МОДЕЛЬНЫЙ СИНТЕЗ И МОДЕЛЬНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ – ТЕХНОЛОГИЯ ИМИТАЦИИ, ОРИЕНТИРОВАННАЯ НА РАСПРЕДЕЛЕННЫЕ И ВЫСОКОПРОИЗВОДИТЕЛЬНЫЕ ВЫЧИСЛЕНИЯ

Ю.И. Бродский (Москва)

Предлагается новый подход к описанию, синтезу и компьютерной реализации имитационных моделей сложных многокомпонентных систем, – модельный синтез и модельно-ориентированное программирование. Центральным для этого подхода является понятие модели-компоненты. Семейство моделей-компонент замкнуто относительно объединения компонент в комплекс, а организация имитационных вычислений однотипна для любого представителя семейства, что позволяет решить задачу синтеза моделей любой сложности. Для описания моделей-компонент и построения из них комплексов предлагается специальный декларативный язык ЯОКК (язык описаний комплексов и компонент). Предлагаемая система модельно-ориентированного программирования ориентирована на распределенные и параллельные вычисления.

Ключевые слова: имитационное моделирование, сложные системы, модельный синтез, модельно-ориентированное программирование, параллельные и распределенные вычисления.

Model Synthesis and Model-Oriented Programming:

simulation technology focused on distributed and high performance computing

Abstract. A new approach proposed to the description, design and implementation for computer simulations of complex multi-component systems, – the model synthesis and model-oriented programming. The central concept of this approach is the concept of the model-component. The models-components family is closed under uniting components into the complex, and the simulation computations organization is an invariant for any member of the family. This fact allows solving the synthesis problem for a simulation model of any complexity. A special declarative language LCCD (language of components and complexes descriptions) proposed for describing the models-components and building models-complexes from them. The model-oriented programming system proposed is focused on parallel and distributed computing.

Keywords: simulation, complex systems, model synthesis, distributed computing, model-oriented programming, high performance computing.

Введение

Работа посвящена проблемам имитационного моделирования достаточно широкого класса сложных систем, который характеризуется атомистическим и фрактальным устройством своих представителей. Такое понимание сложной системы отражено, например, в одноименной статье Н.П. Бусленко в БСЭ [17]: «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» – т.е., компоненты сложной системы сами могут быть сложными системами. При этом считается, что хорошо известно устройство «атомов» системы, их поведение, а также закономерности их взаимодействия между собой. Задачей имитационного моделирования в данном случае является воспроизведение поведения всей системы в целом, с целью выяснения ее возможностей, особенностей, возможно решению относительно нее каких-либо управленческих или оптимизационных задач.

Эта задача далеко не тривиальна. Достаточно сложной является даже входящая в нее задача описания всего, что известно о сложной системе. Выше было сказано о

наличие знания об устройстве «атомов» системы и их взаимосвязях. Однако сложность здесь в том, что такое знание обычно плохо структурировано: мы редко знаем о какой-либо компоненте моделируемой системы ровно столько, сколько необходимо и достаточно для построения имитационной модели. Если мы знаем меньше – хорошо бы понимать, что еще необходимо узнавать, или от каких особенностей будущей модели можно отказаться. Если мы наоборот, знаем что-то сверх необходимого относительно какой-либо составляющей системы, – хорошо бы уметь исключать такое знание из формального описания, так как подобных составляющих могут быть тысячи, и их избыточные описания, скорее всего, осложнят в дальнейшем проектирование модели.

Тем более сложной является компьютерная реализация имитационной модели сложной системы. Хотелось бы, чтобы проектирование и реализация модели осуществлялись максимально автоматизировано, на основании имеющегося формального описания сложной системы. Желательно также максимально исключать из проекта реализации модели наиболее трудоемкое императивное программирование.

Одним из магистральных направлений развития современной информатики являются распределенные и параллельные вычисления. В связи с этим хотелось бы получать имитационные модели сложных систем, ориентированные на распределенные и высокопроизводительные вычислительные системы.

