч классов. Теперь она всегда с нами, все классы отлажены, следовательно, вполне логично использовать их в своей работе, т. е., применять индуктивный способ, пользуясь кирпичиками .NET. Если же мы долгое время развивали индуктивный способ – со временем оказывается, что мы располагаем достаточно хаотичной коллекцией «атомов» с частично дублирующей друг друга функциональностью. Возникает естественное желание навести в этом богатстве определенный порядок – произвести дедуктивный реинжиниринг системы.

В данной работе будет рассматриваться исключительно индуктивный способ построения системы. Не потому, что он чем-то лучше дедуктивного, а потому, что класс задач мощностью примерно 25% (половина от половины) от всего на свете, заслуживает специального рассмотрения. Тем

более что такое мощное средство как ООП, работает для него гораздо хуже, чем для дедуктивного способа.

В качестве примеров корпускулярных индуктивных моделей сложных систем приведем моделирование боевых действий в ходе штабных игр с уровнем детализации полк – дивизия – армия – фронт; изучение возможностей стратегической оборонной инициативы (СОИ) – исследование, в котором участвовал автор в конце 80-х; моделирование работы холдинга, объединяющего ряд промышленных и финансовых предприятий.

Полностью соглашаясь на интуитивном уровне с тем, что сложная система может сама состоять из сложных систем, мы не беремся давать здесь строгое определение этому понятию. Тем не менее понятие модели сложной системы будет введено вполне формально, как семейство родов структур в смысле Н. Бурбаки [33].

С этой точки зрения, сложная система – это то, что достаточно адекватно может быть представлено моделью сложной системы.

Род структуры – развитие понятия множества. Базисное множество снабжается структурой некоторого рода – вводится определенный тип отношений между его элементами, и в зависимости от этого типа отношений, множество может стать, например, группой или решеткой, или векторным пространством, или же в нашем случае – имитационной моделью сложной системы. При этом математический объект, например конкретное линей-

ное пространство, является экземпляром структуры соответствующего рода.

Метод структурализма [39], идейно восходящий к Эрлангенской программе Ф. Клейна [63] и оказавшийся достаточно популярным и продуктивным в XX веке, в том числе и в гуманитарных науках, предлагает далее рассматривать различные преобразования базисных множеств и искать инварианты этих преобразований. Например, роды структур сохраняются при биекциях базисных множеств.

Школа член-корр. РАН Ю.Н. Павловского в ВЦ РАН в последние десятилетия развивала геометрическую теорию декомпозиции [65, 66, 74, 75, 42] где с помощью морфизмов пытаются найти более простые представления различных математических объектов – их редукции и декомпозиции.

В данной работе, посвященной синтезу модели сложной системы из моделей ее компонент, нас более всего будет интересовать возможность распространения рода структуры на объединение базисных множеств математических объектов, наделенных этим родом структуры. Инвариантом относительно объединения базисных множеств имитационных моделей, оказывается предложенный ниже способ организации имитационных вычислений.

Модельный синтез и анализ, как способ описания и синтеза имитационных моделей сложных многокомпонентных систем, развивался в отделе имитационных систем ВЦ РАН с конца 80-х гг. Основные его идеи и мето-

ды изложены в работах [1 - 3, 24 – 32], однако сам термин «модельный синтез» впервые предлагается в работе [24]. В основе модельного синтеза и анализа лежит понятие модели-компоненты.

С точки зрения построения программных систем, модель-компонента подобна объекту объектного анализа, но помимо характеристик снабженному не только методами, способными делать что-то полезное, если их вызовут, а неким аналогом операционной системы, всегда готовым давать стандартные ответы на стандартные запросы внутренней и внешней среды модели.

С формальной точки зрения, модель-компонента есть математический объект, базисным множеством которого является совокупность множеств внутренних и внешних характеристик модели, методов (того, что модель умеет делать) и событий (того, на что модель должна уметь реагировать). На базисном множестве вводится род структуры «модель-компонента», который обладает двумя замечательными свойствами:

1. Род структуры «модель-компонента» позволяет стандартным и однозначным образом организовать вычислительный процесс моделирования для всех объектов, снабженных структурой этого рода. Это означает возможность создания универсальной программы, способной запустить на выполнение любую имитационную модель, если та является математическим объектом, снабженным структурой рода «модель-компонента».

2. Вообще говоря, если рассмотреть два произвольных математических объекта снабженных структурой одного рода (например, структурой абстрактной группы), то распространение этой структуры на объединение их базисных множеств возможно далеко не всегда. Тем не менее для математических объектов, наделенных родами структур из семейства родов структур «модель-компонента», наделение объединения их базисных множеств родом структуры из того же семейства родов структур «модель-компонента» или возможно (если подмножества характеристик их базисных множеств не имеют попарных пересечений), или возможно с некоторыми оговорками (например, при условии пополнения исходных объектов-компонент некоторым количеством дополнительных объектов-компонент, снабженных той же структурой).

Второе свойство позволяет образовывать из моделей-компонент путем объединения их базисных множеств модели-комплексы, которые после распространения общей структуры компонент на объединение их базисных множеств оказываются математическими объектами того же самого семейства родов структур «модель-компонента» и стало быть, снова могут объединяться в модели-комплексы. Первое свойство позволяет не впадать в отчаяние от сложности вычислительного процесса, получающейся в результате таких объединений сверхсложной фрактальной модели.

Для программной реализации сложной многокомпонентной модели предлагается модельно-ориентированная парадигма программирования, где единицей проектирования программного комплекса является модель-компонента конструкция более агрегированная по сравнению с объектом объектного анализа.

Ниже будет показано, что модельно-ориентированное программирование позволяет исключить наиболее сложное как для разработки, так и для отладки императивное программирование [31]. Кроме того, получаемый исполняемый код отличается высокой степенью параллельности, при этом степень параллельности кода возрастает при росте сложности модели. Данный факт может открыть перспективы для применения методов модельно-ориентированного программирования на высокопроизводительных вычислительных системах, в том числе и для задач, не связанных с имитационным моделированием, но имеющих многокомпонентную организацию.

Относительно предлагаемого в работе модельно-ориентированного подхода к программированию следует отметить два важных момента:

1. Несмотря на внешнее сходство терминов под этим подходом подразумевается нечто вполне отличное от MDA (Model Driven Architecture), MDE (Model Driven Engineering) и MDD (Model Driven Development), – различных вариантов концепции разработки программных систем, управляемой моделями. Подробнее об отличиях

будет сказано в специальном разделе, посвященном сравнению модельного синтеза с объектным анализом.

2. Автор (хотя ему и приходилось программировать, быть может, больше и чаще, чем хотелось бы) отнюдь не считает себя профессиональным специалистом в области программирования. Поэтому пропагандирует модельно-ориентированный подход к программированию достаточно осторожно – не как универсальный метод решения всех на свете программистских проблем, а как метод, хорошо зарекомендовавший себя в четко ограниченной сфере его профессиональной деятельности – имитационном моделировании сложных многомпонентных систем. Тем не менее, с определенной обосновываемой в работе надеждой, что предлагаемый метод, быть может, окажется хорош также и для некоторого более широкого класса концептуально близких программных систем, возможно и не связанных с имитационным моделированием.

Глава I. Роды структур и элементы геометрической теории декомпозиции

Оригинальное изложение аппарата родов структур, содержащееся в работе [33], опирается на специфическую «бурбаковскую» терминологию и аксиоматику, отличающуюся от принятой в большинстве учебников. Тем не менее существуют работы, например [74] и [78], где этот аппарат излагается на основании обычной терминологии и аксиоматики. В русле именно этих работ, зачастую прямо цитируя их, мы и дадим здесь определение рода структуры и примеры математических объектов, наделенных различными родами структур. Начальные понятия геометрической теории декомпозиции и касающиеся их утверждения приводятся, следуя работам [65, 74]. Подробно познакомиться с теорией родов структур и с геометрической теорией декомпозиции, в том числе с доказательствами приводимых утверждений, можно в работах [33, 68, 74, 75, 78].

1.1. Определение рода структуры

Род структуры объявляет о том, что математический объект будет состоять из частей $\sigma_1,...,\sigma_r$ некоторых множеств. Это записывается следующим образом

$$\sigma_1 \subset S_1(X_1,...,X_n,A_1,...,A_m),\ldots,\ \sigma_r \subset S_r(X_1,...,X_n,A_1,...,A_m).$$

Здесь множества $X_1,...,X_n$ называются основными базисными, множества $A_1,...,A_m$ − вспомогательными базисными множествами (употребляемая терминология почерпнута из [33]).

Множества $S_k(X_1,...,X_n,A_1,...,A_m)$, $1 \le k \le r$, − так называемые ступени, построенные по схеме S_k из базисных и вспомогательных множеств $X_1,...,X_n,A_1,...,A_m$. Ступени получаются путем применения к исходным множествам и/или уже имеющимся ступеням операций декартова произведения × и взятия множества всех подмножеств $\beta(\cdot)$. А именно:

1. По определению, X_i − ступень при любом $1 \le i \le n$. Аналогично, A_j − ступень при любом $1 \le j \le m$.
2. Если S − ступень, то и $\beta(S)$ − ступень.
3. Если S и S' − ступени, то и $S \times S'$ − ступень.
4. Других ступеней нет.

Схема S_k фиксирует исходные множества и порядок применения к ним двух указанных выше операций.

Основные и вспомогательные базисные множества играют разную роль в построениях родов структур (например, в построениях данной работы вспомогательные множества вообще не используются). Основные базисные множества должны быть обозначены разными буквами. На обозначения вспомогательных множеств таких ограничений не накладывается. Соотношения

$$\sigma_1 \subset S_1(X_1,...,X_n,A_1,...,A_m),\ldots,\sigma_r \subset S_r(X_1,...,X_n,A_1,...,A_m)$$

называются соотношениями типизации. В род структур, кроме соотношений типизации может входить еще некоторое соотношение

$$R(X_1,...,X_n,\sigma_1,...,\sigma_r,A_1,...,A_m,\xi_1,...,\xi_s).$$

Это соотношение предъявляет к роду структуры некоторые требования. Здесь $\xi_1,...,\xi_s$ − соотношения, касающиеся множеств $A_1,...,A_m$. Вспомогательные множества $A_1,...,A_m$ и соотношения $\xi_1,...,\xi_s$ не преобразуются – отображения рассматриваются только над основными базисными множествами.

Соотношение $R(X_1,...,X_n,\sigma_1,...,\sigma_r,A_1,...,A_m,\xi_1,...,\xi_s)$ называется «аксиомой» данного рода структуры. К нему предъявляется требование: оно должно быть переносимо при биекциях. Это означает, что

$$(f_i; X_i \to X_i', i = 1,2,...,n,) - \text{биекции}) \Rightarrow$$

$$(R(X,\sigma,A,\xi) \Rightarrow R(X'\sigma',A,\xi)), \text{ где } \sigma' = S(f,id_A).$$

Здесь $S(f,id_A)$ – так называемое «распространение» [33, 65] отображений f базисных множеств и тождественное распространение вспомогательных множеств по схеме S.

Род структуры будет обозначаться далее следующим образом:

$$\Sigma(A_1,...,A_n,\xi_1,...,\xi_s) =$$

$$=< X_1,...,X_n; [\sigma_1 \subset S_1(X,A),...,\sigma_r \subset S_r(X,A)]; [R(X,\sigma,A,\xi)] >$$

или более коротко:

$$\Sigma[(A,\xi)] = < X; [\sigma \subset S(X,A)]; [R(X,\sigma,A,\xi)] >.$$

Ломаные скобки «<» и «>» здесь играют роль ограничителя. Необязательные элементы рода структуры, как это принято при записи такого сорта конструкций, поставлены в квадратные скобки. Далее по возможности будет использоваться короткая форма всех соотношений.

Пусть имеются множества (E,τ) и выполняются соотношения $\tau \subset S(E,A)$ и $R(E,\tau,A,\xi)$, т. е. соотношение типизации и аксиома рода структуры $\Sigma[(A,\xi)]$ для множеств (E,τ). Тогда говорят, что объект (E,τ) снабжен структурой τ рода $\Sigma[(A,\xi)]$ или что τ есть структура рода $\Sigma[(A,\xi)]$ на множестве E. Обозначение $\Sigma[(A,\xi)]$ содержит то, что не меняется при переходе от одного $\Sigma[(A,\xi)]$-объекта к другому.

Это означает, что множества $A_1,...,A_m$ и соотношения $\xi_1,...,\xi_s$ являются одними и теми же для всех объектов данного рода структуры.

Тем самым, как уже говорилось, математические объекты делятся на классы. К классу относятся объекты, снабженные структурой данного рода $\Sigma(A,\xi)$.

1.2. Примеры родов структур

Приведем примеры родов структур.

$$MAPS = < X; \sigma \subset X \times X; (\forall x \in X)(\exists ! x' \in X)((x,x') \in \sigma)) > -$$

род структуры графика отображения множества в себя. Далее будет удобно оперировать краткими обозначениями аксиом родов структур. Как правило, такие обозначения будут образовываться следующим образом: обозначение будет начинаться с буквы А, далее будет ставиться обозначение рода структуры, далее в скобках будут ставиться не связанные кванторами буквы, фигурирующие в соотношениях, определяющих род структуры.

Например, обозначение для аксиомы

$$(\forall x \in X)(\exists! x' \in X)((x, x') \in \sigma))$$

рода структуры графика отображения в себя в соответствии с этим правилом имеет вид $AMAPS(X, \sigma)$. MAPS-объект – это пара (E, τ), где $\tau \subset E \times E$ – бинарное отношение, называемое графиком отображения и удовлетворяющее аксиоме $AMAPS(E, \tau)$. В «обычных» математических текстах MAPS-объекты обозначаются как $f : E \to E$. Буква f в этих обозначениях иногда трактуется как график отображения, иногда как само отображение, т. е. пара (E, τ).

$$EQ = < X; \sigma \subset X \times X; AEQ(X, \sigma) > -$$

род структуры отношения эквивалентности. Аксиома $AEQ(X, \sigma)$ требует, чтобы отношение σ на X было рефлексивно, симметрично, транзитивно. Для того чтобы не загромождать изложение сложными формулами, здесь и в некоторых случаях далее аксиомы рода структуры формулируются сло-

вами. EQ-объект есть пара (E,Q), где отношение Q есть отношение эквивалентности.

В «обычных» математических текстах принадлежность пары (x,x') отношению эквивалентности Q принято обозначать $x \approx x'$, что читается как x эквивалентно x'. При этом в обычных математических текстах, как правило, не указывается, о каком отношении Q идет речь. Это должно быть ясно из контекста. Множество $\{x' | (x',x) \in Q\}$ называется классом эквивалентности, соответствующим x и обозначается x_Q. Множество $\{x_Q \mid x \in E\}$ называется фактор - множеством множества E по отношению эквивалентности Q и обозначается E_Q или E/Q.

$$PO = < X; \sigma \subset X \times X; APO(X,\sigma) > -$$

род структуры отношения частичного порядка. Аксиома $APO(X,\sigma)$ требует, чтобы отношение $\sigma \subset X$ было рефлексивно, антисимметрично, транзитивно. PO-объект есть пара (E,τ). В обычных математических текстах принадлежность пары (x,x') отношению частичного порядка τ принято обозначать $(x \le x')$, что читается как x предшествует x' или как x' следует за x. О каком отношении τ идет речь, обычно не указывается. Считается, что это должно быть ясно из контекста. Если к аксиоме APO добавить условие, для всякой пары или $x \le x'$, или $x' \le x$, то получится род структуры LP линейного порядка.

$$\text{GR}=< X;\sigma \subset X\times X\times X : AGR(X,\sigma) > -$$

род структуры абстрактной группы. GR-объект есть пара (E,τ), где $(E,\tau)-$ тернарное отношение на E, удовлетворяющее аксиоме $AGR(E,\tau)$. В «обычных» математических текстах принадлежность тройки (x,y,z) отношению τ обозначается как $x\cdot y=z$ и трактуется как постулирование на X ассоциативной алгебраической операции с нейтральным элементом, обозначаемым обычно через e, относительно которого каждый элемент обратим. Какому именно отношению τ, обычно не указывается.

В этих обозначениях $AGR(E,\tau)$ аксиома есть конъюнкция соотношений

$$(\forall x\in E)(\forall y\in E)(\ x\cdot y\in E),(\forall x\in X)(x\cdot e=x),\ (\forall x\in X)(\exists x^{-1})(x\cdot x^{-1}=e).$$

Род структуры AGR абелевой группы получается из рода структуры GR «прибавлением» к аксиоме AGR условия коммутативности алгебраической операции.

$$\text{FLD}=< X;\sigma \subset X\times X\times X,\tau \subset X\times X\times X; AFLD(X,\sigma,\tau) > -$$

род структуры поля. Аксиома $AFLD(X,\sigma,\tau)$ утверждает, что $Card(X)\geq 2$, что σ и τ являются коммутативными, ассоциативными тернарными отношениями, первое из которых принято записывать аддитивно, второе – мультипликативно. Оба отношения обладают нейтральными элементами. Тернарные отношения связаны формулами дистрибутивности.

Самый важный для данной работы пример рода структуры, – семейство родов структур «модель-компонента» будет приведен ниже, в разд. 3.2.2.

1.3. Изоморфизмы, естественные канонические морфизмы

Определение 1.3.1 Пусть (E,τ) и (E',τ') - $\Sigma[(A,\xi)]$-объекты и $i: E \to E'$ отображение. Если это отображение биективно и $S(i,id_A)(\tau) = \tau'$, то i называется изоморфизмом объекта (E,τ) на объект (E',τ'). Если – $f: E \to E'$ произвольное отображение (не обязательно биекция) и имеет место $S(f,id_A)(\tau) \subset \tau'$, то f называется естественным каноническим морфизмом. Естественные канонические морфизмы для сокращения речи далее будут называться ЕКМ. Из этих определений следует, что всякий изоморфизм является естественным каноническим морфизмом.

Для многих родов структур естественные канонические морфизмы имеют специальные названия, возникшие в соответствующих этим родам структур теориях конкретных математических объектов.

Приведем примеры:

1. Для рода структуры SET ЕКМ − произвольные отображения.
2. Для родов структуры PO, LP, L частичного порядка, линейного порядка, решетки ЕКМ − монотонные неубывающие отображения.

3. Для рода структуры MAPS отображений в себя ЕКМ из объекта $f: E \to E$ в объект $f': E' \to E'$ есть отображение $m: E \to E'$, такое, что $f' \circ m = m' \circ f$.
4. Аналогично определяются ЕКМ для рода структуры MAP.
5. Для родов структур GR, AL абстрактных групп, алгебраических решеток и, вообще, для всюду определенных алгебраических структур ЕКМ являются гомоморфизмы.

1.4. P- и F- редукции и декомпозиции

Определение 1.4.1 Пусть (E, τ) – $\Sigma(A, \xi)$-объект, $\tilde{E}$ – подмножество множества E, $\omega: \tilde{E} \to E$ – каноническая инъекция. $\Sigma(A, \xi)$-объект $(\tilde{E}, \tilde{\tau})$ называется P-редукцией (или P-объектом) объекта (E, τ), если $\omega: \tilde{E} \to E$ является ЕКМ и для любого $\Sigma(A, \xi)$-объекта и любого отображения $g: E \to \tilde{E}$ из того, что $g \circ \omega$ является ЕКМ вытекает, что g является ЕКМ. Подмножество $\tilde{E}$ в этом случае называется P-множеством, объект $(\tilde{E}, \tilde{\tau})$ вместе с названием P-редукция называется также подобъектом объекта (E, τ), структура $\tilde{\tau}$ – подструктурой структуры τ.

Множество P-множеств будет обозначаться $P(E, \tau)$. Также будет обозначаться множество P-редукций или P-объектов, поскольку между P-множествами и P-объектами имеется каноническая биекция.

Определение 1.4.2 Пусть (E, τ) – $\Sigma(A, \xi)$-объект, Q – отношение эквивалентности на E. $\pi: E \to E_Q$ – каноническая проекция. $\Sigma(A, \xi)$-объект

(E_Q, τ_Q) называется F-редукцией (или F-объектом) объекта (E, τ), если $\pi : E \to E_Q$ является ЕКМ и для любого $\Sigma(A, \xi)$-объекта (E', τ') и любого отображения $g : E_Q \to E$ из того, что $\pi \circ g$ является ЕКМ вытекает, что g является ЕКМ. Множество E_Q в этом случае называется F-множеством, объект (E_Q, τ_Q) вместе с названием F-редукция называется также фактор-объектом объекта (E, τ), структура τ_Q называется фактор-структурой структуры τ. Множество F-множеств будет обозначаться $F(E, \tau)$, Также будет обозначаться множество F-редукций.

Понятие «редукция» употреблялось многими математиками. Каждый раз при употреблении этого понятия можно было вывести, каким точным смыслом оно наделялось. Впервые как переход от исходного объекта к «такому же объекту», определенному на подмножестве (фактор-множестве) того множества, на котором определен исходный объект, предложено в [42].

Предложение 1.4.1

Если P-объект (F-объект) $\Sigma(A, \xi)$-объекта (E, τ) на множестве $\tilde{E}$ существует, то он единственен. ■

Данное утверждение кратко формулируется следующим образом: если подструктура существует, то она единственна.

Множества P-объектов (F-объектов) $\Sigma(A, \xi)$-объекта (E, τ) не пусты, поскольку по определению (E, τ) является своим подобъектом (знак $\subset$

означает у нас или принадлежность или равенство), а E_{id_E} отождествляется с E.

Множество $\mathrm{P}(E,\tau)$ снабжено структурой частичного порядка по включению Р-множеств. Точно такая же структура вводится на множестве $\mathrm{F}(E,\tau)$. Структуры частичного порядка, (часто это не только частичный порядок, но и решетка) возникающие на $\mathrm{P}(E,\tau)$ и $\mathrm{F}(E,\tau)$, сохраняются при ЕКМ.

Более точно, справедливо следующее

Предложение 1.4.2

Пусть (E,τ) – $\Sigma(A,\xi)$-объект, $(\tilde{E},\tilde{\tau})$ – его Р-редукция, $f:(E,\tau)\to(E_1,\tau_1)$ – изоморфизм, $\tilde{f}:\tilde{E}\to\tilde{E}_1$ – ограничение f на $\tilde{E}$, $\tilde{\tau}_1 = S(\tilde{f}, id_A)(\tilde{\tau})$. Тогда

$$i_1 = f\circ\omega\circ\tilde{f}^{-1}:\tilde{E}_1\to E_1 - \text{ЕКМ. }\blacksquare$$

Аналогичное утверждение имеет место для F-редукций. Эти утверждения говорят о том, что структура частичного порядка «сохраняется» при изоморфизмах.

Определение 1.4.3

Возможен случай, когда имеется набор непересекающихся Р-редукций, дающих в совокупности (при объединении основных базисных множеств), весь исходный объект. Этот случай есть Р-декомпозиция исходного объекта.

Определение 1.4.4

F-декомпозиция объекта – это набор непересекающихся F-редукций, который в совокупности (при декартовом произведении основных базисных множеств), дает весь математический объект.

1.5. Редукции высших уровней

Предложение 1.5.1

P-редукция P-редукции есть P-редукция исходного объекта. F-редукция F-редукции есть F-редукция исходного объекта. ■

Определение 1.5.1

F-редукция P-редукции $\Sigma[(A,\xi)]$-объекта будет называться его PF-редукцией. P-редукция F-редукции $\Sigma[(A,\xi)]$-объекта будет называться его FP-редукцией. PF-редукции и FP-редукции будут называться редукциями второго уровня. Аналогично определяются редукции третьего уровня, к которым, например, относятся PFP-редукции и FPF-редукции, а также редукции более высоких уровней. Совокупность редукций объекта всех уровней будем называть множеством его редукций.

Глава II. Распределенное моделирование сложных систем: проблемы и решения

2.1. Проблемы имитационного моделирования сложных систем

Коль скоро мы заявили, что предметом этой работы 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.

2.1.1. Управление временем в имитационных системах

Простейший способ такого взаимодействия – дискретная модель с постоянным шагом моделирования. 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, характерным 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 работа – 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 мо-

дельный 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: [4 – 7], [10], [43], [61], и в дальнейшем обратим внимание на то, как они решаются в приводимых ниже примерах.

2.1.2 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 задачи. 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 модели, 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.

2.2. Решения. Примеры инструментальных средств моделирования

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 Neighborhoods – разработка Университета Джорджа Мейсона, Вирджиния), SeSAm (**Sh**e**ll** 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 работе 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 [37].

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. Опи-

сания инструментальных систем моделирования приводятся в основном в порядке появления этих систем на свет, все они основаны на публикациях, ссылки на которые приводятся.

2.2.1. 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 [90]. 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 [49], где классический язык GPSS погружен в современную среду программирования, например, Delphi. В общем, проект GPSS продолжает активно жить, включается в вузовские курсы моделирования [81], 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 со-

мнения: 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.

2.2.2. Инструментальная система MISS (Multilingual Instrumental Simulation System)

Многоязыковая инструментальная система имитационного моделирования MISS была создана в конце 80-х гг. в ВЦ АН СССР [30]. 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 [30] 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 сохране-

ния 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:

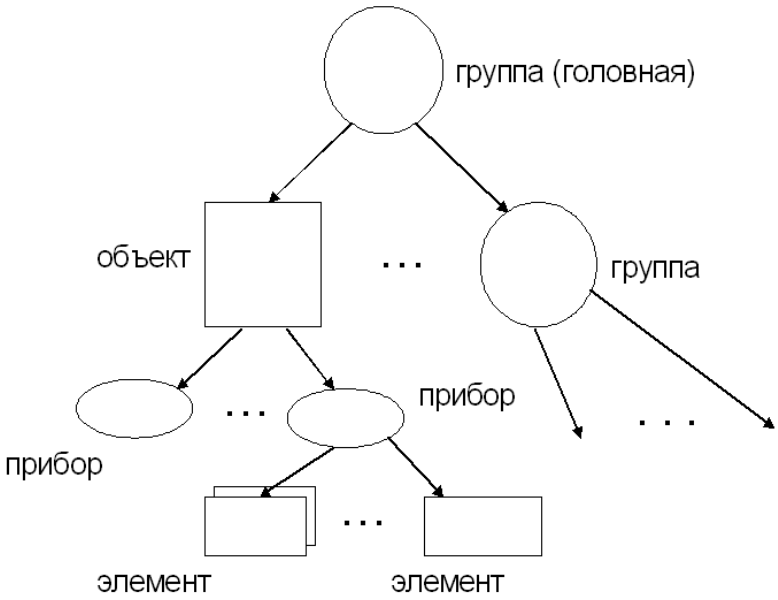


Схема иерархического строения модели.

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 наступившее 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 [34 – 35], весьма похожий механизм впоследствии вошел в спецификацию HLA [10]. 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.

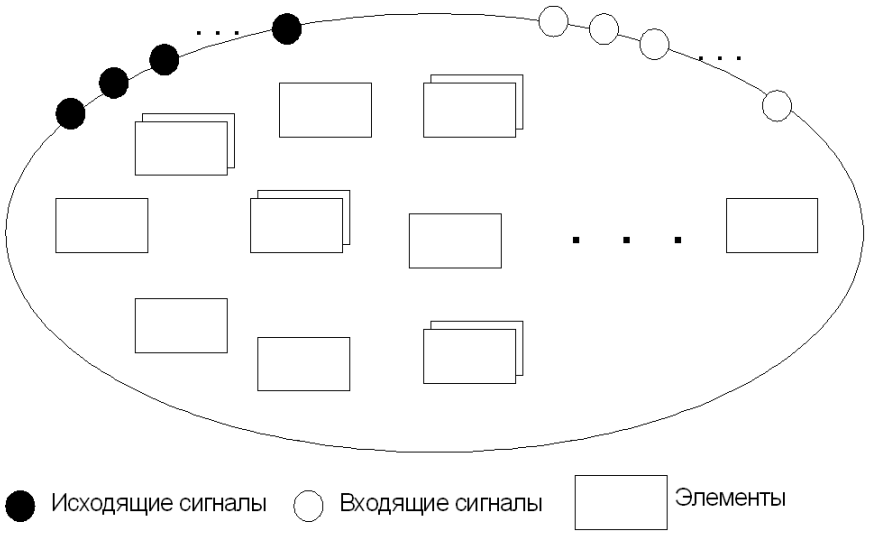


Схема структуры прибора.

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.

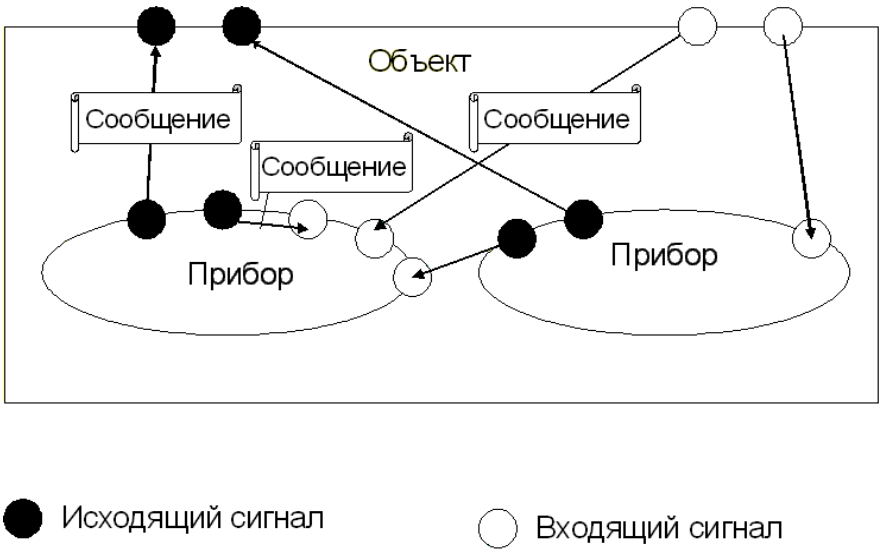


Схема коммутации сигналов на уровне объекта.

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 комму-

тации 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.

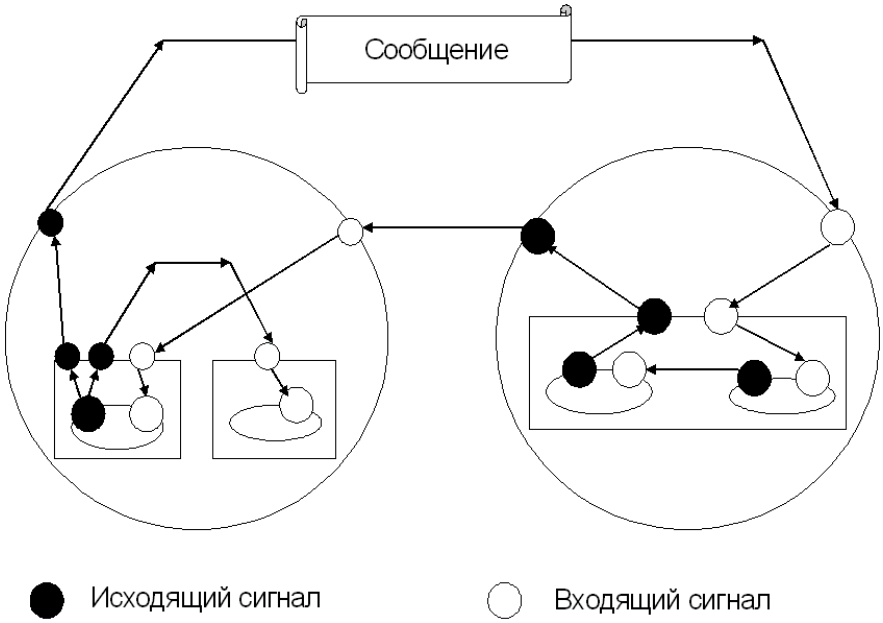
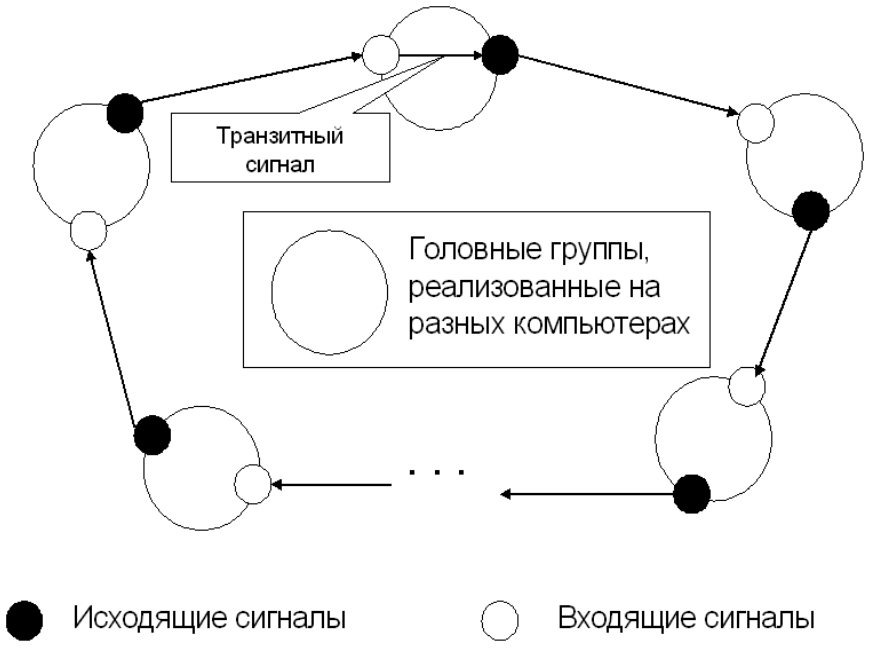


Схема коммутации сигналов на уровне группы.

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.



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, ко-

торую 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-распреде-ленными 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 эле-

мент, 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.

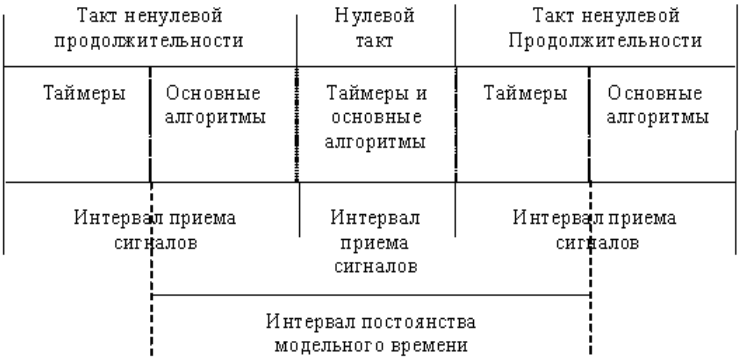
Итак, выбор очередного элемента осуществляется по правилу:

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 опера-

ции, в том числе - пересчет времени, обновление поля сигналов и определение переключений по автоматным функциям.



Принципы синхронизации процессов.

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

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]

[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.

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 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.

2.2.3. Унифицированный язык моделирования UML (Unified Modeling Language)

Унифицированный язык моделирования UML предназначен для описания и документирования объектно-ориентированных систем и бизнес-процессов, с ориентацией создаваемых с его помощью описаний, на последующую их реализацию в виде программного обеспечения. Язык возник в середине 90-х годов XX века, когда трое ведущих специалистов в области объектного анализа, Гради Буч, Джеймс Рамбо и Ивар Якобсон объединили свои усилия для создания инструмента для объектно-ориентированного проектирования и моделирования.

О языке UML - унифицированном языке моделирования сложных программных систем достаточно много и подробно будет сказано в посвященном сравнению концепций модельного синтеза и объектного анализа

разделе 3.4 следующей главы. Здесь мы приведем лишь основные его характеристики.

Анализ и синтез – необходимые этапы проектирования и реализации сложных систем и их имитационных моделей. Анализ позволяет декомпозировать сложную систему на достаточно простые для последующего исследования составляющие, а с помощью синтеза удается собрать из этих составляющих цельную систему. Процессы анализа и синтеза сложных систем и их моделей, вообще говоря, не формализуются, т.е., относятся скорее к искусству, нежели к науке. Тем не менее, имеются средства, призванные облегчить анализ и синтез сложных систем. Одним из главных средств является объектно-ориентированный подход (ООП) или объектный анализ [36, 84]. В рамках этого подхода, результаты анализа системы можно выразить иерархией классов, воплощающей в программном коде методов ее базовые идеи.

Для описания последующего синтеза сложной программной системы из выработанного в результате объектного анализа множества листовых классов, был предложен UML - унифицированный язык моделирования. Язык UML предназначен для описания и документирования объектно-ориентированных систем и бизнес-процессов, с ориентацией создаваемых с его помощью описаний, на последующую их реализацию в виде программного обеспечения.

На наш взгляд проблема здесь в том, что в объектном анализе, реализованном в известных языках программирования высокого уровня, нет должной гармонии между анализом и синтезом – очень сильно перевешивает именно момент анализа. Что касается анализа, – в ООП хорошо развиты методы декомпозиции типа «факторизация», когда базовая идея корневого класса последовательно конкретизируется путем наследования и переопределения методов, пока не воплощается программный код методов множества листовых классов, порожденных корневым. При этом, совсем не развита декомпозиция системы на подобъекты, например, когда «багаж» содержит «диван, чемодан, саквояж, картину, корзину, картонку и маленькую собачонку», которые не выводятся путем наследования ни друг из друга, ни из багажа.

Общая теория декомпозиции математических объектов – геометрическая теория декомпозиции [65], [74], утверждает, что есть два основных типа декомпозиции: F-декомпозиция – декомпозиция факторизации, и P-декомпозиция – декомпозиция разложения на подобъекты. Любая декомпозиция любого математического объекта есть некоторое сочетание этих двух видов декомпозиции. С этой точки зрения набор инструментов объектного анализа, реализованный в объектно-ориентированных языках программирования в виде наследования и переопределения методов, функционально неполон – при наличии средств F-декомпозиции, ему не хватает средств P-декомпозиции.

Кроме того, в известных объектно-ориентированных языках программирования высокого уровня вообще нет никаких средств для синтеза программной системы из листовых классов – это оставляется на усмотрение и искусство разработчика. Средства описания синтеза системы из объектов появляются в языке UML [36], [84]. Однако по вопросу как строить такой синтез с помощью UML, единства мнений нет. Спектр мнений здесь простирается от необходимости компиляции описаний UML в языки программирования высокого уровня, до концепции разработки программных систем, управляемая моделями, известной в различных вариантах под названиями MDA (Model Driven Architecture), MDE (Model Driven Engineering) или MDD (Model Driven Development). Тем не менее, во всех этих подходах не предлагается никакой универсальной методики осуществления синтеза – этот вопрос остается в области искусства.

2.2.4. 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 отдель-

но 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 [10], [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 назы-

вают «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 моделиро-

вании 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 [4 – 7], [10], [43], [61], 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, которым посвящена эта работа, а вовсе не для всего на свете, что может встретиться в моделировании, и что может совместно работать в сети.

