О СТРУКТУРНОМ ПОДХОДЕ К КОНЦЕПТУАЛЬНОМУ МОДЕЛИРОВАНИЮ ШИРОКОГО КЛАССА КРУПНОМАСШТАБНЫХ СИСТЕМ

Бродский Ю.И.[1,2], Круглов Л.В.[2]

[1]*Федеральный исследовательский центр «Информатика и управление» РАН,*
Россия, г. Москва, ул. Вавилова д.44/2,
[2]*Московский педагогический государственный университет,*
Россия, г. Москва, ул. Краснопрудная д.14,
yury_brodsky@mail.ru, leonid.kruglov.cmc@gmail.com

Аннотация: Концептуальное моделирование приобретает важную роль при проектировании крупномасштабных систем, особенно с ростом сложности последних. Средством проектирования таких систем является разработка онтологий их предметных областей. Возникает проблема: в своей деятельности мы рассматриваем все более сложные предметные области и сложность онтологий растет. Есть ли предел этой сложности? Проектирование крупномасштабных систем и так задыхается от сложности - сложно описание компонент предметной области, сложен их синтез, еще сложнее программная реализация модели в рамках императивного объектно-ориентированного подхода. Крайне сложна отладка созданной программной системы. Структурная теория предлагает единый способ описания предметных областей широкого класса крупномасштабных систем, реализуя в этом описании принцип "Одно во всем и все в Одном". На основании этого строится близкая методам САПР сквозная технология описания, синтеза и программной реализации моделей сложных крупномасштабных систем. В реализации программной системы удается оставаться в рамках декларативного программирования, избегая императивного, что существенно упрощает ее отладку.

Ключевые слова сложные системы, концептуальное моделирование, роды структур, модельный синтез, модельно-ориентированное программирование, парадигмы программирования.

Введение

Программирование начиналось с машинных инструкций и данных трех типов (целочисленных, плавающих и булевых), все время, расширяя и усложняя объект деятельности программиста.

Первые языки программирования принесли такие конструкции, как операторы, циклы, массивы данных. До сих пор те, кому достаточно этих инструментов, кто например, решает уравнения математической физики на различных сетках, являются ярыми поклонниками Фортрана и вполне счастливы, не будучи знакомы с такими инновациями, как классы, наследование или онтологии.

Структурное программирование обосновало набор базовых конструкций: последовательность, ветвление, итерация, рекурсия, подпрограмма, блок, достаточный для написания любой программы без использования оператора «goto» – одного из самых популярных в предыдущий период [1]. Организация данных становится более сложной – записи объединяют различные типы данных. Другим важным выводом этого периода является то, что координация между языком программирования и компьютерным оборудованием (например, C и PDP-11) может значительно повысить эффективность вычислений.

Следующим шагом эволюции программирования стал объектный анализ. Концепция класса определяет тип объектов, экземпляры которых объединяют структуру данных с методами их обработки. Кроме того, с помощью механизма наследования вы можете построить таксономию классов, развивая и конкретизируя идеи, заложенные в корневые классы этой иерархии. Когда иерархия классов построена, программа рассматривается как последовательная активация желаемых объектов и вызов их соответствующих методов. Здесь заканчивается технология объектного анализа и начинается неформальное искусство программирования. Мы видим, что анализ объектов – это не сквозная технология, как, например, технологии САПР. Что касается анализа – да, он, несомненно, присутствует, а что касается синтеза – он не формализован. Синтез оставлен искусству программиста. Что касается организации данных, то этот период был отмечен быстрым развитием технологий баз данных, и с тех пор почти любая сложная программная система взаимодействует с одной или несколькими базами данных.

На пороге тысячелетий область программирования становилась все более сложной. Возрос интерес к сверхбольшим формам – мирам. Даже в сфере развлечений мы знаем и любим миры Толкина, Льюиса, Ефремова, Стругацких, Звездных войн, Матрицы, Гарри Поттера, Пиратов Карибского моря, Игры престолов. Чтобы иметь дело с такой сложной областью, необходима ее концептуальная модель. В [2] говорилось о необходимости изучения сложных систем в трех мирах: материальном, информатики и идей (как учил Платон), и предлагалось делать это с помощью родов структур в смысле Н. Бурбаки [3]. В практике программирования в начале текущего тысячелетия это

выражается в создании онтологий для предметной области сложных программных систем. Термин «онтология» неплох и уже прижился в информатике, но философы – первые, кто занимает и давно владеет им. Игнорировать этот факт – все равно, что учить детей считать Бетховена сенбернаром. Тем не менее, во введении мы позволим себе использовать этот термин, в дальнейшем отдавая предпочтение «структуре» или «концептуальной модели».

Онтология помогает ответить на поставленный выше вопрос: что делать после того, как идеи, заложенные в базовые классы, развернуты путем наследования в согласованную систему листовых. Как не потеряться в таком богатстве (их могут быть десятки тысяч, например, .net)? В голове столько не удержать – может взорваться – поневоле приходится вооружаться схемой трех миров Платона. Онтология может оказаться нитью Ариадны, способной вывести программиста из этого лабиринта. Особенно, если она была проработана сначала, а уже потом создавалась таксономия классов.

Возникает проблема: мы замахиваемся в своей деятельности на все более сложные миры – растет сложность онтологий предметных областей. Есть ли предел этой сложности? Современное программирование итак задыхается от сложности: императивный подход сложен; сложная структура программной системы и связи между ее компонентами; сложна организация данных; поведение компонентов системы сложное: сложна его логика и сложна функциональность отдельных действий. Все это делает современное искусство программирования практически недоступным для обычного человека.