Обзор существующих решений

В работе [9] был дан обзор ряда существующих решений в области описания, проектирования и реализации имитационных моделей сложных многокомпонентных систем, кратко перечислим их в порядке появления.

Система GPSS (General Purpose Simulation System [22], 1961 г., Дж. Гордон, IBM) в наибольшей степени решает поставленные во введении задачи, видимо это и позволяет ей сохранять приверженцев на протяжении более полувека. Однако ее предметная область существенно уже, чем заявленная во введении – это различные системы, укладывающиеся в парадигму процессов массового обслуживания.

В основе синтеза многокомпонентной системы лежит идея, высказанная в работе Н.П. Бусленко[18]: дать каждой из компонент, про которые нам все известно, максимально проявить себя, учитывая при этом и все межкомпонентные связи. Это обусловило появление с середины 70-х идей агентного программирования и моделирования, а затем и систем ABMS (Agent-Based Modeling and Simulation), 2-я половина 70-х – 80-е гг., C. Hewitt [2], Y. Shoham [4], системы ABMS, с конца 90-х. Однако на предлагаемых ими агентах лежит некий отпечаток антропоморфности – об их поведении говорится в терминах «убеждений», «обязанностей», «способностей» и т.д. Возникает вопрос – а каким должен быть универсальный агент и его поведение, достаточные для моделирования сложных многокомпонентных систем и в то же время максимально простые, близкие к необходимым условиям такого моделирования?

Про инструментальную систему MISS (Multilingual Instrumental Simulation System, 1990 г., ВЦ РАН [12]) можно сказать, что это была первая действующая система модельно-ориентированного программирования – такого, где основной единицей программы является модель, помимо характеристик и методов, наделенная законченным поведением – умением стандартно отвечать на стандартные запросы. В то же время, этой системе не хватает теоретического обоснования – например, многие вычисления в ней параллелятся естественным образом, но остается неясным, можно ли этим пользоваться, не приведет ли это к конфликтам за ресурсы.

Унифицированный язык моделирования UML (Unified Modeling Language, середина 90-х, Г. Буч, Д. Рамбо, И. Якобсон [18]; OMG, UML Partners), предназначен в первую очередь для проектирования сложных программных систем, хотя может быть применен и для описания имитационных моделей сложных систем. Основные его

недостатки – избыточность и тяжеловесность, существенно затрудняющие сквозные решения: от описания – до программного кода. Также он оставляет весь проект в рамках объектно-ориентированной императивной парадигмы, которая сложна и не всегда адекватна задачам имитационного моделирования [6, 7].

Спецификация HLA (High Level Architecture [3], 2-я половина 90-х, DMSO – Defense Modelling & Simulation Office – Министерства Обороны США) предназначена для создания программного обеспечения, позволяющего объединять в распределенной системе готовые имитационные модели, часть из которых может быть аналоговой – реализованной «в железе».

Инструментальная система моделирования AnyLogic 2000 г., AnyLogic (ранее XJ Technologies) [5, 21] – по-видимому, наиболее мощное решение из имеющихся в настоящее время на рынке инструментальных средств имитационного моделирования. Помимо рассматриваемого в данной работе агентного моделирования, она также дает средства моделирования системной динамики и процессов массового обслуживания (отсюда и название AnyLogic). Оставаясь внутри объектно-ориентированной парадигмы, система дает средства заметного упрощения процесса программирования, разделения логики поведения агентов и их функциональности. Тем не менее, также возникает ряд теоретических вопросов, например, до каких пределов можно довести такое упрощение и для каких классов имитационных моделей.

Основной вывод обзора – ни одна из попавших в него систем не решает полностью все поставленные во введении задачи. Кроме того, все эти системы дают более или менее мощные средства решения некоторых из этих задач, но не дают теоретических обоснований, почему предлагаются именно эти средства, какие средства необходимы и достаточны для построения каких классов имитационных моделей.