2.2.5. Инструментальная система моделирования AnyLogic

Инструментальная система имитационного моделирования AnyLogic – уникальный пример отечественной и притом коммерчески успешной разработки на рынке профессиональных средств моделирования [23], [64]. 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 [88] 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 моде-

ли 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 начинаю-

щему, 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!

2.2.6. 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.

2.3. 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 [51, 52], 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 [4 – 7], [43], [61]?
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 разделах.

Глава III. Модельный синтез и модельно-ориентированное программирование

3.1 Математические модели. Внутренние и внешние характеристики. Гипотеза о замкнутости

Основная задача раздела – дать содержательное пояснение с позиций математического и имитационного моделирования, формальной конструкции, семейству родов структур «модель-компонента», которая будет определена в следующем разделе.

Прежде чем перейти к строгому определению имитационной модели сложной системы как рода структуры, остановимся на таких важных понятиях, связанных с математическим и имитационным моделированием, как замкнутость модели и ее внутренние и внешние характеристики. Поскольку процесс построения математических моделей различных явлений не формализован и в значительной мере остается искусством, обсуждение будет вестись также на не слишком формальном уровне.

3.1.1 Математические модели

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. Характеристики изучаемого явления изменяются со временем, однако в этих изменениях заключается нечто постоянное – закономерность их связи между собой, которая и является предметом выявления и изучения путем математического моделирования. Численное из-

мерение характеристик изучаемого явления задает некоторое их отображение во множество характеристик будущей математической модели. Математическая модель была бы идеальной, если бы это отображение сохраняло связи между характеристиками явления и модели – некий аналог рода структуры.

Однако это мечты сохранить «то – не знаю что», ведь закономерности изменения характеристик явления как раз и есть предмет изучения с помощью математического моделирования. Лучшее, что можно сделать – это придумать такие соотношения между характеристиками модели, которые бы в максимальной степени уподобляли бы их изменения изменениям их прообразов – характеристик изучаемого явления. Правда, к сожалению, такое уподобление – лишь необходимое, а не достаточное условие построения адекватной модели.

В самом деле, если бы мы располагали закономерностью связи характеристик, как инвариантом преобразования характеристик явления в характеристики модели – мы, несомненно, были бы способны давать любые прогнозы динамики этого явления. С другой стороны, располагая даже очень точной, многократно проверенной на практике математической моделью явления, нельзя быть уверенным, что однажды не появится необходимость рассмотрения некоторых не учитывавшихся ранее его характеристик. Учет новых характеристик может потребовать совсем других моделей для построения адекватного прогноза развития явления. Например, с

помощью геоцентрической модели движения планет Птолемея, которой человечество успешно пользовалось более полутора тысяч лет, до сих пор можно весьма точно рассчитать взаимное расположение планет на небосводе или даты наступления лунных и солнечных затмений. Однако запустить с ее помощью ракету на Луну, было бы, по-видимому, проблематично.

3.1.2 Внутренние и внешние характеристики

Предположим, что мы измеряем и изучаем численные характеристики $X_1, X_2, ..., X_n$, нашего явления. Поскольку ни одно явление невозможно полностью изъять из окружающего мира, в наше поле зрения неизбежно попадает некий набор характеристик $a_1, a_2, ..., a_m$ внешнего по отношению к изучаемому явлению мира, так или иначе связанных с характеристиками изучаемого явления. Характеристики внешнего мира, совсем не связанные с нашим явлением, в рамках его изучения мы вправе игнорировать. Отметим, что граница между характеристиками явления и внешнего по отношению к нему мира достаточно размыта, подвижна, условна, – обычно исследователь сам определяет, что включать, а что нет в изучаемое им явление. Остановимся подробнее на возможных типах зависимостей между характеристиками X_i и a_j.

1. Предположим, что некоторое подмножество $\tilde{A} \subset \{a_j\}_{j=1}^{m}$ характеристик внешнего мира, во-первых, влияет на характеристики $\{X_i\}_{i=1}^{n}$

изучаемого явления и, во-вторых, зависит от этих характеристик. Предлагается включить такие характеристики $\tilde{A}$ во множество характеристик изучаемого явления и исключить из рассматриваемых характеристик внешнего мира, тем самым расширив нашу модель, ибо такова структура взаимосвязей между ее характеристиками.

2. Предположим, что некоторое подмножество $A' \subset \{a_j\}_{j=1}^{m}$ характеристик внешнего мира не влияет на характеристики $\{X_i\}_{i=1}^{n}$ изучаемого явления, но зависит от этих характеристик. Предлагается совсем не рассматривать такие характеристики A', поскольку изучение влияния нашего явления на внешний по отношению к нему мир, хотя вполне может быть содержательной задачей, но это самостоятельная задача, отличная от провозглашенной нами программы исследования зависимостей между характеристиками изучаемого явления.
3. Наконец остается последнее предположение: что подмножество $\overline{A} \subset \{a_j\}_{j=1}^{m}$ характеристик внешнего мира влияет на характеристики $\{X_i\}_{i=1}^{n}$ изучаемого явления, но не зависит от этих характеристик. Только такие характеристики внешнего мира мы и будем рассматривать в дальнейшем, называя их внешними характеристиками явления и его моделей. Характеристики $\{X_i\}_{i=1}^{n}$ будем называть внутренними характеристиками.

Внутренние характеристики зависят друг от друга и от внешних характеристик. Внешние характеристики влияют на внутренние, но не зависят от них. Иногда утверждение о независимости внешних характеристик от внутренних называют гипотезой об инвариантности.

Приведенные выше рассуждения показывают, что принятие такой гипотезы не уменьшает общности рассмотрения, – просто те внешние характеристики, которые зависят от внутренних, удобно включать в модель и считать дополнительными внутренними характеристиками.

3.1.3 Гипотеза о замкнутости

Приступая к построению математической модели изучаемого явления, хотелось бы иметь уверенность в том, что такое построение возможно. В «Максимах и отражениях» И.В. Гете писал: «Человек должен непреложно верить, что непостижимое на самом деле постижимо; иначе бы он прекратил исследования». Гипотеза о замкнутости как раз и призвана дать исследователю такую уверенность.

Гипотеза о замкнутости модели постулирует, что знания значений внутренних и внешних характеристик модели в момент t достаточно для вычисления ее внутренних характеристик на некотором интервале $(t, t+\Delta t)$. Таким образом, внутренние характеристики, с одной стороны, определяют состояние модели, а с другой стороны, могут прогнозироваться с помощью этой модели на некотором интервале времени. Внешние характеристики не моделируются, но влияют на модель. Поэтому их можно считать доступ-

ными для восприятия модели характеристиками внешнего по отношению к ней мира (в том числе, например, внешними управляющими воздействиями). Обычно предполагается, что внешние характеристики доступны для измерения в любой момент модельного времени. Иногда предположение о наблюдаемости внешних характеристик включают в гипотезу о замкнутости. В данной работе, упоминая гипотезу о замкнутости, всегда будем считать, что предположение о наблюдаемости внешних характеристик включается в нее.

Из гипотезы о замкнутости следует, что если теперь нас интересует изменение выбранных нами внутренних характеристик, например во времени, то ему просто не от чего больше зависеть, кроме как от самих этих характеристик (в виртуальном мире, образуемом нашей замкнутой моделью, больше просто ничего нет).

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

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

Правда правая часть этой системы дифференциальных уравнений неизвестна, ее еще предстоит придумать, это и будет созданием математической модели.

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

3.1.4 Гипотеза о замкнутости применительно к моделям сложных многокомпонентных систем

Как и прежде, будем считать, что в любой момент времени мы умеем узнавать значения внешних характеристик модели, и предполагать, что знания значений внутренних и внешних характеристик модели в момент t достаточно для вычисления ее внутренних характеристик на некотором интервале $(t, t+\Delta t)$.

Для моделей сложных систем предположение о непрерывной зависимости характеристик от времени не является естественным. Очень часто здесь зависимости некоторых характеристик от времени явно носят дискретный характер. Кроме того, все вычисления характеристик модели осуществляются, как правило, с помощью компьютера. Попробуем обсудить возможности компьютерного построения моделей сложных многокомпонентных систем.

Отметим, что реализуя имитационные вычисления на компьютере, за конечное время мы сможем обработать лишь конечное число разрывов первого рода. Поэтому будем искать траекторию нашей модели $\vec{X}(t)$ в классе кусочно-гладких по t функций.

Далее, поскольку шаг моделирования Δt выбирается так, что на протяжении этого шага изменения модели не слишком заметны, будем всегда

относить разрыв первого рода на начало шага, а далее на протяжении Δt будем считать траекторию модели $\vec{X}(t)$ гладкой. Траекторию модели будем вычислять с некоторой заданной точностью $\varepsilon > 0$. Например, если траектория модели определяется дифференциальным уравнением $\dot{X} = F(X, a)$, то при малых $\tau > 0$ справедливо

$$X(t+\tau) \approx X(t) + F(X(t), a(t))\tau .$$

Попытаемся теперь построить траекторию модели $\vec{X}(t)$ на некотором макроскопическом отрезке модельного времени $[0, T]$, попутно выясняя, в каких предположениях может быть успешным такое построение.

Определение 3.1.1

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:

1. На основании внутренних и внешних характеристик модели $\vec{X}(t)$ и $\vec{a}(t)$ можно определить, есть ли в точке t разрыв траектории $\Delta\vec{X}(t)$, и если он есть, – вычислить его.
2. Далее, на полуинтервале $(t, t + \Delta t]$, траектория модели, выходящая из точки $\vec{X}(t) + \Delta\vec{X}(t)$, является гладкой функцией времени, например, решением системы дифференциальных уравнений $\dot{X} = F_t(X, a)$.

Определение 3.1.2

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

Определение 3.1.3

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.

Очевидно, если модель прогнозируема в смысле последнего определения и удалось найти соответствующее разбиение отрезка $[0,T]$ точками $0 = t_0 < t_1 <,...,< t_n = T$, то задачу построения модели можно считать в принципе решенной. С другой стороны, если не выполнены условия определения 3.1.2., т. е. существуют точки $t \in [0,T)$, откуда невозможно сделать даже малый шаг на $\Delta t > 0$, то задача построения модели представляется безнадежной. Поэтому требование локальной замкнутости на отрезке будет одним из базовых требований, предъявляемых к модели.

Из локальной замкнутости на отрезке, прогнозируемость на этом отрезке, вообще говоря, не следует.

Контрпримером является модель, описываемая условиями задачи, встречающейся 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 задаче спрашивается, сколько пролетит муха до момента встречи пешеходов.)[1]

Здесь всей истории модели недостаточно для определения скорости мухи в момент встречи пешеходов. Действительно, скорость мухи разрывна, 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.

Предложение 3.1.1

Сделать локально замкнутую на отрезке $[0,T]$ модель прогнозируемой на этом отрезке может добавление требования непрерывности слева траектории модели $\Delta\vec{X}(t)$ в любой точке $t \in (0,T]$.

Действительно, начнем продвигаться по оси времени из исходной точки 0 в соответствии с определением замкнутости в точке. Если за конечное число шагов мы не доходим до точки T, это означает, что существует точка накопления $\tilde{t} \in (0,T)$. В силу непрерывности траектории мо-

[1] Иногда эту задачу связывают с именем Джона фон Неймана, – якобы он мгновенно выдал ее ответ, сказав при этом, что просуммировал в уме бесконечный ряд. Этот вариант задачи известен как «муха фон Неймана», в нем вместо пешеходов фигурируют поезда, которые в момент встречи сталкиваются, и в катастрофе гибнет все, в том числе и муха. Поэтому в модели «муха фон Неймана» никакое продолжение ее истории за момент встречи поездов невозможно.

дели слева существует предел слева $\tilde{X} = \lim_{t \to \tilde{t}-0} X(t)$ в точке накопления. При этом для выбранной нами точности вычислений $\varepsilon > 0$ найдется такое $\tilde{t} > \delta > 0$, что для всех $t \in (\tilde{t} - \delta, \tilde{t})$ справедливо $\varepsilon > \left| \tilde{X} - X(t) \right|$. До точки $\tilde{t} - \delta$ мы дойдем за конечное число шагов, иначе она, а не $\tilde{t}$ была бы точкой накопления. На интервале $(\tilde{t} - \delta, \tilde{t})$ положим $X(t) = \tilde{X}$, и, таким образом, дойдем за конечное число шагов до точки $\tilde{t}$, после чего в соответствии с локальной замкнутостью модели на отрезке $[0, T]$ можем продолжить путь дальше, на ненулевой шаг, что противоречит нашему предположению о том, что $\tilde{t}$ – точка накопления. ■

Как мы видим, непрерывность слева является достаточным условием, чтобы локально замкнутая на отрезке модель стала прогнозируемой на этом отрезке. Из определений замкнутости в точке и прогнозируемости модели на отрезке следует также, что непрерывность слева есть и необходимое условие прогнозируемости (возможно, после переопределения траектории модели в конечном числе точек).

Отметим, что приведенное выше рассуждение не просто провозглашает возможность пройти отрезок моделирования за конечное число шагов (при условии непрерывности траектории слева), а дает конструктивный способ, как это сделать. Нужно начинать с левого конца отрезка и «перешагивать» встречающиеся точки накопления – те отрезки, где траектория

системы с выбранной точностью моделирования $\varepsilon > 0$, остается постоянной.

3.1.5 Особенности моделирования сложных многокомпонентных систем

Особенностью многокомпонентных «атомистических» систем является то, что их атомы внутри себя связаны гораздо сильнее, чем между собою. Большую часть времени каждый атом живет сам по себе в соответствии со своим устройством, и лишь время от времени атомы взаимодействуют между собою. Часто такое взаимодействие бывает мгновенным (относительно выбранного в модели масштаба времени) и, следовательно, вызывает разрывность характеристик модели.

Такие особенности устройства многокомпонентных систем дают шанс построения синтеза в духе приводившейся во введении цитаты из Н.П. Бусленко: «…дать компонентам возможность в полной мере проявить себя». А именно в начале шага моделирования обсчитать все межкомпонентные взаимодействия, а затем в течение некоторого Δt до следующего взаимодействия компоненты проявляют себя независимо друг от друга и, скорее всего, такое проявление описывается соответствующими системами дифференциальных уравнений. Поэтому представление траектории системы кусочно-гладкой функцией представляется вполне уместным.

Моделировать сложную систему чаще всего приходится с переменным шагом времени. Очень часто приходится уменьшать стандартный шаг моделирования до ближайшей точки синхронизации компонент. В связи с

этим возникает вопрос, а не зациклится ли модель, как это происходит с упомянутой выше моделью «пешеходы и муха». Другими словами, дойдем ли мы до правого конца отрезка моделирования? Как следует из предыдущего раздела, для успеха этого предприятия кроме достаточно обычного в моделировании требования локальной замкнутости нужно потребовать непрерывности слева траектории системы в любой точке.

Содержательно это означает, что для успеха моделирования мало уметь сделать хотя бы небольшой шаг вперед из любой точки – еще обязательно нужно, чтобы всякое значение траектории модели, кроме начального, было обусловлено некоторой предысторией.

Требование непрерывности слева как достаточного условия возможности построения модели на самом деле близко к условию необходимому. То, что мы в состоянии создать компьютерную реализацию имитационной модели всегда означает, что мы проходим весь рассматриваемый отрезок модельного времени за конечное число шагов, поэтому и число разрывов первого рода траектории модели, которые мы можем распознать и учесть, конечно. В промежутках между разрывами чаще всего используется линейное приближение траектории, вычисляемое по левому концу интервала на основании гипотезы о замкнутости. Подобное необходимое условие возможности построения модели и было зафиксировано в определении 3.1.3 прогнозируемой, или лапласовской модели. Кусочно-гладкую траек-

торию модели всегда можно считать непрерывной слева, относя все разрывы первого рода к началу, а не к концу интервалов непрерывности.

Прогнозируемая модель была названа в соответствующем определении также и лапласовской, потому что для нее оказывается существенным не только прогноз – взгляд в будущее из любой точки, но и обусловленность – взгляд в прошлое, где текущее состояние есть результат некоторых существовавших ранее и осуществившихся в нем причин, его определивших. На этот счет у П.С. Лапласа [51] имеется следующее высказывание: «Современные события имеют с событиями предшествующими связь, основанную на очевидном принципе, что никакой предмет не может начать быть без причины, которая его произвела...».

Кроме того, прогнозируемая на отрезке модель, по-видимому, являет собой образец лапласовского детерминизма. Во всяком случае, именно в прогнозируемой на отрезке имитационной модели в смысле определения 3.1.3 воплощается мечта Лапласа, высказанная им в работе [52]:

«Ум, которому были бы известны для какого-либо данного момента все силы, проявляющиеся в природе, и относительное положение всех ее составных частей (если бы вдобавок этот ум оказался достаточно обширным, чтобы подчинить эти данные анализу), – обнял бы в одной формуле движения величайших тел вселенной наравне с движениями легчайших атомов: не осталось бы ничего, что было бы для него недостоверно, и будущее так же, как и прошедшее, предстало бы перед его взором».

В заключение раздела скажем несколько слов о детерминированности и однозначности вычислительного процесса имитационного моделирования. Детерминированность и однозначность вычислительного процесса соотносится с гипотезой о замкнутости примерно так же, как соотносятся конструктивные и неконструктивные теоремы о существовании. 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 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 (собственно, 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 вычисления 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 быть и вероятностным.

Заметим также, что высказанное выше требование детерминированности и однозначности вычислительного процесса есть некий идеал, к которому следует стремиться при создании моделей. Реально в достаточно сложных системах, например в известных операционных системах, эксплуатирующихся десятилетиями, время от времени находятся уязвимости – с точки зрения моделирования, недокументированные цепочки внешних характеристик, 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.

Далее, 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 разумным и детерминированным образом, например, при выборе единственного значения характеристики, полученной несколькими способами.

3.1.6 Основные выводы из гипотезы о замкнутости

Возможно, все высказанные в данном разделе соображения и предложения покажутся простыми и даже тривиальными. Тем не менее, из них вытекает ряд весьма важных для дальнейшего изложения следствий, которые будут перечислены далее:

1. Коль скоро мы признаем на каждом отрезке прогноза $[t, t+\Delta t]$ функциональную зависимость от значений характеристик модели в момент t как величины возможного скачка траектории в момент t, так и последующей ее гладкой динамики на $(t, t+\Delta t]$, – естественно признать и возможность вычисления этих функциональных зависимостей в функциональной парадигме программирования.
2. Функциональная зависимость однозначна по определению. Отсюда следует, что если по каким-то соображениям (например, проистекающим из предметной области моделирования, или из способа организации вычислительного процесса), нам представляется удобным декомпозировать эту зависимость на ряд параллельно выполняемых

вычислений, то эта декомпозиция не должна нарушать упомянутой однозначности. Таким образом, если два параллельных процесса пытаются одновременно модифицировать одну и ту же характеристику модели – это есть просто ошибка ее проектировщика – нарушение однозначности функциональной зависимости (а отнюдь не отражение объективной сложности модели или моделируемой ею системы). Стало быть, декомпозированные на параллельные процессы вычисления функциональной зависимости, не эквивалентны исходным. Сохранение однозначности вычислений – необходимое условие такой эквивалентности.

3. Из определения 3.1.3. прогнозируемой или лапласовской модели следует, что процесс вычисления ее траектории естественным образом декомпозируется на чередующиеся вычисления «скачков» и интервалов гладкой динамики. Траекторию модели можно считать полунепрерывной слева. Разрывов траектории модели первого рода может быть лишь конечное число, они происходят мгновенно в модельном времени и зависят только от предела слева характеристик модели в момент разрыва. На интервалах гладкой динамики внутренние характеристики модели непрерывно зависят от ее характеристик на левом конце интервала и от времени прошедшего от начала интервала. Именно эта декомпозиция и определяет предлагаемые в

следующем разделе правила организации имитационных вычислений (аксиому поведения).

На сформулированные выше выводы мы будем существенно опираться в последующих разделах работы.

3.2 Модельный синтез – концепция описания и реализации имитационных моделей сложных многокомпонентных систем

Для облегчения понимания довольно громоздких формальных конструкций разд. 3.2, дадим сначала неформальное описание двух самых важных понятий модельного анализа: основного понятия – «модель-компонента» и вспомогательного понятия – «модель-комплекс».

3.2.1 Неформальное описание моделей-компонент и моделей-комплексов

Сначала опишем устройство модели-компоненты.

Характеристики

Модель-компонента, подобно объекту объектного анализа имеет характеристики. Эти характеристики мы будем разбивать на внутренние и внешние. Внутренние характеристики – это то, что модель-компонента моделирует, внешние – это то, что она знает о внешнем мире.

Процессы

Функциональность модели-компоненты удобно структурировать следующим образом. Считается, что модель-компонента реализует один или несколько параллельно выполняющихся процессов. Процесс состоит в последовательном чередовании элементов – алгоритмически элементарных методов. Если какой-либо процесс какую-то часть модельного времени не

выполняется – удобно считать, что в это время он выполняет «пустой» элемент, не меняющий никаких характеристик модели-компоненты.

Элементы

Элементарные, алгоритмически однородные методы, реализующие функциональности модели-компоненты (т.е. то, что компонента умеет делать, получение на основании значений некоторых внутренних и внешних характеристик модели-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 характеристики системы, имеющие разрывы первого рода, или принимающие только целочисленные значения.

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 эле-

менты 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 внутренними характеристиками модели: каждый метод имеет право изменять только «свои» характеристики. 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 модели-компоненты на просторах интернета, по-

этому его параметры таковы, какими их сделал его автор, и, скорее всего, никак не согласованы с характеристиками модели-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.

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 характеристик, на которые обязаны отреагировать некоторые процессы модели-компоненты.

Формально событие – функция значений внутренних и внешних характеристик в начале шага моделирования. С точки зрения организации имитационных вычислений, событие – это метод, входными параметрами которого является подмножество внутренних и внешних характеристик модели-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 $E_{\{A,A\}}$, прерывающие выполнение элемента A, например, если его еще не нужно заканчивать, но он вычислил характеристики модели-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 $\{A, D\}$.

Возможно тем, кто знаком с архитектурой компьютера или с событийно-ориентированными языками программирования такое определение событий покажется странным, и вызовет вопросы типа, а где же здесь что-нибудь подобное обработчикам событий? Дело в том, что в моделировании (если, конечно не брать моделирование полунатурное, когда к компьютерному комплексу подключают реальное изделие «в железе») чаще всего наступление события должна определить сама модель – даже если это событие неожиданное, например, случайное, – все равно, в нужный момент модель вполне детерминированно должна запустить генератор случайных чисел, чтобы выяснить, не произошло ли оно. А подобный порядок действий вполне адекватно ложится на изложенную выше концепцию событий.

Выполнение модели-компоненты

Модель-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 модели-компоненты в моменты событий.

Правила запуска модели-компоненты на выполнение следующие. 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. **Все эти события можно вычислять параллельно.** Если есть наступившие события, проверяется, нет ли переходов к быстрым (сосредоточенным) элементам, если они есть – **параллельно** 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, $\Delta t \le \Delta\tau$ – **параллельно** мо-

делируем текущие распределенные элементы со стандартным шагом Δt. В противном случае – **параллельно** моделируем их с шагом времени до ближайшего спрогнозированного события $\Delta\tau$.

4. Возвращаемся к началу п.1.

Все события и элементы есть функции в математическом смысле и в смысле парадигмы функционального программирования, т.е. не имеют состояний и побочных эффектов, что может дать при их вычислении дополнительные возможности распараллеливания.

В связи с приведенными правилами выполнения модели-компоненты могут возникнуть вопросы: не может ли выполнение быстрых элементов в п. 1, а также уменьшение шага времени в п. 3 привести к зацикливанию программы за счет возникновения точек накопления системных событий. Полную гарантию, как было показано в разд. 2, можно дать лишь для кусочно-гладких непрерывных слева систем.

Еще заметим, что раз мы требовали от элементов однозначности имитационных вычислений и сумели этого добиться – на выполнение все одновременно выполняющиеся в модельном времени элементы можно запускать асинхронно, т.е. загружать ими имеющиеся ядра процессора или же доступные распределенные компьютеры. Если же этого почему-то добиться не удалось – у системы поддержки выполнения модели есть полная информация о том, кто что меняет (собственно, это она должна будет по выполнении элементов обновить соответствующие данные в базе) – и она

обязана выдать соответствующую диагностику ошибки времени выполнения.

Модели-комплексы

Модели-компоненты могут объединяться в модели-комплексы, при этом (необязательно) может оказаться, что некоторые компоненты явно моделируют внешние характеристики некоторых других компонент. Для того чтобы полностью описать модель-комплекс, достаточно указать:

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

При объединении компонент в комплекс, следует иметь в виду, что однозначность вычислительного процесса может быть потеряна. Так может произойти, если по каким-то причинам, несколько компонент вычисляют одну и ту же характеристику моделируемого явления. В таком случае можно ввести в модель-комплекс новую компоненту, которая в качестве внешних характеристик получает весь многозначный набор значений упомянутой характеристики, а в качестве внутренней характеристики каким-то образом вычисляет единственное ее значение.

Комплекс как модель-компонента

Модель-комплекс, состоящий из многих моделей-компонент, вовне может проявляться в качестве единой модели-компоненты.

Введем следующую операцию объединения компонент комплекса:

1. Внутренними характеристиками комплекса объявляется объединение внутренних характеристик 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 характеристиками комплекса объявляется объединение всех внешних характеристик его компонент, из которого исключаются все те характеристики, которые моделируются явно какими-либо компонентами.

Операция объединения превращает модель-комплекс в модель-компоненту. Этот факт позволяет строить модель как фрактальную конструкцию, сложность которой (и соответственно подробность моделирования) ограничивается лишь желанием разработчика.

3.2.2 Семейство родов структур «модель-компонента»

Попробуем формально определить математический объект, представляющий элементарную имитационную модель сложной системы. Предыдущий раздел на неформальном уровне поясняет, почему ее стоит определять именно таким образом.

Введем однопараметрическое семейство родов структур Σ_N «модель-компонента». Параметром N семейства Σ_N является количество процессов модели-компоненты. Что такое процесс – на неформальном уровне было определено в предыдущем разделе. Формально процессы модели-компоненты будет определены ниже соотношениями типизации (10) и аксиомами R_9.

$$\Sigma_N = < X, M, E, \{M_j\}_{j=1}^{N}, \{E_j\}_{j=1}^{N};$$

$$x \subset X,$$

$$a \subset X,$$

$$s \subset M,$$

$$f \subset M,$$

$$\{m_{j,real} \subset M_j \times M\}_{j=1}^{N},$$

$$\{e_{j,real} \subset E_j \times E\}_{j=1}^{N},$$

$$\{m_{j,in} \subset M_j \times \beta(X)\}_{j=1}^{N},$$

$$\left\{m_{j,out} \subset M_j \times \beta(X)\right\}_{j=1}^{N},$$

$$\left\{e_{j,in} \subset E_j \times \beta(X)\right\}_{j=1}^{N},$$

$$\left\{m_j^0 \subset M_j\right\}_{j=1}^{N},$$

$$\left\{sw_j \subset E_j \times M_j \times M_j\right\}_{j=1}^{N},$$

$$\left\{p_j \subset \beta(M_j) \times \beta(E_j) \times M_j \times \beta(E_j \times M_j \times M)\right\}_{j=1}^{N};$$

$$R_1: (x \cup a = X) \,\&\, (x \cap a = \varnothing),$$

$$R_2: (s \cup f = M) \,\&\, (s \cap f = \varnothing),$$

$$R_3: \left\{\left(\forall m \in M_j\right)\left(\exists ! \tilde{m} \in M\right)\left(\{m, \tilde{m}\} \in m_{j,real}\right)\right\}_{j=1}^{N},$$

$$R_4: \left\{\left(\forall e \in E_j\right)\left(\exists ! \tilde{e} \in E\right)\left(\{e, \tilde{e}\} \in e_{j,real}\right)\right\}_{j=1}^{N},$$

$$R_5: \left\{\left(\forall m \in M_j\right)\left(\exists ! r \in \beta(X)\right)\left(\{m, r\} \in m_{j,in}\right)\right\}_{j=1}^{N},$$

$$R_6: \left\{\left(\forall m \in M_j\right)\left(\exists ! r \in \beta(x)\right)\left(\{m, r\} \in m_{j,out}\right)\right\}_{j=1}^{N},$$

$$R_7: \left\{\left(\forall e \in E_j\right)\left(\exists ! r \in \beta(X)\right)\left(\{e, r\} \in e_{j,in}\right)\right\}_{j=1}^{N},$$

$$R_8: \left\{\left(\left(\forall e \in E_j\right)\left(\exists ! r \in M_j \times M_j\right)\left(\{e, r\} \in sw_j\right)\right) \,\&\right.$$

$$\left. \,\&\left(\left(\{e, r\} \in sw_j, \{\tilde{e}, \tilde{r}\} \in sw_j, r = \tilde{r}\right) \Rightarrow \left(e = \tilde{e}\right)\right)\right\}_{j=1}^{N}$$

$$R_9: \left\{p_j = \left\{M_j, E_j, m_j^0, sw_j\right\}\right\}_{j=1}^{N},$$

R_{10}: аксиома однозначности вычисления характеристик модели-компоненты,

R_{11}: аксиома поведения модели-компоненты (организации имитационных вычислений)>.

Обозначение $\{\ldots_j\}_{j=1}^{N}$ используется для краткости и означает, что содержимое скобок повторяется через запятую N раз, при этом индекс j заменяется на $1,\ldots,N$. Например, $\{M_j\}_{j=1}^{N}$ есть краткий вариант записи $M_1,\ldots,M_N$.

Дадим пояснения приведенному выше формальному определению рода структуры «модель-компонента». Оно заключено в угловые скобки «<» и «>» и состоит из трех частей, которые отделяются друг от друга символом «;», а отдельные определения в каждой части разделяются символом «,». В первой части определяются базисные множества. Во второй части перечислены соотношения типизации, которые определяют родовые константы, как части множеств, построенных из базисных специальным образом – возможным последовательным применением операций декартова произведения × и взятия множества всех подмножеств $\beta(\cdot)$. В третьей части перечислены аксиомы рода структуры $R_1 - R_9$, которые устанавливают определенные зависимости между базисными множествами и родовыми константами.

В качестве основных базисных множеств выбраны

$$X, M, E, \{M_j\}_{j=1}^{N}, \{E_j\}_{j=1}^{N}$$

Здесь X – множество характеристик модели, причем иногда будем разделять его на два подмножества: $X = \{x, a\}$, где x – внутренние характеристики модели, то, что в соответствии с гипотезой о замкнутости полностью определяет ее состояние, а a – ее внешние характеристики, то, что в силу той же гипотезы о замкнутости полностью определяет взаимодействие модели с внешним по отношению к ней миром. Первые два из соотношений типизации

$$x \subset X,\ a \subset X, \qquad (1)$$

утверждают, что внутренние характеристики модели x и внешние характеристики a – есть части ее характеристик. Аксиома R_1: $(x \cup a = X)\,\&\,(x \cap a = \varnothing)$ утверждает, что эти части x и a не пересекаются, и в совокупности составляют все множество характеристик модели X.

Далее идет M – множество различных реализаций методов-элементов, элементарных умений нашей модели, среди которых мы также иногда будем выделять два подмножества: $M = \{s, f\}$ – медленных s, реализующих гладкую зависимость внутренних характеристик модели от ее внутренних и внешних характеристик, если например, $\dot{x} = F(x, a)$, то $x(t + \Delta t) \approx x(t) + F(x(t), a(t))\Delta t$, и быстрых f – реализующие скачки внутренних характеристик модели: $\Delta x = G(x(t), a(t))$. Точно так же, как и в

случае с характеристиками, запишем соответствующие соотношения типизации:

$$s \subset M,\ f \subset M, \tag{2}$$

и аксиому: $R_2: (s \cup f = M) \& (s \cap f = \varnothing)$.

Далее, E − множество различных реализаций методов-событий, связанных с моделью. События – это то, на что обязана реагировать наша модель. Метод-событие – функция, которая, получая на входе подмножество характеристик $Y \subset X$, на выходе дает неотрицательное число, означающее, что событие наступило, если число равно нулю, или же прогноз времени до наступления события, если число положительно.

Отметим, что во множествах X, M, E нет одинаковых элементов. Этот факт также утверждается аксиомой рода структуры R_{10}. Для X это следует из того, что речь идет о модели, и принцип неувеличения числа сущностей сверх необходимости (бритва Оккама) побуждает нас избегать ненужного дублирования одних и тех же характеристик как моделируемого явления, так и его связей с внешним миром. Что касается M и E, словами «множество различных реализаций» подчеркивается именно уникальность методов в этих множествах. На самом деле, язык математики хорош именно тем, что иногда позволяет одними и теми же соотношениями записать различные законы, имеющие место в совсем различных предметных областях. Излагаемый здесь подход позволяет учитывать этот факт и пользоваться им. Однако для этого необходимо знать, сколько и каких

различных функциональных зависимостей имеется в нашем распоряжении. Множества M и E как раз и являются такими «хранилищами» различных функциональных зависимостей для модели-компоненты.

Далее будут упоминаться процессы модели-компоненты. Подробное объяснение того, что такое процессы будет дано далее, пока же скажем, что содержательно процессы подобны, например, системным службам операционной системы компьютера. Они работают всегда и при этом одновременно. Будем считать, что число процессов модели – N. Каждый процесс p_j, $j = 1,...,N$, последовательно осуществляет некоторый конечный набор возможных для него элементарных действий M_j, который будем называть множеством его методов-элементов, возможно, в зависимости от возникающих в системе ситуаций E_j, на которые процесс умеет реагировать, их будем называть множеством его методов-событий.

Этим исчерпывается описание основных базисных множеств.

Вспомогательных базисных множеств нет.

Перейдем теперь к соотношениям типизации и аксиомам. Уже упоминавшееся соотношение $x \subset X$ определяет внутренние характеристики модели как подмножество всех ее характеристик.

$$\left\{ m_{j,real} \subset M_j \times M \right\}_{j=1}^{N} \qquad (3)$$

Для каждого процесса p_j задается отображение множества его методов-элементов M_j во множество их реализаций M. Таким образом, для

любого $m \in M_j$ однозначно определяется его реализация $\tilde{m} \in M$. Формально это можно записать как аксиомы

$$R_3 : \left\{ \left(\forall m \in M_j \right) \left(\exists ! \tilde{m} \in M \right) \left(\{ m, \tilde{m} \} \in m_{j,real} \right) \right\}_{j=1}^{N}.$$

Разница между имеющими одинаковую реализацию методами, с точки зрения процесса, может быть, например, в способе коммутации их параметров с характеристиками модели, о чем речь пойдет ниже. Не запрещаются (хотя и не рекомендуются) и просто «синонимы».

$$\left\{ e_{j,real} \subset E_j \times E \right\}_{j=1}^{N} \qquad (4)$$

Для каждого процесса p_j задается отображение множества его методов-событий E_j во множество их реализаций E. Для любого $e \in E_j$ однозначно определяется его реализация $\tilde{e} \in E$. Аксиомы

$$R_4 : \left\{ \left(\forall e \in E_j \right) \left(\exists ! \tilde{e} \in E \right) \left(\{ e, \tilde{e} \} \in e_{j,real} \right) \right\}_{j=1}^{N}$$

утверждают этот факт. Так же, как и в случае методов-элементов, различные с точки зрения процесса события, могут иметь одинаковые реализации.

$$\left\{ m_{j,in} \subset M_j \times \beta(X) \right\}_{j=1}^{N} \qquad (5)$$

Коммутируются внутренние и внешние характеристики модели с входящими параметрами методов-элементов j-го процесса. Включение долж-

но определить, какое подмножество характеристик модели передается каждому из методов-элементов. Это можно выразить аксиомами

$$R_5: \left\{\left(\forall m \in M_j\right)\left(\exists !r \in \beta(X)\right)\left(\{m,r\} \in m_{j,in}\right)\right\}_{j=1}^{N}.$$

$$\left\{m_{j,out} \subset M_j \times \beta(X)\right\}_{j=1}^{N} \tag{6}$$

Коммутируются возвращаемые параметры методов-элементов с внутренними характеристиками модели. При этом выполняются аксиомы

$$R_6: \left\{\left(\forall m \in M_j\right)\left(\exists !r \in \beta(x)\right)\left(\{m,r\} \in m_{j,out}\right)\right\}_{j=1}^{N}.$$

$$\left\{e_{j,in} \subset E_j \times \beta(X)\right\}_{j=1}^{N} \tag{7}$$

Коммутируются внутренние и внешние характеристики модели с входящими параметрами методов-событий j-го процесса. Включение должно определить, какое подмножество характеристик модели передается каждому из методов-событий. При этом должны выполняться аксиомы

$$R_7: \left\{\left(\forall e \in E_j\right)\left(\exists !r \in \beta(X)\right)\left(\{e,r\} \in e_{j,in}\right)\right\}_{j=1}^{N}.$$

Соотношениями

$$\left\{sw_j \subset E_j \times M_j \times M_j\right\}_{j=1}^{N} \tag{8}$$

определяются переключения методов-элементов в каждом процессе. Если при определенных условиях возможно переключение процесса с выполнения метода $A \in M_j$ на выполнение метода $B \in M_j$ (а таким условием в си-

лу гипотезы о замкнутости может быть лишь некое сочетание внутренних и внешних характеристик модели $\{x,a\}\in X$, – в виртуальном мире модели просто ничего больше нет), то должен быть создан единственный метод-событие $e_{AB}(x,a)\in E_j$, прогнозирующий и вычисляющий такое переключение. Возможны также события вида $e_{AA}(x,a)$, не переключающие метод-элемент $A\in M_j$ на другой, а лишь прерывающие его выполнение (например, для синхронизации результатов его вычислений с другими процессами модели). Таким образом, время и порядок переключений элементов каждого процесса определяется тем, что происходит внутри и вне модели. Вообще говоря, в самых простых процессах, событий может и не быть, тогда не будет и переключений. Если же множество событий j-го процесса – E_j, непусто, то соотношение типизации (8) должно удовлетворять следующим аксиомам

$$R_8:\ \Big\{\Big((\forall e\in E_j)\,(\exists\,!\,r\in M_j\times M_j)\,(\{e,r\}\in sw_j)\Big)\&$$

$$\&\,\Big((\{e,r\}\in sw_j,\{\tilde{e},\tilde{r}\}\in sw_j,r=\tilde{r})\Rightarrow(e=\tilde{e})\Big)\Big\}_{j=1}^{N},$$

утверждающим, что отображения множеств методов-событий во множества переключений биективны.

Соотношение

$$\left\{m_j^0\subset M_j\right\}_{j=1}^{N} \tag{9}$$

определяет начальные методы-элементы процессов.

