Структурная теория сложных систем. Модельно-ориентированное программирование

Ю. И. Бродский¹, Л. В. Круглов²

¹ Федеральный исследовательский центр "Информатика и управление" РАН, Москва, Россия

² Московский педагогический государственный университет, Москва, Россия

Аннотация. Предлагается формальное определение компьютерной модели сложной системы, как рода структуры в смысле Н. Бурбаки – род структуры «модель». Класс математических объектов, определяемый этим родом структуры, обладает следующими двумя свойствами: комплекс, созданный объединением математических объектов рода структуры «модель» по определенным правилам, сам является математическим объектом этого рода структуры. Организация вычислительного процесса для всех математических объектов рода структуры «модель» однотипна и поэтому может быть реализована единой универсальной программой, притом ориентированной на параллельные вычисления. Наличие этих двух свойств позволяет построить сквозную технологию разработки, описания, синтеза и программной реализации моделей сложных систем – Модельно-ориентированное (МО) программирование.

Ключевые слова: сложные системы, структурная теория, модельный синтез, модельно-ориентированное программирование.

DOI 10.14357/20718632230409

EDN VTCHQU

Введение

Более полувека назад Д. Кнут начал издавать многотомник «Искусство программирования» [1]. Многое в программировании за это время изменилось, не изменилось одно: программирование как было искусством, так и осталось, не став рутинной технологией, которой можно было бы легко и быстро научиться в высшей и даже средней школе.

Для разработки все более сложных программных комплексов были придуманы языки типа UML [2] и выросшие из него языки концептуального моделирования, однако успех их был достаточно скромным, превратить с их помощью искусство в технологию так и не удалось.

Однако мысль, что можно программировать, не программируя, жила среди программистов с

60 гг XX в. до наших дней. Сейчас очень популярны идеи Low-Code, No-code и Zero-Code [3] в области Web-программирования, да и само Web-программирование – движение в этом направлении: многие вопросы решаются разработанными ранее методами.

Современный зерокодинг – не единственный пример упрощения программирования. Первым, на наш взгляд, был язык GPSS (General Purpose Simulation System) [4], появившийся в 1961 г. и до сих пор сохраняющий круг пользователей. Это язык, ориентированный на область систем массового обслуживания, тем не менее, судя по названию, создатели считали, что системы массового обслуживания покрывают всю область имитации. Программирование на этом языке состоит в описании требуемой комбинации готовых конструкций, таких как генераторы заявок,

очереди различных типов, каналы обработки, и задание числовых параметров этих конструкций. С помощью GPSS можно построить весьма сложную систему, совсем не программируя.

Следующим примером может быть язык DYNAMO Дж. Форрестера [5] (начало 70 гг. XX в.) и связанная с ним система программирования. Здесь системы обыкновенных дифференциальных уравнений описываются в терминах уровней (основные переменные), потоков (их производные по времени), темпов (слагаемых, из которых складываются потоки) и т.д. Пользователю, как и в GPSS, остается собрать систему из готовых конструкций и задать их числовые параметры.

Что общего в приведенных примерах? Их можно назвать предметно-ориентированными [6]. Подобная система программирования предлагается и в данной работе, только ее предметная область существенно шире, чем в приведенных выше примерах.

Структурный подход переводит внимание исследователя с многообразия различных предметных областей в информационное пространство. И здесь оказывается, что для весьма широкого класса сложных систем достаточно весьма небольшого набора способов моделирования происходящего. Теория, позволяющая единообразно конструировать сложные системы, была

описана в работах [7, 8], и названа там модельным синтезом. Данная статья посвящена практическому следствию модельного синтеза, – сквозной технологии описания, синтеза и реализации сложных программных систем. Такой метод был назван в [8] модельно-ориентированным. В нем весьма высокий уровень инкапсуляции – единицей является универсальный агент – модель сложной системы, описываемая родом структуры «модель».