С другой стороны, стремительное развитие информационных технологий требует, чтобы программирование сложных систем стало широко доступным, как например, технологии САПР, которым можно регулярно обучать в высшей школе, а не оставалось искусством, передаваемым от Учителя избранным. Объектный подход, даже подкрепленный элементами концептуального моделирования (построения онтологий), не является такой технологией – слишком многие этапы деятельности программиста остаются неформализованными. Основным инструментом по-прежнему остается императивное программирование, которое крайне сложно отлаживать.

Эта статья призвана показать выход из тупика сложности, если не для всего на свете, то для широкого класса проблем при создании сложных программных систем. Предлагаемая технология реализует технологии САПР в программировании, которые зарекомендовали себя, например, при проектировании микроэлектроники, где процессор содержит десятки миллионов транзисторов. Предлагаемая технология вообще не использует императивного программирования. Кроме того, предлагаются программные решения, позволяющие существенно повысить производительность вычислений при взаимодействии со специально ориентированными аппаратными решениями.

1 Элементы структурной теории многокомпонентных сложных систем

В диалоге Платона «Парменид» именитый элейский софист, его зрелый ученик Зенон (известный нам своими апориями), молодой Сократ и совсем юный Аристотель (не знаменитый философ – тот еще не родился, – но будущий тиран из 30) спорили о природе этого мира - един он или множественен. Поскольку любое из этих утверждений является глубокой истиной (по классификации Н. Бора или К. Гёделя), его отрицание – также глубокая истина. Полемика была долгой, точки зрения многократно подтверждались и опровергались множеством аргументов. Современные философы, поэтому рассматривают диалог «Парменид», как начало всякой диалектики. Тем не менее, некоторым особо циничным личностям кажется, что это лишь выездная кампания по рекламе и продвижению элейской софистики. Как бы то ни было, синтезирующая противоположные утверждения диалога формула: «Одно во всем и все в Одном», была и остается популярной среди мистиков всех времен и народов.

Поскольку моделирование претендует на отражение реальности, эта формула отразилась в теории модельного синтеза, предлагающей сквозную технологию описания, синтеза и программной реализации моделей сложных многокомпонентных систем. [4]. Центральным моментом этой теории является построение универсального агента для агентного моделирования – так называемой «модели - компоненты» (очень бы хотелось назвать ее объектом, и это имя было бы наиболее верным, но, увы, термин уже был занят и используется в объектном анализе, салют Бетховену!).

Модель-компонента – формальный математический объект – род структуры в смысле Н. Бурбаки. Формально она определена, например, в [4]. Семейство моделей-компонент имеет две важные особенности:

- моделей-компонент в модель-комплекс. Комплекс, полученный путем такого Семейство моделей-компонент замкнуто относительно операции конечного объединения, сам является моделью-компонентой и, следовательно, может быть включен в новые комплексы.

- Организация имитационных вычислений однотипна для всех представителей семейства моделей-компонент. Этот факт означает возможность создания универсальной компьютерной программы, способной выполнять любую имитационную модель, если она математический объект, снабженный родом структуры модели-компоненты.

Эти свойства семейства моделей-компонент позволяют решить задачу синтеза сложной системы из ее агентов. Все доступные воображению агенты являются моделями-компонентами, и конечное объединение моделей-компонент также является моделью-компонентой. Таким образом, эту теорию можно рассматривать как математическую модель и интерпретацию (разумеется, на упрощенном и более низком уровне, как и любую модель) приведенной выше фразы: «Одно во всем и все в Одном».

Концепция модельного синтеза дает следствия очень широкого применения. Мы можем сопоставить почти любую сложную агентную систему (физическую, техническую, социальную или смешанную) с ее математической моделью (модель-компонента – это математический объект определенного рода структуры), по крайней мере, в качестве мысленного эксперимента. Мы кратко опишем класс систем, к которым применима теория модельного синтеза, в п. 3.1.

Что это дает для решения поставленных во введении проблем?

- Когда сложность предметной области системы растет, сложность ее концептуальной модели остается прежней качественно – это все та же модель-компонента. Сложность концептуальной модели растет лишь количественно – у нее становится больше базисных множеств, соотношений типизации, аксиом, при этом ее типовые характеристики как рода структуры остаются прежними.
- Следствием этого является то, что сложность организации вычислений также не меняется качественно, при росте сложности системы. Да, вычислений становится количественно больше, но их тип остается прежним, так как работа ведется с той же самой структурой. Это позволяет разработать и отладить до блеска универсальную программу организации вычислений, способную запустить на счет любой объект, снабженный родом структуры модель-компонента.

Таким образом, появляется возможность обойти ловушку сложности при разработке все более сложных программных систем. Нет необходимости придумывать все новые и новые онтологии по мере усложнения и изменения предметных областей. Нужно учиться видеть в них единую и неизменную модель-компоненту. Ну а работать с ней умеет универсальная программа организации вычислений.

А теория Модельного Синтеза дает пошаговую технологию как в практически любой предметной области увидеть все ту же модель-компоненту. Таким образом, единообразие концептуальной модели любой области моделирования, а также способ организации ее вычислений являются основой теории модельного синтеза и технологии модельно-ориентированного программирования.

2 Элементы модельного синтеза и модельно-ориентированного программирования

Агентный подход в моделировании известен со времен Левкиппа и Демокрита – современников Сократа и Элейских софистов. С тех пор и до наших дней придумано немало различных типов агентов для такого моделирования.

Попробуем показать в следующем подразделе, что модель-компонента не очередная эвристическая попытка придумать еще один тип агента, а минималистичное логическое следствие предположений, необходимых для возможности построения модели, в первую очередь, – гипотезы о замкнутости.

2.1 Гипотеза о замкнутости

Что такое гипотеза о замкнутости? Важнейшая составляющая модели – ее характеристики, отражающие состояние моделируемой системы и состояние влияющей на систему части внешнего мира. Первые из них называются внутренними, а вторые – внешними.