Соотношения типизации

$$\left\{p_j \subset \beta(M_j) \times \beta(E_j) \times M_j \times \beta(E_j \times M_j \times M)\right\}_{j=1}^{N} \qquad (10)$$

определяют процессы модели, а соответствующий им набор аксиом

$$R_9 : \left\{p_j = \left\{ M_j, E_j, m_j^0, sw_j \right\}\right\}_{j=1}^{N},$$

уточняет это понятие. Процесс – это множества элементов и событий, начальный элемент и правила переключений. Множества M_j и E_j – это множества элементов и событий j-го процесса; m_j^0 – его начальный элемент;

sw_j – правила переключения элементов, которые были подробно описаны выше.

Формальное определение семейства родов структур «модель-компонента» на этом полностью заканчивается.

Тем не менее дополним набор аксиом семейства родов структур «модель-компонента» еще двумя. Эти аксиомы в некотором смысле стоят особняком, поскольку ничего не добавляют к формальному определению отношений, задаваемых на базисных множествах рассматриваемого семейства родов структур – упомянутые отношения уже полностью определены. Смысл дополнительных аксиом R_{10} и R_{11} в том, чтобы описать, что и как мы собираемся делать дальше с математическими объектами семейства родов структур «модель-компонента».

Аксиома R_{10} требует однозначности вычисления характеристик модели-компоненты. Возможность однозначности вычисления характеристик постулируется гипотезой о замкнутости и подробно обсуждалась в разд. 2.3 – 2.5. Для модели-компоненты это требование означает, например, что два метода-элемента разных процессов не должны одновременно менять одну и ту же характеристику. А гипотеза о замкнутости утверждает, что этого возможно добиться. Собственно, для модели-компоненты этим все и ограничивается, и можно было бы не выделять это требование в аксиому. Однако, при объединении компонент в комплекс, все оказывается не так просто, и эта аксиома начинает работать. Поэтому пока можно считать ее заготовкой на будущее, которая потребуется в следующем разд. 3.3.

Аксиома поведения R_{11} задает следующие правила запуска на счет любого математического объекта семейства родов структур «модель-компонента».

Во-первых, считается, что в начале шага моделирования известны текущие элементы всех процессов и все внутренние характеристики модели (на первом шаге – это начальные значения внутренних характеристик и начальные элементы процессов).

Во-вторых, предполагается, что внешние характеристики модели возможно определить в любой момент модельного времени.

Далее

1. Вычисляются события связанные с текущими элементами процессов. Связь событий с текущими элементами процессов определяется правилами переключений (8). Вычисляться события могут параллельно, однако для продвижения вычислительного процесса далее, следует дождаться завершения вычислений всех событий. Если есть наступившие события, проверяется, нет ли переходов к быстрым элементам из множеств $\{f_j\}_{j=1}^{N}$, если они есть – выполняются соответствующие быстрые элементы (они становятся текущими). Вычисляться они могут также параллельно, однако для продвижения вычислительного процесса далее, следует дождаться завершения вычислений всех быстрых элементов, затем возврат к началу п.1; если нет переходов к быстрым элементам – совершаются переходы к новым медленным элементам из множеств $\{s_j\}_{j=1}^{N}$, затем возврат к началу п.1.
2. Если нет наступивших событий – из всех прогнозов событий выбирается ближайший $\Delta\tau$.
3. Если стандартный шаг моделирования Δt не превосходит прогнозируемого времени до ближайшего события, $\Delta t \le \Delta\tau$ – вычисляем текущие медленные элементы со стандартным шагом Δt. В противном случае вычисляем их с шагом времени до ближайшего спрогнозированного события $\Delta\tau$. Медленные элементы из множеств $\{s_j\}_{j=1}^{N}$, так-

же можно вычислять параллельно, с ожиданием окончания последнего.

4. Возвращаемся к началу п.1.

Еще раз проиллюстрируем соотношения типизации и аксиомы семейства родов структур «модель-компонента» следующей таблицей:

Таблица соотношений типизации и аксиом

Соотношения типизации, пояснение	Аксиома, пояснение
$x \subset X$, $a \subset X$ x и a – множества внутренних и внешних характеристик модели, X – всех ее характеристик.	$R_1: (x \cup a = X) \& (x \cap a = \varnothing)$ Подмножества внутренних и внешних характеристик не пересекаются и дают в совокупности все характеристики.
$s \subset M$, $f \subset M$ s и f – медленные и быстрые методы-элементы, M – все элементы.	$R_2: (s \cup f = M) \& (s \cap f = \varnothing)$ Подмножества медленных и быстрых элементов не пересекаются и дают в совокупности все элементы.
$\{m_{j,real} \subset M_j \times M\}_{j=1}^{N}$ Отображение методов-элементов j-го процесса в общее хранилище методов-элементов M.	$R_3: \{(\forall m \in M_j)(\exists! \tilde{m} \in M)(\{m, \tilde{m}\} \in m_{j,real})\}_{j=1}^{N}$ У каждого элемента из M_j есть единственная реализация в M.
$\{e_{j,real} \subset E_j \times E\}_{j=1}^{N}$ Отображение методов-событий j-го процесса в общее хранилище методов-событий E.	$R_4: \{(\forall e \in E_j)(\exists! \tilde{e} \in E)(\{e, \tilde{e}\} \in e_{j,real})\}_{j=1}^{N}$ У каждого события из E_j есть единственная реализация в E.
$\{m_{j,in} \subset M_j \times \beta(X)\}_{j=1}^{N}$ Коммутация входных параметров методов-элементов.	$R_5: \{(\forall m \in M_j)(\exists! r \in \beta(X))(\{m, r\} \in m_{j,in})\}_{j=1}^{N}$ У каждого $m \in M_j$ есть единственный набор входных параметров из X.

Соотношения типизации, пояснение	Аксиома, пояснение
$\{m_{j,out} \subset M_j \times \beta(X)\}_{j=1}^{N}$ Коммутация возвращаемых параметров методов-элементов.	R_6: $\{(\forall m \in M_j)(\exists! r \in \beta(x))(\{m,r\} \in m_{j,out})\}_{j=1}^{N}$ У каждого элемента из M_j есть единственный набор возвращаемых в x параметров.
$\{e_{j,in} \subset E_j \times \beta(X)\}_{j=1}^{N}$ Коммутация входных параметров методов-событий.	R_7: $\{(\forall e \in E_j)(\exists! r \in \beta(X))(\{e,r\} \in e_{j,in})\}_{j=1}^{N}$ У каждого события из E_j есть единственный набор входных параметров из X.
$\{m_j^0 \subset M_j\}_{j=1}^{N}$ Начальные методы-элементы процессов.	
$\{sw_j \subset E_j \times M_j \times M_j\}_{j=1}^{N}$ Правила переключений методов-элементов j-го процесса.	R_8: $\{(\forall e \in E_j)(\exists! r \in M_j \times M_j)(\{e,r\} \in sw_j)\)\&$ $\&((\{e,r\} \in sw_j, \{\tilde{e},\tilde{r}\} \in sw_j, r=\tilde{r}) \Rightarrow (e=\tilde{e}))\}_{j=1}^{N}$ Каждое событие из E_j, определяет единственное переключение между двумя (возможно, даже одинаковыми), методами-элементами из M_j; и отображения событий в переключения биективны.
$\{p_j \subset \beta(M_j) \times \beta(E_j) \times M_j \times$ $\times \beta(E_j \times M_j \times M_j)\}_{j=1}^{N}$ Процессы.	R_9: $\{p_j = \{M_j, E_j, m_j^0, sw_j\}\}_{j=1}^{N}$ Процесс – это множества элементов и событий, начальный элемент и правила переключений.
	R_{10}: аксиома однозначности вычисления характеристик модели-компоненты
	R_{11}: аксиома поведения модели-компоненты

Таким образом, род структуры «модель-компонента» полностью формально определен и даже частично содержательно объяснен.

3.2.3 Синтез модели-комплекса из моделей-компонент

Пусть теперь есть два математических объекта из семейства родов структур «модель-компонента» Σ_N и $\Sigma_{N'}$, с базисными множествами $X, M, E, \{M_j\}_{j=1}^{N}, \{E_j\}_{j=1}^{N}$ и $X', M', E', \{M'_j\}_{j=1}^{N'}, \{E'_j\}_{j=1}^{N'}$ соответственно. Определим модель-комплекс, составленную из этих двух моделей-компонент. Назовем базисными множествами комплекса следующий набор множеств:

$$X \cup X', M \cup M', E \cup E', \{M_j\}_{j=1}^{N+N'}, \{E_j\}_{j=1}^{N+N'};$$

где $\{M_j\}_{j=1}^{N+N'}$ и $\{E_j\}_{j=1}^{N+N'}$ обозначают $M_1,...,M_N, M'_1,...,M'_{N'}$ и $E_1,...,E_N, E'_1,...,E'_{N'}$ соответственно. Объединения множеств $M \cup M'$ и $E \cup E'$ понимаем в самом обычном смысле:

$$M \cup M' = \{m : (m \in M) \vee (m \in M')\},$$

$$E \cup E' = \{e : (e \in E) \vee (e \in E')\}.$$

Что касается множеств X и X', будем считать, что они не пересекаются, т.е. если даже $X \cap X' \neq \emptyset$, договоримся, что мы умеем различать характеристики из $X \cap X'$ по признаку их происхождения из X или X'. Таким образом, непустые элементы $X \cap X'$ входят в объединение $X \cup X'$ дважды, как $X \cap X' \subset X$ и $X \cap X' \subset X'$, и различаются. Содержательный смысл такого предположения – мы собрали вместе две модели-

компоненты, но они никак не взаимодействуют между собой – каждая живет собственной жизнью так, как если бы она была одна.

Покажем, что на базисных множествах комплекса можно задать род структуры $\Sigma_{N+N'}$ из семейства «модель-компонента». Будем проверять выполнение следующих соотношений типизации:

$$x \cup x' \subset X \cup X',\ a \cup a' \subset X \cup X', \tag{11}$$

$$s \cup s' \subset M \cup M',\ f \cup f' \subset M \cup M', \tag{12}$$

$$\left\{m_{j,real} \subset M_j \times (M \cup M')\right\}_{j=1}^{N+N'}, \tag{13}$$

$$\left\{e_{j,real} \subset E_j \times (E \cup E')\right\}_{j=1}^{N+N'}, \tag{14}$$

$$\left\{m_{j,in} \subset M_j \times \beta(X \cup X')\right\}_{j=1}^{N+N'}, \tag{15}$$

$$\left\{m_{j,out} \subset M_j \times \beta(X \cup X'\right\}_{j=1}^{N+N'}, \tag{16}$$

$$\left\{e_{j,in} \subset E_j \times \beta(X \cup X')\right\}_{j=1}^{N+N'}, \tag{17}$$

$$\left\{m_j^0 \subset M_j\right\}_{j=1}^{N+N'}, \tag{18}$$

$$\left\{sw_j \subset E_j \times M_j \times M_j\right\}_{j=1}^{N+N'}, \tag{19}$$

$$\left\{p_j \subset \beta(M_j) \times \beta(E_j) \times M_j \times \beta(E_j \times M_j \times (M \cup M')\right\}_{j=1}^{N+N'}; \tag{20}$$

и аксиом:

R_1: $(x \cup x' \cup a \cup a' = X \cup X') \,\&\, ((x \cup x') \cap (a \cup a') = \varnothing)$,

R_2: $(s \cup s' \cup f \cup f' = M \cup M') \,\&\, ((s \cup s') \cap (f \cup f') = \varnothing)$,

R_3: $\left\{\left(\forall m \in M_j\right)\left(\exists\, !\tilde{m} \in M \cup M'\right)\left(\{m, \tilde{m}\} \in m_{j,real}\right)\right\}_{j=1}^{N+N'}$,

R_4: $\left\{\left(\forall e \in E_j\right)\left(\exists\, !\tilde{e} \in E \cup E'\right)\left(\{e, \tilde{e}\} \in e_{j,real}\right)\right\}_{j=1}^{N+N'}$,

R_5: $\left\{\left(\forall m \in M_j\right)\left(\exists\, !r \in \beta(X \cup X')\right)\left(\{m, r\} \in m_{j,in}\right)\right\}_{j=1}^{N+N'}$,

R_6: $\left\{\left(\forall m \in M_j\right)\left(\exists\, !r \in \beta(x \cup x')\right)\left(\{m, r\} \in m_{j,out}\right)\right\}_{j=1}^{N+N'}$,

R_7: $\left\{\left(\forall e \in E_j\right)\left(\exists\, !r \in \beta(X \cup X')\right)\left(\{e, r\} \in e_{j,in}\right)\right\}_{j=1}^{N+N'}$,

R_8: $\left\{\left(\left(\forall e \in E_j\right)\left(\exists\, !r \in M_j \times M_j\right)\left(\{e, r\} \in sw_j\right)\right)\,\&\right.$

$\left.\,\&\left(\left(\{e, r\} \in sw_j, \{\tilde{e}, \tilde{r}\} \in sw_j, r = \tilde{r}\right) \Rightarrow \left(e = \tilde{e}\right)\right)\right\}_{j=1}^{N+N'}$

R_9: $\left\{p_j = \left\{ M_j, E_j, m_j^0, sw_j \right\}\right\}_{j=1}^{N+N'}$,

R_{10}: аксиома однозначности вычисления характеристик модели-компоненты,

R_{11}: аксиома поведения модели-компоненты (организации имитационных вычислений).

Очевидно, соотношения типизации (11) – (20) и аксиомы $R_1 - R_9$ выполняются, поскольку как для компоненты Σ_N, так и для $\Sigma_{N'}$, выполняются соотношения (1) – (10) и соответствующие аксиомы $R_1 - R_9$.

Для примера рассмотрим самые громоздкие соотношения типизации (20). Пусть сначала $1 \le j \le N$, тогда в силу (10) справедливо:

$$p_j \subset \beta(M_j) \times \beta(E_j) \times M_j \times \beta(E_j \times M_j \times M) \subset$$

$$\subset \beta(M_j) \times \beta(E_j) \times M_j \times \beta(E_j \times M_j \times (M \cup M')).$$

Пусть теперь $N < j \le N + N'$. Тогда снова в силу (10) заключаем

$$p'_j \subset \beta(M'_j) \times \beta(E'_j) \times M'_j \times \beta(E'_j \times M'_j \times M') \subset$$

$$\subset \beta(M'_j) \times \beta(E'_j) \times M'_j \times \beta(E'_j \times M'_j \times (M \cup M')).$$

Самый громоздкий набор аксиом R_8, равно как и R_9, проверяется совсем уж просто – это всего лишь объединение соответствующих наборов аксиом для объектов Σ_N и $\Sigma_{N'}$, поэтому для примера проверим аксиомы R_6.

Пусть $1 \le j \le N$, тогда, в силу выполнения R_6 для Σ_N справедливо

$$\left(\forall m \in M_j\right)\left(\exists\, ! r \in \beta(x)\right)\left(\{m, r\} \in m_{j,out}\right).$$

Если $r \in \beta(x)$, тем более $r \in \beta(x \cup x')$, поскольку

$$\beta(x) \subset \beta(x \cup x').$$

Следовательно, справедливо:

$$\left(\forall m \in M_j\right)\left(\exists ! r \in \beta(x \cup x')\right)\left(\{m, r\} \in m_{j,out}\right).$$

Аналогично, если $N < j \le N + N'$, тогда в силу выполнения R_6 для $\Sigma_{N'}$ справедливо

$$\left(\forall m' \in M'_j\right)\left(\exists ! r' \in \beta(x')\right)\left(\{m', r'\} \in m'_{j,out}\right).$$

Если $r' \in \beta(x')$, то тем более $r' \in \beta(x \cup x')$, поскольку

$$\beta(x') \subset \beta(x \cup x').$$

Следовательно, справедливо

$$\left(\forall m' \in M'_j\right)\left(\exists ! r' \in \beta(x \cup x')\right)\left(\{m', r'\} \in m'_{j,out}\right),$$

что и завершает проверку выполнения аксиом R_6.

Из сказанного выше заключаем, что наш комплекс является математическим объектом семейства родов структур «модель-компонента» $\Sigma_{N+N'}$.

Аксиома поведения R_{11} по определению такова, что применима к любому объекту семейства «модель-компонента».

Обсудим теперь аксиому однозначности вычисления характеристик модели-компоненты R_{10}.

Поскольку мы «принудительно» решили, что

$$X \cap X' = \varnothing, \qquad (21)$$

то однозначность вычисления характеристик следует из таковой, имевшей место для каждой из компонент. Правда, при этом наши компоненты никак

не взаимодействуют друг с другом, они просто формально объединены вместе.

Обратим внимание на то, что формального критерия выполнения условия (21) быть не может. Формально можно лишь добиться выполнения (21), всегда считая разными характеристики, первоначально принадлежащие разным компонентам, как это и было сделано. Определить, какие характеристики модели одинаковы, а какие отличаются, может лишь разработчик, исходя из своих представлений о ее предметной области. Тем не менее если разработчик указал, что условия (21) нарушены, и конкретизировал, в чем состоят эти нарушения, можно указать на ряд вполне формальных действий, которые следует предпринять, чтобы исправить ситуацию.

Отметим, что если бы мы понимали объединение характеристик комплекса самым обычным образом:

$$X \cup X' = \{y : (y \in X) \vee (y \in X')\},$$

не различая «происхождение» объединенных характеристик, то, как не трудно видеть, соотношения типизации (11) – (20) и аксиомы $R_2 - R_9$ также были бы выполнены. При этом в случае $X \cap X' \neq \emptyset$, имело бы место взаимодействие между компонентами за счет пересечения части их характеристик. Однако гарантировать однозначность вычисления объединенных характеристик (аксиома R_{10}), вообще говоря, было бы невозможно. Действительно, в этом случае уже никакие гипотезы о замкнутости не запре-

щают двум методам-элементам разных процессов (первоначально принадлежащих разным компонентам), одновременно изменять одну и ту же характеристику, если эта характеристика принадлежит $X \cap X'$.

Попробуем найти примирение двух приведенных выше крайностей. Будем считать, во-первых, что по-прежнему все элементы объединения $X \cup X'$ формально различимы, т.е., формально $X \cap X' = \emptyset$. Во-вторых, предположим, что с содержательной точки зрения, наоборот, некоторые характеристики компонент одинаковы (например, обе модели-компоненты пытаются дать прогноз погоды на завтра), т.е., с точки зрения семантики модели $X \cap X' \neq \emptyset$.

Начнем с самого простого. Пусть пересекаются внешние характеристики наших компонент, например, по характеристике $\tilde{a}$. Это означает, что внешняя характеристика $a_i = \tilde{a}$ первой модели и внешняя характеристика $a'_j = \tilde{a}$ второй, отражают один и тот же (иначе они бы различались) фактор внешнего по отношению к нашим моделям мира. Из общих представлений о единстве и объективности внешнего мира следует, что значения этих характеристик (конечно, правильно идентифицированные) должны совпадать, и поэтому, в модели-комплексе вполне достаточно одной из них. Исключим для определенности a' из $X \cup X'$. Тогда во всех соотношениях коммутации (15) и (17), куда входит a', следует заменить a' на a, что позволит этим соотношениям сохранить прежний смысл в модели-комплексе. Если кому-то соображения о «единстве и объективности мира»

кажутся не слишком убедительными, – ниже для устранения пересечения внутренних характеристик компонент, будет предложен иной способ, который вполне может быть применен также и в данном случае.

Пусть теперь пересекается множество внутренних характеристик одной модели-компоненты с множеством внешних характеристик другой.

Например, пусть $x \cap a' = \tilde{y} = x_i = a'_j$. Это означает, что если во второй модели-компоненте характеристика a'_j не моделировалась, а являлась фактором внешнего по отношению к этой компоненте мира, то в модели-комплексе та же характеристика моделируется явно, так как явно моделируется в первой модели-компоненте внутренней ее характеристикой x_i. Стало быть, в модели-комплексе внешняя характеристика второй компоненты $\tilde{y} = a'_j$ становится внутренней характеристикой первой модели-комплекса $\tilde{y} = x_i$. Стало быть, из объединения $X \cup X'$ следует исключить a'_j, а во все соотношения коммутации из (15) и (17), куда входит $\tilde{y} = a'_j$, вместо него включить $\tilde{y} = x_i$.

Таким образом, в полном соответствии с шутливой программистской поговоркой: «Описанная ошибка становится новой полезной возможностью», – можно не только объединять компоненты в комплекс, но и использовать в модели-комплексе то, что некоторые из компонент, быть может, моделируют нечто, что без них не умели другие, извлекая определенные преимущества из усложнения модели путем создания комплексов.

Пусть, наконец, пересекаются внутренние характеристики моделей-компонент. К сожалению, звучавшие в случае внешних характеристик соображения о единстве и объективности мира, здесь не убеждают. Наоборот, всем хорошо известно, что могут быть десятки различающихся прогнозов погоды или курсов валют на завтра. Пусть внутренние характеристики двух моделей-компонент пересекаются по характеристике $\tilde{x} = x_1 = x_2$. Предлагается дополнить множество $X \cup X'$ характеристикой $\tilde{x}$. Вопрос вычисления значения этой характеристики поручить специальной модели-компоненте, внешними характеристиками которой будут x_1 и x_2, алгоритм ее вычисления по x_1 и x_2 оставляется на усмотрение разработчика. В соотношениях коммутации входящих параметров – это (15) – (17), где присутствуют x_1 и x_2, они должны быть заменены на $\tilde{x}$.

Теперь основной результат данного раздела можно сформулировать в виде следующего предложения.

Предложение 3.3.1

Пусть имеются n математических объектов семейства родов структур $\Sigma_{N_1},...,\Sigma_{N_n}$ «модель-компонента» с базисными множествами

$$X_1, M_1, E_1, \{M_{1,j}\}_{j=1}^{N_1}, \{E_{1,j}\}_{j=1}^{N_1}; \ldots; X_n, M_n, E_n, \{M_{n,j}\}_{j=1}^{N_n}, \{E_{n,j}\}_{j=1}^{N_n}.$$

Тогда возможно распространение рода структуры $\Sigma_{N_1+...+N_n}$ семейства «модель-компонента» на базисные множества модели-комплекса, составленной из этих моделей-компонент:

$$X_1 \cup ... \cup X_n, M_1 \cup ... \cup M_n, E_1 \cup ... \cup E_n, \{M_j\}_{j=1}^{N_1+...+N_n}, \{E_j\}_{j=1}^{N_1+...+N_n},$$

при этом:

1. В объединение характеристик $X_1 \cup ... \cup X_n$ входят все характеристики компонент, т.е., формально различными считаются даже одинаковые в предметной области характеристики разных компонент. Объединения реализаций методов-элементов $M_1 \cup ... \cup M_n$ и методов-событий $E_1 \cup ... \cup E_n$ понимаются в обычном смысле.
2. Если множества внутренних характеристик моделей-компонент $x_1 \subset X_1, ..., x_n \subset X_n$ имеют непустые попарные пересечения, то исходный набор из n моделей-компонент придется пополнить еще некоторым количеством L объектов семейства родов структур «модель-компонента». Первоначальное множество характеристик комплекса также пополняется одной новой характеристикой на каждую дополнительную компоненту.
3. Если попарно пересекаются внутренние и внешние характеристики компонент: $x_i \subset X_i$, $a_j \subset X_j$, $i \neq j$ и $x_i \cap a_j \neq \emptyset$, то эти характеристики коммутируются, при этом коммутируемые внешние характеристики компонент исключаются из набора внешних характеристик комплекса, так как становятся его внутренними характеристиками.
4. Если попарно пересекаются внешние характеристики моделей-компонент – они либо отождествляются, либо, как и в случае с внут-

ренними характеристиками, неоднозначность разрешается путем добавления в комплекс некоторого количества дополнительных компонент и характеристик. ■

Таким образом, модели-компоненты допускают объединение в комплексы с возможной (но не всегда обязательной) коммутацией некоторых внутренних характеристик некоторых компонент с некоторыми внешними характеристиками других, возможным дополнением первоначального набора компонент некоторыми новыми и коррекцией части соответствий в соотношениях (5) – (7) для некоторых компонент. Как было показано, полученная таким образом модель-комплекс обладает структурой рода «модель-компонента» (быть может, после добавления в комплекс еще некоторого количества компонент, разрешающих возникшие при объединении базисных множеств неоднозначности), и, стало быть, в свою очередь может входить в новые модели-комплексы.

Этим способом можно строить сколь угодно сложные фрактальные конструкции комплексов, состоящих из компонент, которые сами затем становятся компонентами комплексов. При этом, тем не менее все вычислительные процессы конструкций произвольной сложности, полностью определяются аксиомой поведения R_{11} рода структуры «модель-компонента».

3.3. Модельно-ориентированное программирование

Построение имитационных моделей сложных многокомпонентных систем невозможно без компьютера. При этом компьютерная реализация имитационной модели, скорее всего, является достаточно большой программной системой, в силу сложности ее предметной области. Сказанное поднимает проблему программной реализации имитационных моделей. Основным средством реализации больших программных систем является в последние десятилетия объектно-ориентированный подход, воплощенный в ряде популярных языков программирования высокого уровня.

3.3.1 Декларативное и императивное программирование

Философ времен окончательного упадка Римской империи А. Боэций сказал: «Гораздо важнее знать, что делается, чем делать то, что знаешь». Можно соглашаться или спорить с ним на предмет что же важнее. Однако самым существенным здесь нам кажется указание на два вида знания:

1. Про некоторые вещи мы точно (конечно, в пределах точности наших моделей) знаем, каковы они. Например, свое имя, паспортные данные и банковские реквизиты. Или каковы характеристики и способы взаимодействия «атомов» корпускулярной модели. Или какова должна быть страничка Web-сайта, которую мы описываем на языке HTML.

2. Про некоторые вещи мы не знаем, каковы они, а знаем (или только считаем, что знаем) лишь, как об этом узнавать. Например, каково решение достаточно сложной системы дифференциальных уравнений — существуют различные алгоритмы решения таких систем, для некоторых из них известны строгие обоснования их применимости в определенных случаях. Или, например, как работает и на что способна предложенная Р. Рейганом СОИ – можно построить ее имитационную модель и поиграть на ней в звездные войны. Здесь проблема с обоснованием применимости модели, по-видимому, будет сложнее.

Следует отметить, что граница между этими видами знаний достаточно подвижна. Например, легко можно забыть свои паспортные данные и банковские реквизиты. Однако если вместе с ними не утрачены основные навыки бытовой культуры, то известно, какие стандартные действия нужно предпринять, чтобы восстановить эти знания. С другой стороны, многие скажут, что знают, что такое π, или $\sqrt{2}$, хотя в очень строгом смысле, мы всего лишь знаем, как вычислять эти числа с некоторой (вовсе не какой угодно!) степенью точности. Аналогично для специалистов в области вычислительной математики численное решение некоторых классов систем дифференциальных уравнений, особенно если под рукой имеются соответствующие инструментальные средства, является почти такой же рутиной,

как вычисление $\sqrt{2}$, и в этом смысле можно сказать, что такие специалисты «знают» решения упомянутых систем уравнений.

С приведенными выше двумя типами знания достаточно естественно связываются две известные и очень важные для нас парадигмы программирования – декларативная и императивная.

Вообще-то различных парадигм программирования гораздо больше. Например, в википедии их можно насчитать около трех десятков. В данной работе будут упомянуты лишь те из них, которые имеют отношение к исследуемой теме – построению корпускулярной индуктивной модели сложной системы.

Под парадигмой программирования, вслед за Памелой Зейв (Pamela Zave), автор склонен понимать набор представлений о некотором классе программных систем, допускающих реализацию с помощью этой парадигмы «way of thinking about computer systems» [15]. Таким образом, парадигма не есть язык или система программирования, а некий порядок в голове программиста, позволяющий ему даже на ассемблере писать структурно, или объектно, или функционально, или же в рамках какой-то еще парадигмы. Тем не менее многие современные языки программирования ориентированы на ту или иную парадигму программирования, и, следовательно, предоставляют специальные средства и удобства для ее реализации. Например, C++, Java, C# реализуют ООП (разновидность императивного

программирования); LISP, F#, РЕФАЛ – функциональное программирование (разновидность декларативного); PROLOG – логическое.

При декларативном программировании описывается то, каким должен быть известный заранее желаемый результат, например статическая страничка HTML или документ в системе LaTeX, или образ DVD-диска со всей сложной системой его меню и видеоматериалов в системе авторинга Scenarist. При этом выбор последовательности действий, приводящей к описанному в системе декларативного программирования результату, как правило, оставляется на усмотрение соответствующего компилятора или интерпретатора.

Под императивным программированием будем понимать самый распространенный подход к написанию программ, где программа есть последовательность инструкций-приказов, которые должен выполнить компьютер. Таким образом, применяя императивное программирование, мы описываем последовательность действий, достаточную (на наш взгляд) для получения результата проекта. Результат при этом не предполагается известным заранее, наоборот, скорее всего, целью проекта императивного программирования как раз и является получение этого результата.

Мы видим, что декларативная парадигма программирования достаточно естественна для описания наших знаний о системе первого типа, т.е. того, что мы знаем наверняка. Императивное программирование хорошо

для описания знаний второго типа – алгоритмов узнавания того, что мы не знаем непосредственно, но хотим и умеем (считаем, что умеем) узнавать.

Заметим, что как декларативное, так и императивное программирование дают достаточно простора для ошибок. Об этом знает всякий, кому приходилось программировать, хотя бы даже на HTML. Однако отлаживать декларативную программу гораздо легче – очень сильно помогает знание о том, как все должно быть. Императивные программы отлаживать очень трудно. Результат работы программы не известен заранее. В случае достаточно сложной программы, даже правильно работающей, этот результат может оказаться весьма неожиданным для исследователя (например, в оптимальном управлении вместо обывательских представлений о плавном восхождении от хорошего к еще лучшему могут возникать скользящие режимы управления, весьма далекие от подобных представлений). Так что если даже императивная программа не вылетела и не зациклилась, а благополучно что-то сосчитала – всегда остается сомнение: полученный ею результат действительно верен, или же программа где-то врет? Для решения этой проблемы разрабатываются специальные системы тестирования, иногда сопоставимые по затратам с разработкой основной программы. Заметим еще что предполагаемое «знание о том, как узнавать» в императивных программах может оказаться ложным – здесь также достаточно обширное поле для разного рода иллюзий и ошибок.

Скажем еще несколько слов о функциональном программировании – разновидности декларативного. Здесь процесс вычисления трактуется как вычисление значений функций в математическом понимании последних. Математическая функция отличается от функции-подпрограммы императивного программирования тем, что осуществляет отображение своей области определения в область допустимых значений. Функция-подпрограмма, вообще говоря, устроена сложнее. Она также может оказаться отображением множества своих входящих параметров в множество возвращаемых, однако вполне может и не оказаться. Например, у нее могут существовать различные внутренние состояния, связанные со статическими переменными или со значениями записей базы данных, с которой она работает. И в зависимости от этих состояний, функция-подпрограмма при одном и том же входе может выдавать, вообще говоря, различные выходы. Функциональную парадигму легко осуществить средствами большинства современных универсальных языков программирования, ограничившись константами вместо переменных и следя за тем, чтобы функции-подпрограммы были отображениями в математическом смысле. Функциональные программы хороши тем, что по сравнению с императивными легче отлаживаются и тестируются: никаких побочных эффектов, никаких состояний – выход зависит лишь от входа, ошибка достаточно легко локализуется.

Из всего сказанного в этом разделе следует вывод: если что-то в системе допускает декларативное описание – надо это ценить и стараться всячески использовать, отладка будет гораздо легче. Аналогично, если есть возможность сократить императивное программирование, ни в коем случае нельзя ее упускать – это существенно упростит последующую жизнь. Если есть возможность написать подпрограмму в функциональной парадигме, обязательно нужно этим воспользоваться – во время отладки и тестирования окупится сторицей.

Почему так трудно программировать модель сложной системы?

Потому что она и в самом деле очень сложная!

Попробуем перечислить основные сложности:

1. Сложная структура системы и сложные связи между ее компонентами;

2. Сложная организация данных;

3. Сложное поведение компонент системы:

   а) сложная логика поведения;

   б) сложная функциональность действий.

Здесь мы выделили несколько типов сложности, а в модели сложной системы они еще все тесно переплетены между собой. Психологи говорят, что средний человек уверенно способен оперировать не более чем 4 – 6 объектами одновременно. Если объектов становится больше, человек начинает ошибаться. Быть может, природная одаренность в данной обла-

сти или же опыт и тренировка способны повысить этот порог с 4 – 6 даже до 15 – 20 объектов. Беда только в том, что в сложных системах приходится оперировать с сотнями и тысячами объектов.

Особенно сложным является императивное программирование, о чем уже говорилось выше. Кроме упомянутых сложностей выбора алгоритма, действительно приводящего к желаемому результату, отладки и тестирования – именно императивные программы обычно сочетают в себе сложную логику со сложной функциональностью и сложной организацией данных.

Возникает вопрос, что же тогда делать, как найти технологию программирования, которая поможет преодолеть перечисленные сложности? На самом деле, такие технологии существуют и достаточно давно применяются в области автоматизации проектирования.

3.3.2 Декомпозиция и инкапсуляция

Проблема синтеза сложных систем из заданных компонент достаточно успешно решается в различных системах автоматизации проектирования (САПР). Основная идея САПР – модульность разработки. Система проектируется не из самых мелких своих составляющих, а из модулей (блоков) самого высокого уровня, которые могут состоять в свою очередь из модулей более низкого уровня, и, наконец, модули самого низкого уровня состоят из самых мелких «атомов» системы. Как только тот или иной модуль отлажен – про его внутреннее устройство можно забыть. Для

работы с ним на более высоком уровне важны лишь его интерфейс и функциональность. Такой подход позволяет создавать процессоры, содержащие десятки миллионов транзисторов, что было бы немыслимо при проектировании их на уровне отдельных транзисторов.

Заметим, что, по крайней мере, на первый взгляд, эта задача хорошо укладывается в императивную парадигму программирования. В самом деле, отталкиваясь от имеющихся знаний об отдельных компонентах сложной системы, мы беремся построить синтез всей системы, основанный на воспроизведении известного нам поведения каждой ее компоненты, и при этом собираемся наблюдать и изучать поведение системы в целом, не известное нам заранее. В основе синтеза многокомпонентной системы лежит идея, высказанная еще в работах Н.П. Бусленко: дать каждой из компонент, про которые нам все известно, максимально проявить себя, учитывая при этом и все межкомпонентные связи [34].

Основным инструментом реализации проектов императивного программирования являются объектно-ориентированные языки программирования. В свое время эти языки возникли в значительной мере как ответ на запросы исследователей, занимавшихся имитационным моделированием. Одним из самых первых объектно-ориентированных языков можно считать Симулу-67. Структура сложной системы, состоящей из многих экземпляров различных типов компонент хорошо описывается иерархией классов объектно-ориентированного программирования. У ООП масса извест-

ных всем достоинств, которые и сделали этот подход бесспорным лидером в программировании последних 30 лет. Однако применительно к нашей задаче есть у ООП и существенные недостатки.

Главный из них (конечно, применительно к классу задач, о которых идет речь в данной работе) – отсутствие у объекта поведения. У объекта есть характеристики, есть масса всяческих умений – это его методы – а поведения, в смысле умения давать ответы на стандартные запросы окружающей среды, у него нет. Конечно, ООП в соединении с соответствующим программным обеспечением промежуточного уровня, позволяет объектам даже в распределенных системах взаимодействовать между собою, наблюдать чужие характеристики, пользоваться чужими методами, однако все это делается в некотором смысле «вручную».

В этом смысле коллектив программистов, приступающий к созданию сложного проекта, и вооруженный набором библиотек самых полезных классов, похож на человека, которому выдали несколько компьютеров, каждый из которых оснащен большим набором полезных прикладных программ, однако из системных программ имеющий только загрузчик, и поставили задачу сделать из этого распределенную систему. Понятно, что современный программист возмутился бы страшно: «Так не бывает! Где моя сетевая операционная система? Где, наконец, ПО промежуточного уровня?». Однако при создании сложной системы средствами ООП дело обстоит именно так – к сложности содержательных вычислений предмет-

ной области добавляется еще и сложность организации поведения объектов а также их связей и взаимодействий между собою.

3.3.3 Поведение математических моделей сложных систем

Что же такое поведение сложной системы?

На наш взгляд, поведение сложной системы есть функциональный аналог операционной системы в области системного программного обеспечения компьютера, и функциональный аналог бытовой культуры в социальных системах – способность давать стандартные ответы на стандартные запросы внутренней и окружающей среды. В имитационном моделировании сложных систем очень важно уметь моделировать поведение компонент. Как уже говорилось выше, именно из их поведения естественно синтезировать поведение системы в целом.

В истории развития информатики известно несколько подходов к объектному программированию, где объекты обладают поведением. Подходы эти зародились в среде исследователей, занимавшихся проблемами искусственного интеллекта, и известны как модель акторов [8] и агентное программирование [11, 12]. Однако хотя автор согласен с главным посылом этих подходов – важностью моделирования поведения агентов системы, детали описания этого поведения в [8, 11, 12], на наш взгляд, слишком окрашены спецификой проблематики искусственного интеллекта, т. е. недостаточно универсальны. Так, у И. Шохема [11, 12] поведение – это поведение ментальное, о состоянии которого рассуждают в категориях «убеж-

дения», «обязанности», «способности» и т.д. Акторная модель Хьюитта [8] интересна своей ориентированностью на параллельные вычисления, однако асинхронный обмен акторов сигналами-сообщенями, а также лавинообразное порождение новых акторов представляются нам избыточными конструкциями, затрудняющими реализацию подобных систем. Подробно этот вопрос обсуждался в [25].

Проблема здесь в том, что, согласившись на наличие поведения у компонент системы, нужно учиться описывать и реализовывать это поведение. Это поведение, с одной стороны достаточно сложно, так как это поведение элементарной сложной системы. Заранее оно не может быть известно, поскольку может зависеть от внутренних и внешних по отношению к рассматриваемой компоненте условий, на которые она должна уметь реагировать должным образом. Следовательно, такие поведенческие моменты должны вычисляться и, казалось бы, алгоритмы их вычисления должны быть описаны на императивных языках программирования.

С другой стороны, кое-что о поведении компоненты мы всегда знаем заранее, так как она в силу исходных предположений работы имеет в предметной области моделирования достаточно хорошо изученный прообраз. Поэтому обычно мы заранее можем перечислить весь набор ее «умений» – элементарных действий, а также сказать, какое из этих элементарных действий может перейти в какое и под действием каких обстоятельств

– просто уж так устроен прообраз этой компоненты в предметной области моделирования.