1. Род структуры «модель»

Для удобства последующих ссылок мы процитируем из [7] формальное определение модели сложной системы с минимальными пояснениями. Оно было нужно для доказательства в [7, 8] на языке теории множеств, что объединение конечного числа «моделей» в комплекс по определенным правилам, также является объектом рода структуры «модель» и для определения единых правил организации вычислений для любого объекта этого рода структуры. Для разработчика предлагается более привычный декларативный язык программирования ЯОКК, которому далее будет посвящен раздел, а для универсальной программы организации вычислений существует третье эквивалентное описание «модели» – ее база данных.

$$\Sigma(\mathbb{N}, \xi) = \langle \mathbf{X}, M, E, \{M_j\}_{j=1}^N, \{E_j\}_{j=1}^N, \mathbb{N};$$

$$N \subset \mathbb{N},$$

$$\mathbf{x} \subset \mathbf{X}, \mathbf{a} \subset \mathbf{X}, \quad (1)$$

$$s \subset M, f \subset M, \quad (2)$$

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

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

$$\{m_{j,in} \subset M_j \times \mathfrak{B}(\mathbf{X})\}_{j=1}^N, \quad (5)$$

$$\{m_{j,out} \subset M_j \times \mathfrak{B}(\mathbf{X})\}_{j=1}^N, \quad (6)$$

$$\{e_{j,in} \subset E_j \times \mathfrak{B}(\mathbf{X})\}_{j=1}^N, \quad (7)$$

$$\left\{ sw_j \subset E_j \times M_j \times M_j \right\}_{j=1}^N, \quad (8)$$

$$\left\{ m_j^0 \subset M_j \right\}_{j=1}^N, \quad (9)$$

$$\left\{ p_j \subset \mathfrak{B}(M_j) \times \mathfrak{B}(E_j) \times M_j \times \mathfrak{B}(E_j \times M_j \times M_j) \right\}_{j=1}^N; \quad (10)$$

$$\xi : |N| = 1,$$

$$R_1 : (\mathbf{x} \cup \mathbf{a} = \mathbf{X}) \& (\mathbf{x} \cap \mathbf{a} = \emptyset),$$

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

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

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

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

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

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

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

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

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

R_{10} : аксиома однозначности вычисления характеристик сложной системы,

R_{11} : аксиома организации имитационных вычислений>.

Здесь угловая скобка «<>» обозначает начало описания рода структуры, а парная угловая скобка «>>» – его конец. Три раздела этого описания: базисные множества, соотношения типизации и аксиомы разделяются между собой двумя точками с запятой – после разделов базисных множеств и соотношений типизации. Обозначение $\left\{ \dots_j \right\}_{j=1}^N$ используется для краткости,

заменяя повторение содержимого скобок N раз через запятую, с изменением индекса j последовательно на $1, \dots, N$. $\mathfrak{B}(\cdot)$ – множество всех подмножеств множества в скобках.

$\mathbf{X}$ – множество характеристик модели. Соотношения типизации (1) и аксиома R_1 утверждают, что все $\mathbf{X}$ состоит из двух

непересекающихся подмножеств: $\mathbf{X}$ – внутренних и $\mathbf{a}$ – внешних характеристик. Далее идет M – множество различных реализаций методов-элементов, элементарных умений нашей модели, которое также состоит из двух непересекающихся подмножеств: s – медленных элементов, вычисляющих гладкую эволюцию траектории на положительном отрезке прогноза, и f – быстрые методы, вычисляющие величины разрывов первого рода, о чем говорят соотношения типизации (2) и аксиома R_2 .

Далее $\{M_j\}_{j=1}^N$ – методы процессов. В отличие от M , где все методы уникальны, в M_j методы могут повторяться. Смысл повторений – например, один и тот же алгоритм может работать с разными данными. Соотношение типизации (3) и аксиома R_3 гласят, что все методы процессов берутся из единого хранилища M , у каждого там есть единственный прообраз. Точно так же $\{E_j\}_{j=1}^N$ – события процессов, отвечающие за переключение текущих методов-элементов модели. Так же события в E_j могут и повторяться. Соотношения типизации (4) и аксиома R_4 гласят, что все события процессов берутся из единого хранилища E .

Соотношения типизации (5), (6) вместе с аксиомами R_5 и R_6 описывают, как методам процессов передаются характеристики и как от них принимаются внутренние характеристики.

Соотношения типизации (7) вместе с аксиомой R_7 описывают, как событиям процессов передаются характеристики. Возвращают события всегда неотрицательное число, которое есть либо прогноз времени до его наступления, если положительно, либо ноль, если оно наступило.