Гипотеза о замкнутости предполагает, что знания внутренних характеристик $x(t)$ и внешних характеристик $a(t)$ модели в момент t достаточно для детерминированного и однозначного вычисления ее внутренних характеристик на некотором интервале времени $(t, t+\Delta t)$ положительной длины Δt. Внешние характеристики $a(t)$ считаются наблюдаемыми в любой момент времени t. Здесь $x(t)$ и $a(t)$ – векторы.

Однозначность и детерминированность здесь относится именно к процессу вычислений. Их предметная область может быть стохастической, но вычислитель должен определенно знать, когда включить генератор случайных чисел, и какое распределение потом использовать.

Задача моделирования – составить прогноз эволюции системы на макроскопическом отрезке времени $[0,T]$. Это подразумевает вычисление значений траектории системы $x(t)$ на этом отрезке. Эти вычисления могут быть выполнены только на компьютере, если система достаточно сложна, а именно о таких и идет речь здесь. Это означает, что за конечное время мы можем вычислить значение траектории $x(t)$ лишь в конечном числе точек $t_1 < t_2 < ... < t_n$, $t_i \in [0,T]$, $i = 1,...,n$. Мы должны уметь получать представление о значениях траектории в остальных точках, на основании вычисленных, например, линейно аппроксимируя промежуточные значения.

Из сказанного следует, что мы в состоянии строить лишь модели с кусочно-гладкими траекториями. На более сложные объекты не хватит компьютерного времени. Возможность конечного числа разрывов первого рода следует из того, что если система сложна, то некоторые процессы будут всегда происходить мгновенно по сравнению с продолжительностью шага моделирования, при любом разумном шаге.

Еще о разрывах: поскольку мы моделируем приближенно, шаг моделирования Δt считается столь малым, что мы различаем лишь наличие – отсутствие разрыва на отрезке шага Δt, а где конкретно он произошел, в начале, середине или конце шага Δt – не различаем. Это позволяет нам для определенности всегда относить разрыв к левому концу отрезка Δt. Из этих же соображений считаем, что на шаге моделирования Δt может быть только один разрыв (если бы их было больше – мы все равно восприняли бы их слившимися в один). Таким образом, наличие разрыва на шаге моделирования подобно ординарному событию в теории потоков событий.

Теперь мы можем сформулировать гипотезу о замкнутости для исследуемых сложных систем.

Определение 1

Назовем модель замкнутой в точке $t \in [0,T)$, если на основании значений ее внутренних $x(t)$ и внешних $a(t)$ характеристик

1. мы можем определить, есть ли разрыв траектории Δx в точке t и если он есть – вычислить его однозначно;
2. мы можем найти число $\Delta t > 0$, такое, что на отрезке $(t, t + \Delta t] \subseteq (0,T]$, который назовем отрезком прогноза,
3. мы можем однозначно вычислить траекторию модели, начиная с точки $\{t, x(t) + \Delta x(t)\}$, как гладкую функцию времени на отрезке прогноза $(t, t + \Delta t]$. ■

Определение замкнутости в точке иллюстрирует Рис. 1.

Важное замечание: Рис. 1 также показывает, что шагом 4, мы переводим системное время вперед на Δt, и обнаруживаем себя в начале следующего шага моделирования, где снова начинаем с п. 1.

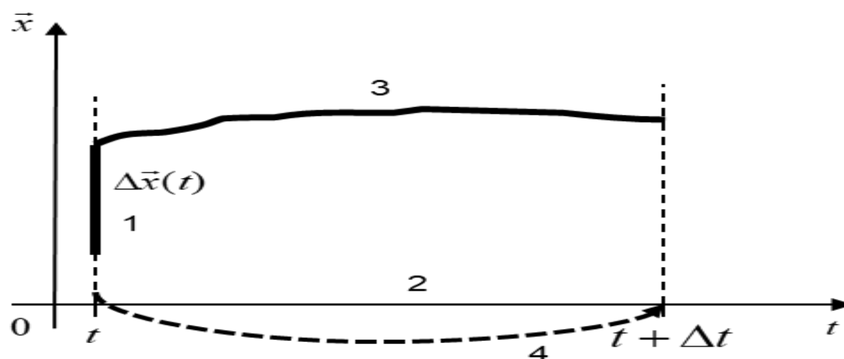

Рис. 1. Замкнутость в точке

Определение 2

Модель локально замкнута на отрезке $[0,T]$, если она замкнута в каждой точке $t \in [0,T)$. ■

Локальная замкнутость представляется нам необходимым условием возможности построения модели: сложно представить себе, как ее строить, если из какого-то момента времени невозможен положительный шаг вперед. Требование локальной замкнутости на отрезке $[0,T]$ покрывает каждую точку $t \in [0,T)$. связанным с ней отрезком прогноза $[t, t+\Delta t]$, на котором мы умеем вычислять траекторию системы. Это бесконечное покрытие. Если добавить требование непрерывности траектории системы слева в любой точке $t \in (0,T]$, этого будет достаточно для нахождения конечного подпокрытия (без этого требования могут возникать ситуации типа мухи фон Неймана – см. ниже – или лампы Томсона [5]). Возможность выбора конечного подпокрытия из покрытия отрезка моделирования отрезками прогноза точек – означает теорему существования для задачи моделирования.

Теорема существования несомненна, важна, однако для данной работы наиболее важным будет определение замкнутости модели в точке, замечание к нему и иллюстрирующий их Рис. 1. Что это означает для реализации системы?