Гипотеза о замкнутости и ее следствия

Из работ Н.П. Бусленко (например, [16, 17]), мы знаем, что если имитационная модель реализуется на компьютере (а если она достаточно сложна, то только так ее и можно реализовать), то ее траектория будет из класса кусочно-линейных агрегатов, просто потому что количество вычислений должно быть конечным.

Гипотеза о замкнутости есть предположение о том, что наших знаний о сложной системе достаточно для того, чтобы построить ее имитационную модель, т.е., исходя из начальных значений характеристик, воспроизвести их динамику на отрезке $[0,T]$. Таким образом, задача построения имитационной модели всегда есть задача создания «демона Лапласа» [19, 20].

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 характеристик модели $\vec{X}(t)$ можно определить, есть ли в точке t разрыв траектории первого рода $\Delta\vec{X}(t)$, и если он есть – вычислить его. 2). Далее, на интервале $(t, t+\Delta t)$, траектория модели, выходящая из точки $\vec{X}(t)+\Delta\vec{X}(t)$, является непрерывной функцией времени и однозначно вычисляется по начальным значениям характеристик модели $\vec{X}(t)$.

По-видимому, требование замкнутости модели в любой точке $t \in [0,T)$ есть необходимое условие построения модели – непонятно, как ее строить, если есть точки, из которых невозможен даже малый шаг вперед.

В работах [1, 7] показано, что достаточным это условие не является. Там же показано, что локальная замкнутость становится достаточным условием построения модели, если дополнить ее требованием непрерывности траектории модели слева (обусловленности любого состояния модели некоторой его предысторией). Это – некий

аналог теоремы существования для имитационной модели. Весьма важные вопросы единственности, устойчивости и зависимости от начальных значений характеристик, по-видимому, должны выясняться в ходе имитационных экспериментов с моделью.

Весьма важны (быть может, даже важнее теоремы существования) следствия гипотезы о замкнутости – функциональная (следовательно, однозначная) зависимость скачка траектории, ее последующей непрерывной эволюции и размера отрезка прогноза, от значений характеристик модели в начальной точке. Во-первых, функциональные зависимости естественно реализовывать в функциональной парадигме программирования; во-вторых, если удается эквивалентно распараллелить данные вычисления, они должны быть бесконфликтны – конфликт всегда есть нарушение однозначности вычислений; в третьих, необходимость на каждом шаге моделирования вычислять возможный разрыв, непрерывную эволюцию траектории и окончание отрезка прогноза, определяет облик предлагаемого ниже универсального агента.

Модельный синтез

Модельный синтез и модельно-ориентированное программирование, как методы описания, синтеза и программной реализации имитационных моделей сложных многокомпонентных систем, развивались в отделе Имитационных систем ВЦ АН СССР и затем ВЦ РАН, с конца 80-х гг. Основные их идеи изложены в работах [1, 6-12], а сами термины впервые введены в работе [7]. В основе модельного синтеза лежит понятие модели-компоненты – универсального агента. Модель-компонента подобна объекту объектного анализа, но снабженному не только характеристиками и способными делать что-то полезное, если их вызовут, методами, а неким аналогом системных служб операционной системы, всегда функционирующим и готовым давать стандартные ответы на стандартные запросы внутренней и внешней среды модели.

Предлагается, основываясь на гипотезе о замкнутости и ее следствиях, формализовать семейство имитационных моделей сложных систем семейством родов структур [6, 7] в смысле Н. Бурбаки [15]. Базисными множествами представителей семейства являются совокупности множеств характеристик модели, методов (того, что модель умеет делать) и событий (того, на что модель должна уметь реагировать). Семейство родов структур «модель-компонента» обладает двумя важными свойствами:

1. Организация имитационных вычислений однотипна для всех представителей семейства. Притом значительная часть этих вычислений может выполняться параллельно. Это означает возможность создания универсальной программы, ориентированной на высокопроизводительные или распределенные вычисления, способной запустить на выполнение любую имитационную модель, если та является математическим объектом, снабженным структурой рода семейства «модель-компонента».
2. Семейство родов структур «модель-компонента» оказывается замкнутым, относительно операции объединения моделей-компонент в модель-комплекс. Комплекс, полученный объединением моделей-компонент, сам принадлежит семейству родов структур «модель-компонента», и, следовательно, может включаться в новые комплексы, а организация процесса его имитационных вычислений может осуществляться той же самой универсальной программой.