Далее, $\mathbb{N}$ – вспомогательное базисное множество натуральных чисел, из которого вспомогательное соотношение типизации $N \subset \mathbb{N}$ выделяет подмножество N , которое в силу

вспомогательной аксиомы $\xi : |N| = 1$ оказывается просто натуральным числом (содержательно это количество параллельных процессов в сложной системе).

Процессы определяются соотношениями типизации (10) и аксиомами R_9 (косвенно в их определения входят также соотношения (8) с аксиомами R_8 , определяющие правила переключения элементов в процессах и соотношения (9), определяющие начальные элементы процессов. Каждый процесс p_j , $j = 1, \dots, N$, определяемый (10) и R_9 , 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. Соотношения (8) и R_8 задают правила переключения элементов под влиянием событий.

2. Модель, комплекс и язык программирования ЯОКК

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 мы приведем: это SQL, [9]; язык инструментальной системы MISS, [10]; язык ОМТ в HLA, [11]; язык Slice фирмы ZeroC, [12]; язык XML, [13]; язык UML [2, 14] и т.д.

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

областей одной универсальной структурой – родом структуры «модель», большого разнообразия языков нам не надо.

Насколько хорош язык родов структур для доказательства теорем, настолько он малоприменим для описания человеком результатов анализа и синтеза сложной системы. Поэтому предлагается минималистичный декларативный язык ЯОКК (язык описания компонент и комплексов), являющийся упрощением языка системы MISS, реализованного в 1990, [10]. Его часть, отвечающая за описания компонент – эквивалентна приведенному выше описанию рода структуры «модель». Часть, отвечающая за описание комплексов – воплощает правила объединения компонент в комплекс, описанные в [7, 8].

В ЯОКК четыре типа дескрипторов:

1. Дескриптор типа – описатель пользовательского типа данных, содержащий информацию о входящих в него полях.
2. Дескриптор методов и событий – описатель, содержащий информацию и об их реализации (адрес, входные и выходные параметры).
3. Дескриптор компонент – описатель структуры и поведения модели-компоненты.
4. Дескриптор комплексов – описатель структуры модели-комплекса.

Все дескрипторы ЯОКК компилируются не в код, а в таблицы базы данных. Это снимает немаловажный вопрос о качестве компиляции: все вычисления производятся универсальной программой в соответствии с аксиомой R_{II} , а также входящими в базисные множества методами-элементами и методами-событиями. Таким образом, существуют три эквивалентных способа описания «модели»: родом структуры, описателями ЯОКК и базой данных модели. Первый способ хорош для теоретизирования, второй – инструмент разработчика для описания предметной области моделирования, третий – кристаллизованная в базе данных концептуальная модель, с которой способна работать универсальная программа организации вычислений. Мы попытаемся приводить соответствия всех трех версий описания. 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 конструкциям других. Важно, что это три способа исчерпывающего описания.

Проиллюстрируем конструкции ЯОКК на примере описания модели мухи фон Неймана [7]. 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 разворачивается и летит к другому.

Комплекс «Муха фон Неймана» состоит из двух экземпляров «модели» Man и одного экземпляра «модели» Fly.

Начнем с основных базисных множеств модели: $\mathbf{X}, M, E, \{M_j\}_{j=1}^N, \{E_j\}_{j=1}^N$. Проще

всего с $\mathbf{X}$ – 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 таблице характеристик базы данных (Табл. 1). Первая строка этой таблицы заполняется вручную. Последующие – это результаты моделирования. Данные, относящиеся к конкретной модели, должны быть дополнены шагом моделирования Δt и модельным временем.

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.

Табл. 1. Таблица характеристик

№	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

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 SLOW {
  ADDRESS "192.168.1.75:3000";
  INPUT { double x, v; }
  OUTPUT { double x; }
}
```

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

```
METHOD Turn FAST {
  ADDRESS "localhost";
  INPUT { double v; }
  OUTPUT { double v; }
}
```

Метод-событие «Reaching» (муха долетает до пешехода) управляет переходом от медленного метода-элемента “move” к быстрому «Turn». Обратный переход – безусловный, т.е. после «Turn» всегда «Move». У всех событий имеется единственный выходной параметр «double dt» по умолчанию, обозначающий ожидаемое время до наступления события.