То, что какова бы ни была область моделирования (техническая система, борьба популяций за выживание, распространение эпидемии или социальные процессы), вся динамика модели весьма проста и одинакова для любой предметной области. Это четырехтактный цикл, подобный циклу Карно или работе двигателя внутреннего сгорания, изображенный на Рис. 1 и описанный в определении замкнутости модели в точке (с добавлением перевода модельного времени).

2.2 Модель-компонента, модель-комплекс и язык программирования ЯОКК

Модель-компонента была формально описана как род структуры в смысле Н. Бурбаки в работе [4]. За недостатком места, мы не будем повторять это достаточно громоздкое описание, но иногда будем его цитировать. Скажем только, что оно было нужно, прежде всего для того, чтобы на языке теории множеств доказать, что объединение по определенным правилам нескольких моделей-компонент в комплекс, снова будет моделью-компонентой. Формальное описание модели-компоненты как рода структуры Н. Бурбаки было нужно в первую очередь для доказательства, что комплекс, объединяющий несколько компонент по определенным правилам, сам будет моделью-компонентой. И хотя формальное описание является адекватным и исчерпывающим, для программистской работы по описанию сложной системы и ее компонент, оно было бы слишком непривычным. Гораздо естественней для этого использовать специальный декларативный язык программирования ЯОКК (язык описаний компонент и комплексов).

Необходимость в непроцедурных языках, где описывается не то, что надо выполнить, а кто и как устроен, наряду с тем, кто, как и с кем связан, назрела достаточно давно. Несколько языков подобной ориентации приведено ниже с указанием разработчиков и областей применения.

- SQL (IBM, ANSI, базы данных) – 1986.
- Язык инструментальной системы MISS (ВЦ АН СССР) – 1986-1990.
- Язык IDL (CORBA, OMG) – 1991.
- OMT (HLA – Архитектура высокого уровня, – DMSO, IEEE) – 2000.
- Язык Slice промежуточного ПО ICE (ZeroC) – 2003.
- Язык XML (W3C, SOAP, Майкрософт) – 1996-2005.
- Язык UML (OMG, Партнеры UML) – 1997-2005.

В наши дни все более популярными становятся языки онтологий, такие как DOGMA, OntoUML, Common logic и многие другие. Поскольку направление модное, предметных областей, нуждающихся в концептуальном моделировании много, а смысл такого моделирования понимают далеко не все, то и языков возникает много.

Поскольку мы утверждаем возможность концептуального моделирования широкого класса предметных областей одной универсальной структурой – моделью-компонентой, большого разнообразия языков нам не надо. В модельном синтезе есть лишь одно основное понятие – модель-компонента, и одно вспомогательное – модель-комплекс, который при ближайшем рассмотрении оказывается тоже моделью-компонентой. Поэтому предлагается минималистичный язык ЯОКК, являющийся упрощением языка системы MISS, реализованного в 1990, [6].

В ЯОКК четыре типа дескрипторов:

1. Дескриптор типа данных.
2. Дескриптор методов и событий.
3. Дескриптор компонент.
4. Дескриптор комплексов.

Все дескрипторы, кроме дескриптора типа данных, состоят из заголовка и нескольких параграфов. У дескриптора типа данных только один параграф. Этот дескриптор – необязательная конструкция.

Все дескрипторы ЯОКК компилируются не в код, а в таблицы базы данных. Таким образом, существуют три эквивалентных способа описания модели-компоненты: родом структуры, описателем ЯОКК и таблицами базы данных. Первый способ хорош для теоретизирования, второй – инструмент разработчика для описания предметной области моделирования, третий – кристаллизованная в базе данных концептуальная модель предметной области, уже настолько конкретная, что универсальная программа организации вычислений способна работать на ее основе. Далее мы попытаемся приводить все три версии описания в соответствии друг другу. Заметим, что если и есть взаимно-однозначное соответствие этих описаний – то лишь на самом верхнем уровне. На более низких уровнях, конструкция одного способа описания может соответствовать нескольким частям конструкций других описаний и наоборот.

За недостатком объема статьи, проиллюстрируем конструкции ЯОКК на примере описания модели мухи фон Неймана. Два пешехода идут с постоянными скоростями навстречу друг другу. Между ними летает муха с постоянной по абсолютной величине скоростью, большей абсолютной скорости любого пешехода. Как только муха долетает до пешехода, она мгновенно разворачивается и летит к другому.

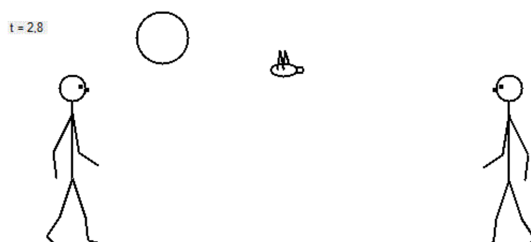

Рис. 2. Муха фон Неймана

Модель-комплекс «Муха фон Неймана» состоит из двух экземпляров модели-компоненты «Man» и одного экземпляра модели-компоненты «Fly».

Начнем с базисных множеств: $X, M, E, \{M_j\}_{j=1}^{N}, \{E_j\}_{j=1}^{N}$. Проще всего с X – характеристиками комплекса. Это объединенные гетерогенные данные (как **struct** в C). Эквивалент в базе данных модели – строка, где под каждый тип данных зарезервировано поле соответствующего размера.

Ниже – характеристики модели «Муха фон Неймана» в базе данных

Таблица 1. Таблица характеристик

No	**Fly_0_x**	**Fly_0_y**	**Man_0_x**	**Man_0_v**	**Man_1_x**	**Man_1_v**	**Dt**	**t**
1	1	3	0	1	50	-1.5	0.3	0

Первая строка этой таблицы заполняется вручную. Последующие-это результаты моделирования. Данные, относящиеся к конкретной модели, должны быть дополнены шагом моделирования Δt и модельным временем t.