Приведенные выше свойства семейства родов структур «модель-компонента» позволяют предложить новый модельно-ориентированный метод программирования для программной реализации имитационных моделей сложных систем.

Программный комплекс при этом подходе видится как комплекс моделей-компонент, чье поведение нет необходимости специально организовывать (например, вызывая какие-либо методы) – все компоненты всегда ведут себя так, как умеют. Программирование состоит в описании устройства и поведения компонент (по сути – в

описании соответствующего рода структуры) и в описании построения комплексов из компонент.

Что такое модель-компонента

Понятие модели, о котором говорилось выше, можно определить вполне формально, как математический объект, снабженный определенным родом структуры, на языке формализма Н. Бурбаки [10] и основанной на нем геометрической теории декомпозиции [11]. Такому ее определению и некоторым свойствам, вытекающим из него, посвящена работа [2]. Здесь мы дадим менее строгое, более содержательное определение.

Модель-компонента должна иметь внутренние и внешние характеристики – этим она похожа на объект объектного анализа, также имеющий набор характеристик. Кроме характеристик объект объектного анализа имеет методы – набор умений. Собственного поведения объект, как правило, не имеет – он лишь хранилище умений, из которых, наряду с умениями других объектов будет складываться поведение программной системы в целом. Модель, в отличие от объекта, наделена собственным поведением. Ее умения, в отличие от объекта, невозможно вызвать извне. Зато она, подобно операционной системе компьютера, готова в любой момент стандартным образом ответить на любой стандартный запрос внутренней и внешней среды (что в силу гипотезы о замкнутости есть просто некоторое стандартное сочетание значений ее внутренних и внешних переменных). У операционных систем, для возможности обслуживания стандартных запросов, постоянно и параллельно друг другу, работает несколько системных служб. У нашей модели параллельно могут развиваться несколько процессов. Каждый из этих процессов состоит из чередования методов-элементов – элементарных действий процесса, реализуемых функциональной зависимостью. Чередование методов-элементов, в силу гипотезы о замкнутости модели, определяется состоянием внутренних и внешних переменных модели. Подробнее на правилах чередования элементов мы остановимся далее.

Одной из особенностей моделирования сложных систем можно считать необходимость учета разномасштабных по времени явлений, происходящих в них, проведем классификацию элементов модели по отношению к модельному времени:

1. Сосредоточенные или быстрые элементы. Это элементы, которые происходят мгновенно по отношению к модельному времени. Сосредоточенные элементы – один из способов вычисления заведомо дискретных характеристик модели.
2. Распределенные или медленные элементы. Эти элементы имеют ненулевую продолжительность в модельном времени, т.е., занимают время не меньшее, характерного для данной модели интервала осреднения по времени. Кроме того, для таких элементов всякому разумному интервалу времени можно сопоставить результат выполнения элемента за этот интервал времени. Распределенные элементы – наиболее естественный способ вычисления непрерывных характеристик модели.

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

Формально событие – функция значений внутренних и внешних переменных в начале шага моделирования. С точки зрения организации имитационных вычислений, событие – это метод, входными параметрами которого является подмножество внутренних и внешних характеристик модели-компоненты, а выходной параметр один

– прогнозируемое время до наступления этого события. Если это прогнозируемое время равно нулю – значит, событие уже наступило.

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

Итак, подведем итог. Модель, являющаяся основным понятием излагаемого здесь модельного синтеза, характеризуется:

1. Набором внутренних и внешних характеристик.
2. Набором процессов, каждый из которых есть чередование методов-элементов, каждый из которых реализует некую функциональность. Входящими параметрами элементов являются подмножества внутренних и внешних характеристик модели, их возвращаемые параметры меняют подмножества внутренних характеристик модели.
3. Методы-события управляют переключениями элементов в процессах или прерывают выполнение элементов в целях синхронизации. Входящими параметрами событий являются подмножества внутренних и внешних характеристик модели. Если возможен переход от одного элемента процесса к другому – с такой упорядоченной парой должно быть связано единственное событие. Возможно событие прерывающее выполнение элемента с последующим его продолжением.