```
METHOD Reaching EVENT {
  ADDRESS "simul.ccas.ru";
  INPUT { double x, v, m0x, m0v, m1x, m1v; }
}
```

Один из важных результатов компиляции приведенных примеров дескрипторов методов – заполнение Табл. 2, чьи столбцы показывают, где искать реализации тех или иных методов. Мы видим, что не вся информация из дескрипторов присутствует в таблице. Она будет отражена в других таблицах.

Модель-комплекс «Муха фон Неймана» состоит из двух экземпляров «модели» Man и одного экземпляра «модели» Fly. «Модель» Man имеет единственный процесс, состоящий из единственного медленного метода Move. «Модель» Fly имеет единственный процесс, состоящий из медленного метода Move, быстрого метода Turn и управляющего их переключением события Reaching. Поэтому у 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 ме-

тоды-элементы и события процессов, информация о которых содержится в Табл. 3 ниже. Кроме того, в нее попала информация о типе метода из дескрипторов методов, не вошедшая в таблицу реализаций (столбец Type). Она также включает информацию о начальных методах процессов, которая задается соотношением типизации рода

структуры $\{m_j^0 \in M_j\}_{j=1}^N$. В ЯОКК начальные

методы задаются в параграфе элементов дескриптора компоненты.

Табл. 2. Таблица реализаций методов и событий

№	Method name	Assembly	Address
1	Move	Man	192.168.1.75:3000
2	Turn	Fly	localhost
3	Reaching	Fly	simul.ccas.ru

Табл. 3. Таблица методов и событий

№	Method/event name	Type	Current	Process	Realization
1	Man_0_move	Slow	☐	1	1
2	Man_1_move	Slow	☐	2	1
3	Fly_0_move	Slow	☐	3	1
4	Fly_0_turn	Fast	☐	3	2
5	Fly_0_reaching	Event	☐	3	3

Табл. 4 Таблица переключений

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

Имена методов и событий в Табл. 3 достаточно сложные, так как получены автоматически при компиляции дескриптора комплекса в соответствующий ему дескриптор компоненты. Столбец Supgent указывает, является ли данный метод текущим, на самом первом такте указываются начальные методы. В столбце Process указывается принадлежность элемента процессу. Столбец Realization содержит вторичные ключи, указывающие позицию элемента в таблице реализаций.

Таблица переключений задает соответствие между переключаемыми методами и событиями. В нашем случае событие лишь одно, а обратный переход – безусловный, что показано в таблице заданием пустого значения вторичного ключа. В столбцах с 3-го по 5-й – вторичные ключи, указывающие положение элемента в таблице методов и событий.

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

Отметим, что конструкция «модели» Fly сложнее, чем «модели» Man, что нашло отражение в большей сложности соответствующего дескриптора. Обратим внимание на параграф SWITCHES – это эквивалент Табл. 4 переключений в базе данных и соотношений типизации (8) с аксиомами R_8 в описании рода структуры. В базе данных операторам коммутации соответствуют таблицы входной и выходной коммутации, в описании рода

структуры – соотношения типизации (5) – (7) 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).PARAM.man0Phase.x =
Man(0).PHASE.x;
    Fly(0).PARAM.man1Phase.x =
Man(1).PHASE.x;
  }
}
    
```

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.

Теперь посмотрим, как выглядит коммутация возвращаемых методом параметров с характеристиками модели в таблице выходных коммутаций базы данных (Табл. 5) 1-й столбец – ключи, 2-й – вторичные ключи, указывающие положение элемента в Табл. 3 методов и событий, 3-й столбец – смещение в байтах от начала записи параметров метода, 4-й – смещение в байтах от начала записи характеристик модели, 5-й – длина параметра в байтах.

Табл. 5. Таблица выходных коммутаций

№	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

Табл. 6 Таблица входных коммутаций

№	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 (Табл. 6) такая же, как и выходных, только больше. 1-й столбец – ключи, 2-й – вторичные ключи, указывающие положение элемента в Табл. 3 методов и событий, 3-й столбец – смещение в байтах от начала записи параметров метода, 4-й столбец – смещение в байтах от начала записи характеристик модели, 5-й – длина параметра в байтах.