Заметим, что из всех таблиц, приводимых здесь – эта единственная определяемая конкретной моделью – видом ее характеристик, остальные – универсальны для любых моделей.

Следующие базисные множества M – множество реализаций методов-элементов E – множество реализаций методов-событий модели. Соответствующие конструкции в ЯОКК – дескрипторы методов, хотя их функция несколько шире, чем просто описание соответствующих базисных множеств.

Движение мухи и обоих пешеходов (вычисление гладкой эволюции траектории на отрезке прогноза) обеспечивает медленный метод “move”. Он вычисляет координату x' в конце отрезка

прогноза, основываясь на значениях координаты x и скорости v в начале шага моделирования, а также на его продолжительности Δt по формуле $x' = x + v\Delta t$.

```
METHOD move;
//Если тип метода не указан, –
// по умолчанию он SLOW
ADDRESS: 192.168.1.75;
//где находятся реализации
INPUT
double x, v;
OUTPUT
double x;
END;
```

```
METHOD Uturn: FAST;
// ADDRESS: local; по умолчанию
INPUT
double v;
/**************
Если про OUTPUT ничего не сказано – он по
умолчанию такой же как INPUT, т.е.
OUTPUT
double v;
***************/
END;
```

У мухи есть быстрый элемент “Uturn”. Он разворачивает муху – меняет скорость v на $-v$.

Метод-событие “reaching” (муха долетает до пешехода) управляет переходом от медленного метода-элемента “move” к быстрому “Uturn”. Обратный переход – безусловный, т.е. после "Uturn" всегда "move".

Алгоритм события "reaching" наиболее сложный в этой модели. Муха должна знать координаты и скорости обоих пешеходов. Она ищет пешехода со скоростью противоположного знака и делит расстояние до него на сумму абсолютных величин своей скорости и скорости этого пешехода. Если расстояние между ними нулевое – событие уже наступило, если положительное – вычислено время до его наступления.

```
METHOD reaching: EVENT;
ADDRESS: simul.ccas.ru;
INPUT
double x, v, m0x, m0v, m1x, m1v;
// OUTPUT всех событий всегда –
// double dt; – время до наступления.
END;
```

Один из важных результатов компиляции приведенных примеров дескрипторов методов – заполнение Таблицы 2, чьи столбцы показывают, где искать реализации тех или иных методов.

Таблица 2. Таблица реализаций методов и событий

No	Method	Assembly	Address
1	move	Man	192.168.1.75
2	Uturn	Fly	local
3	reaching	Fly	simul.ccas.ru

Мы видим, что не вся информация из дескрипторов присутствует в таблице. Она будет отражена в других таблицах. Между отдельными конструкциями рода структуры, языка ЯОКК и базы данных может не быть взаимно-однозначных соответствий. Каждый из этих методов описания модели-компоненты имеет свою логику. Важно, что это три способа ее исчерпывающего описания.

Модель-комплекс «Муха фон Неймана» состоит из двух экземпляров модели-компоненты «Man» и одного экземпляра модели-компоненты «Fly». Модель-компонента «Man» имеет единственный процесс, состоящий из единственного медленного метода «move». Модель-компонента «Fly» имеет единственный процесс, состоящий из медленного метода «move», быстрого метода «Uturn» и управляющего их переключением события «reaching». Поэтому у модели-комплекса три процесса: два достались от экземпляров «Man» и один – от «Fly».

Продолжаем искать соответствия базисным множествам. $\{M_j\}_{j=1}^{N}$ и $\{E_j\}_{j=1}^{N}$ – это методы и события процессов. Таблица 3 ниже как раз о методах и событиях процессов. Кроме того, в нее попала часть информации из дескрипторов методов, не вошедшая в таблицу реализаций. Она также включает информацию о начальных методах процессов, которая задается соотношением типизации рода структуры $\{m_j^0 \subset M_j\}_{j=1}^{N}$, а в ЯОКК начальные методы задаются в параграфе элементов дескриптора компоненты.

Таблица 3. Таблица методов и событий

No	Method / Event name	Type	Current	Process	Realization
1	Man_0_move	1	☑	1	1
2	Man_1_move	1	☑	2	1
3	Fly_0_move	1	☑	3	1
4	Fly_0_Uturn	2	☐	3	2
5	Fly_0_reaching	3	☐	3	3

Первый столбец таблицы – ключи. Второй – имена методов и событий. Они достаточно сложные, так как получены автоматически при компиляции дескриптора комплекса в соответствующий ему дескриптор компоненты. Третий столбец – тип элемента: 1 – медленный метод; 2 – быстрый; 3 – событие. Четвертый столбец – булевская переменная указывает, является ли данный метод текущим. На самом первом такте указываются начальные методы. Пятый столбец указывает принадлежность элемента процессу. Последний шестой столбец – вторичные ключи, указывающие позицию элемента в таблице реализаций.

Таблица переключений задает соответствие между переключаемыми методами и событиями. Событие лишь одно, а обратный переход – безусловный, что показано в таблице заданием невозможного значения вторичного ключа. В столбцах с третьего по пятый – вторичные ключи, указывающие положение элемента в таблице методов и событий.

Таблица 4. Таблица переключений

No	Process	Current method	Next method	Event
1	3	3	4	5
2	3	4	3	0

В описании модели-компоненты родом структуры, аналогом этой таблицы будут соотношения типизации $\left\{ sw_j \subset E_j \times M_j \times M_j \right\}_{j=1}^{N}$ с довольно громоздкими аксиомами R_8. В ЯОКК – параграф SWITCHES в дескрипторе компоненты.