Вот это-то знание, коррелирующее с пониманием устройства в предметной области как прообраза отдельной компоненты, так и всей многокомпонентной модели, как раз наличествует заранее еще до построения модели и не зависит от хода имитационных экспериментов. Поэтому такое знание об устройстве модели сложной системы вполне может быть выражено на декларативном языке программирования.

Такой подход позволяет получить «ортогональные» (подобным образом «ортогональны», например, описания стиля и наполнения в Word- или HTML-документах) описания модели сложной системы: на декларативном языке описывается то, как устроена система, и правила поведения ее составляющих. На императивном языке описываются отдельные законченные элементарные алгоритмы, не взаимодействующие ни с чем в модели, кроме собственных входных и выходных параметров, и, стало быть, вполне укладывающиеся в парадигму функционального программирования.

Предлагаемое разделение описаний на декларативные и функциональные оказывается весьма технологичным: декларативные и функциональные части можно отлаживать независимо друг от друга. При этом самая сложная составляющая программы, которая пишется на обычных языках программирования, распадается на ряд независимых друг от друга за-

конченных элементарных алгоритмов. При этом она редуцируется настолько, что чаще всего возможности ООП оказываются слабо востребованными: алгоритмически цельную программу с фиксированным набором входных и выходных параметров чаще всего нетрудно реализовать в функциональной парадигме. И это существенно облегчает отладку системы. Вся объектная ориентированность, идущая от предметной области моделирования, оказывается в декларативном описании модели.

Одной из первых сред программирования, воплотивших описанную выше концепцию разделения описания сложной системы на декларативную и функциональную составляющие, была разработанная в конце 80-х в ВЦ АН СССР инструментальная система имитационного моделирования MISS (Multilingual Instrumental Simulation System). Концепция программирования, лежащая в ее основе была полностью описана на уровне руководства пользователя в работе [30]. В системе MISS была реализована в среде MS-DOS система программирования на специальном декларативном языке описания сложных систем, интегрированная с базой данных, системой поддержки выполнения модели и системой презентации результатов моделирования. В качестве императивных языков программирования, на которых писались вычислительные алгоритмы, были разрешены две версии языка MODULA-2, а также языки C и C++ в Борландовской реализации. Следует заметить, что до этого моделирующее сообщество создавало исключительно языки моделирования императивного типа.

Впоследствии идея разделения описания сложной системы на декларативную и императивную части применялась неоднократно. Так в спецификации архитектуры верхнего уровня (High Level Architecture – HLA) [10] – средстве создания сложных распределенных моделей – устройство системы, ее компонент и связей между ними описывается специальными шаблонами. В коммерчески успешной отечественной инструментальной системе имитационного моделирования AnyLogic [23, 64] в качестве декларативного языка описания сложной системы применяется интерактивная графическая система, где на экране размещаются пиктограммы компонент будущей системы и тут же проводятся связи между ними. В качестве императивного языка программирования используется родной для системы AnyLogic язык Java. Наконец, появился и даже стал международным стандартом язык моделирования объектных систем UML [36, 83], однако автору он не симпатичен в силу своей громоздкости и порой за счет этого неоднозначности. В качестве результатов компиляции транслятор UML может выдавать заготовки модулей для императивных языков.

В настоящее время в ВЦ РАН продолжается развитие концепции разделения описания сложной системы на декларативную и функциональную составляющие. Функционирует макет инструментальной системы распределенного моделирования [25], основанный на этой концепции.

3.3.4 Модельно-ориентированная парадигма программирования

Исторически смена парадигм программирования сопровождалась укрупнением, агрегированием основного инструмента деятельности программиста.

Начиналось все с машинной команды, затем, с появлением языков программирования высокого уровня – таким инструментом стал оператор языка, реализующий некое законченное действие, возможно, с помощью нескольких машинных команд.

С победой идей структурного программирования – на смену отдельным операторам и переменным пришли стандартные конструкции типа «цикл», «ветвление», подпрограммы-функции и структуры данных.

С появлением объектного анализа основной единицей конструирования стал объект, объединяющий некую структуру данных с набором необходимых для их обработки методов. Кроме того, с помощью механизма наследования можно строить иерархии классов объектов, развивающих, конкретизирующих и воплощающих некоторый набор базовых идей, заложенных в корневых классах такой иерархии. Данная парадигма программирования в настоящее время является господствующей и ее базовые понятия, такие как класс, объект, типизация, наследование, инкапсуляция, полиморфизм реализованы с некоторыми нюансами в большинстве современных императивных языков программирования, таких как C++, Java, C#, Delphi и многих других.

Модельно-ориентированная парадигма программирования предлагает снова увеличить степень агрегирования основной единицы программирования. Предлагается конструировать программную систему из моделей-компонент – сущностей, помимо характеристик и отдельных умений, обладающих собственным поведением, т.е. способных во всех ситуациях, которые могут им встретиться, давать стандартные для них ответы на стандартные запросы внутренней и внешней среды. Формально этот факт следует из уже упоминавшейся выше гипотезы о замкнутости модели.

Воспользовавшись описанной выше инвариантностью организации вычислительного процесса имитационной модели (аксиома поведения R_{11}) относительно операции объединения моделей-компонент в модель-комплекс, можно предложить новый подход к программированию, во-первых, имитационных моделей сложных систем, и, во-вторых, сложных программных систем, быть может, не имеющих отношения к имитационному моделированию, но имеющих выраженную многокомпонентную организацию, где уместен синтез целого из составляющих его частей, природа которых, а также способы взаимодействия между собой, известны заранее настолько, что для них выполнены требования раздела 3.1 этой работы.

В отличие от объекта объектного анализа, методы модели-компоненты не нужно (да и невозможно) вызывать извне. Не нужно заботиться и об организации функционирования модели-компоненты. Модель-компонента функционирует всегда (как всегда функционирует, например,

операционная система компьютера), в соответствии с аксиомой R_{11} рода структуры «модель-компонента», и всегда готова отреагировать заложенным в ее конструкцию способом на происходящие внутри и вне ее изменения, на которые способны реагировать ее методы-события. Поэтому, про внутреннее устройство однажды отлаженной модели-компоненты можно полностью забыть, используя ее в дальнейших конструкциях как готовый функциональный блок – просто нужно правильно коммутировать входы и выходы – и все будет работать. Все происходит примерно так же, как при включении готовой микросхемы в микросборку и размещении затем этой микросборки на печатной плате. Если при этом на каждом уровне проекта коммутация разумна – микросхема будет правильно функционировать в составе гораздо более сложного электронного устройства.

Описания моделей-компонент, как математических объектов соответствующего рода структуры, декларативны. Также декларативны описания объединения моделей-компонент в модели-комплексы. Для таких описаний предлагается специальный декларативный язык [32, 24] ЯОКК (язык описания компонент и комплексов). Значительный удельный вес в этом языке имеют операторы коммутации.

Декларативные описания модели-компоненты на ЯОКК, эквивалентные соотношениям типизации (1)-(10), вполне определяют ее устройство и логику функционирования, при этом полностью абстрагируясь от содержательной стороны, как действий модели-компоненты, осуществляемых ее

методами-элементами, так и причин этих действий, выявляемых методами-событиями.

В силу все той же гипотезы о замкнутости, все методы модели-компоненты, как быстрые и медленные методы-элементы, так и методы-события, вычисляют некоторые функции (в математическом, а не программистском понимании термина «функция») от некоторых подмножеств внутренних и внешних характеристик модели-компоненты. Следовательно, эти методы вполне могут быть реализованы в функциональной парадигме программирования (что, между прочим, может еще сильнее увеличить степень параллельности получаемого кода). Последнее вовсе не означает призыва к всеобщему переходу на лямбда-исчисление – функциональная парадигма вполне может быть реализуема и на всеми любимых C, C++, C# и Java.

В результате оказывается, что императивное программирование – камень, о который больше всего спотыкаются как при разработке, так и при отладке сложных программных систем – в модельно-ориентированном программировании не применяется вовсе. Кроме того, реализация даже весьма сложной программной системы методом модельно-ориентированного программирования декомпозируется на ряд вполне обозримых декларативных описаний моделей-компонент и составляемых из них моделей-комплексов и не зависящих от этих описаний и друг от друга функциональных программ, допускающих независимое написание и от-

ладку. Подобная декомпозиция весьма технологична при коллективной разработке большой программной системы.

Из аксиомы поведения R_{11} рода структуры «модель-компонента», определяющей вычислительный процесс функционирования любой модели, следует, что чем больше у модели-компоненты процессов, тем большее число методов можно запускать на выполнение параллельно. Поскольку при объединении моделей-компонент в модель-комплекс число процессов полученного комплекса есть сумма процессов его компонент, то можно заключить, что при усложнении модели количество методов, допускающих параллельное выполнение, также увеличивается. Еще более увеличиться оно может от реализации методов в функциональной парадигме программирования. Все это позволяет надеяться на плодотворное применение методов модельно-ориентированного программирования на высокопроизводительных вычислительных системах.

3.3.5 Модельно-ориентированный декларативный язык описания компонент и комплексов (ЯОКК)

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

Основная задача рассматриваемого языка ЯОКК – описать род структуры «модель-компонента», введенный в разд. 3.2, и некоторые связанные с ним понятия, например, организацию внутренних и внешних характери-

стик модели и их коммутацию. Также с помощью этого языка описываются модели-комплексы. Их описания затем можно откомпилировать в соответствующие описания моделей-компонент.

Рассматриваемый язык в полной мере выражает предлагаемую в данной работе концепцию имитационного моделирования сложных многокомпонентных систем. 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.

Возможно, в дальнейшем над языком ЯОКК будет создана надстройка в виде графического интерфейса пользователя (GUI), где мышкой на рабочее поле вытаскивались бы нужные сущности в виде соответствующих пиктограмм и мышкой проводились бы необходимые связи между ними. Такая надстройка переводила бы графические манипуляции с пиктограммами в соответствующие конструкции ЯОКК, – можно считать, что это задача следующего этапа реализации системы.

Потребность в непроцедурных языках, на которых описывается не что нужно выполнить, а то, кто и как устроен и как и с кем связан внутри и снаружи, возникла достаточно давно. Ниже перечислены несколько языков

подобной направленности с указанием их разработчиков и иногда систем, где они применялись:

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

По функциональности выразительных средств, для наших целей подошли бы кроме второго, пожалуй, два последних. Однако из них UML, несмотря на статус международного стандарта, 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 [30], – 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).

3.3.6 Пример использования ЯОКК – пешеходы и муха

Определения конструкций ЯОКК мы будем параллельно иллюстрировать примерами описания на этом языке одной учебно-тестовой модели – уже упоминавшейся выше модели про пешеходов и муху. В связи с этим сначала опишем на неформальном уровне реализованный вариант этой модели.

Модель «Пешеходы и муха» или вариант этой модели, известный как «муха фон Неймана», – простейшая нелапласовская модель. Она уже упоминалась ранее. В ее основе лежит задача, встречающаяся 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. И в школьной задаче и в легенде про то, как фон Нейман ее решал, спрашивается, какое расстояние пролетит муха за время от начала движения до момента встречи пешеходов.

С точки зрения имитационного моделирования сложных систем, 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 не является лапласовской: момент встречи пешеходов является точкой накопления системных собы-

тий, а скорость мухи в этот момент невозможно обусловить всей предшествовавшей историей модели. Если имеется желание продолжить моделирование на время после встречи пешеходов – необходимо задать мухе в точке встречи совершенно новую скорость, никак не вытекающую из всего того, что было до встречи. Связано это с тем, что скорость мухи разрывна и в точке встречи пешеходов имеет два равноправных предельных значения, – по непрерывности в точку встречи ее продолжить невозможно.

Модель была 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.

Модель «пешеходы и муха» представляет собой модель-комплекс, состоящий из двух экземпляров модели-компоненты «пешеход» и одного экземпляра модели-компоненты «муха». Компонента «пешеход» устроена совсем просто: она реализует единственный процесс, состоящий из единственного метода-элемента «движение». «Движение» – медленный элемент, по текущим значениям внутренней характеристики X – координаты пешехода и внешней характеристики V – его скорости, а также по величине текущего шага модельного времени Δt, этот метод вычисляет значение внутренней характеристики X в конце шага модельного времени по формуле $X(t+\Delta t) = X(t) + V \cdot \Delta t$. Никаких других методов, а следователь-

но, и переходов, и связанных с ними событий единственный процесс модели-компоненты «пешеход» не имеет.

Модель-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 характеристикой, а не параметром, как у "пешехода". «Разворот» – быстрый 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 сначала 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.

В дальнейшем именно эту модель в описанном выше виде мы будем использовать для иллюстрации тех или иных конструкций ЯОКК.

3.3.7 Общий синтаксис ЯОКК

Отметим, что хотя аккуратнее было бы употреблять термины «описание класса моделей-компонент» или «описание класса моделей-комплексов», мы ради краткости будем пользоваться словосочетанием «описание компоненты» и «описание комплекса». Лингвистические 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.

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 собой единственный параграф. Описатель типа данных – необязательная конструкция, в некоторых случаях она удобна, но, вообще говоря, можно обойтись и без нее.

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 «*/».

Попробуем формально определить основные синтаксические единицы ЯОКК.

Начнем с метасимволов, которые сами не входят в ЯОКК, но используются в синтаксических формулах, его описывающих.

| – «или» – знак альтернативы. Не входит в число символов ЯОКК. Через этот знак будут перечисляться возможные альтернативы той или иной синтаксической конструкции.

::= – «это есть» – определение синтаксической единицы, стоящей слева от знака через синтаксическую конструкцию, стоящую справа от него. Используется только в метаформулах.

«» – кавычки. Сами кавычки не входят в число символов ЯОКК. В метаформулах в кавычки будут заключены разделители ЯОКК, в основном с целью привлечения к ним внимания. Например, «» или « » означают пробел, «,» – запятая.

[] – квадратные скобки. Не входят в число символов ЯОКК. В метаформулах в квадратные скобки заключается конструкция возможная, но не обязательная в данном месте.

{} – фигурные скобки. Не входят в число символов ЯОКК. В метаформулах в фигурные скобки заключается конструкция, которая может быть повторена в данном месте любое конечное число раз, в том числе и ни одного раза.

<> – угловые скобки. Не входят в число символов ЯОКК. В метаформулах в угловые скобки заключается название той или иной синтаксической единицы.

Определим теперь символы и общие синтаксические конструкции ЯОКК.

<символ ЯОКК> ::= <буква> | <цифра> |

| <разделитель>

<буква> ::= a | … | z | A | … | Z | а | … | я | А | … | Я | _

<цифра> ::= 0 | <значащая цифра>

<значащая цифра> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<разделитель> ::= « » | «.» | «,» | «;» | «:» | «(» | «)» |

| «=» | <ключевое слово>

<ключевое слово> ::= **END;** | **TYPE** | **int** | **double** |

| **long** | **char** | **byte** | **bool** | **uint** | **short** | **ulong** | **decimal** |

| **float** | **address** | **sbyte** | **ushort** | **COMPLEX** |

| **COMPONENTS** | **COMMUTATION** | **SAMENESS** |

| **METHOD** | **FAST** | **SLOW** | **EVENT** | **ADDRESS** | **local** |

| **INPUT** | **OUTPUT** | **COMPONENT** | **PHASE** |

| **PARAMETERS** | **phase** | **param** | **ELEMENTS** |

| **EVENTS** | **SWITCHES**

<идентификатор> ::= <буква>

[{<буква> | <цифра>}]

При этом идентификатор не может совпадать ни с одним ключевым словом.

< целое без знака > ::=

::= 0 | <значащая цифра>[{<цифра>}]

Синтаксические единицы, названия которых начинаются с <имя …> – всегда идентификаторы.

Перейдем теперь к синтаксису описателей.

3.3.8 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.

Описания данных в ЯОКК базируются на встроенных типах данных, совпадающих с аналогами в языках семейства С. Набор встроенных типов может меняться, в зависимости от платформы.

Синтаксис оператора описания данных следующий:

<идентификатор типа> « » <описание данных>

{«**,**»<описание данных >}«**;**»

Идентификатор типа отделяется пробелом от одного или нескольких разделенных запятой описаний данных, заканчивается оператор точкой с запятой. Здесь идентификатор типа – либо одно из встроенных имен типов, приведенных в таблице ниже, либо идентификатор 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.

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

Встроенные типы определяются приведенной ниже таблицей:

Встроенные типы ЯОКК

Тип в ЯОКК	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
sbyte	sbyte	System.SByte	1
ushort	ushort	System.UInt16	2

Пример описателя типа данных:

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

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).

3.3.9 Описатель комплекса

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

За заголовком идут параграфы:

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

   внешних характеристик.]

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 характеристики каких компонент вычисляют какие внешние характеристики каких компонент. Параграф открывается ключевым словом **COMMUTATION**, за которым следуют операторы коммутации. Операторы коммутации имеют следующий вид:

<идентификатор компоненты> «(»

<экземпляр компоненты>«)»«**.**»

[{<имя типа данных> «**.**»}]

<поле параметра компоненты> «**=**»

<идентификатор компоненты> «(»

<экземпляр компоненты >«)»«**.**»

[{<имя типа данных> «**.**»}]

<поле внутренней характеристики

компоненты>«**;**»

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

{<идентификатор>«**.**»}<поле>

где поле – либо идентификатор, либо элемент массива.

Необязательный параграф отождествления внешних характеристик компонент открывается ключевым словом **SAMENESS**, за которым следуют операторы отождествления. Операторы отождествления имеют следующий вид:

<идентификатор компоненты> «(»

<экземпляр компоненты>«)»«**.**»

[{<имя типа данных> «**.**»}]

<поле параметра компоненты> {«=»

<идентификатор компоненты> «(»

<экземпляр компоненты >«)**.**»

[{<имя типа данных> «**.**»}]

< поле параметра компоненты >}«**;**»

Результат компиляции комплекса – текстовый описатель этого комплекса как модели-компоненты.

Пример описания модели-комплекса

Модель-комплекс «пешеходы и муха»:

COMPLEX menANDfly;

COMPONENTS

Fly(1), Man(2);

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;
```

3.3.10 Описатель метода

Методы – это или элементы процессов, или методы, вычисляющие наступление событий. Это те части модели, которые могут вызываться как локально, так и распределенно. 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 характеристик и/или параметров компоненты. Кроме того, методы различаются на события и методы, реализующие элементы процессов, а последние, еще и по отношению к модельному времени на:

- сосредоточенные (быстрые) – происходящие чены для вычисления возможных разрывов первого рода характеристик модели-компоненты в начале шага моделирования;
- распределенные (медленные) – занимающие не менее одного модельного такта и дающие определенный результат своего выполне-

ния в виде изменений внутренних характеристик модели в конце каждого такта. Они предназначены для вычисления характеристик модели-компоненты на интервалах их непрерывности.

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 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 URL (Uniform Resource Locator) – единый указатель ресурсов или 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 идентификатор типа после ключевых слов необязателен. Если тем не менее он 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 параметров, так как известно, что они всегда возвращают значение типа **double** – прогнозируемое время до наступления соответствующего события.

Примеры описаний методов-элементов моделей-компонент «пешеход» и «муха» и метода-события компоненты «муха» –

Медленный метод-элемент «движение»:

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

Быстрый метод-элемент «разворот»:

METHOD Uturn: FAST;

// по умолчанию ADDRESS: local;

INPUT

double v;

/**********

Поскольку про OUTPUT ничего не сказано – по умолчанию он такой же, как и INPUT, т.е.

OUTPUT

double v;

**********/

END;

Метод-событие «долет до пешехода»:

METHOD reaching: EVENT;

ADDRESS: simul.ccas.ru;

INPUT

double x,v,m0x,m0v,m1x,m1v;

// У всех методов-событий OUTPUT всегда – double dt;

// – прогноз времени его наступления

END;

3.3.11Описатель 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 характеристик.
3. [Параграф параметров.]
4. Параграф элементов.
5. [Параграф событий]
6. Параграф коммутации.
7. [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 характеристик и параметров. Синтаксис их был описан выше в описателе типов.

Синтаксис параграфов внутренних характеристик и параметров (внешних характеристик) такой же, как и параграфов типов – разница лишь в ключевых словах параграфов: **PHASE** и **PARAMETERS** соответственно. Кроме того, идентификатор типа после ключевого слова необязателен. Если он присутствует – тип данных с таким именем появится в базе данных рабочей станции в результате успешной компиляции описателя. Если нет, то чтобы идентифицировать 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.

Приведем формальное описание параграфа элементов:

< параграф элементов> ::= **ELEMENTS**

{<описатель процесса> «;»}

[**END;**]

<описатель процесса> ::= [<метка процесса> «:»]

<элемент> [{«,» <элемент>}]

<элемент> ::= < имя элемента>

[«:» <имя реализации>]

< имя элемента> ::= <идентификатор>

< имя реализации> ::= <идентификатор>

< метка процесса > ::= <идентификатор> |

| <целое без знака>

Параграф методов-элементов открывается ключевым словом **ELEMENTS** и заканчивается ключевым словом **END;**, если он последний.

На самом деле, этот параграф определяет не только методы-элементы, но и процессы модели-компоненты. В этом параграфе через «**;**» – точку с запятой перечислены процессы компоненты. Каждый процесс в этом перечислении – это метка процесса, не обязательная в простейших случаях, например, когда процесс один. Метка процесса – целое без знака или идентификатор, после которого следует двоеточие. После метки процесса идут перечисленные через запятую имена образующих его элементов с возможным указанием через двоеточие имени реализации метода. Имя реализации должно совпадать с именем, появлявшемся в одном из ранее откомпилированных описателей методов. Если имя реализации явно не указано – реализация будет искаться в базе данных под именем элемента в процессе.

Порядок элементов в их перечислении в составе процесса произволен, за исключением того, что первым в списке идет так называемый начальный элемент, который будет считаться текущим при первом запуске модели на выполнение. Имена элементов должны быть уникальными в пределах описания процесса: нельзя одинаково назвать два разнотипных эле-

мента одного процесса, но использование одного имени элемента в описаниях разных процессов не возбраняется. Приведем пример параграфа элементов:

ELEMENTS

1: Man_0_move: move;

2: Man_1_move: move;

3: Fly_0_move: move, Fly_0_Uturn: Uturn;

END;

Приведем формальное описание параграфа событий:

< параграф событий > ::= **EVENTS**

{<описатель событий процесса> «;»}

[**END;**]

<описатель событий процесса> **::=**

[<метка процесса> «:»]

<событие> [{«,» <событие>}]

<событие> ::= <имя события>

[«:» <имя реализации>]

<имя события> ::= <идентификатор>

<имя реализации> ::= <идентификатор>

<метка процесса> ::= <идентификатор> |

| <целое без знака>

Синтаксис параграфа событий очень похож на синтаксис параграфа элементов. В этом параграфе через «**;**» – точку с запятой перечислены наборы событий процессов компоненты. Каждый набор событий процесса в этом перечислении – это метка процесса, не обязательная в простейших случаях, например, когда процесс один. Метка процесса – целое без знака или идентификатор, после которого следует двоеточие. После метки процесса идут перечисленные через запятую имена событий процесса с возможным указанием через двоеточие имени реализации метода-события. Имя реализации должно совпадать с именем, появлявшемся в одном из ранее откомпилированных описателей методов. Если имя реализации явно не указано – реализация будет искаться в базе данных под именем события в процессе.

Порядок событий в их перечислении в составе процесса произволен. Имена событий должны быть уникальными в пределах описания процесса: нельзя одинаково назвать два разнотипных события одного процесса, но использование одного имени события в описаниях разных процессов не возбраняется.

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 типа данных> «**.**»}]<имя поля>«**;**»

Входящий параметр метода связывается с внутренней (ключевое слово **phase**) или внешней (ключевое слово **param**) характеристикой компоненты. Если в описателе компоненты нет параграфа параметров, квалификатор «**phase.**» можно опустить, оставив в операторе коммутации лишь имя поля.

2. Коммутация возвращаемых параметров методов, ее синтаксис:

   [{<имя типа данных> «**.**»}]

   <имя поля внутренней характеристики> «=»

   <имя метода>«**.**»

   <имя возвращаемого параметра>«**;**»

Возвращаемый параметр метода связывается с полем внутренней характеристики 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 от-

компилированы 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:

[<метка процесса> «:»]

<имя текущего элемента> «,»

<имя следующего элемента >

[«**:**»<имя события>] «**;**»

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

Примеры описаний моделей-компонент –

Модель-компонента «пешеход»:

```
COMPONENT Man;
 PHASE
 ManPhase:
 double x;
 PARAMETERS
 ManParam:
 double v;
 ELEMENTS
 move;
 COMMUTATION
```

```
move.x = x;
move.v = v;
x = move.x;
END;
```

Модель-компонента «муха»:

```
COMPONENT Fly;
PHASE
FlyPhase:
double x,v;
PARAMETERS
FlyParam:
FlyPhase man0Phase, man1Phase;
ELEMENTS
move, Uturn;
EVENTS
reaching;
SWITCHES
Move, Uturn: reaching;
Uturn, move;
COMMUTATION
move.x=FlyPhase.x;
move.v=FlyPhase.v;
```

```
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;
```

Модель-компонента «пешеходы и муха».

Модель-комплекс «пешеходы и муха», рассматриваемая как компонента, в результате синтеза из своих компонент (распространения рода структуры «модель-компонента») – описание генерируется автоматически, в результате компиляции описателя модели-комплекса «пешеходы и муха», при условии, что все входящие в модель-комплекс модели-компоненты были успешно откомпилированы:

```
COMPONENT menANDflyAScomp;
PHASE
menANDflyPhase:
double FlyPhase_0_x, FlyPhase_0_v, ManPhase_0_x,
ManPhase_1_x;
```

PARAMETERS

menANDflyParam:

double ManParam_0_v, ManParam_1_v;

ELEMENTS

1: Man_0_move: move;

2: Man_1_move: move;

3: Fly_0_move: move, Fly_0_Uturn: Uturn;

EVENTS

3: Fly_0_reaching: reaching;

SWITCHES

3: Fly_0_move, Fly_0_Uturn: Fly_0_reaching;

3: Fly_0_Uturn, Fly_0_move;

COMMUTATION

Man_0_move.x=ManPhase_0_x;

Man_0_move.v=ManParam_0_v;

ManPhase_0_x=Man_0_move.x;

Man_1_move.x=ManPhase_1_x;

Man_1_move.v=ManParam_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=ManParam_0_v;
Fly_0_reaching.m1x=ManPhase_1_x;
Fly_0_reaching.m1v=ManParam_1_v;
END;
```

3.3.12 Компиляция описателей ЯОКК

Основой компиляции описателей ЯОКК можно назвать компиляцию типов данных. Для каждого проекта имитационной модели сложной системы, состоящего из набора комплексов и компонент, в базе данных заводится таблица типов данных, в которую автоматически заносятся встроенные в ЯОКК типы данных.

Таблица типов данных отличается от приведенной выше наличием еще одного столбца, где для каждого типа данных хранится в текстовом виде результат его полного синтаксического анализа – схема типа данных. Для встроенных типов данных схема совпадает с именем встроенного типа. Для более сложных типов данных схема содержит все имена промежуточных типов данных и имена их полей, начиная от имени типа в целом, которое может быть сгенерировано автоматически, если не было задано

явно в соответствующем описателе, и кончая терминальными полями встроенных типов.

По схеме типа можно однозначно определить, имеются ли у рассматриваемого типа данных поля с заданным именем, сколько таких полей, а также вычислить величину смещения в байтах любого поля типа от начала данных.

Основной задачей компиляции явного описания типа данных (описатель типа данных), или неявного их описания (описания параметров методов, внешних и внутренних характеристик компонент) как раз и является построение схемы типа данных и определение размера данных этого типа в байтах.

Поскольку новый тип данных определяется как набор полей и/или массивов существующих типов - процесс компиляции состоит в поиске в уже упоминавшейся таблице базы данных имен этих типов. Если тип найден в таблице, его схема включается в схему вновь создаваемого типа, а размер учитывается при определении размера нового типа данных. Если же какой-либо из типов данных, используемых в описании компилируемого типа, отсутствует в таблице, – компиляция описателя заканчивается аварийно с соответствующей диагностикой.

Схемы типов данных оказываются очень важными при компиляции операторов коммутации во всех описателях ЯОКК, где такие операторы встречаются – в описателях комплексов и компонент. Прежде всего, по-

скольку в операторах коммутации требуется не полная квалификация имен коммутируемых полей (хотя она, конечно же, не возбраняется), а минимальная, не приводящая к неоднозначности, то по соответствующим схемам типов проверяется, сколько полей с запрашиваемыми именами существует в схеме рассматриваемого типа данных. Если их не оказывается или оказывается больше одного – компиляция завершается аварийно с соответствующей диагностикой. Если и слева и справа в операторе коммутации находится ровно одно поле, поля проверяются на соответствие типов. Здесь может быть несколько уровней строгости контроля их соответствия:

1. Самый строгий – проверяется соответствие ключей в таблице типов базы данных.
2. Проверяется соответствие схем данных, т.е. разрешается коммутировать синонимы типов данных.
3. Проверяется соответствие коммутируемых полей, выраженных через встроенные типы данных, т.е. разрешается коммутировать поля, возможно, существенно различных типов – не синонимов, но одинаково представляемые встроенными типами данных (например, массив **double**, и столько же отдельных полей этого типа).
4. Проверяется соответствие размеров коммутируемых полей.
5. Иногда могут иметь смысл коммутации даже при несоответствии размеров коммутируемых полей.

В случае несоответствия полей при выбранном уровне проверки строгости, происходит аварийное завершение компиляции с соответствующей диагностикой. В случае соответствия заполняется соответствующая таблица базы данных. Например, в случае коммутации параметров метода с характеристиками модели-компоненты можно хранить в строке таблицы коммутации, соответствующей оператору коммутации, номера метода и модели-компоненты в проекте, смещения от начала данных для типов параметров метода и характеристик компоненты, и, наконец, номер в проекте типа коммутируемого поля. Ссылку на тип, а не длину коммутируемого поля имеет смысл хранить потому, что она может пригодиться, например, при дальнейшей коммутации моделей-компонент в модель-комплекс.

При компилировании операторов коммутации описателя компоненты, также должно проверяться:

1. Все ли характеристики модели задействованы, т.е. если какая-либо внутренняя характеристика не изменяется ни одним из методов-элементов – об этом должно быть диагностическое сообщение (возможно, это внешняя характеристика модели). Если внешняя характеристика компоненты не передается ни одному из методов-элементов или методов-событий – об этом также должно быть диагностическое сообщения.

2. Нет ли попыток изменять внешние характеристики модели. Если такие попытки имеются, должна быть соответствующая диагностика (возможно, эти характеристики внутренние).
3. Должна быть предупреждающая диагностика, если методы-элементы разных процессов изменяют одну и ту же внутреннюю характеристику модели-компоненты. В этом может и не быть криминала, если поведение модели таково, что эти методы никогда не вызываются одновременно в модельном времени. Однако если такие методы могут попасть в набор методов-элементов, вызываемых параллельно, должна быть соответствующая диагностика времени выполнения. Вообще говоря, такая ситуация говорит о том, что модель спроектирована неверно: если одну и ту же характеристику пытаются одновременно получить несколькими различными способами, то это свидетельствует о том, что нарушена однозначность вычислительного процесса моделирования. О способах преодоления подобных неоднозначностей подробно говорилось в разд. 3.3.

Что касается компиляции описателей моделей-компонент, нерассмотренными остались параграфы элементов, событий и переключений. Вместе эти три параграфа полностью описывают процессы модели-компоненты, т.е. ее поведение. На стадии компиляции здесь необходимы следующие проверки:

1. У процессов не должно быть изолированных методов-элементов, т.е. таких, в которые принципиально невозможно попасть из начального элемента за конечное число шагов – не описаны соответствующие переключения.
2. Для быстрых элементов наряду с переходами по событиям, желателен безусловный переход. Отсутствие такового не будет криминалом, если всегда реализуется одно из событий, обеспечивающих переход из быстрого элемента. Если же нет, возникает диагностика времени выполнения: элемент закончился, а к какому переходить далее, не указано.
3. Не должно быть неиспользуемых событий, т.е. событие, появившееся в параграфе событий, должно встретиться и в параграфе переключений. И наоборот, встречающиеся в параграфе переключений события должны быть описаны в параграфе событий, при этом события должны относиться к одному и тому же процессу.

Наконец, несколько слов о компиляции описателя модели-комплекса. Одно из важных отличий его компиляции от компиляции всех остальных описателей ЯОКК состоит в том, что ее результат – не записи в тех или иных таблицах связанной с проектом базы данных, а текстовый файл описателя модели-компоненты, в которую превращается модель-комплекс. В дальнейшем этот описатель модели-компоненты, полученный автоматиче-

ски, подлежит обычной компиляции, как и любой описатель модели-компоненты.

Неоднозначности внешних характеристик комплекса можно разрешать с помощью операторов коммутации параграфа **SAMENESS**. К этим операторам, а также к операторам обычной коммутации относятся все обычные проверки, относящиеся к коммутации, о которых уже было сказано выше.

Для разрешения неоднозначностей, связанных с методами комплекса (см. разд. 3.3) возможно, придется скорректировать указатели реализаций этих методов в соответствующих описателях, а еще проще – в базе данных проекта.

Неоднозначности внутренних характеристик комплекса нужно разрешать вручную, так, как это было описано в разд. 3.3, т.е. вводя в комплекс новые компоненты, чья задача – по правилам, определенным разработчиком модели, на основании набора характеристик, имеющих одинаковый смысл в данной модели, выдавать единственную характеристику, несущую этот смысл.

Для компиляции описателя модели-комплекса необходимо, чтобы все входящие в него модели-компоненты были до этого успешно откомпилированы.

3.4 Модельный синтез и объектный анализ

Прежде всего следует отметить, что объектный анализ последние тридцать лет является основным средством проектирования и реализации любых программных систем. В то же время модельный синтез не претендует на освоение всего поля разработки сложных программных систем. Изложенная выше концепция предлагается в первую очередь для синтеза имитационных моделей сложных многокомпонентных систем – систем, состоящих из множества агентов, каждый из которых обладает собственным поведением – способностью определенным образом откликаться на происходящее внутри и снаружи этого агента. При этом поведение модели всей сложной системы в целом интересует нас в первую очередь как изменения во времени ее характеристик.

Далеко не все сложные программные системы таковы. Например, при разработке транслятора, если действующих агентов еще иногда можно придумать, то разворачивание изменения характеристик во времени не интересно совсем – в идеальном случае хорошо бы все транслировать мгновенно – здесь может происходить лишь чередование различных стадий обработки исходных данных.

Однако можно надеяться, что предлагаемый подход применим для разработки некоторого не очень малого класса сложных программных си-

стем, организация которых атомистична, допускает выделение агентов, обладающих поведением, и разворачивается индуктивно снизу вверх.

3.4.1 Элементы модельного анализа и проектирования имитационных моделей сложных систем

До сих пор мы все время говорили о синтезе более сложных моделей-комплексов из более простых моделей-компонент. На самом деле, на основе методов модельного синтеза вполне возможно проектировать имитационные модели сложных многокомпонентных систем сверху вниз, начиная с самых агрегированных представлений о модели и постепенно переходя к детальной проработке состава и организации внутренних структуры и связей каждого из таких агрегатов.

Пусть модель-компонента будет терминальным символом некоторого алфавита, а модель-комплекс – нетерминальным. Определим следующее правило вывода, применимое к нетерминальному символу: модель-комплекс может состоять из моделей-комплексов и/или моделей-компонент. Правило вывода применяется, пока имеется хотя бы один нетерминальный символ.

Указанное выше правило вывода порождает древовидные структуры, подобные файловой системе компьютера, где модель-комплекс есть аналог папки, а модель-компонента – файла. Применяя это правило вывода и выводя в конечном итоге с его помощью модели-компоненты, тем не менее запоминаем и все вложенности компонент и комплексов в те комплексы более высокого уровня, в которые они входят.

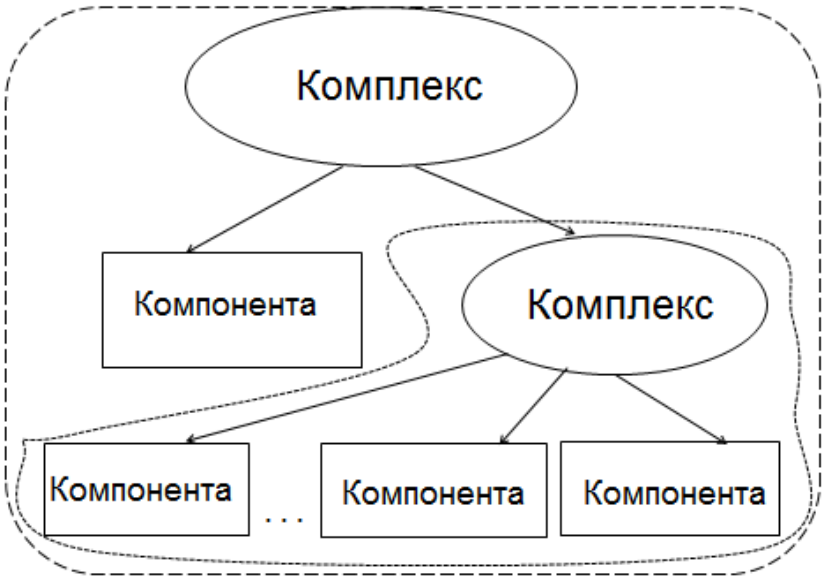


Рисунок 1. Иерархическое проектирование имитационной модели сложной системы.

На рис. 1 пунктиром очерчено то, что входит в различные комплексы.

Таким образом, мы проектируем сложную систему, каждый раз конкретизируя, из чего состоит комплекс и как взаимосвязаны его компоненты. Когда мы дойдем до комплексов, состоящих из одних только моделей-компонент, можно по выработанным в разд. 3.3 правилам синтеза начинать обратное движение снизу вверх, представляя модели-комплексы моделями-компонентами, и, таким образом, строить снизу вверх синтез нашей модели сложной многокомпонентной системы.

3.4.2 Модельный синтез vs объектный анализ

Теперь остановимся на сравнении основных понятий модельно-ориентированного и объектно-ориентированного программирования. Прежде всего отметим, что объектно-ориентированный подход в настоя-

щее время представлен в двух видах: один из них можно назвать «базовым», в его основе лежат такие понятия, как класс, объект, типизация, наследование, инкапсуляция, полиморфизм[1], которые реализованы с некоторыми нюансами в большинстве современных императивных языков программирования, таких как C++, Java, C#, Delphi и многих других.