Мы видим, что модели широкого класса предметных областей 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. Подробно он был описан в [7], здесь мы попробуем дать описание этого алгоритма с привязкой его к базе данных.

3. Организация вычислений и параллельность

В работе [7] был приведен алгоритм универсальной программы организации имитационных вычислений с привязкой к описанию «модели» родом структуры (аксиома R_{11}). Здесь мы добавим привязку к базе данных, что ближе к программированию.

Сперва 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 методов и событий. Предполагаем, что внешние характеристики модели доступны для наблюдения всегда.

Далее:

1. Вычисляем события, связанные с текущими методами всех процессов. Правила переходов (текущий метод в столбце Current method таблицы переключений и соответствующее событие в столбце Event) определяют связь событий с текущими методами процессов. События можно вычислять параллельно, дожидаясь вычисления последнего для продолжения процесса. Если есть наступившие события, проверяется нет ли переходов (столбец Next method таблицы переключений) к быстрым методам из

$\{f_j\}_{j=1}^N$. Если таковые есть – считаются быстрые методы (они становятся текущими). Их

тоже можно вычислять параллельно, но для продолжения процесса вычислений нужно дождаться их выполнения, а затем вернуться к началу п. 1. Если нет переходов к быстрым методам – переходим к новым медленным методам

из $\{s_j\}_{j=1}^N$ 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 $\Delta t \leq \Delta \tau$, – 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 $\{s_j\}_{j=1}^N$ также могут вычисляться параллельно, с ожиданием завершения последнего.

4. Возвращаемся к началу п. 1.

Почему мы имеем право вычислять параллельно методы процессов «моделей»? Для «моделей» 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 [7], и в них же показана достаточность правил для сохранения однозначности.

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

4. Модельно-ориентированная и объектно-ориентированная парадигмы

Итак, была описана новая, модельно-ориентированная парадигма программирования, существенно отличающаяся как от объектно-ориентированного, так и от похожих по названиям концепций разработки программных систем, управляемых моделями – MDA (Model Driven Architecture), MDE (Model Driven Engineering) и MDD (Model Driven Development), которые являются всего лишь надстройками над объектно-ориентированным подходом. МО-программирование реализует методы САПР в области программирования.

Остановимся на сравнении основных понятий МО-программирования и объектно-ориентированного программирования. Прежде всего отметим, что объектно-ориентированный подход в настоящее время представлен в двух видах. Один из них можно назвать «базовым», в его основе лежат такие понятия, как класс, объект, типизация, наследование, инкапсуляция, полиморфизм, 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 [2], а также выросшими из него более современными языками концептуального моделирования. 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 отношении 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.

В отличие от объектного анализа и UML, модельный синтез минималистичен в наборе базовых понятий: в нем имеется единственное основное понятие – модель (объект рода структуры «модель») и одно вспомогательное понятие – комплекс, состоящий из «моделей», который, в конце концов, в результате устранения возникших неоднозначностей также может стать объектом рода структуры «модель» [7]. 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. Примерно одинаково понимаются также характеристики и методы. Что же касается использования методов, здесь имеется существенная разница. Она состоит в том, что в

объектно-ориентированном программировании объект затем и нужен, чтобы вызывать его методы в различных программах, причем в качестве параметров методу можно передать, а также принять от него любые переменные, удовлетворяющие его сигнатуре – не обязательно характеристики его объекта. В МО-программировании метод «модели» 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. Инкапсуляция в МО-программировании такова, что не позволяет напрямую добраться до методов. Оперировать можно лишь «моделями».

Однако это вовсе не означает, что методы совсем недоступны. Наоборот, например, в реализованном макете распределенной системы имитационного моделирования, библиотеки методов публикуются в интернете для общего использования, и возможны «модели», 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, 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» – она всегда ведет себя так, как умеет, и поэтому включение ее в любой комплекс всего лишь вопрос правильной коммутации.

Переопределение методов в МО-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. С точки зрения геометрической теории декомпозиции [15], набор базовых классов представляет собой F-редукции будущих наборов листовых классов. 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».

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 собачонка». 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 (P-наследование, с позиций работы [15]).