Теперь можно сформулировать правила запуска модели-компоненты на вычисление.

Поведение модели

Сформулируем теперь общие для всех моделей «правила поведения», которые есть ничто иное, как правила организации для них имитационных вычислений.

Во-первых, задается стандартный шаг моделирования Δt. Во-вторых, считается, что в начале шага моделирования известны текущие элементы всех процессов и все внутренние характеристики модели. Считается, что мы умеем получать внешние характеристики модели в любой интересующий нас момент модельного времени. Далее,

1. Вычисляются события, связанные с текущими элементами процессов. 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, если они есть – выполняются 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, 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, $\Delta t \le \Delta \tau$ – вычисляем текущие медленные элементы со стандартным шагом Δt. В противном случае – вычисляем их с шагом времени до ближайшего спрогнозированного события $\Delta \tau$. Медленные элементы также можно вычислять параллельно, с ожиданием завершения последнего.
4. Возвращаемся к началу п.1.

В соответствии с описанными выше правилами может быть написана универсальная программа, способная осуществить имитационные вычисления для любой модели, описанной выше конструкции.

Что такое модель-комплекс

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

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

Комплекс состоящий из многих компонент, вовне может проявляться в качестве единой компоненты. Введем следующую операцию объединения компонент комплекса:

1. Внутренними переменными комплекса объявляется объединение внутренних переменных всех его компонент.
2. Процессами комплекса объявляется объединение всех процессов его компонент.
3. Методами комплекса объявляется объединение всех методов его компонент.
4. Событиями комплекса объявляется объединение всех событий его компонент.
5. Внешними переменными комплекса объявляется объединение всех внешних переменных его компонент, из которого исключаются все те переменные, которые моделируются явно какими-либо компонентами.

Отметим, что подобное объединение может вызвать неоднозначности, похожие на те, что возникают в объектном анализе при множественном наследовании. Например, несколько компонент дают прогноз погоды или валютных курсов. Однако подобные неоднозначности можно устранить. Вполне строго этот вопрос решается в работе [6], здесь, на содержательном уровне, скажем, что например, можно добавить в комплекс еще одну компоненту, внешними переменными которой будут все имеющиеся в комплексе различные прогнозы, а в качестве внутренней, она будет выдавать один, полученный из них известным разработчику комплекса способом.

Операция объединения, после устранения всех возникших в ее результате неоднозначностей, превращает модель-комплекс в модель-компоненту, что позволяет строить модель как фрактальную конструкцию, сложность которой (и соответственно подробность моделирования) ограничивается лишь желанием разработчика.

Превращение модели-комплекса в модель-компоненту и является центральной идеей модельного синтеза как метода описания и реализации имитационных моделей сложных многокомпонентных систем.

Модельно- и объектно- ориентированное программирование

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, объединяющие 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.

Отношение наследования для множества классов объектно-ориентированного языка программирования есть отношение частичного 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. Однако 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.

Для описания сложных систем в объектной парадигме в конце 90-х был предложен унифицированный язык моделирования UML [8]. Создатели языка UML пошли по пути резкого увеличения числа исходных понятий и представлений объектного анализа, 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 весьма затрудняет новичку работу с ним. Сами создатели [8] говорят, что: «UML подчиняется правилу 80/20, т.е., 80% большинства проблем можно смоделировать, используя 20% средств 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 авторы (например, [9]) считают, что основная ценность UML как раз в применении как средства документирования и обмена формализованными описаниями на стадиях эскиза и проектирования сложных систем. 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 данной работе будет описан иной подход к описанию и построению моделей сложных систем, который называется модельным синтезом, потому что в его основе лежит понятие модели, весьма близкое тому, что присутствует в математическом и особенно в имитационном моделировании. Модель – более сложная и агрегированная конструкция, нежели объект объектного анализа. Главное ее отличие от объекта – обладание собственным поведением, в том смысле в каком обладает поведением, например, компьютер с загруженной операционной системой – способностью стандартным, заранее заданным способом отвечать на известный заранее набор внешних и внутренних запросов. При этом оказывается, что способ организации поведения в указанном смысле также может быть стандартным – одинаковым для любой модели, сколь бы сложной она ни была.