Второй, который можно назвать «расширенным», представлен унифицированным языком моделирования UML [36, 83]. Язык UML включает все перечисленные выше базовые понятия, кроме того, его создатели пошли по пути резкого увеличения числа исходных понятий и представлений. Например, кроме «вертикального» отношения наследования, которое в UML чаще называется отношением обобщения (при этом отношение обобщения направлено от потомка к предку), имеются отношения ассоциации, композиции, агрегации и зависимости. Появилась возможность описывать поведение систем, причем даже несколькими способами: диаграммы взаимодействия, диаграммы состояний и диаграммы деятельности.

Вообще в UML очень многое можно делать несколькими способами, что делает его удобным инструментом знатока, но весьма затрудняет жизнь новичку. Сами авторы языка говорят, что: «UML подчиняется пра-

[1] Термин «полиморфизм», хотя и является общепринятым, не представляется нам удачным, поскольку его название неверно передает суть обозначаемого им явления – переопределения методов. Действительно, то, что сохраняет объект после переопределения методов, – это именно форма: унаследованные от предка характеристики, а также сигнатуры методов. А то, что при этом изменяется – не форма, а содержание вычислений – то, что переопределенные методы выполняют. Поэтому в дальнейшем, вместо полиморфизма мы будем говорить о переопределении методов.

вилу 80/20, т.е. 80% большинства проблем можно смоделировать, используя 20% средств UML» [36].

В отличие от объектного анализа UML модельный синтез минималистичен в наборе базовых понятий: в нем имеется единственное основное понятие – модель-компонента и одно вспомогательное понятие – модель-комплекс, состоящий из моделей-компонент, который, в конце концов, в результате устранения возникших неоднозначностей также может стать моделью-компонентой. Правда, на широту охвата «всего на свете» модельный синтез не претендует, область его применения – сложные многокомпонентные «атомистические» системы, которые естественно строить снизу вверх.

Продолжим сравнение базовых понятий. Такие понятия, как класс, объект, типизация в двух рассматриваемых подходах понимаются примерно одинаково. Следует лишь заметить, что корни этих понятий следует искать конечно не в C++ и даже не в Симуле-67, а, скорее, в работах Н. Бурбаки [33], структуралистов XX века [39], вплоть до Эрлангенской программы Ф. Клейна [63].

Примерно одинаково понимаются также характеристики и методы. Что же касается использования методов, здесь имеется существенная разница. Она состоит в том, что в объектном программировании объект затем и нужен, чтобы вызывать его методы в различных программах, причем в качестве параметров методу можно передать, а также принять от него лю-

бые переменные, удовлетворяющие его сигнатуре – не обязательно характеристики его объекта. В модельно-ориентированном программировании метод модели-компоненты может работать лишь с ее характеристиками и вызывать его «вручную» нет ни возможности, ни необходимости; он будет вызван автоматически тогда, когда этого потребует логика поведения модели-компоненты. Инкапсуляция в модельно-ориентированном программировании такова, что не позволяет напрямую добраться до методов. Оперировать можно лишь моделями-компонентами.

Это вовсе не означает, что методы совсем недоступны. Наоборот, например, в реализованном макете распределенной системы имитационного моделирования [25, 32], библиотеки методов публикуются в интернете для общего использования, и возможны модели-компоненты, не имеющие ни одного локального метода, притом все реализации методов могут физически находиться в разных местах сети. Тем не менее воспользоваться методом можно, лишь включив его в состав некоторой модели-компоненты, и работать он в ней будет не сам по себе, а в соответствии с ее логикой поведения – таков уровень инкапсуляции. В некоторых случаях это может показаться усложнением, но оно оплачено отсутствием забот об организации поведения модели-компоненты – она всегда ведет себя так, как умеет, и поэтому включение ее в любой комплекс всегда всего лишь вопрос правильной коммутации.

Переопределение методов в модельно-ориентированном программировании может быть легко достигнуто путем указания иной реализации в соотношении типизации (3) из разд. 3.2, если речь идет об элементе, или (4), если речь идет о событии.

Скажем несколько слов о наследовании. Определим базисное множество рода структуры «объект» как совокупность его характеристик X и методов M. Вводится отношение наследования, которое по базисному подмножеству характеристик есть включение множества характеристик предка во множество характеристик потомка, а по базисному подмножеству методов есть инъекция множества методов предка во множество методов потомка, при этом сохраняющая так называемую сигнатуру, т.е. названия методов и типы их параметров. Сами же реализации методов потомка могут отличаться в силу возможности переопределения потомком методов предка. Отношение наследования, рассматриваемое в направлении от потомка к предку, в объектном анализе называют также отношением обобщения. В объектно-ориентированных языках программирования по своему типу потомок является предком, но предок не является потомком (аксиома антисимметрии отношения частичного порядка), поэтому переменные типа потомка могут быть использованы как значения переменных или параметров типа предка, но не наоборот.

Таким образом, отношение наследования для множества классов объектно-ориентированного языка программирования есть отношение частич-

ного порядка. Классы, не имеющие предков, но обладающие потомками, называются по отношению к ним корневыми или базовыми. Классы, не имеющие потомков, называются листовыми.

Проектирование в объектно-ориентированной парадигме большой программной системы состоит в воплощении базовых понятий и представлений этой системы в базовые классы объектов и построении затем иерархии классов, развивающих, конкретизирующих и воплощающих эти идеи во множестве листовых классов, с помощью которых и будет строиться целевая программная система.

С точки зрения геометрической теории декомпозиции, набор базовых классов представляет собой F-редукции будущих наборов листовых классов, которые разворачивается из набора базовых сверху вниз. Наследование в объектно-ориентированном программировании носит характер постепенного выявления различий в первоначально однородном фактор-множестве, поэтому такой тип наследования можно назвать F-наследованием. Такой дедуктивный способ проектирования большой программной системы хорош, когда она творится «с чистого листа», подобно миру у Платона.

В имитационном моделировании, однако, гораздо чаще возникает задача не создания новых миров, а воспроизведения фрагментов уже существующего. В подобном фрагменте запросто могут быть собраны вместе «…диван, чемодан, саквояж, картина, корзина, картонка и маленькая соба-

чонка». Друг из друга они не выводятся, а восходить вверх до их общих предков – достаточно бессмысленно. Для подобных задач «базовому» варианту объектного подхода недостает наследования снизу вверх (в UML такое наследование имеется, что впрочем, все равно не делает его удобным инструментом проектирования имитационных моделей).

Возникает вопрос, возможно ли в модельно-ориентированном программировании дедуктивное F-наследование сверху вниз? Включая компоненту в комплекс, мы тем самым всегда получаем в распоряжение ее характеристики и методы. Как уже говорилось выше, методы переопределяются легко, например путем указания других URL их реализаций. Однако просто так добавить характеристик мы не можем, их всегда должен кто-то обрабатывать. Но мы можем добавить новую компоненту, в состав которой входят дополнительные характеристики, при этом возможно коммутировать эти характеристики со старыми, унаследованными.

Таким образом, в модельно-ориентированном программировании объединение моделей-компонент в модель-комплекс можно считать множественным наследованием снизу вверх. Точно так же, как в языках программирования, где множественное наследование разрешено, в модельно-ориентированном программировании оно может приводить к некоторой неоднозначности, которую тем не менее можно разрешить способом, показанным в разд. 3.3. При этом модели-компоненты, хотя могут и не являться точными P-подобъектами содержащей их модели-комплекса из-за возмож-

ных межкомпонентных связей, тем не менее очень на такие подобъекты похожи. Поэтому объединение компонент в комплекс с позиций геометрической теории декомпозиции можно назвать P-наследованием.

Геометрическая теория декомпозиции [65] утверждает, что F- и P-декомпозициями и их сочетаниями исчерпываются всевозможные декомпозиции математических объектов, при этом F- и P- конструкции двойственны. Поэтому отсутствие одной из них (например, в механизмах наследования), является признаком функциональной неполноты любого метода, в особенности претендующего на универсальность.

Даже если в объектно-ориентированном проекте с помощью наследования построена самая замечательная иерархия классов, все равно все сложности организации вычислительного процесса, состоящего в использовании разработанных и отлаженных методов листовых классов, лежит на разработчике системы: чтобы система что-то делала, необходимо организовать вызов нужных методов в нужной последовательности. Между прочим, на этом этапе и теряется модульность объектно-ориентированной программы: обычно все используемые методы так или иначе связаны друг с другом. Для настоящей модульности (например, такой как в САПР микроэлектроники), не хватает уровня инкапсуляции: из объектов приходится вручную в нужной последовательности вызывать их методы. Самый сложный этап построения большой программной системы остается неформализованным – это искусство.

Попытки формализовать процесс проектирования сложных программных систем и породили UML. По-видимому, и в самом деле на UML можно описать любую систему и даже с нескольких точек зрения. Вопрос в том, что делать дальше с таким описанием, здесь нет единства мнений.

Некоторые специалисты (например, [83]) считают, что основная ценность UML как раз в применении как средства документирования и обмена формализованными описаниями на стадиях эскиза и проектирования сложных систем. Судя по некоторым изменениям работы [36] (2-е изд.), при переизданиях создается впечатление, что и сами авторы языка UML начинают склоняться к такому мнению.

Тем не менее имеется и ряд средств, позволяющих компилировать UML-описания в заготовки классов универсальных языков программирования, и в этом случае можно говорить о режиме использования UML в качестве языка программирования. Однако здесь мы снова остаемся в рамках объектной парадигмы, получаем иерархию классов и заготовок классов, но не избавляемся от необходимости писать императивные программы, вызывающие в нужном порядке методы этих классов.

В рамках объектно-ориентированной императивной парадигмы с F-наследованием сверху вниз остается и возникшая на основе UML концепция разработки программных систем, управляемая моделями, известная в различных вариантах под названиями MDA (Model Driven Architecture), MDE (Model Driven Engineering) или MDD (Model Driven Development).

Эта концепция предполагает, вооружившись всей мощью UML, начинать разработку с построения абстрактной модели программной системы, которая затем воплощается в платформенно-независимую модель, которая преобразуется (возможно, автоматически) в платформенно-зависимую, из которой (возможно, автоматически) генерируется программный код. Здесь снова разворачивание проекта происходит сверху вниз, и все его дальнейшее развитие остается в парадигме объектно-ориентированного программирования. Следовательно, неизбежно должно завершиться написанием императивных программ, вызывающих методы как полученных автоматически, так и созданных вручную классов.

При компиляции языка UML и при автоматической трансформации в код моделей MDA/MDD/MDE неизбежно возникает вопрос о качестве этой компиляции и эффективности получаемого кода – UML достаточно сложный язык, располагающий множеством различных не слишком простых понятий и представлений. Собственно, именно сложности такого рода и заставляют многих специалистов считать UML в первую очередь, именно средством документирования и эскизного проектирования сложных программных комплексов. Программную реализацию комплекса при этом создают вручную на стандартных объектно-ориентированных языках, пользуясь описанием эскиза программного комплекса, выполненного на UML.

Наконец, следует заметить, что в последнее время все более востребованными становятся высокопроизводительные многопроцессорные и мно-

гоядерные вычислительные системы. Связано это в том числе и с тем, что на горизонте уже начали вырисовываться физические ограничения для увеличения производительности однопроцессорных компьютеров. Объектный подход ни в «базовой», ни в «расширенной» версии здесь ничего особенного не предлагает – задача параллелизации вычислений остается одной из самых сложных и трудно формализуемых.

Модельный синтез и модельно-ориентированное программирование предполагают более высокий уровень инкапсуляции, нежели принят в объектном анализе. Основная единица программы – модель-компонента – наделена самостоятельным поведением, поэтому организовывать вычислительный процесс путем вызова каких-либо методов не нужно и невозможно, он всегда организуется автоматически в соответствии с описанным поведением компонент (правилами переключения методов-элементов в процессах моделей-компонент, – соотношения типизации (8) в разд.3.2).

Наличие у модели-компоненты собственного поведения позволяет последовательно провести идею модульности (до сравнимой, например, с САПР микроэлектроники степени) разрабатываемой с помощью модельного синтеза программной системы. Как в микроэлектронике однажды разработанную микросхему можно включить в любую микросборку или печатную плату, зная ее функцию и правила соединения и не заботясь о ее внутреннем устройстве на уровне транзисторов – просто правильно припаять, так и в модельном синтезе однажды разработанную и отлаженную модель-

компоненту можно включать в любой комплекс, зная правила коммутации и выполняемую функцию, позабыв полностью подробности ее реализации. При этом работать она всегда будет сама, как и микросхема, в полном соответствии с описанными где-то ранее правилами поведения, дополнительно заботиться об этом не нужно. Играющие роль «паяльника» операторы коммутации, например языка ЯОКК, полностью декларативны, т.е. программировать в привычном смысле императивного программирования, для построения сложных программных комплексов из готовых моделей-компонент, не приходится.

Вообще, модельный синтез полностью исключают императивное программирование. Это следует из того, что гипотеза о замкнутости (разд. 3.1.) гарантирует нам функциональную зависимость на интервале модельного времени $(t, t+\Delta t]$ внутренних характеристик модели от ее внутренних и внешних характеристик на левом конце этого интервала – в точке t.

Если зависимость возвращаемых параметров методов-элементов от их входящих параметров функциональная, ее вычислительный процесс вполне может быть осуществлен в функциональной парадигме программирования (пусть и на обычных универсальных языках), что существенно упрощает отладку программы, а в некоторых случаях может к тому же увеличить степень параллельности получаемого кода.

В рамках концепции модельного синтеза, устройство модели последовательно описывается на декларативном языке ЯОКК. Описатели ЯОКК

компилируются в таблицы базы данных совершенно определенной структуры. Вопрос о качестве такой компиляции, как это часто бывает с декларативными описаниями, не стоит: компиляция либо верна, если компилятор работает правильно, а если неверна, значит, компилятор нужно отлаживать. Вопрос об эффективности получаемого кода здесь не стоит, потому что этого кода просто нет – генерируются и заполняются таблицы базы данных известной формы.

Наконец, последнее преимущество модельного синтеза – ориентированность программы организующей имитационные вычисления (аксиома поведения модели-компоненты R_{11} из разд. 3.2) на параллельное выполнение множества методов-элементов и методов-событий. При этом при усложнении модели, количество методов, которые допускают параллельное вычисление, также растет, пропорционально росту количества всех процессов модели.

3.5 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 ранее 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 [24], [25] (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.

3.5.1 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 моделировать 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 выше, 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 [10], где распределенно 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 ЯОКК, описанный ранее.

3.5.2 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 вы-

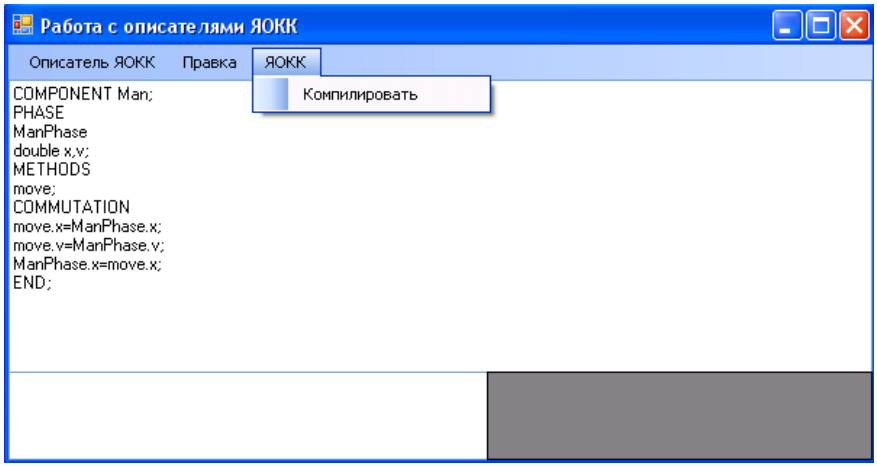
зовов, где запоминается имя метода, время вызова и IP-адрес вызвавшей метод рабочей станции.

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.



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 «Про-

цесс» – это номер соответствующего ключа из таблицы процессов. Наконец, шестое числовое поле «Реализация» – номер соответствующего ключа из таблицы реализаций.

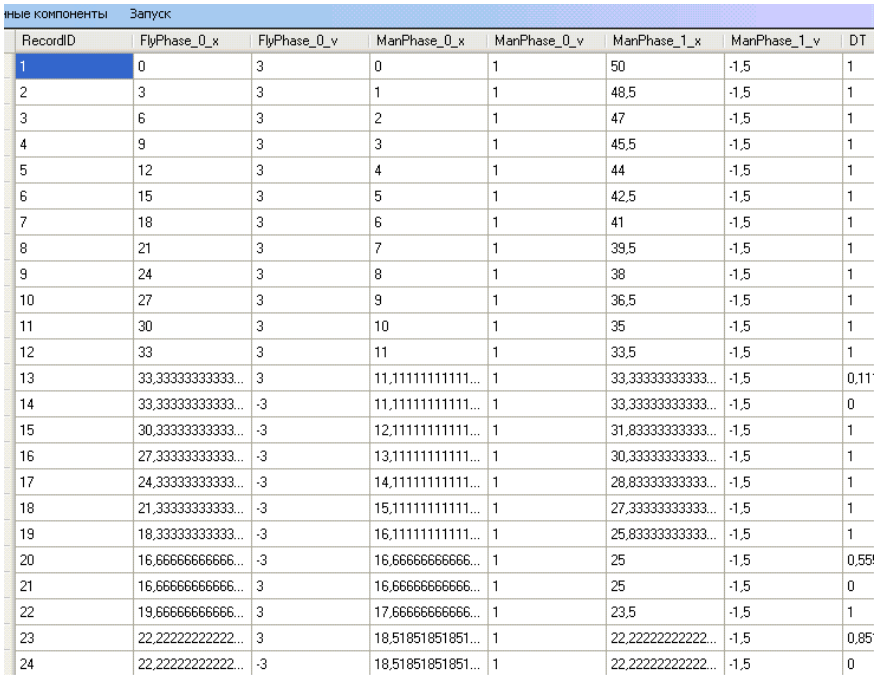
5. Следующая таблица – таблица переходов между элементами, в ней 5 полей. Первое – ключ «НомерПерехода». Второе числовое «Процесс» – номер ключевого поля в таблице процессов. Третье, четвертое и пятое поля числовые «ТекущийЭлемент», «СледующийЭлемент» и «Событие» – все это номера ключевого поля в таблице 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 [30]. 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 основных 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 спис-

ками. 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

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 работе модели «Пешеходы и муха», приводится пример реализации распределенной модели в описываемой инструментальной системе моделирования.

Заключение

В настоящее время область построения моделей сложных многокомпонентных систем относится в значительной мере к области искусства, поскольку все решения основаны на эвристических представлениях их авторов о моделировании сложных систем и эвристических способах, пусть даже и весьма успешных, решения различных проблем, возникающих при таком моделировании.

В данной работе, оттолкнувшись от некоторых общепризнанных в кругах разработчиков имитационных моделей положений, таких как требование замкнутости модели, однозначности и детерминированности имитационных вычислений, а также необходимости реализации имитационных вычислений на компьютере за конечное время, мы попробовали вывести некоторые необходимые свойства получаемых моделей. Одно из таких свойств – кусочно-непрерывный, с не более чем конечным числом разрывов первого рода, характер траектории модели. При этом, переопределяя траекторию модели не более чем в конечном количестве точек, можно считать, что траектория модели полунепрерывна слева.

Выделен класс моделей, (а именно удовлетворяющих в каждой точке гипотезе о замкнутости и имеющих кусочно-гладкую, непрерывную слева траекторию), для которых из локальной гипотезы о замкнутости модели будет следовать возможность успешного синтеза модели на конечном отрезке моделирования (разд. 3.1, предложение 3.1.1).

Предложена организация имитационных вычислений, ориентированная на модели с кусочно-гладкими траекториями, как инварианта относительно объединения моделей-компонент в модели-комплексы, позволяющая полностью решить задачу описания и синтеза имитационных моделей сложных многокомпонентных систем.

Все это позволяет формально определить класс имитационных моделей сложных многокомпонентных систем, как род структуры «модель-компонента» в смысле Н. Бурбаки, и на основе такого определения построить новые концепции формального описания, синтеза и реализации имитационных моделей – модельный синтез и модельно-ориентированное программирование, являющиеся альтернативой повсеместно используемых объектного анализа и объектно-ориентированного программирования, прежде всего для задач имитационного моделирования сложных многокомпонентных систем.

Концепция модельного синтеза минималистична в отношении базовых понятий и представлений: в ней всего одно основное понятие – модель-компонента, и одно вспомогательное – модель-комплекс, который путем несложных операций синтеза, описанных в разд. 3.3, в конечном итоге также превращается в модель-компоненту.

Концепции модельного синтеза и модельно-ориентированного программирования полностью исключают из проекта создания имитационной модели наиболее трудоемкое в проектировании, реализации и отладке им-

ративное программирование, ограничиваясь лишь программированием декларативным и функциональным. Кроме того, модельно-ориентированное программирование производит код высокой степени параллельности.

Описание модели сложной системы состоит из набора декларативных описаний моделей-компонент и составленных из них моделей-комплексов на специальном декларативном языке ЯОКК. При этом многие из этих описаний (например, компоненты одного комплекса) не зависят друг от друга и поэтому могут создаваться и отлаживаться независимыми группами разработчиков. Синтез имитационной модели сложной многокомпонентной системы строится путем компиляции этих описаний. Результатом компиляции любой модели-компоненты (а таковой, как было показано, является, в том числе и вся модель сложной системы), является некий набор таблиц постоянной структуры в базе данных. Для того чтобы запустить модель-компоненту на счет, остается внести в базу данных начальные значения ее характеристик, а также написать на универсальных языках программирования (можно в функциональной парадигме) и отладить ее методы.

То, что конкретно делает модель (методы-элементы, проявляющие ее поведение), полностью отделено от того, почему она что-то делает (методы-события, реагирующие на изменения внутренней и внешней среды), и

от того, каким образом она что-то делает (описания логики поведения модели, соотношениями типизации).

Таким образом, построение большой программной системы, реализующей имитационную модель, разбивается на вполне обозримые, слабо зависящие друг от друга фрагменты декларативного описания моделей-компонент и моделей-комплексов, а также написание на универсальных языках программирования, но в функциональной парадигме, методов-элементов и методов-событий моделей-компонент.

Отладка декларативных описаний моделей-компонент, моделей-комплексов, а также функциональных программ, реализующих методы, происходит по принципу – отладил и забыл. Так же как, например, при проектировании изделий микроэлектроники, про модель-компоненту достаточно знать, какие функции она выполняет, какие у нее входы и выходы, после чего ее смело можно включать в любой комплекс с полной уверенностью, что она будет функционировать правильно, если ее входы и выходы правильно скоммутированы внутри этого комплекса. Императивное программирование при этом полностью исключается из проекта, что облегчает его отладку.

Вычислительный процесс имитационного моделирования (аксиома поведения R_{11} рода структуры «модель-компонента»), организуется так, что чем сложнее фрактальная конструкция модели, тем более высокая степень параллельности кода, который производит программа, реализующая

аксиому R_{11}, тем большее количество методов может быть вызвано параллельно.

Изложенная концепция модельного синтеза применима в первую очередь для синтеза моделей сложных многокомпонентных систем. Однако можно надеяться, что подобный подход применим и для разработки сложных программных систем, организация которых атомистична и укладывается в описанные выше требования гипотезы о замкнутости, а также требования однозначности и детерминированности вычислений.

Также есть надежда применения предложенных методов модельно-ориентированного программирования для программных систем, ориентированных на высокопроизводительные многопроцессорные вычислительные системы.

Разработанная концепция моделирования сложных систем была реализована на практике в виде ряда имитационных систем, реализованных под влиянием модельно-ориентированной парадигмы программирования, инструментальной системы имитационного моделирования MISS [30], а также в виде программного обеспечения макета рабочей станции пиринговой системы распределенного имитационного моделирования [25].

На рабочей станции системы распределенного имитационного моделирования, 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 настоящее время в отделе имитационных систем и исследования операций ВЦ РАН ведутся работы по реализации системы модельно-ориентированного программирования для высокопроизводительных вычислительных систем.

Приложение. Примеры имитационных моделей, реализованных с помощью методов модельного синтеза и модельно-ориентированного программирования

Прежде чем перейти к описанию примеров имитационных моделей, отметим, что все эти модели были разработаны с середины 80-х гг прошлого века до настоящего времени в отделе «Имитационные системы» (с 2009 г. «Имитационные системы и исследование операций») ВЦ РАН. Все они являются продуктами коллективного труда сотрудников отдела, в том числе и автора, а иногда также и сотрудников дружественных организаций. Вклад автора в создание этих моделей (кроме тестовых моделей для пиринговой сети распределенного моделирования, описанных в разделе П.3 и целиком выполненных автором) состоял, в основном, в общей организации их имитационных вычислений и системной интеграции.

П.1. Моделирование элементов Стратегической Оборонной Инициативы Р. Рейгана

Стратегическая Оборонная Инициатива (СОИ), известная также в народе как программа «Звездных войн» (первые фильмы культовой саги Дж. Лукаса как раз уже успели набрать популярность) – долгосрочная программа создания системы противоракетной обороны (ПРО), с элементами космического базирования, позволяющая поражать также наземные цели из космоса. Была провозглашена президентом США Р. Рейганом в марте 1983. В октябре 1986г. М. Горбачев заявил, что если США создадут, реализуя СОИ и растрачивая деньги, трехслойную систему ПРО, то ответ Советский Союз найдет – ответ асимметричный. Программа СОИ так и не была

развернута, не в последнюю очередь из-за того, что имитационное моделирование показало ее неэффективность против одновременного массового запуска баллистических ракет.

П.1.1. Закон сохранения момента количества движения

Первая из имитационных моделей, реализованных в отделе «Имитационные системы» ВЦ АН СССР на тему «Звездных войн», воспроизводила сценарий нападения на навигационную систему противника силами наземного и космического базирования. В этом сценарии участвует множество различных объектов как с одной, так и с другой стороны. Перечислим эти объекты.

Сторона А.

Орбитальная группировка из 18 навигационных спутников, летающих на высоких орбитах порядка 20000 км. В данном сценарии их навигационная активность не моделировалась – они служили лишь объектами нападения. В этом качестве были важны их возможности защиты, которые состояли в ресурсе для маневра – возможности некоторого изменения траектории за счет включения двигателя маневрирования, возможности выпуска четырех ложных целей – объектов трудноотличимых от самих навигационных спутников, и, наконец, наличие противоракеты, которую можно направить на нейтрализацию атакующей спутник ракеты.

Орбитальная группировка из 3 спутников наблюдения на геостационарной орбите. Эти спутники отслеживают все наземные и космические

старты, а также передвижения космических объектов, что позволяет с помощью наземного аналитического центра рассчитывать траектории чужих космических объектов и, например, своевременно обнаруживать угрозу нападения на навигационные спутники.

Орбитальная группировка из 3 спутников-ретрансляторов на геостационарной орбите. Эти спутники позволяют постоянно осуществлять двустороннюю связь между наземным командным пунктом и любым из спутников любой орбитальной группировки.

Наземные командный и аналитический центры. Аналитический центр рассчитывает траектории космических объектов противника. Командный центр отдает команды орбитальным группировкам, например, включить средства защиты, или использовать ресурс для маневра. Панель оператора командного центра выводится на экран компьютера с целью задания команд стороны А во время имитационного эксперимента.

Сторона Б.

Сторона Б также располагает орбитальными группировками трех спутников наблюдения и трех спутников-ретрансляторов на геостационарной орбите, которые несут те же функции и имеют те же характеристики, что и у стороны А.

Орбитальная группировка из 18 боевых станций, летающих на низких орбитах порядка 500 км, каждая из которых вооружена тремя боевыми ракетами – одно из возможных средств нападения на навигационные спут-

ники противника. При выходе боевой ракеты в район атакуемого спутника, она имеет некоторый ресурс топлива для наведения на цель. При приближении ее к спутнику, вероятность его поражения считалась равной 0,8. Если за отведенное для атаки время сближение с атакуемым объектом не происходило – ракета самоликвидировалась.

Наземная группировка ракет шахтного базирования – еще одно из возможных средств нападения на навигационные спутники противника. Также как и ракета космического базирования, при подлете к объекту имеет некоторый ресурс для наведения на него, при сближении поражает объект с вероятностью 0,8, а в случае неудачи – самоликвидируется.

На земле, кроме ракет шахтного базирования, находятся аналитический и командный центры. Аналитический центр с помощью результатов наблюдения рассчитывает траектории космических объектов противника, решает задачу целераспределения (кто на кого будет нападать), в случае принятия решения о нападении на орбитальную группировку противника. Командный центр принимает решения и выдает команды. Панель оператора командного центра выводится на экран. На ней показывается сколько траекторий спутников противника уже известно (рассчитано). Имеется возможность инициировать нападение на спутники с уже известными траекториями, при этом можно выбирать следующие варианты нападения:

1. Нападение только космическими средствами.
2. Нападение только наземными средствами.

3. Нападение наземными и космическими средствами с преимуществом космических средств (сначала рассчитывается задача целераспределения для космических средств, а затем для неохваченных целей – для наземных).
4. Нападение наземными и космическими средствами с преимуществом наземных средств.

Имитационные эксперименты с моделью

Одна из неожиданностей, с которыми столкнулись разработчики модели, когда та была уже более или менее отлажена – то что при решении задачи целераспределения, практически все цели отдаются наземной группировке. В то же время мириться с этим не хотелось – слишком много в то время говорилось и писалось о «Звездных войнах» – разработчикам модели было интересно попробовать, что это такое. Собственно этим и объясняется наличие четырех способов нападения в приведенном выше меню оператора командного центра стороны Б.

При попытке напасть на спутники противника исключительно силами космической группировки, чаще всего оказывалось, что сразу после отдания команды никакого нападения не происходит (в отличие от работы наземной группировки, где дана команда напасть – и ракеты сразу полетели). Старты ракет космического базирования могли произойти с различными задержками до нескольких часов. Попытки объяснить, почему так

происходит, вывели на известный всем со школы закон сохранения момента количества движения.

Действительно, боевые станции движутся по орбите с первой космической скоростью порядка 8 км/сек, а энерговооруженность размещенных на них ракет сравнительно невысока – они могут дать приращение скорости до 2 км/сек. Отсюда следует, что ракета космического базирования в известной мере «привязана» к плоскости своей орбиты и может оперировать лишь в некоторой не слишком большой окрестности этой плоскости. Следовательно, чтобы поразить спутник противника, она должна лететь не куда попало – а в одну из точек пересечения орбиты атакуемого спутника с плоскостью орбиты самой станции, куда она имеет шанс долететь. При этом начинать полет приходится не когда угодно, а так, чтобы к моменту подлета к упомянутой точке пересечения орбит, спутник противника тоже бы подлетал к ней и точка пересечения орбит стала бы также и точкой встречи атакующей ракеты с навигационным спутником. Понятно, что время старта при этом будет квантоваться, и квант по порядку величины сравним со временем прохождения спутником витка своей орбиты. Именно этим и объясняются задержки старта ракет космического базирования после отдания приказа об атаке.

Данный факт, как мы увидим в следующем разделе, существенно ограничивает применение боевых ракет космического базирования.

Выводы

Основной вывод из создания и эксплуатации описанной выше модели, применительно к целям и задачам настоящей работы состоит в том, что это было первое применение методов модельного синтеза и модельно-ориентированного программирования для создания имитационных моделей сложных многокомпонентных систем. И это применение оказалось весьма успешным.

Разработка модели велась с осени 1986 г. коллективом отдела «Имитационные системы» ВЦ АН СССР, состоящим тогда более чем из 20 человек. До января 1987 г. это были в основном теоретические работы посвященные погружению в предметную область в виде семинаров по изучению небесной механики [41]. Вычислительные мощности отдела в то время были более чем скромны – одна PC XT, а конкуренция за машинное время среди сотрудников отдела очень высока – при равномерном его распределении каждому доставалось около часа в сутки. Тем не менее, к началу 1988 г. модель уже функционировала. В значительной мере успех разработки был основан на применении парадигмы модельно-ориентированного программирования (без применения каких-либо специальных инструментальных средств, облегчающих применение этой парадигмы). Положительную роль сыграла характерная для модельно-ориентированного программирования декомпозиция задачи на множество слабо связанных блоков, каждый из которых можно отлаживать по отдель-

ности независимой группой разработчиков, не заботясь при этом о будущем их синтезе.

В дальнейшем, найденные при создании описываемой модели идеи и методы моделирования сложных многокомпонентных систем, были воплощены в инструментальной системе MISS [30], которая функционировала уже в 1989 г.

П.1.2. «Бриллиантовые камни»

После распада СССР и фактического прекращения военного противостояния двух сверхдержав, идея СОИ в большой мере потеряла внутри США привлекательность и актуальность. Тем не менее, работы в этом направлении не прекратились сразу. В течении нескольких лет эти работы приняли в США форму проектирования так называемой системы GPALS (Global Protection Against Limited Strikes - Глобальная защита от ограниченных ударов). Система GPALS мыслилась как защита от ядерного терроризма, вероятность которого в связи с распространением ядерного оружия неуклонно повышается. Основная часть этой системы - ее космический рубеж, по проекту авторов должен был состоять из большого количества (до 100000) легких (около 40 кг.) противоракет, снабженных инфракрасной системой самонаведения. Эти противоракеты получили в США название "brilliant pebblles" (бриллиантовые камни).

Модель «Бриллиантовые камни» была реализована в самом начале 90-х А.А. Левиковым [60]. К этому времени из экспериментов с предыду-

щей моделью было известно основное «узкое место» рассматриваемой предметной области – низкая энерговооруженность противоракет, по сравнению с имеющимся у них моментом количества движения. Поэтому была возможность не воспроизводить все действо в точности, как в предыдущей модели, а лишь схематично отразить его важнейшие этапы. Кроме того, к этому времени в распоряжении исследователей уже имелась инструментальная система имитационного моделирования MISS – инструмент, поддерживающий концепцию модельно-ориентированного программирования.

Для изучения возможностей системы GPALS «Бриллиантовые камни», в рамках интегрированной инструментальной системы имитационного моделирования MISS, А.А. Левиковым была разработана проблемно-ориентированная интерактивная имитационная система PEBBLES.

Ниже описываются модели, лежащие в основе системы PEBBLES и результаты расчетов, выполненных с ее помощью и предназначенных для предварительного анализа эффективности системы ПРО космического базироваия, основанной на технологии "brilliant pebbles".

В состав космического рубежа ПРО входят находящиеся на околоземных орбитах называемые далее "камнями" объекты-исполнители атаки на взлетающие межконтинентальные баллистические ракеты (МБР). Прочие объекты космического рубежа в описываемой системе агрегированы в один объект, называемый центр ПРО. На первом этапе расчетов для каж-

дой из стартовавших МБР определяется список "камней", которые могут атаковать данную МБР на стадии активного участка (АУ) ее полета. Данная задача для рассматриваемых ниже расчетов решалась при следующих предположениях.

Относительно конфигурации рубежа ПРО космического базирования предполагалось, что

- конфигурация имеет симметричную структуру и состоит из круговых орбит одинакового радиуса и одинакового наклонения;
- все орбиты одинаково разнесены друг от друга по долготе восходящего узла;
- на каждой орбите находится одинаковое количество станций, которые равномерно разнесены по широте.

Конфигурацию рубежа ПРО задают параметры:

- высота орбит "камней" (км)
- наклонение орбит "камней"(градус)
- количество орбит "камней"
- количество"камней" на орбите

В проведенной серии вычислений позиционный район (область, откуда стартуют МБР) считается точечным. Его местоположение определяется двумя параметрами

широта позиционного района (градус)

долгота

позиционного района (градус)

Все стартующие МБР из позиционного района МБР летят по идентичным траекториям.

Считается, что каждая из станций системы ПРО на своем борту имеет единственную ракету, обладающую запасом характеристической скорости V_x: предполагается, что в момент старта изменение скорости ракеты происходит мгновенно, причем модуль изменения не превосходит V_x. После выстрела ракета летит к цели по какой-либо из кеплеровских траекторий.

Для каждой станции ПРО начало атаки на МБР становится возможным лишь после того как будет идентифицирована траектория МБР.

Идентификация считается законченной, если приводимые ниже условия наблюдения за МБР выполняются в течение промежутка времени, длительность которого больше или равна величине

время идентификации траектории для МБР (с)

Считается, что на борту каждой стации системы ПРО жестко закреплен прибор наблюдения, поле обзора которого задается пересечением конуса с осью, находящейся в плоскости движения станции, и шара с центром в местоположении станции радиуса

максимальная дистанция наблюдения (км)

Конус обзора задается углом между его осью и направлением к центру земли и углом между его осью и образующей :

угол идентификации траектории для МБР (градус)

угол оси визирования с радиусом (градус)

Считается, что условия наблюдения МБР станцией выполняются, если МБР находится в поле обзора прибора наблюдения станции.

Активный участок траектории задается параметрически. При этом как явные функции времени задаются модуль радиус-вектора МБР в геоцентрической прямоугольной системе координат, ось Z которой направлена по оси вращения Земли, ось X - в точку весеннего равноденствия, ось Y - перпендикулярно плоскости XZ и так, чтобы оси X,Y,Z составляли правую тройку, и угол между этим векторами и вектором, характеризующим местоположение МБР в момент ее старта.

Параметризация устроена так , чтобы выполнялись условия непрерывности по положению и скорости МБР на концах АУ. Диапазоны изменения параметризуемых величин задаются следующими позициями :

время активного участка МБР (с)

угол поворота активного участка МБР (градус)

максимальная высота активного участка МБР (км)

Особой частью АУ является его начало, длительность которого задается позицией

время" задержки" при наблюдении МБР (с)

Эта величина есть минимально возможное время, разделяющее старт и начало наблюдения за МБР каким-либо "камнем". Содержательная ин-

терпретация данной величины зависит от моделируемой схемы управления атакой, из которых далее рассматриваются следующие варианты

Существует централизованное управление "камнями", цель которого – нейтрализовать произошедший старт МБР. Первый критерий, посредством которого можно формализовать данную цель и который, по-видимому, наиболее приемлем для оценки качества управления в случае широкомасштабных боевых действий, выглядит следующим образом: максимизировать математическое ожидание уничтоженных в результате атаки МБР.

Однако возможна ситуация, когда для противоположной стороны факт выживания хотя бы одной (в общем случае k) МБР столь же трагичен, как и факт выживания большего числа МБР. В такой ситуации уместна следующая формализация: минимизировать вероятность события, состоящего в том, что после атаки уцелеет хотя бы одна (в общем случае k) из стартовавших МБР. Этот критерий, по-видимому, особенно уместен при анализе надежности космического рубежа, проектируемого в рамках международной системы безопасности.

В модели предполагается, что для любой пары "камемнь"-МБР вероятность успеха атаки одна и та же и равна некоторому P = const (в расчетах P = 0.8). Оказывается, что в этом случае оптимальное целераспределение одно и то же для любого из перечисленных критериев.