Приведем примеры дескрипторов компонент. Компонента «Fly»

```
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 = x;
move.v = v;
 x = move.x;
Uturn.v = v;
v = Uturn.v;
reaching.x = x;
reaching.v = v;
reaching.m0x = man0Phase.x;
reaching.m0v = man0Phase.v;
reaching.m1x = man1Phase.x;
reaching.m1v = man1Phase.v;
END;
```

Далее, дескриптор компоненты «Man»

```
COMPONENT Man;
PHASE
double x;
PARAMETERS
double v;
ELEMENTS
move;
COMMUTATION
move.x = x;
move.v = v;
x = move.x;
END;
```

Далее, дескриптор компоненты «Man»

```
COMPONENT Man;
PHASE
double x;
PARAMETERS
double v;
ELEMENTS
move;
COMMUTATION
move.x = x;
move.v = v;
x = move.x;
END;
```

Мы видим, что в ЯОКК характеристики компонент разбиты на внутренние (PHASE) и внешние (PARAMETERS). Это сделано для дополнительного контроля во время компиляции дескриптора: например, внешние переменные не должны находиться в левых частях операторов коммутации возвращаемых методами параметров.

Отметим, что конструкция компоненты “Fly" сложнее, чем компоненты “Man", что нашло отражение в большей сложности соответствующего дескриптора. Обратим внимание на параграф SWITCHES – это эквивалент Таблицы 4 переключений в базе данных и соотношений типизации $\left\{sw_j \subset E_j \times M_j \times M_j\right\}_{j=1}^{N}$ с аксиомами R_8 в описании рода структуры.

Можно заметить, что операторы коммутации начинают играть все большую роль в дескрипторах компонент. Чтобы не умножать число таких операторов, можно укрупнять описания типов, и стараться коммутировать крупные агрегаты переменных. В базе данных операторам коммутации соответствуют таблицы входной и выходной коммутации. В описании рода структуры – соотношения типизации $\left\{m_{j,in} \subset M_j \times \beta(X)\right\}_{j=1}^{N}$, $\left\{m_{j,out} \subset M_j \times \beta(X)\right\}_{j=1}^{N}$, $\left\{e_{j,in} \subset E_j \times \beta(X)\right\}_{j=1}^{N}$ и аксиомы R_5 – R_7 В базе данных – это таблицы входных и выходных коммутаций.

Мы приведем пример таблиц коммутации, но для всего комплекса «Муха фон Неймана» Поэтому, сперва приведем ЯОКК дескриптор этого комплекса.

```
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;
```

В операторах коммутации компонент комплекса внешние переменные компонент, наоборот, абсолютно законны в левых частях операторов коммутации. Тогда в правой части таких операторов должны быть внутренние переменные других компонент, вычисляющие внешние переменные, указанные в левой части.

Таблица 5. Таблица выходных коммутаций

No	**Method**	**Output offset**	**Phase offset**	**Length**
1	1	0	16	8
2	2	0	32	8
3	3	0	0	8
4	4	0	0	8

Теперь посмотрим, как выглядит коммутация возвращаемых методом параметров с характеристиками модели в базе данных. Первый столбец – ключи. Второй столбец – вторичные ключи, указывающие положение элемента в Таблице 3 методов и событий. Третий столбец – смещение в байтах от начала записи параметров метода. Четвертый столбец – смещение в байтах от начала записи характеристик модели. Пятый – длина параметра в байтах.

Таблица 6. Таблица входных коммутаций

No	**Method**	**Input offset**	**Phase offset**	**Length**
1	1	0	16	8
2	1	8	24	8
3	2	0	32	8
4	2	8	40	8
5	3	0	0	8
6	3	8	8	8
7	4	0	8	8
8	5	0	0	8
9	5	8	8	8
10	5	16	16	8
11	5	24	24	8
12	5	32	32	8
13	5	40	40	8

Таблица входных коммутаций такая же, как и выходных, только больше. Первый столбец – ключи. Второй столбец – вторичные ключи, указывающие положение элемента в Таблице 3 методов и событий. Третий столбец – смещение в байтах от начала записи параметров метода. Четвертый столбец – смещение в байтах от начала записи характеристик модели. Пятый – длина параметра в байтах. Обратим внимание, если заменить число 8 на 48 в строке с первичным ключом 8, строки 9-13 можно удалить. Это говорит о просторе для оптимизации компилятора ЯОКК.

Мы видим, что онтологии широкого класса предметных областей моделирования представимы сочетанием базы данных достаточно простой структуры (почти все ее таблицы были проиллюстрированы на примере модели мухи фон Неймана) с не слишком сложной универсальной программой организации вычислений, которая обеспечивает всю динамику.

Алгоритм универсальной программы следует из гипотезы о замкнутости, он четырехтактный, как показано на Рис. 1. Подробно он был описан в [4], здесь мы попробуем дать описание этого алгоритма с привязкой его к базе данных.

2.3 Программа организации вычислений и ориентированность на высокопроизводительные системы

В работе [4] был приведен алгоритм универсальной программы организации имитационных вычислений с привязкой к описанию модели-компоненты родом структуры (аксиома R_{11}). Здесь мы добавим привязку к базе данных, что ближе к программированию.

Сначала выбираем стандартный шаг моделирования Δt.

Считаем, что в начале шага моделирования известны текущие методы и значения внутренних переменных (на первом шаге это начальные значения внутренних характеристик и начальные методы всех процессов). Эти методы находим в столбце “Current” Таблицы 3. – Таблицы методов и событий.

Предполагаем, что внешние характеристики модели доступны для наблюдения всегда.