Предлагаемый модельный подход минималистичен. По существу, в нем нет других понятий, кроме понятия модели. Модели-компоненты могут объединяться в модели-комплексы, но модель-комплекс также формально принадлежит семейству моделей-компонент, следовательно, сам может быть включен в модель-комплекс следующего уровня в качестве компоненты и т.д. Любая модель выполняется по одним и тем же стандартным правилам, т.е., может быть выполнена однажды написанной и отлаженной программой выполнения моделей. Кроме того, эта программа выполнения моделей такова, что все ее содержательные вычисления допускают распараллеливание. Все программирование модели сложной системы распадается, во-первых, на ряд декларативных описаний моделей-компонент и моделей-комплексов объединяющих модели-компоненты, и во-вторых, на программирование некоторых функциональных зависимостей, которые являются функциями в математическом а не только программистском смысле (т.е. однозначны и не имеют состояний и побочных эффектов), и, следовательно могут быть запрограммированы в функциональной парадигме. Это приводит к тому, что программная реализация даже сверхсложной фрактально устроенной имитационной модели обходится без императивного программирования – самого сложного на стадии отладки. Кроме того, про устройство однажды отлаженной компоненты можно навсегда забыть после завершения отладки. Во всех моделях-комплексах, куда она входит, она будет вести себя ровно так, как она умеет себя вести – это будет обеспечено автоматически программой выполнения моделей. Плата за это – более высокий уровень инкапсуляции – в отличие от объектного анализа методы модели-компоненты невозможно (да и ненужно) вызвать «вручную» с каким-нибудь «своим» набором параметров – они всегда вызываются автоматически и только с характеристиками модели в качестве параметров – в соответствии с описанным поведением модели.

Для подобных описаний предлагается декларативный язык ЯОКК (язык описания комплексов и компонент), подробно описанный в работах [7, 9, 11]. Описатели ЯОКК компилируются не в машинный код, а в базу данных модели (что снимает вопрос о качестве компиляции – остается лишь вопрос о ее правильности). Первым воплощением идеи модельно-ориентированного программирования можно считать систему MISS [12], в настоящее время в ВЦ РАН ведутся работы по созданию современной версии подобной системы [13, 14].

Выводы

Предложенные концепции модельного синтеза и модельно-ориентированного программирования позволяют полностью решить все сформулированные во введении задачи – формально описать на языке ЯОКК имеющиеся знания об «атомах» сложной многокомпонентной системы и их связях между собой; автоматически по этим описаниям построить синтез модели сложной системы, путем компиляции описателей ЯОКК в базу данных модели; далее остается запрограммировать методы модели

(ориентируясь на функциональную парадигму) и заполнить в базе данных начальные значения ее характеристик. После этого модель готова к имитационным экспериментам. Из проекта программной реализации имитационной модели сложной системы удается полностью исключить императивное программирование. Универсальная программа выполнения моделей может быть ориентирована на высокопроизводительные и распределенные вычисления.

Предложенная концепция моделирования сложных систем была воплощена в серии имитационных моделей, реализованных под влиянием модельно-ориентированной парадигмы программирования. Например, моделировались некоторые эпизоды СОИ (Стратегическая Оборонная Инициатива) Рейгана, модель функционирования и взаимодействия нескольких стран. Кроме того были созданы инструментальная система имитационного моделирования: система MISS [12], и программное обеспечение для рабочей станции пиринговой системы распределенного моделирования [9].