Процесс управления состоит из следующих этапов. После того, как взлетающие МБР проходят слой облаков, регистрируется факт их старта. Далее, центр ПРО решает задачу оптимального распределения атакующего потенциала, учитывая при этом ограничения, обусловленные возможностями выполнения успешной атаки, и доводит результат до исполнителей. В модели перечисленные этапы управления учитываются неявным образом – параметру "задержка" наблюдения присваивается значение равное сумме времен протекания всех этапов.

Централизованное управление атакой отсутствует - каждый "камень" автономно выбирает цель. Считается, что в качестве объекта атаки выбирается равновероятно любая из тех МБР, на которые возможна успешная атака. В этом случае параметру "задержка" наблюдения присваивается значение равное времени прохождения взлетающими МБР слоя облаков.

Отметим, что варьируя параметры блоков модели можно количественно оценить некоторые гипотезы относительно информированности центра управления и различных схем обмена информацией в системе ПРО с точки зрения эффективности системы в целом.

В сответствии с приведенными ранее соображениями в качестве показателей эффективности космического рубежа, действующего против плотного старта МБР из точечного позиционного района, рассматривались следующие величины:

FL – математическое ожидание числа уцелевших МБР;

F1(F2, F3) – вероятность наступления события, состоящего в том, что после атаки уцелеет 1(2, 3) МБР и более. Отметим, что для данного космического рубежа каждый показатель является монотонно возрастающей функцией от числа стартовавших МБР.

Для определения возможностей конкретного космического рубежа каждому из приведенных показателей, исходя из содержательных соображений, a priori задается некоторое пороговое значение. Далее для каждого показателя рассчитывается наибольшая интенсивность старта МБР, при которой значение показателя меньше выбранного порога. Определенная таким образом для каждого из показателей величина интерпретируется как максимальная интенсивность старта, которая еще успешно нейтрализуется космическим рубежом ПРО с данными техническими характеристиками. Таким образом, эту величину можно считать показателем тех целей, которые предназначен решать космический рубеж ПРО.

В том случае, если определяемая таким образом максимальная интенсивность старта равна нескольким единицам МБР, условимся считать, что данный космический рубеж предназначен для борьбы с возможными актами терроризма и несанкционированными пусками МБР. Если рубеж нейтрализует старт нескольких десятков МБР, будем считать, что он предназначен для блокировки массового старта МБР.

В расчетах для критериев FL, F1, F2, F3 использовались пороговые значения 1, 0.1, 0.1, 0.1 соответственно.

Основные внешние величины изложенной модели, называемые далее "базовые параметры", для которых выполнялись расчеты, приведены в следующей таблице

Высота орбит "камней" (км)	500
Наклонение орбит "камней" (градус)	90
Широта позиционного района (градус)	45
Долгота позиционного района (градус)	30
Время активного участка МБР (с)	290
Время " задержки" при наблюдении МБР (с)	72
Максимальная дистанция наблюдения (км)	2000
Угол поворота активного участкаМБР (градус)	7
Максимальная высота активного участка МБР (км)	350
Время идентификации траектории МБР (с)	30
Угол идентификации траектории МБР (градус)	60
Угол оси визирования "камней" с радиусом (градус)	30
Вероятность успеха единичной атаки	0.8

Приведем некоторые результаты расчетов для базовых значений параметров. Рассматриваются рубежи ПРО космического базирования, имеющие следующую структуру.

Номер рубежа ПРО	Число орбит	Количество на орбите	Всего
1	60	60	3600
2	90	60	5400
3	120	60	7200
4	120	120	14400
5	180	120	21600
6	180	180	32400

В следующей таблице для случая централизованного управления камнями для ряда значений характеристической скорости и для каждого из перечисленных выше шести номеров рубежей ПРО указано количество камней, которые атакуют МБР.

	Номер рубежа ПРО					
V_x км/сек	1	2	3	4	5	6
2	1	2	4	7	13	20
4	7	12	21	39	55	83
6	15	24	35	69	107	164
8	24	38	50	101	155	231

В следующей таблице при тех же условиях, для тех же значений внешних величин и для каждого из критериев --- FL, F1, F2, F3 указано максимальное количество МБР (стартующих одновременно из точечного позиционного района) при котором значение соответствующего показателя меньше порогового значения (порог для FL был равен 1, для F1, F2, F3 порог был равен 0.1).

		Номер рубежа ПРО					
V_x км/сек	Критерий	1	2	3	4	5	6
2	FL	1	2	4	5	8	11
2	F1	0	1	2	3	5	7
2	F2	1	2	3	4	7	10
2	F3	2	2	4	6	9	12
4	FL	5	8	12	20	26	35
4	F1	3	4	7	13	16	23
4	F2	4	6	10	17	22	30
4	F3	6	8	12	20	27	36
6	FL	9	13	18	30	42	60
6	F1	5	8	11	20	29	42
6	F2	8	12	15	26	38	55
6	F3	9	13	18	31	43	61
8	FL	13	19	24	40	57	80
8	F1	8	12	15	27	40	58
8	F2	12	16	20	36	62	73
8	F3	13	20	25	41	68	81

Согласно принятой выше точке зрения, можно считать, что космические рубежи ПРО, характеризующиеся парами

V_x	2	2	2	2	4
Структура	1	2	3	4	1

предназначены для борьбы с актами терроризма и несанкционированными пусками МБР.

Космические рубежи ПРО, характеризующиеся парами

V_x	4	4	4	6	6	6	6	8	8	8	8	8
Структура	4	5	6	3	4	5	6	2	3	4	5	6

нейтрализуют старт нескольких десятков МБР и, следовательно, предназначены для блокировки массового старта баллистических ракет.

Выводы

Как уже было отмечено выше – основная проблема рассматриваемой предметной области – низкая энерговооруженность противоракет, по сравнению с имеющимся у них моментом количества движения. Если обратить внимание на один из ключевых параметров модели – V_x, то про значение $V_x = 2$, можно сказать что это то, что реально выпускалось промышленностью того времени. Про значение $V_x = 4$ можно сказать, что за этим значением стоит естественное желание улучшения тактико-технических характеристик противоракет, плюс надежды на научно-технический прогресс. Что же касается значений $V_x = 6$ и $V_x = 8$, – это уже совсем из области фантастики, т.е. теоретически и такое возможно, но это уже были бы не

дешевые «камни», а гораздо более сложные технически и, соответственно, намного более дорогие устройства. При этом выводить на орбиту и затем годами эксплуатировать пришлось бы десятки тысяч подобных устройств. И что самое обескураживающее – даже если предположить, что реализован самый мощный рубеж ПРО из 32400 самых энерговооруженных «камней»,

V_x	8
Структура	6

то при массовом одновременном пуске сотни ракет (что вовсе не является чем-то совсем уж невероятным в случае серьезного противника) – несколько ракет его обязательно преодолеют.

Собственно, по этой причине подобные системы ПРО космического базирования и не были развернуты.

П.2. Распределенная модель взаимодействия нескольких государств на основе Экономико-Демографо-Экологической Модели (ЭДЭМ)

Описываемая ниже модель взаимодействия трех стран была выполнена в отделе «Имитационные системы» ВЦ РАН в 2004-2010 гг. в рамках проводившихся с 2001 г. исследований проблематики устойчивого развития мирового сообщества, описанных в работах [18 – 22, 28, 29, 44, 72]. В контексте данной работы, помимо исследований с помощью имитации предметной области модели, нас в значительной мере будут интересовать такие технические вопросы, как различные способы организации распределенных вычислений модели трех стран, где применялись идеи модельного синтеза и модельно-ориентированного программирования.

«Устойчивое развитие мирового сообщества» так, как оно здесь понимается, является гуманитарным понятием, означающее его «плавное» развитие без революций в отдельных его частях и прочих «потрясений». В возникновении этого понятия некоторую роль сыграло понятие «sustainable development», которое принято (у нас) переводить именно как «устойчивое развитие» и использующееся при описании влияния человеческого общества на окружающую среду. Однако «устойчивое развитие», это не очень точный перевод термина «sustainable development». Более точный перевод этого понятия является «самоподдерживающееся развитие». Под термином «sustainable development» понимается такое влияние человеческого общества на окружающую среду, которое не оказывает отрицательного воздействия на ее основные параметры.

Принцип ─ «sustainable development» был декларирован на международной конференции глав государств в Рио-де-Жанейро, посвященной проблемам окружающей среды. Этот принцип подразумевает следующее:

- человечество стремится к выживанию и удовлетворению своих потребностей и в то же время не ставит под угрозу способность будущих поколений выживать и удовлетворять их собственные потребности, а также заботиться о сохранении биоразнообразия на Земле, так как все живое имеет право на жизнь, как сейчас так и в будущем;
- человек зависти от Земли и ее ресурсов и не рискует превышать ограничения и поддерживающую емкость ее систем, налагаемые ко-

нечностью Земли, ее живых и неживых составляющих, подвергая опасности сам факт существования жизни на Земле;

- мир человека и природы, мир человека в природе характеризуется устойчивостью, способностью к самоподдержанию и отсутствием кризисов антропогенного происхождения.

Было решено сохранить за словосочетанием «sustainable development» смысл «самоподдерживающееся развитие», понимая этот термин как взаимное влияние человеческого общества и окружающей среды, а термину «устойчивое развитие» придать тот смысл, который был описан выше, т.е. как «плавное» развитие мирового сообщества и его составных частей без революций и резких потрясений.

Проблема устойчивого развития мирового сообщества, так как она будет здесь пониматься, носит системный характер: она имеет экономический, финансовый, политический, экологический, военный, миграционный, культурологический, образовательный аспекты. Все эти аспекты проблемы взаимосвязаны и их системному анализу средствами МГ будет посвящен специальный раздел. В возникновении представления об экологическом аспекте проблемы устойчивого развития, т.е. проблемы «sustainable development», существенную роль сыграла модель «Мировая динамика» (МД) Дж. Форрестера [85], а также работы Д.Л. Медоуза [54]. Модель МД предсказывает кризис в середине текущего столетия, связанный с ростом

численности людей на Земле, загрязнением окружающей среды, исчерпанием природных ресурсов.

Эта модель, однако, не учитывает технологический прогресс, т.е. появление новых технологий, эволюцию производственной структуры и структуры потребления. Мировая производственная структура является «организмом», т.е. содержит в себе механизмы самосохранения, саморегулирования. Например, исчерпание запасов нефти и газа, предсказываемое как раз к середине текущего столетия, является, конечно, серьезной проблемой, однако, возможно сделать так, чтобы это исчерпание не было катастрофично. Есть много альтернативных источников энергии. Все дело в развитии соответствующих технологий и получении знаний, которые необходимы для этого. Без учета тех факторов, о которых шла речь выше (новые источники энергии, новые технологии, новые товары, изменение структуры потребления и т.д.), проблемы устойчивого развития мирового сообщества решать неестественно.

Экологическая часть проблемы устойчивого развития осложнена тем, что в настоящее время мы не располагаем знаниями о физических, химических, биологических, экологических процессах на Земле, достаточных для достоверного прогноза эволюции этих процессов в результате антропогенного влияния на них. Может быть, не нужно вовсе заботиться о влиянии производственной деятельности на окружающую среду. Но может быть и так, (это маловероятно, но не исключено полностью) что человече-

ство не сумеет приспособиться к неожиданным и «быстрым» процессам в биосфере и климатической системе, связанными с таким влиянием. Особое место занимают социально-экономические проблемы, которые обостряются в связи с ростом населения, его миграцией, возрастающим разрывом между богатыми и бедными странами.

Высказанные соображения явились основанием для попытки разработать более сложную модель, ориентированную по-прежнему на проблему «sustainable development», которая учитывает технологический прогресс, появление новых товаров, изменение структуры потребления и т.д.

На первых порах главным в этой модели были выбросы загрязнителей в окружающую среду и платы фирм за эти выбросы. Платы за выбросы загрязнителей в окружающую среду означают, что человечество, чтобы «существовать», должно «производить» параметры окружающей среды также, как оно производит другие материальные блага. Модель пытается ответить на вопрос, до какой степени сейчас при данном уровне развития технологнй рационально производить эти параметры.

Построенная модель была названа ЭДЭМ (экономико-демографо-экологическая модель). В ее построении участвовали наряду со специалистами ВЦ РАН также специалисты РХТУ [18 – 22, 28, 29, 44, 72]. Эта модель описана ниже на гуманитарном уровне. Она дала возможность авторам провести системный анализ проблемы устойчивого развития мирового

сообщества, понимаемого (см. выше) как его «плавное развитие» без революций и прочих резких потрясений.

Описание этой достаточно сложной модели начнем с описания взаимодействующих экономических, демографических и экологических процессов внутри одной страны.

П.2.1. Общее описание имитационной эколого - демографо - экономической модели

На самом общем уровне имитационную эколого - демографо – экономическую имитационную модель (ЭДЭМ) можно описать следующим образом. В модели имеются объекты двух видов - люди и производственные фонды, которые относятся к факторам производства в экономической подмодели. Каждый из этих двух видов объектов обладает присущими им характеристиками: люди - полом, возрастом, экономической активностью и уровнем образованности; производственные фонды - мощностью, эксплуатационным возрастом и технологией.

Мощность - это максимальное количество продукта, который в единицу времени (год) способны произвести данные фонды при полной обеспеченности их ресурсами.

Технология - это количество продукта, необходимое для строительства единицы производственных фондов (фондообразующие коэффициенты), количество природных и трудовых ресурсов различного уровня образованности, необходимых для выпуска единицы продукции по данной тех-

нологии (коэффициенты ресурсоемкости и трудоемкости) и уровень токсичности выбросов загрязнителей на единицу выпускаемой продукции.

Технологии делятся на «чистые» и «грязные». И те, и другие виды объектов (люди и производственные фонды) рождаются в некоторый момент, живут некоторое время и умирают (естественным образом для людей, путем списания - для производственных фондов). Для того чтобы производственные фонды рождались и функционировали необходимы инвестиции, и труд людей разных уровней образованности в соответствии с фондообразующими коэффициентами и коэффициентами трудоемкости. Чтобы люди рождались и предлагали свой труд, необходима выпускаемая имеющимися и функционирующими (т.е. обеспеченными трудовыми ресурсами) фондами продукция, «чистая» и «грязная», - в соответствии со структурой потребления. Ни технологии, ни структура потребления на характерных временах порядка нескольких десятилетий не являются постоянными. Технологии производятся фундаментальной и прикладной наукой в соответствии со средствами, которые в это вкладываются.

Структура потребления, т.е. распределение спроса между «чистой» и «грязной» продукцией, расходы на здравоохранение и образование зависят от уровня образованности людей и от государственных средств, которые выделяются на образование. Доля чистой продукции в потреблении людей характеризует степень, в которой чистота окружающей среды является для них потребительским качеством.

Ниже дается более подробное описание ЭДЭМ с указанием, каким образом бюджетный процесс связан с характеристиками, обеспечивающими устойчивое развитие моделируемого мира.

П.2.2. Модель демографического процесса

Модель демографического процесса воспроизводит эволюцию половой и возрастной структуры населения в странах. Внутренними и одновременно прогностическими характеристиками являются количества в данном году женщин и мужчин, имеющих данный возраст (от 1 до 100). К внешним характеристикам относятся распределение населения в стране по полу и возрасту в начальном году, т.е. в году, с которого начинается модельное воспроизведение демографического процесса, а также зависимости коэффициентов рождаемости (количество родившихся детей от женщин данного возраста (от 15 до 50)в течение года в расчете на одну женщину этого возраста) и смертности (количество умерших женщин (мужчин) в течение года в расчете на одну женщину (мужчину)) от возраста, уровня образованности, уровня токсичности окружающей среды, расходов на здравоохранение.

Основа модели - пересчет распределения населения по полу и возрасту в данном году на следующий год. Для этого сначала с помощью известных коэффициентов рождаемости рассчитывается количество мальчиков и девочек, которые родились в данном году и поэтому будут иметь в следующем году возраст в 1 год (те из них, кто доживет до этого возраста).

Затем с помощью известных коэффициентов смертности для каждого пола, каждого возраста рассчитывается количество людей, которые в следующем году станут старше на год, т.е. которые доживут до своего следующего дня рождения. Прогностическими характеристиками, кроме упомянутых выше, также являются:

- общая численность населения;
- общая численность мужчин;
- общая численность женщин;
- количества родившихся и умерших в данном году;
- численность детей (возраст от 1-го года до 6);
- численность школьников (возраст от 7 лет до 17)
- численность студентов (возраст от 18 лет до 22); эта величина определяется по расходам государства и домашних хозяйств на высшее образование;
- численность аспирантов (возраст от 22 лет до 24); эта величина определяется по расходам государства и домашних хозяйств на послевузовское образование;
- численность рабочих (возраст от 18 лет до 59); рабочими считаются мужчины и женщины, окончившие школу, но не получившие высшего образования; по численности рабочих вычисляется предложение труда рабочих на рынок труда;

- численность служащих (возраст от 23 лет до 59); служащими считаются мужчины и женщины, получившие высшее образование, но не окончившие аспирантуру; по численности служащих вычисляется предложение труда служащих на рынок труда;
- численность ученых (возраст от 25 лет до 59); учеными считаются мужчины и женщины, окончившие аспирантуру; по численности ученых вычисляется по соответствующей модели предложение труда ученых на рынок труда;
- численность активного населения, т.е. суммарная численность рабочих, служащих, ученых;
- численность пенсионеров;
- отношение численности пенсионеров к численности активного населения.

П.2.3. Производственные мощности, технологии, экономические агенты

В модели экономики учитываются следующие виды экономической деятельности: добыча природных ресурсов, производство предметов потребления («чистых» и «грязных»), образование, здравоохранение, экологическая деятельность, производство новых знаний и новых технологий (фундаментальная и прикладная наука). Вся производственная структура, которая загрязняет окружающую среду, сагрегирована, таким образом, в один вид экономической деятельности. Тем самым считается, что в моде-

лируемом виртуальном мире образование, здравоохранение, производство новых знаний и новых технологий окружающую среду не загрязняют.

Основными характеристиками, описывающими производственный процесс, загрязняющий среду в виртуальном мире, являются производственные мощности, характеризуемые технологией. Экономические агенты - это субъекты, влияние которых на функционирование экономики описывается в модели. В ЭДЭМ имеется три типа экономических агентов:

- правительство виртуального мира (олицетворяемое игроками),
- частные фирмы,
- домашние хозяйства.

Правительство имеет возможность получать денежные средства из рядя источников, а также распределять эти средства на следующие мероприятия (см. таблицу):

Распределение денежных средств

Поступление денежных средств:	Распределение денежных средств:
сбор налогов с прибылей фирм (налог на прибыль)	восстановление расходуемого природного ресурса
штрафы на выбросы загрязнителей (экологический налог)	мероприятия по очистке окружающей среды
налог с получаемой в домашних хозяйствах зарплаты (подоходный налог)	образование, здравоохранение
платы фирм за использование природных ресурсов (рента)	инвестиции в технологии и на производство самих технологий

Ставки налогообложения и распределение расходов по указанным статьям являются государственным управлением (т.е. внешними характеристиками). Количество природного ресурса в ЭДЭМ зависит от его использования для производства продукции в соответствии с коэффициентами ресурсоемкости, процессом самовосстановления природного ресурса и процессом его восстановления посредством соответствующих мероприятий, на которые должны быть выделены средства из бюджета. Соответствующие коэффициенты относятся к внешним характеристикам. Частные фирмы инвестируют в технологии, приносящие прибыли. Механизм, в силу которого инвестиции распределяются по имеющимся и новым технологиям, описывается ниже. Домашние хозяйства тратят получаемую заработную плату в соответствии со структурой потребления.

Структура потребления в ЭДЭМ - это доли истраченных домашними хозяйствами заработанных продажей своего труда на рынке труда средств (т.е. зарплаты) на приобретение «чистой» и «грязной» продукции, а также на образование и здравоохранение. Другими словами, предполагается, домашние хозяйства не делают сбережений и весь свой доход тратят на потребление.

П.2.4. Модель производственных процессов

Основная характеристика производственного процесса - распределение производственных фондов по времени, эксплуатационному возрасту и технологиям. Напомним, что основным значением этой характеристики

является мощность, т.е. максимально возможный выпуск продукции данными фондами с данным эксплуатационным возрастом и данной технологией в единицу времени (т.е. в год).

В каждом году для каждого эксплуатационного возраста каждой технологии вычисляется рентабельность данной мощности. Для этого от рыночной стоимости единицы продукции (см. ниже описание модели рынка) вычитаются затраты, которые необходимо осуществить для того, чтобы эту единицу продукции произвести. Затраты состоят из стоимости природных ресурсов, необходимых для производства единицы продукции по данной технологии, из рыночной стоимости труда различного уровня образованности, необходимого для производства единицы продукции по данной технологии, из стоимости эксплуатации данной мощности, зависящей от ее эксплуатационного возраста, а также из выплат за уровень токсичности загрязнителей, выбрасываемых в окружающую среду в расчете на единицу выпуска. Если рыночная стоимость единицы продукции, произведенной данной мощностью, превышает затраты, то данная мощность считается рентабельной.

Рентабельные мощности предъявляют спрос на труд для людей различного уровня образованности на рынке труда в соответствии с коэффициентами трудозатрат (см. ниже описание модели рынка труда). Предложение труда на рынок труда вычисляется в демографической модели. Таким образом, определяется спрос и предложение труда данного уровня об-

разованности. Это дает возможность найти цену труда (ставку заработной платы) в следующем году в соответствии с моделью рынка труда. В ЭДЭМ считается, что спрос на труд всегда удовлетворяется, а при превышении трудовых ресурсов цена труда возрастает

Далее вычисляется потребление продукции. Произведенную чистую и грязную продукцию потребляют домашние хозяйства в соответствии со структурой потребления, частные фирмы в соответствии с инвестициями, ценами на продукт и коэффициентами фондоемкости, государство в соответствии с распределением бюджетных средств, определяющим спрос на продукт (считается, что государство потребляет только чистый продукт). Потребление продукции домашними хозяйствами характеризуется структурой потребления. ***Структура потребления*** --- это доля зарплаты, которая тратится домашними хозяйствами на приобретение чистой и грязной продукции, на образование и здравоохранение.

Цены на чистую и грязную продукцию определяются в соответствии с изменением соответствующих запасов. Именно, произведенная продукция пополняет запасы на рынке. Относительное изменение цены единицы продукции пропорционально разности между так называемым «нормативным» запасом и текущим запасом продукции на рынке.

Цена труда людей различного уровня образованности определяется на соответствующих рынках труда. Считается, что относительное изменение заработной платы в течение года пропорционально разности между

спросом и предложением рабочей силы в случае, если эта разность положительна или равна нулю, в противном случае заработная плата не может уменьшаться.

П.2.5. Модель процесса загрязнения окружающей среды

Уже указывалось, что каждая технология характеризуется уровнем токсичности выбросов загрязнителей в окружающую среду на единицу производимого продукта. В модели процесса загрязнения окружающей среды в каждом году вычисляется уровень токсичности окружающей среды. Его изменение в течение года определяется суммарными выбросами функционирующих (обеспеченных рабочей силой) производственных мощностей, процессом самоочищения окружающей среды (уменьшение уровня токсичности пропорционально с соответствующим коэффициентом этому уровню) и мероприятиями по очистке среды обитания, финансируемыми правительством. Кроме того, вычисляется показатель --- качество окружающей среды, являющийся функцией от уровня токсичности. От уровня токсичности среды обитания зависят коэффициенты рождаемости и смертности в модели демографического процесса. Считается, что имеется критический уровень токсичности окружающей среды, означающий `экологическую' катастрофу в моделируемом мире. Если уровень токсичности окружающей среды достигает критического уровня, то это означает, что игроки не справились с задачей обеспечения устойчивого развития моделируемого мира. Точное значение критического уровня токсичности неиз

П.2.6. Уравнения имитационной модели

К внешним характеристикам в модели демографического процесса относятся количества $x_{m,t,a}$, $x_{f,t,a}$,соответственно, мужчин и женщин, которым в году t исполняется a лет. Характеристики $x_{m,t,a}$, $x_{f,t,a}$ являются распределениями, о которых говорилось выше. Здесь $t=t_0,t_0+1, \ldots, T$, а $a=1,2,\ldots, 100$. Любому человеку, которому в году t исполняется a лет, в следующем $t+1$ - году должно исполнится $a+1$ лет. Поэтому, если бы люди не умирали, то имели бы место следующие равенства $x_{m,t+1,a+1}=x_{m,t,a}$, $x_{f,t+1,a+1}=x_{f,t,a}$. В реальности, однако, только часть людей доживает до своего следующего дня рождения. Обозначим через $\beta_{m,t,a}$ долю мужчин среди тех, которые в момент исполнения им a лет в году t были живы, но которые не дожили до своего следующего дня рождения. По определению имеем

$$\beta_{m,t,a} = \frac{x_{m,t,a} - x_{m,t+1,a+1}}{x_{m,t,a}} \ .$$

Предположим, что характеристика $\beta_{m,t,a,ql}$, которую принято называть силой смертности, уже не зависит от $x_{m,t,a}$, $x_{f,t,a}$.Характеристика $\beta_{m,t,a}$, конечно, существенно зависит от возраста a и от качества окружающей среды. Зависимость этой характеристики от номера года t определяется политическими и социальными условиями. Обозначим через $\beta_{f,t,a}$ аналогичную величину для женщин. Тогда

$$\beta_{f,t,a} = \frac{x_{f,t,a} - x_{f,t+1,a+1}}{x_{f,t,a}} \ .$$

Перепишем эти соотношения в более удобном виде

$$x_{m,t+1,a+1}=x_{m,t,a}(1-\beta_{m,t,a,}) \,, \qquad (П.2.1)$$

$$x_{f,t+1,a+1}=x_{f,t,a}(1-\beta_{f,t,al}) \,, \qquad (П.2.2)$$

$t=t_0, t_0+1, \ldots T; a=1, 2, \ldots, 100.$

В этих уравнениях не учитывается миграция населения. Обозначим через $y_{m,t,a}$, $y_{f,t,a}$ разность между въехавшими в страну (ставшими ее гражданами) и выехавшими из страны (сменившими гражданство данной страны на гражданство другой страны), соответственно, мужчинами и женщинами, которым в году t исполняется a лет. Соотношения (П.2.1), (П.2.2) примут вид

$$x_{m,t+1,a+1}=x_{m,t,a}(1-\beta_{m,t,a,})+ y_{m,t+1,a+1} \,, \qquad (П.2.3)$$

$$x_{f,t+1,a+1}=x_{f,t,a}(1-\beta_{f,t,a})+ y_{f,t+1,a+1} \,, \qquad (П.2.4)$$

$t=t_0, t_0+1, \ldots T; a=1, 2, \ldots, 100.$

Считается, что характеристики $y_{m,t,a}$, $y_{f,t,a}$ конкретизируются на уровне имитационных экспериментов.

Обозначим через $\gamma_{m,t,a}$ количество мальчиков, родившихся от женщин, имеющих в году t возраст a в расчете на одну женщину, имеющую в году t такой возраст. Обозначим через $\gamma_{f,t,a}$ аналогичную величину для девочек. Естественно предположить, что значения характеристик $\gamma_{m,t,a}$ и $\gamma_{f,t,a}$, называемые коэффициентами рождаемости, при $a<15$ и $a>50$ равны нулю. Предположим, что характеристики $\gamma_{m,t,a}$ и $\gamma_{f,t,a}$ не зависят от $x_{m,t,a}$, $x_{f,t,a}$. Введенные характеристики позволяют написать соотношения

$$x_{m,t+1,1} = \sum_{a=15}^{a=50} x_{f,t,a}\gamma_{m,t,a}, \qquad t = t_0, t_0+1,...,T \qquad (П.2.5)$$

$$x_{f,t+1,1} = \sum_{a=15}^{a=50} x_{f,t,a}\gamma_{f,t,a}, \qquad t = t_0, t_0+1,...,T \qquad (П.2.6)$$

Для того, чтобы дать прогноз эволюции распределения населения по возрасту и полу, необходимо знать это распределение в начальном году t_0 , т.е. необходимо знать значение характеристик $x_{m,t_0,a}$, $x_{f,t_0,a}$, $a = 1, 2, ...,100$. Кроме того, необходимо знать коэффициенты смертности $\beta_{m,t,a,}$, $\beta_{f,t,al}$, $t = t, t+1, ...,T, a = 1, 2, ...,100$ и рождаемости $\gamma_{m,t,a,ql}$, $\gamma_{f,t,al}$, $t = t_0, t_0+1,...,T, a = 15, 16, ...,50$, а также величины $y_{m,t,a}$, $y_{f,t,a}$,$t = t_0, t+1, ...,T, a = 1, 2, ...,100$, характеризующие миграцию населения. В самом деле, по формулам (П.2.3) и (П.2.4), написанным для $t = t_0$ вычисляются фигурирующие в левых частях этих формул величины $x_{m,t_0+1,1}$ и $x_{f,t_0+1,1}$, поскольку все величины в правых частях этих формул известны. Аналогичным образом вычисляются величины $x_{m,t_0+2,1}$ и $x_{f,t_0+2,1}$, а затем величины $x_{m,t_0+2,a}$ и $x_{f,t_0+2,a}$, $a = 1, 2, ...,100$.

Таким образом, составленная модель замкнута, если считать в ней внутренними характеристиками $x_{m,t,a}$, $x_{f,t,a}$, $t = t_0+1,...,T, a = 1, 2, ...,100$, а внешними характеристиками $x_{m,t_0,a}$, $x_{f,t_0,a}$, $a = 1, 2, ...,100$ и $\beta_{m,t,a}$, $\beta_{f,t,a}$, $y_{m,t,a}$, $y_{f,t,a}$, $t = t_0, t_0+1, ...,T, a = 1, 2, ...,100, \gamma_{m,t,a}, \gamma_{f,t,a}, t = t_0, t_0+1,...,T, a = 15, 16, ...,50$.

Здесь $\beta_{m,a,q,t}, \gamma_{m,a,q,t}$ - коэффициенты смертности и рождаемости, соответственно. Коэффициент смертности вычисляется по формуле:

$$\beta_{m,a,q,t} = death_{m,a} \cdot deathq_q \cdot deathql(ql_t) \cdot deathwh(wh_t) \qquad (П.2.7)$$

Здесь:

$death_{m,t}$ - «номинальный» коэффициент смертности;

$deathq_q$ - мультипликативная поправка, учитывающая зависимость коэффициента смертности от уровня образованности;

$deathq(ql_t)$ - мультипликативная поправка, учитывающая зависимость коэффициента смертности от уровня качества *ql* окружающей среды;

$deathwh(wh_t)$ - мультипликативная поправка, учитывающая зависимость коэффициента смертности от расходов wh_t на здравоохранение.

Коэффициент рождаемости вычисляется по формуле

$$\gamma_{m,a,q,t} = birth_{m,a} \cdot birthq_q \cdot birthql(ql_t) \cdot birthwh(wh_t) \qquad (П.2.8)$$

Здесь:

$birth_{m,a}$ - «номинальный» коэффициент рождаемости;

$birthq_q$ - мультипликативная поправка, учитывающая зависимость коэффициента рождаемости от уровня образованности;

$birthql(ql_t)$ - мультипликативная поправка, учитывающая зависимость коэффициента рождаемости от уровня качества *ql* окружающей среды;

$birthwh(wh_t)$ - мультипликативная поправка, учитывающая зависимость коэффициента смертности от расходов wh_t на здравоохранение.

Распределения $x_{m,t,a}$, $x_{f,t,a}$, $t = t_0+1,...,T$, $a = 1,2,...,100$, позволяют рассчитать следующие демографические характеристики стран, относящиеся к каждому году:

$pop_t = \sum_{m=1}^{2}\sum_{a=1}^{100} x_{m,a,t}$ - общую численность населения;

$men_t = \sum_{a=1}^{100} x_{m,a,t}$ - общую численность мужчин;

$women_t = \sum_{a=1}^{100} x_{f,a,t}$ - общую численность женщин;

$ch_t = \sum_{m=1}^{2}\sum_{a=1}^{100} x_{m,a,t}$ - численность детей ($1 \le a \le 6$);

$sc_t = \sum_{m=1}^{2}\sum_{a=7}^{17} x_{m,a,t}$ - численность школьников ($7 \le a \le 17$);

$pens_t = \sum_{m=1}^{2}\sum_{a=apens}^{100} x_{m,a,t}$ - численность пенсионеров ($60 \le a \le 100$);

Кроме того, в модели демографического процесса вычисляются также следующие показатели, зависящие не только от возрастного распределения, но и от характеристик производственного процесса:

$LifeS_t$ - средняя продолжительность жизни;

$st_{s,t,a}$ — численность и распределение по возрасту студентов ($18 \le a \le 22$);

$as_{s,t,a}$ — численность и распределение по возрасту аспирантов (на основании данных о расходах государства и домашних хозяйств на получение высшего образования) ($23 \le a \le 25$);

$lb_{s,t,a,pr}$ — численность и распределение по возрасту рабочих (рабочими считаются люди со средним образованием старше 17 лет и моложе 60 лет, не получившие высшего образования);

$ow_{s,t,a,pr}$ — численность и распределение по возрасту служащих (служащими считаются люди с высшим образованием старше 23 лет и моложе 60 лет, не получившие высшего образования);

$sci_{s,t,a,pr}$ — численность и распределение по возрасту ученых (учеными считаются люди с послевузовским образованием старше 25 лет и моложе 60 лет).

$pact_{t,q}$. – численность активного населения, имеющих «образованность» q;

$pact_t = \sum_{q=1}^{3} pact_{t,q}$ - общая численность активного населения;

$rt_t = pens_t / pact_t$ - отношение численности пенсионеров к численности активного населения.

Рассмотрим схему функционирования «стандартной» отрасли производства в базовой модели, рынок продукции которой является мировым. На уровне реализуемой модели эта схема подвергается коррекции для каждой конкретной отрасли. Далее представим распределения, характеризующие отрасль, и запишем соотношения, позволяющие рассчитать функ-

ционирование отрасли, т.е. вычислить эволюцию прогностических характеристик отрасли.

Следующие соотношения описывают эволюцию производственных фондов:

$$m_{t+1,ag+1} = m_{t,ag} \cdot (1 - dC_{ag}) \qquad \text{(П.2.9)}$$

$$t = t_0, t_0 + 1, ..., T - 1; ag = 0,1,2,..., agm - 1$$

$$m_{t,0} = \frac{invp_t}{nF_{0,c} \cdot P_{t,c}} + \frac{\delta F_t \cdot invg_t}{nF_{0,c} \cdot p_{t,c}} + \frac{(1 - \delta F_t) invg_t}{nF_{0,c} \cdot p_{t,c}}$$

$$t = t_0, t_0 + 1, ..., T; m_{t_0,ag} = m0_{ag}; ag = 1,2,..., agm$$

Здесь:

dC_{ag} - доля выбывших в течение года производственных мощностей вследствие старения, стихийных бедствий и техногенных катастроф;

$nF_{0,cd}$ - коэффициент фондоемкости, т.е. количество продукта, необходимое для строительства единицы мощности той технологии, которая вступает в строй в момент времени t; этот коэффициент является продуктом деятельности фундаментальной и прикладной наук;

$invp_t$ - инвестиции частных фирм в строительство новых мощностей; в данном варианте модели считается, что всю прибыль, полученную от продажи произведенного товара на рынке, частные фирмы тратят на инвестиции;

$invg_t$ - инвестиции государства в строительство новых мощностей; эти инвестиции осуществляются частным инвестором, который назначается государством;

δF_t - доля тех инвестиций, на которые приобретается чистый фондообразующий продукт; эта доля характеризует экологические предпочтения инвесторов и зависит от уровня образованности населения в целом и расходов государства на образование;

$p_c(p_d)$ - мировые цены на чистый (грязный) продукты; динамика их изменений описывается аналогичными уравнениями, поэтому в дальнейшем мы будем опускать нижний индекс, если это не вызывает недоразумений.

$$p_{t+1} = p_t \cdot \left(1 - \frac{Q_t - Q0_t}{p\theta \cdot Q_t}\right); p_{t+1} \geq 0 \qquad \text{(П.2.10)}$$

Здесь:

$Q0_t$ - нормативные запасы продукции на рынке (чистой и грязной, соответственно); эти величины вычисляются по формуле:

$$Q0_t = \alpha \cdot mAll_t$$

где α - положительная постоянная, $p\theta$ - характерное время изменения запасов; $mAll_t$ - суммарная мощность производственных фондов; эта величина вычисляется по формуле:

$$mAll_t = \sum_{ag=0}^{agm} m_{t,ag}$$

Эволюция запасов Q как чистой, так и грязной продукции на рынке описывается соотношением:

$$Q_{t+1} = \max(0, Q_t + Y_t - S_t)$$

Здесь:

Y_t - общее количество произведенной продукции (чистой, грязной) в году t;

S_t - общий спрос на продукцию (чистую, грязную) в году t

Эволюция стоимости труда $w_{t,q}$ в году t и квалификации q определяются соотношениями:

$$w_{t_0,q} = w0_q$$

$$w_{t+1,q} = \begin{cases} w_{t,q}, если (lbD_{t,q} / lbS_{t,q} - 1) \le 0 \\ w_{t,q} \cdot (1 + (lbD_{t,q} / lbS_{t,q} - 1)/\theta), если \quad 1 \le (lbD_{t,q} / lbS_{t,q)} \le 2 \\ 2w_{t,q}, если (lbD_{t,q} / lbS_{t,q} - 1) > 1 \end{cases}$$

$$q = 1,2,3; t = t_0, t_0 + 1, ..., T - 1;$$

Здесь:

$lbD_{t,q}$ - спрос на труд в году t и квалификации q

$lbS_{t,q}$ - предложение труда в году t и квалификации q.