Далее:

1. Вычисляем события, связанные с текущими методами всех процессов. Правила переходов (в Таблице 4. – Таблице переключений. Текущий метод – в столбце “Current method”, соответствующее событие – в столбце “Event”) определяют связь событий с текущими методами процессов. События можно вычислять параллельно, но дожидаясь вычисления последнего, для продолжения процесса. Если есть наступившие события, проверяется нет ли переходов (столбец “Next method” той же Таблицы 4) к быстрым методам из $\left\{f_j\right\}_{j=1}^{N}$ (По вторичному ключу из столбца “Next method” в Таблице 4, проверяем поле “Type” метода в Таблице 3. Величина 2 означает быстрый метод). Если таковые есть – считаются быстрые методы (они становятся текущими). Их тоже можно вычислять параллельно, но для продолжения процесса вычислений, нужно дождаться выполнения всех, а затем вернуться к началу п. 1. Если нет переходов к быстрым методам – переходим к новым медленным методам из $\left\{s_j\right\}_{j=1}^{N}$ (значение 1 поля “Type” в Таблице 3) и переходим к началу п. 1.
2. Если нет наступивших событий, из всех прогнозов наступления выбираем ближайший $\Delta\tau$.
3. Если стандартный шаг Δt не превосходит прогноза наступления ближайшего события $\Delta t \le \Delta\tau$, – вычисляем текущие медленные элементы со стандартным шагом Δt. Иначе – уменьшаем шаг моделирования до $\Delta\tau$. Теорема существования, упомянутая в подразделе 3.1, дает шанс избежать бесконечного цикла, уменьшая шаг моделирования. Медленные методы из множества $\left\{s_j\right\}_{j=1}^{N}$ (находим их по вторичному ключу столбца “Next method” Таблицы 4, проверяя поле “Type” метода в Таблице 3. Значение 1 этого поля означает медленный метод), также могут вычисляться параллельно, с ожиданием завершения последнего.
4. Возвращаемся к началу п. 1.

Почему мы имеем право вычислять параллельно методы параллельных процессов модели-компоненты? Для компонент самого низкого уровня надежду на верное распараллеливание дает гипотеза о замкнутости, предполагающая однозначность вычислительного процесса. Понятно, что все можно легко испортить даже и с таким предположением, но это лишь будет означать

необходимость отладки дескрипторов. Чтобы сохранить однозначность вычислений при объединении компонент в комплекс, придуманы специальные правила такого объединения [4] и там же доказана достаточность этих правил для сохранения однозначности.

Мы видим, что чем сложнее модель – тем больше у нее параллельных процессов, и тем большее количество вычислений можно осуществлять параллельно.

2.4 Базы данных и некоторые приемы программирования

Коль скоро оказалось, что база данных составляет основу концептуальной модели практически любой предметной области (динамика, конечно, тоже очень важна, но если программа организации имитационных вычислений написана, отлажена, работает и никогда не меняется, – про нее можно забыть), важным вопросом становится оптимизация работы базы данных и удобство работы с этой базой – качество СУБД.

Здесь мы представим некоторые программистские решения, реализованные при создании инструментальной системы имитационного моделирования MISS, более 30 лет назад. Они помогли решить многие до сих пор актуальные проблемы, касающиеся, в том числе и баз данных и СУБД, тем не менее, применение подобных методов в последующие годы нам не известно.

Разработка системы MISS велась на PC XT. Одна из серьезных проблем этой архитектуры – невозможность прямого использования памяти компьютера свыше 640 K, даже если такая память имеется. Операционные системы, работающие в защищенном режиме, появились в 90-х, после окончания разработки MISS.

Выходом из этого затруднения оказалось создание софтверной системы виртуальной памяти. Доступная оперативная и дисковая память разбивалась на страницы по 32 или 64 K. Доступные для адресации 640 K служили окном отображения виртуальных страниц. Виртуальных страниц могло быть до 256 в каждом из 8 классов памяти. Таким образом, можно было работать не более чем с 128 M виртуальной памяти. По нынешним временам, это конечно, совсем немного – 2 Г оперативной памяти компьютера сейчас считается минимумом. Но тогда 100 M считалось очень неплохой емкостью съемного диска для тогдашних мейнфреймов. Гораздо важнее то, как можно было работать с виртуальной памятью.

Каждый байт виртуальной памяти мог иметь постоянный 32 разрядный адрес. Два байта – адрес на виртуальной странице, еще байт – номер страницы, еще 3 бита – номер класса памяти, остальное – на системные нужды. Постоянство адреса означает, что он продолжает работать при следующем запуске программы (при условии штатного завершения предыдущего). Виртуальный адрес всегда был адресом того или иного типа данных, определенного ранее. Это позволяло легко находить в базе данных тип адресуемой записи и использовать для работы с ней соответствующие ему методы.

Была реализована библиотека модулей (сейчас бы сказали набор классов) для работы с виртуальной памятью по виртуальным адресам – выделение и освобождение памяти, копирование в запись и из записи и т.д. В библиотеку входили различные средства программирования: работа со списками; с блокированными списками (способ ускорения работы списков); хранение в виртуальной памяти, загрузка и выполнение компьютерных программ (напомним, методы и события входят в базовые множества модели-компоненты); работа с картинками – средства формирования видеокадров в виртуальной памяти.

Что это могло дать для создания, управления и работы баз данных? Например, в реляционных базах данных немало времени уходит на поиск по вторичным ключам. В примерах предыдущего подраздела немало таблиц, состоящих почти полностью из вторичных ключей. В базе данных MISS все вторичные ключи были виртуальными адресами соответствующих кортежей, доступ к которым был просто по виртуальному адресу – без какого-либо поиска. Связь адреса с типом того, что он адресует, позволяла создавать универсальные СУБД. Например, в MISS база данных модели создавалась автоматически в результате компиляции дескрипторов и операции сборки модели. В результате, в редакторе базы данных был доступ не только к полям кортежей встроенных типов (integer, double, Boolean, etc.), но и к гораздо более сложным полям. Например, если тип поля был виртуальный адрес списка – можно было путешествовать по записям этого списка, далее раскрывая любое из полей этих записей.