Концепция модельного синтеза и модельно-ориентированного программирования применима в первую очередь для описания, проектирования и программной реализации имитационных моделей сложных многокомпонентных систем. Однако можно надеяться, что аналогичный подход может использоваться и для разработки сложных программных систем, с организацией, соответствующей гипотезе о замкнутости, включая программные системы ориентированные на распределенные и высокопроизводительные вычисления.

Работа выполнена при финансовой поддержке РФФИ в рамках научного проекта №13-01-00499-а.

Литература

1. Brodsky Yu.I. Model synthesis and model-oriented programming – the technology of design and implementation of simulation models of complex multicomponent systems //In the World of Scientific Discoveries, Series B, 2014, Vol. 2, No 1, P. 12-31.
2. Hewitt C. Viewing Control Structures as Patterns of Passing Messages //Journal of Artificial Intelligence. June 1977.
3. Kuhl F., Weatherly R., Dahmann J. Creating Computer Simulation Systems: An Introduction to the High Level Architecture NY: Prentice Hall PTR, 1999. 212 p.
4. Shoham Y. MULTIAGENT SYSTEMS: Algorithmic, Game-Theoretic, and Logical Foundations Cambridge: Cambridge University Press, 2010, 532 p.
5. Боев В.Д., Кирик Д.И., Сыпченко Р.П. Компьютерное моделирование: Пособие для курсового и дипломного проектирования. — СПб.: ВАС, 2011. — 348 с.
6. Бродский Ю.И. Роды структур Н. Бурбаки в задаче синтеза имитационных моделей сложных систем и модельно-ориентированное программирование //ЖВМ и МФ, 2015, том 55, № 1, с. 153–164.
7. Бродский Ю.И. Модельный синтез и модельно-ориентированное программирование М.: ВЦ РАН, 2013, 140 с.
8. Бродский Ю.И. Модельный синтез и модельно-ориентированное программирование как технология реализации имитационных моделей сложных многокомпонентных систем, с ориентацией на параллельные и распределенные вычисления //Материалы конференции «Имитационное моделирование. Теория и практика». ИММОД-2013. – Казань: Изд-во «Фэн» Академии наук РТ, 2013. – Т1, С. 114-118.
9. Бродский Ю.И. Распределенное имитационное моделирование сложных систем М.: ВЦ РАН, 2010, 156 с.

10. Бродский Ю.И., Мягков А.Н. Декларативное и императивное программирование в имитационном моделировании сложных многокомпонентных систем //Инженерный журнал: наука и инновации. 2012. № 2 (2). С. 33.
11. Бродский Ю.И., Павловский Ю.Н. Разработка инструментальной системы распределенного имитационного моделирования. //Информационные технологии и вычислительные системы, №4, 2009, С. 9-21.
12. Бродский Ю.И., Лебедев В.Ю. Инструментальная система имитации MISS М.: ВЦ АН СССР, 1991, 180 с.
13. Бродский Ю.И. Программное обеспечение для макета рабочей станции пиринговой сети распределенного имитационного моделирования «Прототип-Д» //Свидетельство №2015613336 о государственной регистрации компьютерной программы от 12.03.2015.
14. Бродский Ю.И. Анализатор типов данных для компилятора с языка описаний комплексов и компонент (ЯОКК) //Свидетельство №2015613334 о государственной регистрации компьютерной программы от 12.03.2015.
15. Бурбаки Н. Теория множеств. М.: Мир. 1965. 456 с.
16. Бусленко Н.П. Моделирование сложных систем М.: Наука, 1978, 400 с.
17. Бусленко Н.П. Сложная система //Статья в Большой Советской Энциклопедии, 3-е изд., М.: Советская энциклопедия, 1969-1978.
18. 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.
19. Лаплас П.С. Изложение системы мира М.: Наука, 1982. 676 с.
20. Лаплас П.С. Опыт философии теории вероятностей Пер. с фр., Изд.2, М.: URSS, 2011. 208 с.
21. Осоргин А.Е. AnyLogic 6. Лабораторный практикум Самара: ПГК, 2011. 100 с.
22. Шрайбер Т. Дж. Моделирование на GPSS М.: Машиностроение, 1980, 592 с.