Предложение труда рассчитывается по формуле:

$$lbS_{t,q} = \varpi_{t,q} \cdot pact_{t,q} ,$$

где $\varpi_{t,q}$ - доля активного населения, предлагающего свой труд на рынке труда в году t и квалификации q

Для расчета величин Y_t, S_t, $lbD_{t,q}$, фигурирующих во введенных соотношениях, необходимо ввести ряд характеристик. Прежде всего, надо описать способ определения рентабельных мощностей, что даст возможность вычислить выпуск фирмами чистой и грязной продукции. Для этого введем следующие переменные:

$nL_{q,ag}$ - коэффициенты трудоемкости (количество нормо-часов работы людей с уровнем образования q, необходимое для выпуска единицы продукции) по чистой (грязной) технологиям; начальное значение $nL_{q,0}$ является продуктом деятельности фундаментальной и прикладной науки;

nD_{ag} - уровень загрязнения, производимый на единицу выпуска по технологии возраста ag; значение nD_0 является продуктом деятельности фундаментальной и прикладной науки;

D_t - уровень загрязнения окружающей среды;

DL критический уровень загрязнения окружающей среды, при котором происходит экологическая катастрофа;

dl_t - уровень загрязнения на единицу произведенной продукции, определяющий чистоту технологии: если $nD_{ag} \le dl_t$, то технология в момент времени t считается чистой, в противном случае – грязной;

tD_t - ставка государственного штрафа за выброс единицы загрязнителя;

$g_{t,ag}$ - часть мощности $m_{t,ag}$, которая считается чистой. У этой части мощности $nD_{ag} \le dl_t$, а остальная часть мощности $(1-g_{t,ag})m_{t,ag}$ является грязной, и выбросы загрязнителя превышают критический уровень;

nN_{ag} - коэффициент ресурсоемкости;

$z_{t,ag,cd} = \sum_{q=1}^{3} nL_{q,ag,cd} \cdot w_{t,q} + tD_t \cdot nD_{ag,cd} + nN_{ag,cd} \cdot pN_t$ - производственные затраты;

$$nD_{ag,cd} = \begin{cases} nD_{ag} & при \quad cd = d \\ dl_t & при \quad cd = c \end{cases}$$

$prf_{t,ag}$ - рентабельность производственной структуры, имеющий эксплуатационный возраст ag; эта величина вычисляется по формуле:

$$prf_{t,ag} = (p_{t,c} - z_{t,ag,c}) \cdot g_{t,ag} + (p_{t,d} - z_{t,ag,d}) \cdot (1 - g_{t,ag});$$

$y_{t,ag}$ - количество произведенной продукции; эти величины вычисляются по формулам:

$$y_{t,ag,c} = \begin{cases} m_{t,ag} \cdot g_{t,ag}, если \quad prf_{t,ag} \ge 0 \\ 0, если \quad prf_{t,ag} < 0 \end{cases};$$

$$y_{t,ag,d} = \begin{cases} m_{t,ag}(1-g_{t,ag}), если \quad prf_{t,ag} \ge o \\ 0, \quad если \quad prf_{t,ag} < 0 \end{cases}. \qquad (П.2.11)$$

Используя эти выражения, можно записать общее количество произведенной продукции

$$Y_t = \sum_{ag=0}^{agm} y_{t,ag} \ ;$$

спрос на рабочую силу $lbD_{t,q} = \sum_{ag=1}^{agm} y_{t,ag} \cdot nL_{q,ag}$,

а также суммарный выброс загрязнителя $pll_{t,q} = \sum_{ag=1}^{agm} y_{t,ag} \cdot nD_{ag}$

В излагаемом варианте модели считается, что спрос на рабочую силу всегда удовлетворяется.

Для вычисления общей прибыли от произведенной продукции ($profit_t$) используется следующая формула:

$$profit_t = \sum_{ag=1}^{agm} y_{t,ag} \cdot (p_{t,cd_{ag}} - z_{t,ag}) \qquad (П.2.12)$$

Прибыль частных фирм ($income_t$) вычисляется по формуле:

$$income_t = (1 - tP_t) \cdot profit_t \qquad (П.2.13)$$

Эта прибыль определяет инвестиции частных фирм $invp_t = income_t$, а также формирует спрос на продукцию, предъявляемую частными фирмами:

$$ownCD_{t,c} = \frac{\delta_t \cdot invp_t}{p_{t,c}} \; ; \quad ownCD_{t,d} = \frac{(1-\delta_t) \cdot invp_t}{p_{t,d}} .$$

Спрос на продукцию со стороны государства ($gCD_{t,cd}$) вычисляется по формулам:

$$gCD_{t,d} = 0 \; ; \quad gCD_{t,c} = \frac{RI_t + RC_t + RN_t}{p_{t,c}} .$$

Здесь:

RI_t - государственные расходы на инвестиции в новые технологии;

RN_t - государственные расходы на мероприятия по восстановлению природного ресурса;

RC_t - государственные расходы на мероприятия по очистке окружающей среды.

Зарплата ($FL_{q,t}$), выплачиваемая занятым на производстве, зависит от уровня образованности и определяется по формуле: $FL_{q,t} = \sum_{q=1}^{3} lbS_{t,q} \cdot w_{t,q}$.

Средства, которые тратят домашние хозяйства на потребление $dh_{t,q}$, вычисляются по формуле: $dh_{t,q} = (1 - tI_t) \cdot FL_{q,t}$, где tI_t - подоходный налог.

Рассмотрим затраты домашних хозяйств. Для этого введем следующие переменные:

$prte_q$ -доля зарплаты, которая тратится домашними хозяйствами на образование;

$prth_q$ -доля зарплаты, которая тратится домашними хозяйствами на здравоохранение;

$dhCD_{t,q}$ - средства, которые тратятся домашними хозяйствами на приобретение продукции (чистой и грязной); эта величина определяется по формуле:

$dhCD_{t,q} = (1 - prte_q - prth_q) \cdot dh_{t,q}$.

$\chi_{t,q}$ - доля средств, которая тратится на чистую и грязную продукцию

$lCD_{t,q,cd}$ - спрос домашних хозяйств на чистую и грязную продукцию; эти величины вычисляются соответственно по формулам:

$$lCD_{t,q,c} = \frac{dhCD_{t,q} \cdot \chi_{t,q}}{p_{t,c}} \quad ; \quad lCD_{t,q,d} = \frac{dhCD_{t,q} \cdot (1 - \chi_{t,q})}{p_{t,d}} .$$

Душевое потребление продукции ($soul_t$) определяется выражением:

$$soul_t = \frac{\sum_{q=1}^{3} lCD_{t,q,c} + \sum_{q=1}^{3} lCD_{t,q,d}}{pop_t}$$

Тогда общий спрос на продукцию производства вычисляется по формуле:

$$S_{t,cd} = \sum_{q=1}^{3} lCD_{t,q,cd} + ownCD_{t,cd} + gCD_{t,cd}$$

Следующие характеристики необходимы для вычисления показателя, характеризующего «жизненный уровень» (QL_t).

$unemplq_{t,q} = \max(0, pact_{t,q} - dpq_{t,q})$ - количество безработных, имеющих уровень образованности q.

$unemplleveldq_{t,q} = \frac{unemplq_t}{pactq_t}$ - уровень безработицы от уровня образованности q.

$unempl_{t,q} = \sum_{q=1}^{3} unemplq_{t,q}$ - общее количество безработных.

$unemplleveld_{t,q} = \frac{unempl_t}{pact_t}$ - общий уровень безработицы.

Динамика государственных доходов определяется выражением:

$$B_t = tP_t \cdot profit_t + pll_t \cdot tD_t + tI_t \cdot FL_t$$

где tP_t - налог на прибыль; эта величина является государственным управлением.

В модели считается, что государственные расходы R_t - это доходы, полученные в предыдущем году. То есть справедливо $R_t = B_{t-1}$.

Государственные расходы делятся по следующим статьям:

$artI$ - доля государственных расходов на строительство новых мощностей;

$artN$ - доля государственных расходов на мероприятия по восстановлению природного ресурса;

$artC$ - доля государственных расходов на мероприятия по очистке окружающей среды;

$artE$ - доля государственных расходов на образование;

$artH$ - доля государственных расходов на здравоохранение;

$artT$ - доля государственных расходов на разработку новых технологий;

$artS$ - доля государственных расходов на науку.

$RI_t = artI \cdot R_t$ - государственные расходы на инвестиции (строительство новых мощностей);

$RN_t = artN \cdot R_t$ - государственные расходы на мероприятия по восстановлению природного ресурса;

$RC_t = artC \cdot R_t$ - государственные расходы на мероприятия по очистке окружающей среды;

$RE_t = artE \cdot R_t$ - государственные расходы на образование;

$RT_t = artT \cdot R_t$ - государственные расходы на разработку новых технологий;

$RH_t = artH \cdot R_t$ - государственные расходы на здравоохранение;

$RS_t = artS \cdot R_t$ - государственные расходы на науку.

Теперь можно написать формулу для вычисления доли $\chi_{t,q}$, которая тратится домашними хозяйствами на приобретение чистой продукции:

$$\chi_{t,q} = \begin{cases} 1, если \quad p_{t,c} < p_{t,d} \\ \dfrac{\chi 0_q \cdot \exp(\alpha_q \cdot s_t) \cdot \exp(p_{t,d} - p_{t,c})}{1 - \chi 0_q + \chi 0_q \cdot \exp(\alpha_q \cdot s_t)}, если \quad p_{t,c} \geq p_{t,d} \end{cases} \qquad (П.2.14)$$

Здесь характеристика s_t описывает влияние экологического воспитания на желание приобретать экологически чистую продукцию. Эта характеристика вычисляется по формуле: $s_{t+1} = s_t \cdot (1 + e\alpha \cdot RE_t), \quad s_{t_0} = s0$

Аналогично вычисляется доля инвестиций частных фирм (δ_t), которые тратятся на приобретение чистого фондообразующего продукта:

$$\delta_{t,q} = \begin{cases} 1, если \quad p_{t,c} < p_{t,d} \\ \dfrac{\delta 0_q \cdot \exp(\delta\alpha_q \cdot s_t) \cdot \exp(p_{t,d} - p_{t,c})}{1 - \delta 0_q + \delta 0_q \cdot \exp(\delta\alpha_q \cdot s_t)}, если \quad p_{t,c} \geq p_{t,d} \end{cases} \qquad (П.2.15)$$

Количество студентов st_t вычисляется по формуле:

$$st_t = \begin{cases} \dfrac{(RE_t + \sum_{q=1}^{3} prte_q \cdot FL_q) \cdot 0{,}9}{\cos tedust} \end{cases} \qquad (П.2.16)$$

Количество аспирантов as_t вычисляется по формуле:

$$as_t = \begin{cases} \dfrac{(RE_t + \sum_{q=1}^{3} prte_q \cdot FL_q) \cdot 0{,}1}{\cos teduas} \end{cases} \qquad (П.2.17)$$

В знаменателях последних дробей стоят стоимости обучения одного студента и одного аспиранта, соответственно.

В каждый момент времени появляется новая технология, характеризуемая следующими величинами:

$nL_{q,0}$ - коэффициенты трудоемкостей;

nN_0 - коэффициент ресурсоемкости;

nF_0 - коэффициент фондоемкости;

nD_0 - уровень выбросов загрязнителей на единицу произведенной продукции.

Эти коэффициенты определяются следующими выражениями:

$$nL_{q,0,cd} = nL_{q,1,cd} \frac{1 + RT_t^2}{1 + CL_{q,cd} \cdot n \cdot RT_t^2}, \quad (q = 1,2,3) \qquad (П.2.18)$$

$$nN_{o,cd} = nN_{1,cd} \frac{1 + RT_t^2}{1 + CN_{cd} \cdot n \cdot RT_t^2} \qquad (П.2.19)$$

$$nF_{o,cd} = nF_{1,cd} \frac{1 + RT_t^2}{1 + CF_{cd} \cdot n \cdot RT_t^2} \qquad (П.2.20)$$

$$nD_0 = nD_1 \frac{1 + RT_t^2}{1 + CD \cdot n \cdot RT_t^2}$$

Здесь $n = psci_t / pop_t$, а коэффициенты $CL_{q,cd}, CN_{cd}, CF_{cd}$ выбраны таким образом, что все коэффициенты $nL_{q,0}$, nN_0, nF_0 nD_0, за исключением коэффициента $nL_{3,0}$ уменьшаются с увеличением вложений RT_t в новые технологии. Другими словами, считается, что технический прогресс приводит к уменьшению затрат неквалифицированного труда и увеличению затрат квалифицированного труда на единицу производимой продукции.

В разработанном варианте программы сначала вычисляется значение $nL_{q,0,d}$, а затем вычисляется значение $nL_{q,0,c}$ по формуле

$$nL_{q,0c} = nL_{q,0,d} * \frac{TL_q}{1 + SL_q * dl_t}, \qquad (П.2.21)$$

где TL_q, SL_q - положительные внешние коэффициенты. Аналогично вычисляются значения $nF_{q,0,c}$ и $nN_{q,0,c}$.

П.2.7. Расчет эволюции природных ресурсов и характеристик окружающей среды

Эволюция природного ресурса определяется соотношением:

$$N_{t+1} = N_t - dmN_t + rstN_{t\cdot} \cdot N_t + dRst \cdot RN_t \qquad (П.2.22)$$

Здесь:

$rstN_t$ - темп самовосстановления природного ресурса; он вычисляется по формуле

$$rstN_t = rstN0 \cdot rstNql(D_t), \qquad (П.2.23)$$

где $rstN0$ - «нормальный» темп самовосстановления природного ресурса;

$rstNql(D_t)$ - мультипликативный коэффициент, учитывающий зависимость темпа восстановления природного ресурса от качества окружающей среды.

$dRst$ - коэффициент эффективности вложений средств в восстановление природного ресурса; эта величина считается заданной.

$ntrdm_t$ - спрос на природный ресурс; эта величина вычисляется по формуле:

$$ntrdm_t = \sum_{ag=1}^{agm} y_{t,ag} \cdot nN_{ag} \qquad (П.2.24)$$

Эволюция цены pN_t на природный ресурс определяется соотношением:

$$pN_{t+1} = pN_t \cdot (1 + N\gamma \cdot (1 - \frac{N_{t+1}}{N_t})) \qquad (П.2.25)$$

Эволюция уровня загрязнения D_t окружающей среды определяется соотношением:

$$D_{t+1} = D_t + pll_t - rstD \cdot D_t - DRst \cdot RG_t \qquad (П.2.26)$$

Здесь $rstD$ - коэффициент самоочищения окружающей среды, $DRst$ - коэффициент эффективности вложений средств в очистку окружающей среды.

П.2.8. Взаимодействие нескольких стран

Макет распределенной эколого-социально-экономической модели, описанной в разд. 1 и 2 реализован на нескольких (в настоящее время до семи) компьютерах локальной сети или глобальной сети Интернет (в последнем случае, эти компьютеры должны иметь маршрутизируемые в Интернет IP-адреса).

Технической основой взаимодействия распределенных компонент модели является разработанная в ИСА РАН архитектура IARnet, описанная в [16]. Реализация, о которой здесь идет речь, выполнена пока без полного использования описанной в разд. 3 технологии. В частности, отсутствует описание моделируемого комплекса на специализированном непроцедурном языке, позволяющем автоматически сгенерировать его базу данных.

Однако как следует из приводимого ниже описания распределенной реализации, основная идея, заложенная в разбиение модели на распределенные компоненты, полностью соответствует представленной выше технологии модельного синтеза. Она основана на представлении, позволяющим выделить ряд классов компонент, которые достаточно независимы друг от друга, т. е. взаимодействуют друг с другом лишь в определенные моменты жизни модели. Экземпляры этих классов (компонент) могут быть

реализованы на отдельных компьютерах. Ниже кратко описываются эти классы:

- Компонента "информационный сервер" присутствует в единственном экземпляре. Этот объект собственно, является не частью модели, а надстройкой над ней, осуществляющей взаимодействие пользователя с моделью, начиная от создания реализации модели для конкретного имитационного эксперимента и кончая заданием управлений функционированием стран и отображением результатов имитации. Данный объект реализован как HTTP-сервер, основу которого составляют странички, написанные на языке HTML, усиленные клиентскими скриптами, написанными на языке JavaScript, и серверными скриптами, написанными на языке Perl. Странички информационного сервера отражают состояния демографии, экономики и экологии участвующих в имитационном эксперименте стран. Для связи с остальными распределенными компонентами на информационном сервере, в соответствии с архитектурой IARnet постоянно работает **агент доступа**, обеспечивающий такую связь.

- Компонента "страна", который в макете представлен несколькими экземплярами (от одного до четырех). У объектов этого типа два основных метода – это, во-первых, метод-конструктор, позволяющий создать новую страну, т. е. создать и заполнить начальными данными соответствующие информационные структуры. Данный метод

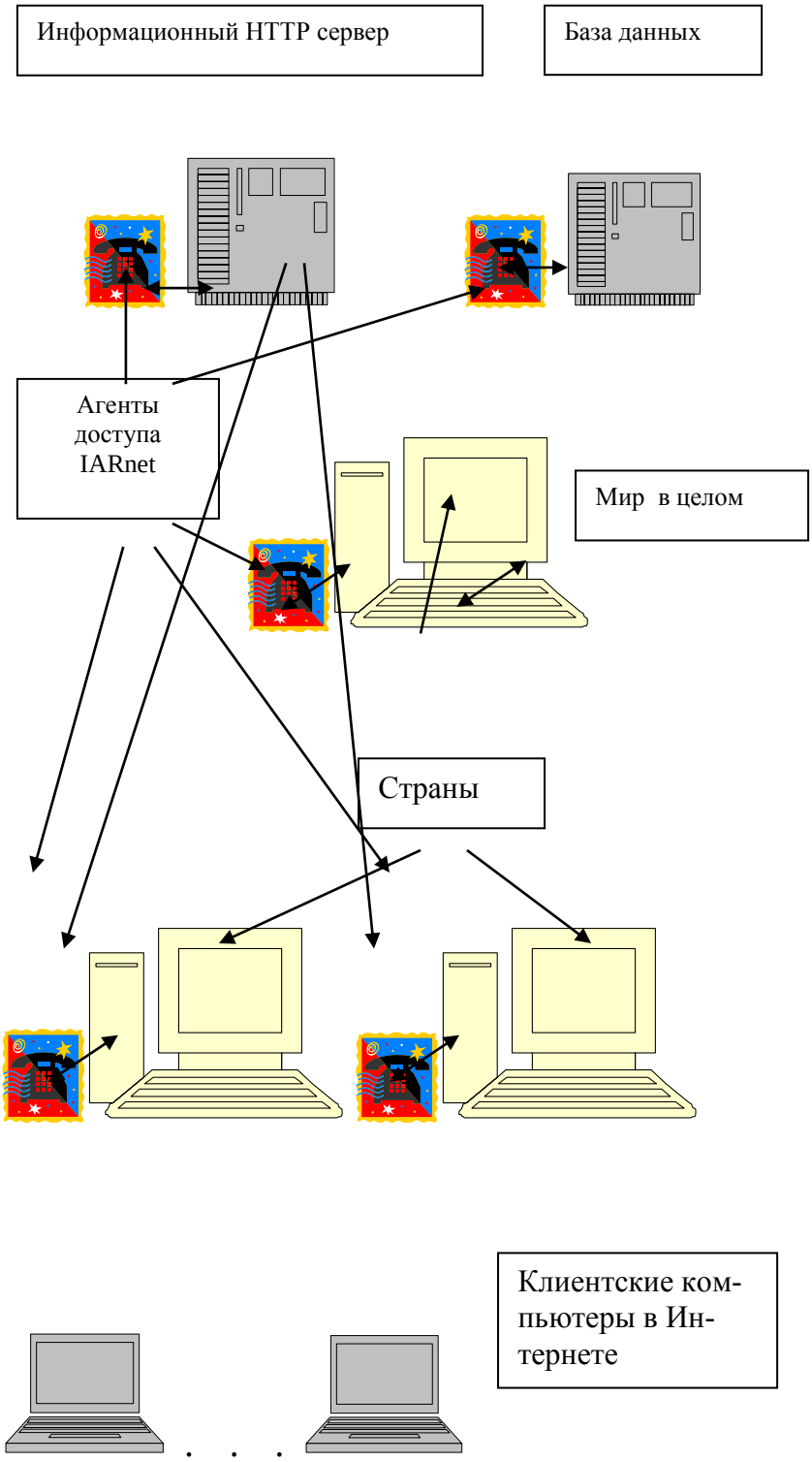
поддержан на информационном сервере соответствующим серверным скриптом, с помощью которого пользователь в начале имитационного эксперимента может вызвать метод-конструктор и создать страну – участницу эксперимента. Второй метод – это шаг моделирования, т. е. метод, позволяющий модельной "стране" прожить очередную единицу модельного времени. Этот метод на информационном сервере также поддержан серверным скриптом, позволяющим лицу, принимающему решения за данную страну, задать управления на очередной модельный год. Считается, что могут быть реализованы различные вычислительные алгоритмы, связанные с методами стран. В данном макете реализованы три таких алгоритма, отражающие тот факт, что, создавая в начале эксперимента очередную страну-участницу, можно выбирать один из трех возможных типов ее экономики. Экземпляры объекта "страна" могут быть размещены на отдельных компьютерах сети. Они также должны быть снабжены агентами доступа.

- Компонента "хранилища демографической информации" представлен в макете одним экземпляром, в котором может храниться демографическая информация трех различных типов, т. е. создавая страну, ее начальную демографическую ситуацию, можно выбирать из трех качественно различных вариантов. В этом хранилище содержится демографическая информация, относящаяся к соответствую-

щей стране, и оно также оснащено агентом доступа к этой информации в соответствии с архитектурой IARnet. Хранилище демографической информации также может располагаться на отдельном компьютере.

- Компонента "мир" представлен в макете единственным экземпляром. Метод этого класса моделирует взаимодействия стран такие как, например, миграция населения, импорт/экспорт товаров. Объект "мир" также должен иметь агент доступа, и может располагаться на отдельном компьютере.

Графически архитектура макета распределенной эколого-социально-экономической модели представлена на рис.1.



..

Рис. 1. Структура макета эколого – демографо -экономической распределенной модели

На рис. показаны только компоненты самого макета. Не показано взаимодействие лиц, принимающих решение за моделируемые страны, с информационным сервером, осуществляющим пользовательский интерфейс с макетом. Такое взаимодействие может быть осуществлено с любой рабочей станции, имеющей выход в Интернет и браузер, графически схема этого взаимодействия показана на рис. 2.

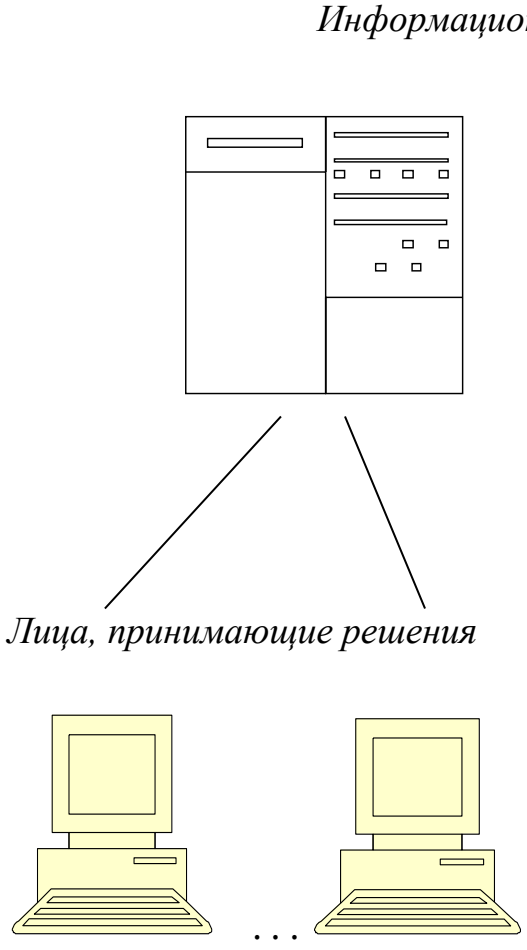


Рис. 2. Схема взаимодействия игроков с моделью.

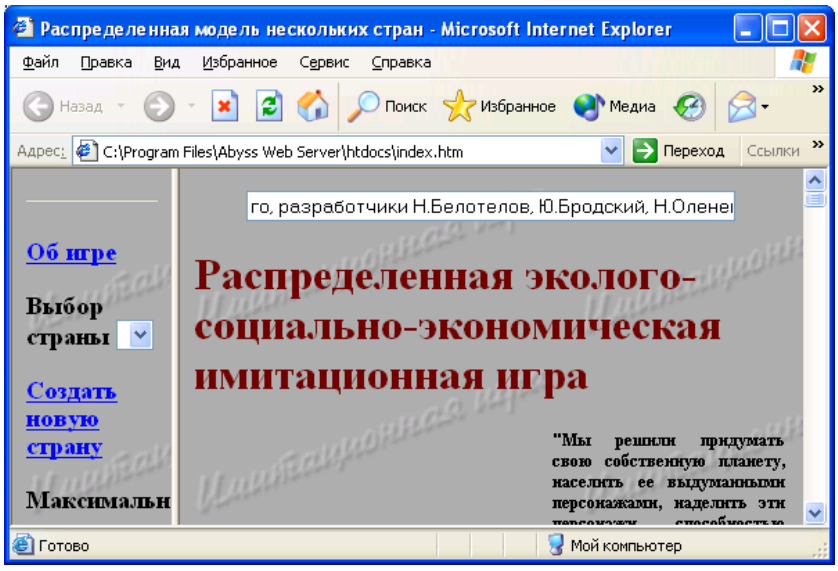
Макет распределенной эколого-социально-экономической модели может быть реализован на семи различных компьютерах. Каждый

информационно-алгоритмический ресурс, расположенный на отдельном компьютере, снабжен соответствующим агентом доступа.

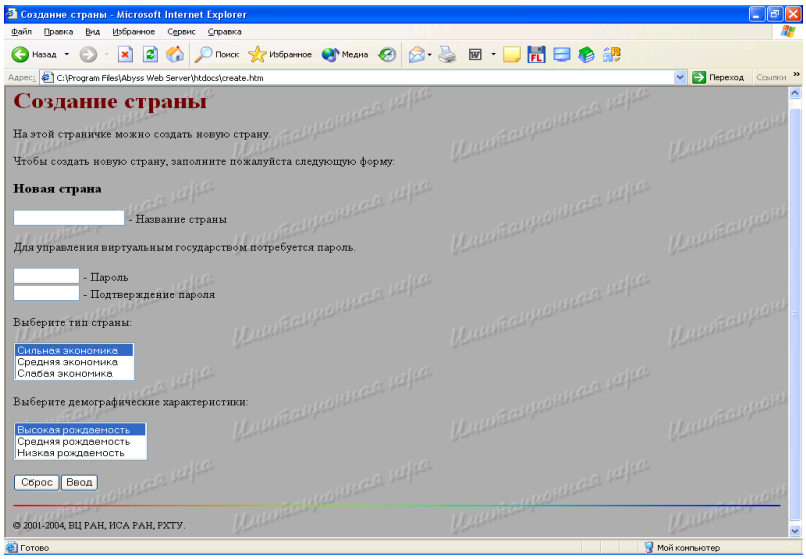
П.2.9. Интерфейс модели

Основой пользовательского интерфейса модели является сайт в Интернете, посвященный данной модели и реализованный на информационном сервере. В настоящее время адрес этого сайта – http://simul.ccas.ru/Distr. На этом сайте можно ознакомиться с проектом создания распределенной имитационной модели эколого-социально-экономических процессов, с подробным описанием всех компонент модели, правилами и инструкциями по работе с ней.

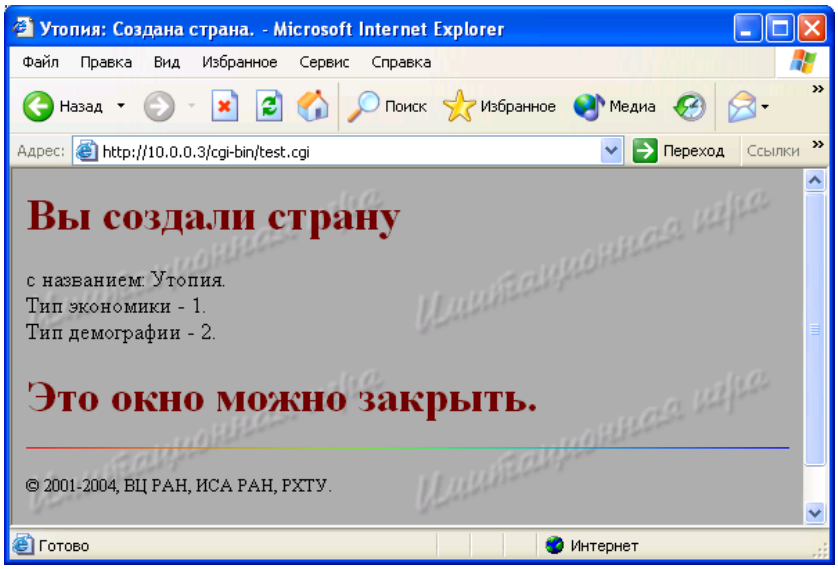
В начальный момент в модели стран еще нет, их предстоит создать. Страничка макета в Интернете имеет вид, показанный на рисунке ниже. Из всех возможностей управления активна лишь возможность создать новую страну.



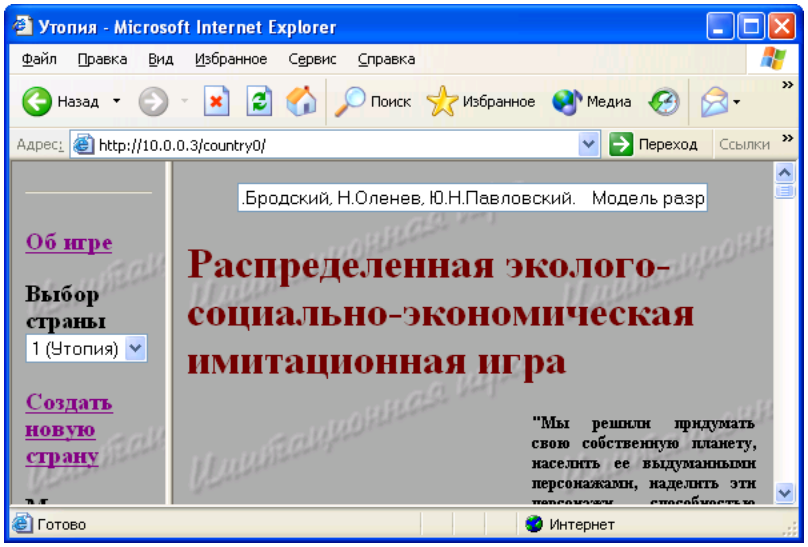
Если выбрать эту возможность, откроется окно создания страны:



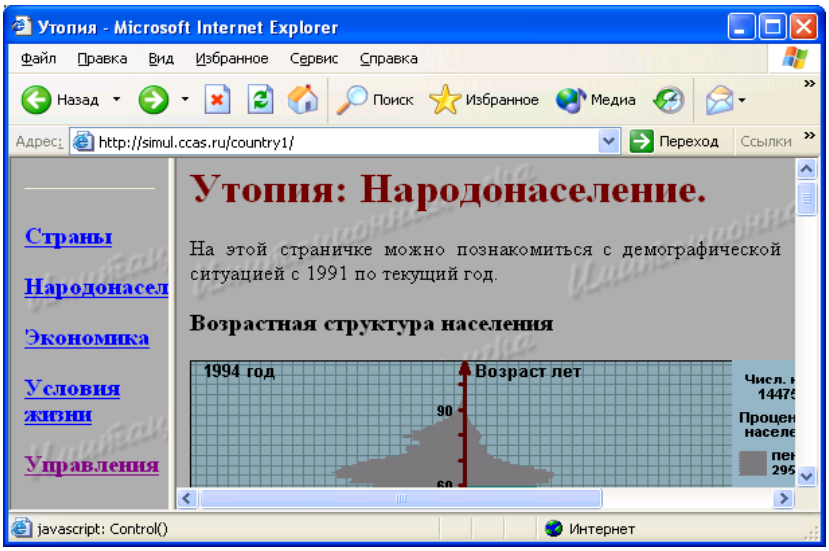
В окне создания страны нужно задать имя страны (под этим именем она будет далее известна на протяжении этого имитационного эксперимента), затем нужно задать пароль и подтвердить его во избежание случайной ошибки. По этому паролю в дальнейшем ходе имитационного эксперимента, можно будет управлять данной страной, т. е. для успешного задания управлений за эту страну необходимо указать пароль, заданный во время создания страны. Далее нужно задать один из трех возможных типов экономики и один из трех возможных типов начальной демографической ситуации. После ввода перечисленных выше сведений появится окно подтверждения создания страны:



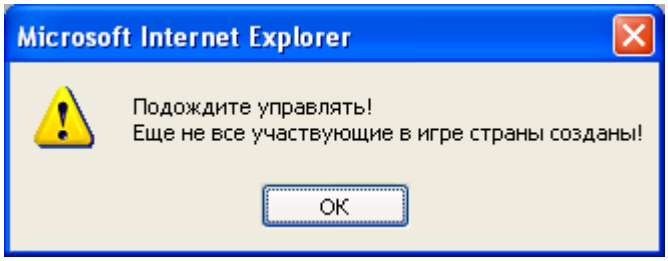
В главном окне макета также появится название созданной страны:



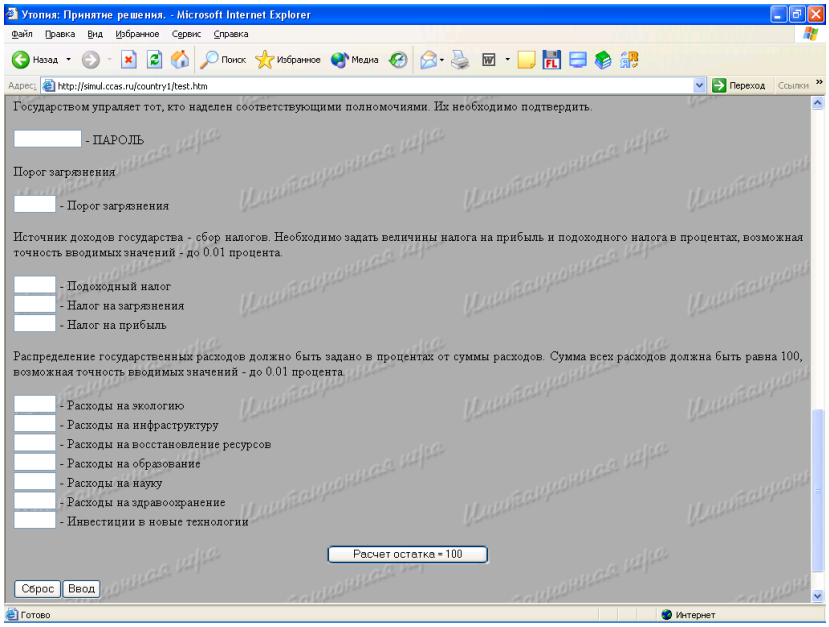
Начать имитационный эксперимент можно только после того, как созданы все четыре участвующие в нем страны. Для этого нужно выбрать нужную страну в меню "выбор страны", затем в меню страны выбрать пункт "управления".



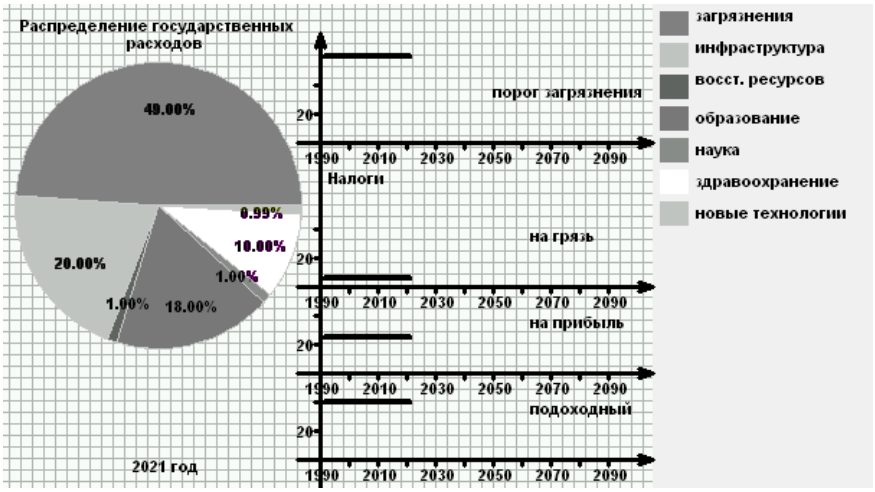
Если попытаться выбрать пункт "управления" какой-либо страной до того, как созданы все четыре страны, появится следующее окно аварийной диагностики:



Если же все страны созданы, появится консоль управления страной:



На этой консоли можно познакомиться с ранее задававшимися управлениями (историей управлений, она показана на следующем рисунке)

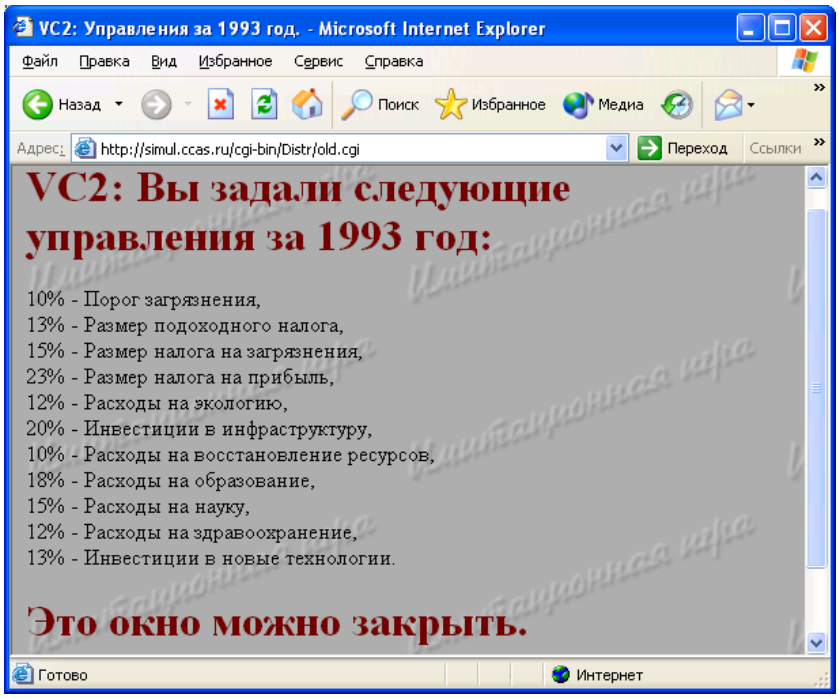


и задать новые управления. Для этого нужно, во-первых, знание пароля, который был задан, когда страна создавалась. Далее задается порог загрязнения, т. е. государство задает границу между экологически чистым и грязным продуктом (что считается чистым, что грязным). Далее государство задает источник своих доходов (величины налогов – подоходного, на загрязнения и на прибыль). Далее государство распределяет свои доходы на семь статей расходов:

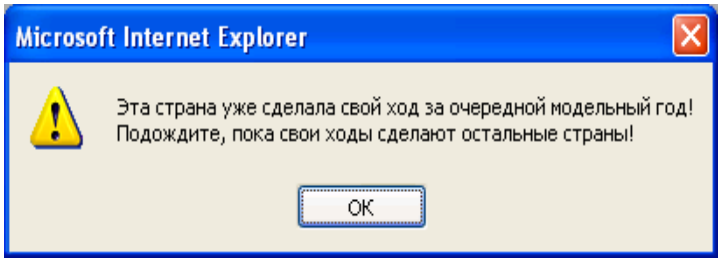
1. Экология.
2. Развитие производства.
3. Восстановление ресурсов.
4. Образование.
5. Наука.
6. Здравоохранение.