Или если поле – картинка, она открывалась в графическом редакторе.

База данных международных авиарейсов в бывшем СССР [7], реализованная средствами MISS, работала достаточно быстро на не слишком мощном ноутбуке на базе 80386.

Любая сколь угодно сложная структура, построенная на виртуальной адресации, мгновенно сохраняется на диск. Для этого нужно переписать на диск все виртуальные страницы из RAM и запомнить состояния окна отображения и процессора. Так же мгновенно она восстанавливается с

диска – надо переписать нужные виртуальные страницы в окно отображения и восстановить состояние процессора. Это решает вопрос сериализации / структуризации данных, над которым многие годы бьются разработчики операционных систем (почему так медленно загружается / выключается мой компьютер?!).

Заключение

Теория модельного синтеза предлагает законченную сквозную технологию описания, синтеза и программной реализации (модельно-ориентированное программирование) моделей сложных систем. Важным следствием этой теории является возможность поставить в соответствии (хотя бы в качестве мысленного эксперимента) практически любой агентной сложной системе ее математическую модель – соответствующую ей модель-компоненту.

Можно сказать, что теория модельного синтеза для проекции этого мира в мир систем, обладающих реактивным поведением, решает спор диалога Платона «Парменид»: все на свете есть модель-компонента, и любой агент, и любой комплекс агентов, и любой комплекс комплексов, и комплекс всего на свете (конечно, лишь как мысленный эксперимент). Таким образом, модельный синтез реализует в области имитационного моделирования формулу: «Одно во всем и все в Одном».

В плане программной реализации имитационных моделей, модельный синтез предлагает новую парадигму программирования – модельно-ориентированное программирование. Несмотря на похожесть названий, модельно-ориентированное программирование радикально отличается от известных концепций разработки программных систем управляемой моделями, – MDA (Model Driven Architecture), MDE (Model Driven Engineering) и MDD (Model Driven Development). Последние являются всего лишь надстройками над объектно-ориентированным подходом и не выводят программирование из области искусства в область технологий. Основным средством разработки в этих концепциях по-прежнему является императивное программирование на языках типа С или Java.

Модельно-ориентированное программирование впервые полностью реализует в программировании идеи САПР, со всеми их достоинствами и недостатками. При этом программирование на всех уровнях становится полностью декларативным, что существенно облегчает последующую отладку программной системы.

Программная система видится состоящей из «атомов» – моделей-компонент, которые могут быть объединены в комплексы, которые сами, являясь моделями-компонентами, могут быть объединены в комплексы более высокого уровня и т.д. Сравнивая с объектно-ориентированном подходом, можно сказать, что в модельно-ориентированном программировании господствует множественное наследование снизу-вверх, т.е. модельный синтез.

Разработан декларативный язык программирования ЯОКК (язык описания компонент и комплексов). На нем описывается устройство компонент (некий более близкий для восприятия и работы программиста аналог родов структур Н. Бурбаки) и образования из них комплексов. Важной особенностью МО-программирования является то, что описатели ЯОКК компилируются не в компьютерный код, а в базу данных. Известно, что язык UML и связанные с ним концепции управляемой моделями разработки ПО погубило плохое качество кода после двойной компиляции. В МО-программировании вопрос качества кода не стоит – языковые конструкции компилируются в базу данных или верно, или неверно. Эффективность вычислений зависит от универсальной программы организации вычислений, которую можно «вылизать до блеска». Заметим, что база данных – третий (наряду с ЯОКК) аналог описания универсального агента бурбаковскими родами структур.

Следует отметить, что описанная выше сквозная технология описания, синтеза и программной реализации имитационных моделей сложных многокомпонентных систем не является лишь теоретическими мечтами – как раз, наоборот, все начиналось именно с полной практической реализации инструментальной системы имитационного моделирования (с декларативным языком программирования типа ЯОКК, средствами отладки и компиляции, базой данных, и, конечно же, универсальной программой организации имитационных вычислений) в среде MS-DOS. Эта система заняла I место в категории профессиональных программ в проводившемся в 1990 г. в СССР японской фирмой ASCII corporation (в то время крупнейший разработчик компьютерных игр в Японии, сейчас – подразделение издательства) Всесоюзном конкурсе компьютерных программ. Правда, тогда совсем не было теоретического обоснования этой технологии, оно было дано позже в [4], – лишь практическая реализация.

Литература

1. *Böhm C., Jacopini G.* Flow diagrams, Turing machines and languages with only two formation rules. Communications of the ACM, Vol. 9, No 5, 1966, pp. 366–371. DOI:10.1145/355592.365646.
2. *Бродский Ю.И.* О сложных процессах, аналогиях, структурах, математическом моделировании, трех мирах и информатике. // Моделирование, декомпозиция и оптимизация сложных динамических процессов, 2016. Т.31, №1(31), С. 86-108.
3. *Бурбаки Н.* Теория множеств. М.: Мир. 1965. 456 с.
4. *Бродский Ю.И*. Модельный синтез и модельно-ориентированное программирование. М.:ВЦ РАН, 2013,142с.
5. *Thomson, J. F.:* Tasks and Super-Tasks. Analysis 1(15), 1-13 (1954). DOI:10.1093/analys/15.1.1.
6. *Бродский Ю.И., Лебедев В.Ю.* Инструментальная система имитации MISS. М.: ВЦ АН СССР, 1991, 180 с.
7. *Бродский Ю.И., Лебедев В.Ю*. О технологии разработки баз данных на основе инструментальной системы MISS. // Моделирование, декомпозиция и оптимизация сложных динамических процессов. 1996. Т. 11, №1-1(11), С. 61-67.