7. Новые технологии.

Расходы задаются в процентах от бюджета, для удобства расчетов имеется калькулятор остатка. После задания управлений появляется окно подтверждения:

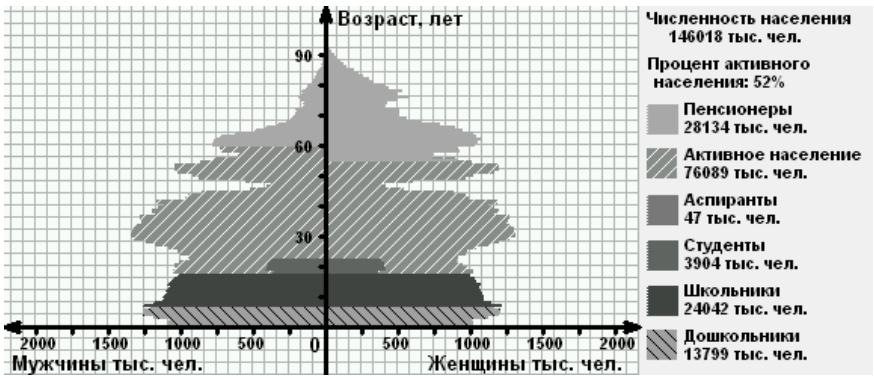


После того как управления заданы за все участвующие в эксперименте страны, проходит очередной модельный год, вызываются соответствующие методы, на информационном сервере появляются соответствующие картинки, снова появляется возможность задавать управления. Если попытаться снова войти в консоль управления, до того как зададут управления остальные участники, появится следующая аварийная диагностика:

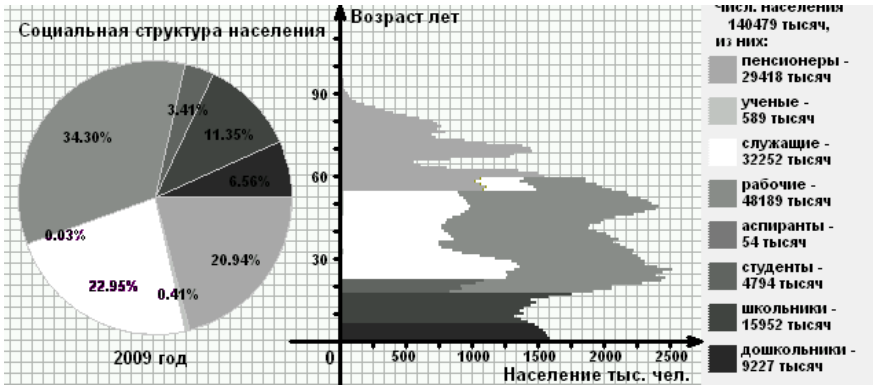


Во время имитационного эксперимента можно следить за демографической, экономической и экологической ситуацией любой из стран на соответствующих страницах информационного сервера. На демографических страничках в виде диаграмм и графиков отображены возрастная и социальная структуры населения, также динамика народонаселения по группам: все население, активное население, пенсионеры, безработные, - и уровень безработицы по социальным группам: рабочие, служащие, научные работники.

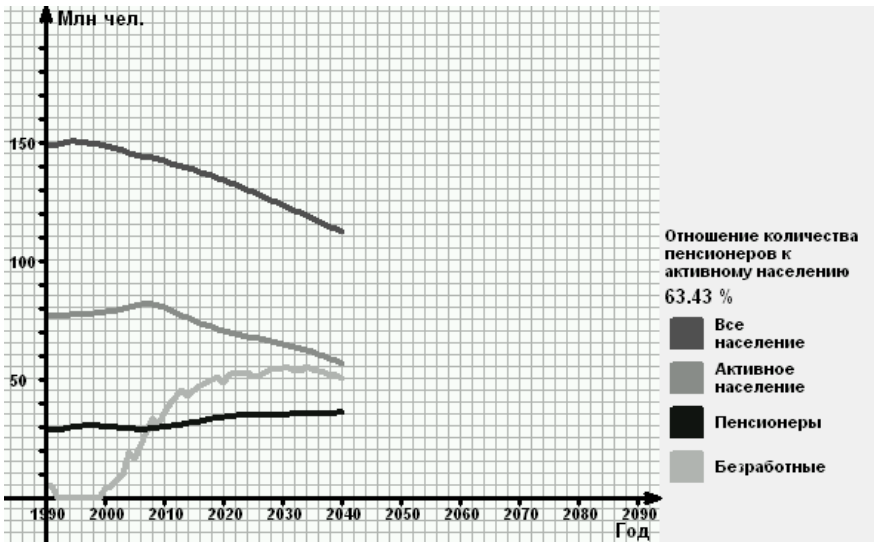
Возрастная структура населения



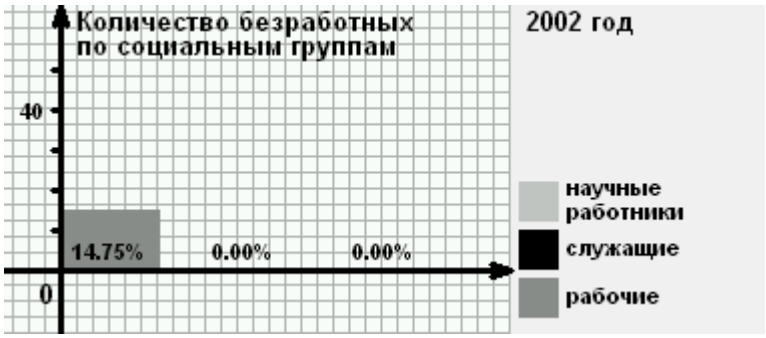
Социальная структура населения



Динамика народонаселения

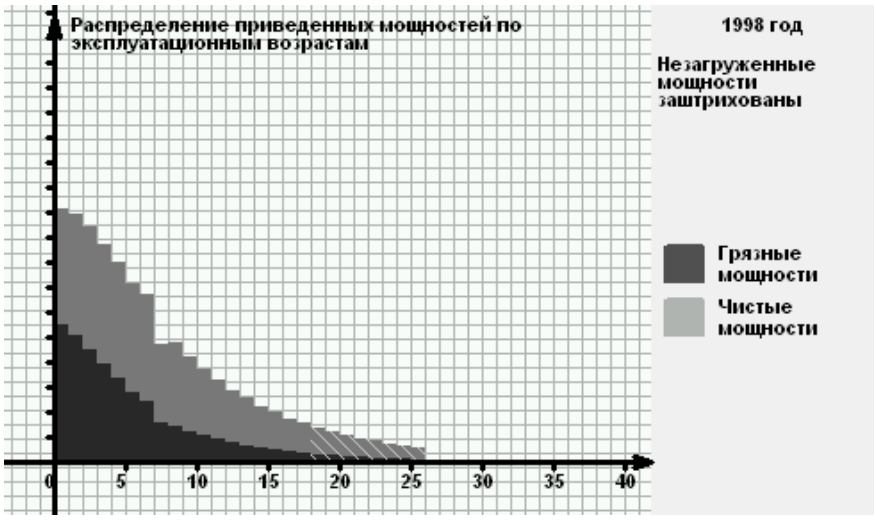


Безработица по социальным группам

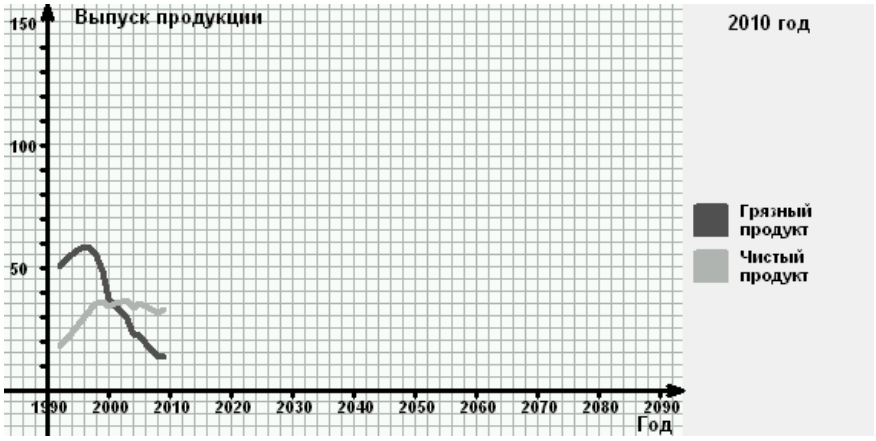


На экономических страничках отображаются возрастная структура производственных мощностей, выпуски продукции и цены на продукцию.

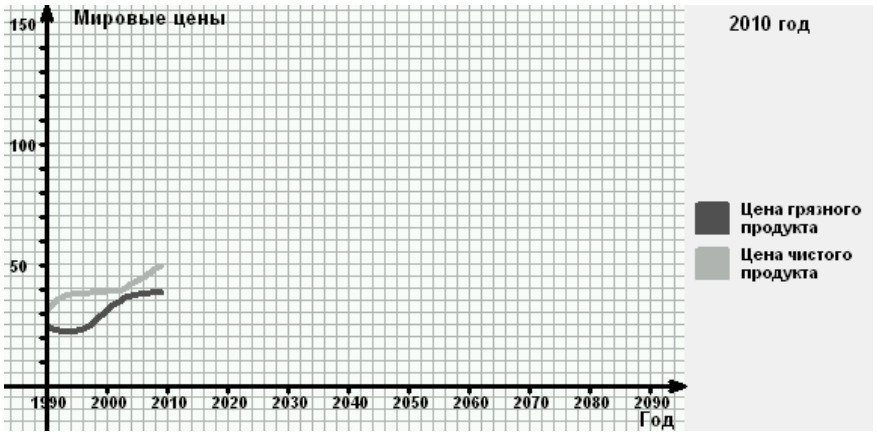
Распределение мощностей по эксплуатационным возрастам



Выпуск продукции

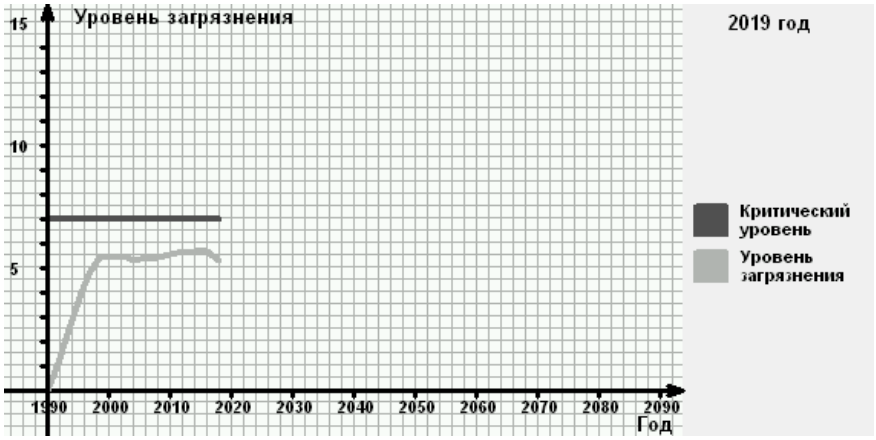


Цены на продукцию



На страничках, посвященных условиям жизни в странах, отражаются экологические условия и уровень потребления населения.

Уровень загрязнения окружающей среды



Потребление на душу населения

П.2.10. Организация имитационного эксперимента

В начальный момент исходные значения характеристик стран задаются участниками имитационного эксперимента и методом-конструктором класса "страна". Метод компоненты "мир" устанавливает начальное модельное время и вызывает программу визуализации, которая переводит наблюдаемую часть фазовых переменных игры в вид, принятый для отображения хода игры на Интернетовском сайте (диаграммы, графики, таблицы, числа, картинки...), и засылает этот материал на сайт игры, - начинается первый такт имитации. В дальнейшем участники эксперимента могут сделать свой ход, посетив сайт игры. Метод компоненты "мир" отслеживает ходы игроков.

Следующий такт игры (перевод часов модельного времени) наступает, когда все участники эксперимента сделали свой ход (либо он может быть привязан к физическому времени и наступать раз в его единицу: час, сутки, неделю... . В этом случае, если кто-то из участников эксперимента не сделал своего хода, метод компоненты "мир" определенным образом

назначает ему управление, например прошлое, или усреднение прошлых управлений, либо назначает некоторое управление, специально заданное для таких случаев).

При наступлении очередного такта эксперимента метод компоненты "мир" снова ожидает, пока методы компонент-стран пересчитают фазовые переменные в соответствии с заданными оператором управлениями, затем переводит модельные часы и вызывает программу визуализации, - наступает следующий такт жизни модели.

П.2.11. Некоторые результаты работы с моделью

На основе базовой модели ЭДЭМ, в 2001 – 2010 гг был построен ряд моделей различной направленности. В данном разделе подробно была описана одна из них – модель взаимодействия трех стран, относящаяся к экспериментам 2004 – 2006 гг. Системный анализ результатов имитационных экспериментов позволил выделить и проанализировать ряд важных аспектов проблемы устойчивого развития, таких как:

- Экономические и финансовые аспекты проблемы устойчивого развития.
- Политический аспект проблемы устойчивого развития.
- Экологический аспект проблемы устойчивого развития.
- Миграционный аспект проблемы устойчивого развития.
- Военный аспект проблемы устойчивого развития.
- Культурологический аспект проблемы устойчивого развития.

- Образовательный аспект проблемы устойчивого развития.

Подробно с результатами анализа различных аспектов проблематики устойчивого развития можно познакомиться в работах [21, 22].

С точки зрения данной работы на этой модели отрабатывались методы модельного синтеза и модельно-ориентированного программирования, а также создания распределенной вычислительной среды на основе предложенной коллегами из ИСА РАН архитектуры IARnet [16].

П.3. Тестовые модели для макета рабочей станции пиринговой сети распределенного моделирования

Приведенные здесь две модели были разработаны как тестовые для проверки и отладки работы макета рабочей станции пиринговой сети распределенного имитационного моделирования [25].

П.3.1. Распределенная модель «Пешеходы и муха» (муха фон Неймана)

Модель «Пешеходы и муха» – неоднократно упоминавшаяся в этой работе 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 требова-

ния 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.

П.3.1.1 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 единствен-

ный 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 пе-

шеходом. 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.

П.3.1.2. 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;
```

П.3.1.3. Реализация методов модели на языке C#

Абстрактные классы "пешехода" и "мухи"

Вообще говоря, подобные абстрактные классы необязательны. Здесь они играют роль заголовков C 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 пла-
нируем 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 интер-
вал 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 пе-
шехода.
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();
}
}
}
```

П.3.1.4. 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 вызывающимися 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");
**********************************************/
}
}
}
```

Посмотреть работу данной модели можно в Интернете по адресу:

http://simul.ccas.ru/Distr/fly/fly.htm

П.3.2. Модель простейшего бизнес-процесса "Передел рынка"

В настоящем разделе излагается модель одного простого бизнес-процесса и описывается процедура его компьютерной реализация с помощью средств инструментальной распределенной системы имитационного моделирования. Перспективность использования таких средств при изучении бизнес-процессов вытекает из того, что в настоящее время в производственных структурах становится все более заметно образование экономических кластеров. Как правило, предприятия, составляющие кластеры, обладают компьютерными системами поддержки и анализа протекающих в них бизнес-процессов. В большинстве случаев эта компьютерная поддержка основана на средствах системой динамики (Дж. Форрестер [85], инструментальная система Ithink. http://www.forekc.ru/31). Инструментальная

распределенная система имитации может позволить объединять средства различных предприятий для решения задач, относящихся к функционированию кластера.

Анализируется функционирование двух конкурирующих фирм, действующих на общем для них рынке. Кроме этих фирм на данном рынке других фирм нет. Поэтому ситуация полностью описывается долями рынка, которые контролируют эти фирмы. Эти доли являются фазовыми переменными соответствующей модели. Поскольку сумма этих долей равна 100 (проценты), то фазовой переменной модели можно было бы считать долю рынка одной из них. Однако, для целей демонстрации распределенной инструментальной системы имитации, более естественно считать фазовыми переменными обе эти доли. Кроме того, такой способ введения фазовых переменных делает модель перспективной для различных усложнений. С течением времени эти доли меняются описанным ниже образом. Обозначим через $f(t)$, $r(t)$ – доли рынка, контролируемые фирмами. Тогда

$$f(t+1) = f(t) + 0{,}1 * k(t), \qquad (1)$$

$$r(t+1) = r(t) - 0{,}1 * k(t), \qquad (2)$$

где $k(t)$ – так называемое конкурентное преимущество. В описываемой модели оно определяется соотношением

$$k(t) = f(t) - r(t).$$

Понятно, что эта модель дает преимущество той фирме, которая вначале имела большую долю рынка. В самом деле, вычитая соотношение (2) из соотношения (1), получим

$k(t+1) = k(t) + 0{,}2k(t) = 1{,}2k(t)$, и

$k(t) = k(0) \cdot (1{,}2)^t$.

Эта модель усложнена следующим условием. Если доля рынка некоторой фирмы опускается ниже определенного уровня (в приводимом варианте - 30%), через некоторое случайное время происходит «передел» рынка, так, что доля рынка проигрывающей фирмы увеличивается на случайную величину, равнораспределенную в пределах от 0, до 0,7.

П.3.2.1. Неформальное описание модели

Модель представляет из себя комплекс, состоящий из двух экземпляров компоненты "фирма". Компонента "фирма" реализует единственный процесс, состоящий из двух методов-элементов: медленного "работа" и быстрого "передел рынка".

•"работа" - медленный метод, реализующий для компоненты динамику в соответствии с формулами (1)-(2) за время DT.

•"передел рынка" - быстрый метод, прибавляющий компоненте некоторую долю рынка случайно распределенную от 0 до 0,7. Эта же доля рынка отнимается у другой компоненты.

Элемент "передел рынка" всегда переходит в элемент "работа". Элемент "работа" переходит в элемент "передел рынка" по наступлению события "инновация". Таким образом, с компонентой "фирма" связано событие

"инновация". Алгоритм его вычисления следующий: если доля рынка занимаемая фирмой не меньше 0,3 - фирма не думает об инновациях, и событие выдает время наступления большее стандартного шага моделирования. Если доля рынка стала меньше 0,3 - генерируется случайное число равнораспределенное от 0 до стандартного шага моделирования. Если оно не превосходит половину стандартного шага - считается, что время инновации наступило (событие произошло), если больше - это и будет прогнозом наступления события (который, однако, сбудется лишь с вероятностью 1/2).

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. Здесь сделаем замечание. Казалось бы в данной реализации модели компонента выполняя быстрый элемент не вполне корректно "лезет в чужую фазу", когда изменяет долю рынка фирмы-конкурента. Однако это происходит на быстром такте моделирования, который не пересекается с медленными. Для данной же модели быстрый такт всегда состоит лишь из одного метода той фирмы, доля рынка которой меньше 0,3. Сле-

довательно никаких недоразумений с одновременным изменением фазы здесь быть не может.

Весь набор необходимых описаний модели и ее компонент уже приводился выше для модели "пешеходы и муха". Здесь мы приведем только реализации методов и сервисного модуля на языке C#.

П.3.2.2. Описания методов "фирмы" на языке C#

Определение методов "фирмы"

```
using System;
 using System.Collections.Generic;
 using System.IO;
 namespace DefFirm
 {
 public abstract class Firm
 {
 public struct FirmPhase
 {
 public double f;
 public double r;
 }
 public struct FirmParam
 {
 public double r;
 }
```

```
public abstract void working(MemoryStream inStream, MemoryStream outStream);
public abstract void revision(MemoryStream inStream, MemoryStream outStream);
public abstract void invention(MemoryStream inStream, MemoryStream outStream);
}
}
```

Реализация методов "фирмы"

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
namespace DefFirm
{
class ImpFirm : Firm
{
public ImpFirm()
{
}
```

```
        public override void working(MemoryStream inStream, MemoryStream
outStream)
        {
        FirmPhase phase = new FirmPhase();
        FirmParam param = new FirmParam();
        double dt;
        BinaryReader inp = new BinaryReader(inStream);
        phase.f = inp.ReadDouble();
        param.r = inp.ReadDouble();
        dt = inp.ReadDouble();
        inp.Close();
        inStream.Flush();
        inStream.Close();
        BinaryWriter outp = new BinaryWriter(outStream);
        phase.f += (dt * 0.1 * (phase.f - param.r));
        outp.Write(phase.f);
        outp.Close();
        outStream.Flush();
        }
        public override void revision(MemoryStream inStream, MemoryStream
outStream)
        {
```

```
        FirmPhase phase = new FirmPhase();
        BinaryReader inp = new BinaryReader(inStream);
        phase.f = inp.ReadDouble();
        inp.Close();
        inStream.Flush();
        inStream.Close();
        BinaryWriter outp = new BinaryWriter(outStream);
        Random random = new Random();
        phase.f += 0.4 * random.Next() / int.MaxValue;
        phase.r = 1.0 - phase.f;
        outp.Write(phase.f);
        outp.Write(phase.r);
        outp.Close();
        outStream.Flush();
        }
        public override void invention(MemoryStream inStream, MemoryStream
outStream)
        {
        FirmPhase phase = new FirmPhase();
        FirmParam param = new FirmParam();
        double dt;
        BinaryReader inp = new BinaryReader(inStream);
```

```
phase.f = inp.ReadDouble();
param.r = inp.ReadDouble();
inp.Close();
inStream.Flush();
inStream.Close();
BinaryWriter outp = new BinaryWriter(outStream);
if (phase.f <= 0.3)
{
Random random = new Random();
dt = random.Next() / int.MaxValue;
if (dt < 0.5)
dt = 0.0;
}
else
if (phase.f >= 0.5)
{
dt = 1.5;
}
else
{
dt = (10.0 * phase.f - 3.0) / (param.r - phase.f);
}
```

```
outp.Write(dt);
outp.Close();
outStream.Flush();
}
}
}
```

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

Ниже приводится пример сервисного модуля для модели "Передел рынка".

Сервисный модуль модели "Передел рынка"

```
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.Threading;
namespace FIRMS
{
public partial class FIRMS : Form
{
public FIRMS()
{
InitializeComponent();
}
object[] para = new object[5];
private float previous=0;
private bool execute = true;
private readonly Brush first = new SolidBrush(Color.Red);
private readonly Brush second = new SolidBrush(Color.Blue);
private void DrawAll(Graphics dc)
{
```

```
lock (this)
{
float kX = this.Width / 100;
float kY = this.Height / 100;
double firstPart = (double)para[1];
float now = (float)firstPart;
now *= 360f;
RectangleF boxArea = new RectangleF(12 * kX, 12 * kY, 90 * kX, 90 * kY);
if (previous == 0)
dc.FillEllipse(second, boxArea);
if (now > previous)
dc.FillPie(first, boxArea.Left, boxArea.Top, boxArea.Width, box-Area.Height, 360 - now, now - previous);
else
if (previous > now)
dc.FillPie(second, boxArea.Left, boxArea.Top, boxArea.Width, box-Area.Height, 360 - previous, previous - now);
string time = ((double)para[4]).ToString();
if (time.Length > 5)
time = time.Substring(0, 5);
label1.Text = "t = " + time;
```

```
}
}
public bool Prepare(object[] param)
{
for (int i = 0; i < 5; i++)
{
para[i] = param[i];
}
this.Refresh();
return (execute);
}
public bool AfterFast(object[] param)
{
lock (this)
{
for (int i = 0; i < 5; i++)
{
para[i] = param[i];
}
this.Refresh();
}
return (execute);
```

```
}
public bool AfterSlow(object[] param)
{
lock (this)
{
int index = (int)param[0];
if (index > 24) execute = false;
for (int i = 0; i < 5; i++)
{
para[i] = param[i];
}
this.Refresh();
}
return (execute);
}
public bool Finish(object[] param)
{
return (true);
}
protected override void OnPaint(PaintEventArgs e)
{
base.OnPaint(e);
```

```
        Graphics dc = e.Graphics;
        DrawAll(dc);
        /*********************************************
        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, Sys-
tem.Drawing.Imaging.PixelFormat.DontCare);
        int i=(int)para[0];
        bmf.Save("firm" + (i<10 ? "0" : "") + i.ToString() + ".bmp");
        *****************************************************/
        }
        private void button1_Click(object sender, EventArgs e)
        {
        execute = false;
        Close();
        }
        }
        }
```

Посмотреть работу данной модели можно в Интернете по адресу: http://simul.ccas.ru/Distr/firm/firm.htm

Литература

1. Brodsky Yu.I. Bourbaki's structure theory in the problem of multi-component simulation models synthesis //Innovative Information Technologies, Part 2, M.: HSE, 2014, P. 234-244.
2. Brodsky Yu.I. Simulation Software //System Analysis and Modeling of Integrated World Systems – V 1, Oxford: EOLSS Publishers Co. Ltd., 2009. P. 287-298.
3. Brodsky Yu.I., Tokarev V.V. Fundamentals of simulation for complex systems. //System Analysis and Modeling of Integrated World Systems – V 1, Oxford: EOLSS Publishers Co. Ltd., 2009. P. 235-250.
4. 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.
5. Chandy, K. M. and Misra J. Asynchronous Distributed Simulation via a Sequence of Parallel Computations //Communications of the ACM 24(4), 1981: 198-205.
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. Hewitt Carl Viewing Control Structures as Patterns of Passing Messages Journal of Artificial Intelligence. June 1977.
9. Hogeweg P. Cellular Automata as a Paradigm for Ecological Modeling. Applied Mathematics and Computation, 1988, 27, P. 81-100.
10. Kuhl F., Weatherly R., Dahmann J. Creating Computer Simulation Systems: An Introduction to the High Level Architecture NY: Prentice Hall PTR, 1999. 212 p.

11. Shoham Y. Agent-oriented programming //Artificial Intelligence, vol. 60, 1993. P. 51-92.
12. Shoham Y. MULTIAGENT SYSTEMS: Algorithmic, Game-Theoretic, and Logical Foundations. Cambridge: Cambridge University Press, 2010, 532 p.
13. Von Neumann, J. The General and Logic Theory of Automata in Cerebral Mechanisms in Behavior: The Hixon Symposium, New York: John Wiley&Sons, 1951.
14. World Science Report/Paris: UNESCO Publishing, 1996. 356 pp.
15. Zave, P. A compositional approach to multiparadigm programming. IEEE Software, 6(5): 15—25, September 1989.
16. Афанасьев А.П., Волошинов В.В., Кривцов В.Е., Рогов С.В., Сухорослов О.В. Использование информационно-алгоритмических ресурсов для организации распределенных вычислений. // Проблемы вычислений в рапределенной среде: организация вычислений в глобальных сетях. Сборник трудов ИСА РАН, М.: УРСС, 2004.
17. Ахромеева Т.С., Курдюмов С.П., Малинецкий Г.Г., Самарский А.А. Нестационарные структуры и диффузионный хаос. –М.: Наука, 1992, 544 с.
18. Белотелов Н.В. Модель миграционных процессов между несколькими странами. В сб. Моделирование, декомпозиция и оптимизация сложных динамических процессов, М., ВЦ РАН, 2010. 112-119
19. Белотелов Н.В. Эколого-демографо-экономическая модель с учетом миграционных процессов между странами. Всероссийская научная конференция «Математическое моделирование развивающейся экономики и экологии, ЭКОМОД 2010/ Сборник трудов, Киров, изд-во гоу ВПО «Вят ГУ»», 2010, 19-27 с.
20. Белотелов Н.В., Бродский Ю.И., Оленев Н.Н., Павловский Ю.Н. Эколого-социально-экономическая модель: гуманитарный и инфор-

мационный аспекты. // Информационное общество. № 6. 2001. С. 43-51.

21. Белотелов Н.В., Бродский Ю.И., Оленев Н.Н., Павловский Ю.Н., Тарасова Н.П. Проблема устойчивого развития: гуманитарный и естественно-научный анализ. М.: Фазис. 2004. 105 с.
22. Белотелов Н.В., Бродский Ю.И., Павловский Ю.Н. Сложность. Математическое моделирование. Гуманитарный анализ. М.: Книжный дом «ЛИБРОКОМ», 2013, 318 с
23. Боев В. Д., Кирик Д. И., Сыпченко Р. П. Компьютерное моделирование: Пособие для курсового и дипломного проектирования. — СПб.: ВАС, 2011. — 348 с.
24. Бродский Ю.И. Модельный синтез и модельно-ориентированное программирование М.: ВЦ РАН, 2013, 142 с.
25. Бродский Ю.И. Распределенное имитационное моделирование сложных систем М.: ВЦ РАН, 2010, 156 с.
26. Бродский Ю.И. Толерантность и нетерпимость с точки зрения системной динамики и исследования операций М.: ВЦ РАН, 2008, 53 с.
27. Бродский Ю.И. О модельном анализе как альтернативе объектному, в задаче описания и синтеза имитационных моделей сложных многокомпонентных систем //Инновации на основе информационных и коммуникационных технологий: Материалы международной научно-практической конференции. М.: МИЭМ НИУ ВШЭ, 2013. С. 181-183.
28. Бродский Ю.И. Проблемы создания центра имитационного моделирования в Internet, Моделирование, декомпозиция и оптимизация сложных динамических процессов. М.:ВЦ РАН, 1998. С.29-35
29. Бродский Ю.И. Эколого-социально-экономическая имитационная модель: технология реализации. Моделирование, декомпозиция и оптимизация сложных динамических процессов. М.:ВЦ РАН, 2001. С.29-35

30. Бродский Ю.И., Лебедев В.Ю. Инструментальная система имитации MISS. М.: ВЦ АН СССР, 1991, 180с.
31. Бродский Ю.И., Мягков А.Н. Декларативное и императивное программирование в имитационном моделировании сложных многокомпонентных систем //Вестник МГТУ им. Н.Э. Баумана. Сер. Естественные науки. Спец. выпуск № 4 «Математическое моделирование». 2012. С.178-187.
32. Бродский Ю.И., Павловский Ю.Н. Разработка инструментальной системы распределенного имитационного моделирования. //Информационные технологии и вычислительные системы, №4, 2009. С. 9-21.
33. Бурбаки Н. Теория множеств. М.: Мир. 1965. 456 с.
34. Бусленко Н.П. Моделирование сложных систем М.: Наука, 1978, 400 с.
35. Бусленко Н.П. Сложная система //Статья в Большой Советской Энциклопедии, 3-е изд., М.: Советская энциклопедия, 1969-1978.
36. Буч Г., Рамбо Д., Якобсон И. Введение в UML от создателей языка. 2-е изд.: Пер. с англ. Н. Мухин М.: ДМК Пресс, 2012. 494 с.
37. Воротынцев А.В. Концепция сетевых информационно-вычислительных библиотек М.: ВЦ РАН, 2009, 108 с.
38. Горшков В.Г., Макарьева А.М., Лосев К.С. В повестке дня - стратегия выживания человечества//Вестник РАН. Т. 76, № 4, 2006. С. 309-314.
39. Грецкий М.Н. Структурализм (философ.) //Статья в Большой Советской Энциклопедии, 3-е изд., М.: Советская энциклопедия, 1969-1978.
40. Григоров Ю.Н., Данилов П.В., Павловский Ю.Н. Проблемы использования технологии математического моделирования в народном хозяйстве (на примере моделей возрастных структур парков техниче-

ских систем). Математическое моделирование//т.3, № 4, 1991, с. 56-66.

41. Дубошин Г.Н. Небесная механика. М.: Физматгиз. 1963.588 с.
42. Елкин В.И. Редукция нелинейных управляемых систем. Декомпозиция и инвариантность по возмущениям. М.: ФАЗИС, 2003. 207 с.
43. Замятина Е.Б. Современные теории имитационного моделирования (Специальный курс для магистров второго курса) Пермь: ПГУ, 2007, 119 с.
44. Имитационная игра на основе эколого-демографо-экономической модели (ЭДЭМ): описание и инструкция пользователю. Белотелов Н.В., Бродский Ю.И., Е.Б. Кручина, Оленев Н.Н., Павловский Ю.Н. М.: РХТУ. Институт проблем устойчивого развития. Издательский центр. 2003. 84 с.
45. Йоханнесбургский саммит ООН: анализ итогов// Вестник РАН. №11, Т73, 2003. С. 1010-1015.
46. Капица С.П. Общая теория роста человечества. М.: Наука. 1999. 190 с.
47. Китинг М. Повестка дня на XXI век и другие документы конференции в Рио-де-Жанейро в популярном изложении. ─ Публикация центра «Наше будущее», с.27, 1993.
48. Комаров В.Ф. Управленческие имитационные игры и АСУ. Новосибирск, Изд-во НГУ, 1979, 234 с.
49. 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.
50. Краснощеков П.С., Петров А.А. Принципы построения моделей. Изд. 2-е, пересмотр. и дополнен., М.: Фазис. 2000. 412 с.
51. Лаплас П.С. Изложение системы мира М.: Наука, 1982. 676 с.
52. Лаплас П.С. Опыт философии теории вероятностей Пер. с фр., Изд.2, М.: URSS, 2011. 208 с.

53. Масалович А.И., Шебеко Ю.И. Моделирование и анализ поведения бизнес-процессов. М.: ТОРА. 2002. 219 с.
54. Медоуз Д.Х., Медоуз Д.Л., Рендерс Й., Беренс В.В. Пределы роста. Доклад по проекту Римского клуба `Сложное положение человечества'. Изд. МГУ, 1991. 208 с.
55. Моисеев Н.Н. Идеи естествознания в гуманитарной науке: о единстве естественно-научного и гуманитарного знания//Человек. 1992. Вып. 2. С. 5-16.
56. Моисеев Н.Н. Математика ставит эксперимент. М.: Наука, 1979. 224 с.
57. Моисеев Н.Н. Человек, среда, общество. М.: Наука, 1982. 238 с.
58. Моисеев Н.Н. Алгоритмы развития. М.: Наука, 1987. 303 с.
59. Моисеев Н.Н., Александров В.В., Тарко А.М. Человек и биосфера: Наука, 1982. 238 с.
60. Моисеев Н.Н., Левиков А.А., Павловский Ю.Н., Черевков К.В. Новый виток гонки вооружений или новое мышление //ПОЛИС. 1991, С.5-15.
61. Мягков А. Н., Бродский Ю.И. Об управлении временем в распределённых имитационных моделях //ТРУДЫ МФТИ, Т. 4, № 3(15), 2012, С. 181-186.
62. Нейлор Т. Машинные эксперименты с моделями экономических систем, М. Мир, 1975,397с.
63. Об основаниях геометрии //Сборник классических работ по геометрии Лобачевского и развитию ее идей /Ред. и вступ. статья А.П. Нордена, М.: Гос. изд. технико-теоретической литературы, 1956. 533 с.
64. Осоргин А.Е. AnyLogic 6. Лабораторный практикум Самара: ПГК, 2011. 100 с.
65. Павловский Ю.Н. Геометрическая теория декомпозиции и некоторые ее приложения. М.: ВЦ РАН, 2011. 93 с.

66. Павловский Ю.Н. Формальная математика Н.Бурбаки и роды структур. М: ВЦ РАН. 2012. 110 с.
67. Павловский Ю.Н. О вооруженной борьбе. //В кн. Свободная мысль. №9/10, 2012. С.129-138.
68. Павловский Ю.Н. Имитационные модели и системы. М.: Фазис. 2000. 166 с.
69. Павловский Ю.Н. Имитационные системы и модели. М.: Знание. № 6. 1990. 46 с.
70. Павловский Ю.Н. Методологические вопросы разработки системы математических моделей, предназначенных для исследования проблем стратегической стабильности, национальной безопасности, строительства вооруженных сил. //В кн. Стратегическая стабильность межгосударственных отношений и безопасность России. Проблемы формализации. М.: Академия естественных наук РФ. Секция геополитики и безопасности. 1992. С.22-29.
71. Павловский Ю.Н. Экологический контроль — составная часть индустрии// Вестник РАН. № 2, 1993. С.29-38.
72. Павловский Ю.Н., Белотелов Н.В., Бродский Ю.И. Имитационное моделирование: учеб. пособие для студ. высш. учеб. заведений М.: Издательский центр «Академия», 2008. 236 с.
73. Павловский Ю.Н., Белотелов Н.В., Бродский Ю.И., Оленев Н.Н. Опыт имитационного моделирования при анализе социально-экономических явлений. М.,МЗ Пресс, 2005, 136 с.
74. Павловский Ю.Н., Смирнова Т.Г. Введение в геометрическую теорию декомпозиции. М.: Фазис, ВЦ РАН, 2006. 169 с.
75. Павловский Ю.Н., Смирнова Т.Г. Проблема декомпозиции в математическом моделировании. М.: ФАЗИС, 1998. 272 с.
76. Пегов С.А. Устойчивое развитие в условиях глобальных изменений природной среды. // Вестник РАН. № 12, 2004. С.1082-1089.

77. Петров А.А., Поспелов И.Г., Шананин А.А. Опыт математического моделирования экономики. М.: Энергоатомиздат. 544с.
78. Пономарев И.Н. Введение в математическую логику и роды структур: Учебное пособие. М.: МФТИ, 2007. 244 с.
79. Попов Э.В. Экспертные системы. М.: Наука. 1987. 283 с.
80. Самарский А.А., Михайлов А.П. Математическое моделирование: Идеи. Методы. Примеры. 2-е изд., испр. М.: Физматлит, 2001. 320с.
81. Советов Б.Я., Яковлев С.А. Моделирование систем: Учеб. для вузов – 3-е изд., перераб. и доп. – М.: Высш. шк., 2001, 343 с.
82. Староверов О.В. Азы математической демографии. М.: Наука. 1997. 161 с.
83. Фаулер М. UML. Основы., 3-е издание. Пер. с англ. СПб: Символ-Плюс, 2004. 192 с.
84. Форрестер Дж. Основы кибернетики предприятия. (Индустриальная динамика). М: Прогресс. 1971, 340 с.
85. Форрестер Дж. Мировая динамика. М. Физматгиз. 1978. 168 с.
86. Форсайт Дж., Молер К. Численное решение систем линейных алгебраических уравнений. М.: Мир, 1969. 168с.
87. Френкель А.А.,Бар-Хиллел И. Основания теории множеств. М.: URSS, 2006. 552 с.
88. Харитонов В.В. Распределенное моделирование гибридных систем. //Материалы межвузовской научной конференции, СПб.: СПбГТУ, 2002, С. 128-129.
89. Шенон Р. Имитационное моделирование систем: исскусство и наука. М., Мир, 1978, 297с.
90. Шрайбер Т. Дж. Моделирование на GPSS М.: Машиностроение, 1980, 592 с.