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. На этом этапе и теряется модульность объектно-ориентированной программы: все используемые методы так или иначе связаны друг с другом. Для настоящей модульности не хватает уровня инкапсуляции: из объектов приходится вручную в нужной последовательности вызывать их методы. Самый сложный этап построения большой программной системы остается неформализованным искусством.

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, [14]) 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 изменениям работы [2] при переизданиях, создается впечатление, что и сами авторы начинают склоняться к этому.

Имеется ряд средств, позволяющих компилировать UML-описания в заготовки классов

обычных 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. В рамках этой парадигмы остается и возникшая на основе UML концепция разработки программных систем, управляемая моделями, MDA, MDE или MDD.

При компиляции языка UML и автоматической трансформации в код описаний MDA/MDD/MDE неизбежно возникает вопрос о их качестве и эффективности получаемого кода – UML достаточно сложный язык. Именно эти сложности и заставляют считать UML в первую очередь средством документирования и эскизного проектирования [14].

Следует заметить, что в последнее время все более востребованными становятся высокопроизводительные многопроцессорные и многоядерные и распределенные вычислительные системы. Одно из преимуществ МО-программирования – нацеленность программы, организующей имитационные вычисления (аксиома поведения «модели» R_{11}), на параллельное выполнение множества методов-элементов и методов-событий параллельно развивающихся процессов. При усложнении модели, количество методов, допускающих параллельное вычисление растет.

МО-программирование предполагает высокий уровень инкапсуляции. Основная единица, «модель», наделена поведением и этим похожа на ядро операционной системы [16], поэтому вызывать какие-либо ее методы извне не нужно и невозможно, они всегда активизируются автоматически, в соответствии с программой поведения модели (правилами переключения методов-элементов в процессах «моделей» – соотношения типизации (8) и аксиома R_8).

Наличие у модели поведения позволяет последовательно провести идею модульности программной системы, до степени сравнимой, например, с САПР микроэлектроники. 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 можно включать в любой комплекс, зная правила коммутации и выполняемую функцию, забыв подробности ее реализации. При этом работать она будет всегда как умеет, в соответствии с описанной ранее программой поведения. Играющие роль «паяльника» операторы коммутации, например языка ЯОКК, полностью декларативны.

Модельный подход исключают императивное программирование. Это следует из того, что гипотеза о замкнутости [7] гарантирует нам функциональную зависимость на интервале модельного времени $(t, t + \Delta t]$ внутренних характеристик модели от ее характеристик на левом конце этого интервала – в точке t . Поэтому вычислительный процесс вполне может быть осуществлен в функциональной парадигме (пусть и на обычных языках), что существенно упрощает отладку, а иногда может и увеличить степень параллельности получаемого кода.

Заключение

Структурная теория предлагает единый способ описания предметных областей широкого класса сложных систем. На его основе строится сквозная технология разработки, описания, синтеза и программной реализации сложных программных систем – МО-программирование. Несмотря на похожесть названий, МО-программирование в корне отличается от известных концепций разработки программных систем, управляемых моделями, – MDA, MDE и MDD. Последние являются надстройками над объектно-ориентированным подходом и не выводят программирование из области искусства в область технологий.

МО-программирование реализует в программировании идеи САПР со всеми их достоинствами и недостатками. Программирование на всех уровнях становится декларативным, что существенно облегчает последующую отладку программной системы. Система видится состоящей из моделей, которые объединяются в комплексы, которые сами являясь моделями, могут быть объединены в комплексы более высокого уровня и т.д.

Разработан декларативный язык программирования ЯОКК, более близкое для программиста средство описания объектов рода структуры «модель» и образования из них комплексов. Описатели ЯОКК компилируются не в компьютерный код, а в базу данных. Эффективность вычислений зависит от универсальной программы их организации, позволяющей добиваться высокой степени параллелизма.

В работе [17] доказана алгоритмическая полнота «модели» – основной конструкции МО-программирования. Это говорит об алгоритмической полноценности предлагаемой парадигмы.

Отметим, что описанная технология не является чистой теорией, наоборот, все начиналось с ее практической реализации в среде MS-DOS [10]. В настоящее время разработан макет МО системы программирования под ОС Ubuntu [17].

Литература

1. Кнут Д. Искусство программирования. Т. 1. Основные алгоритмы. - М.: Мир, 1976. 735 с.
2. 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.
3. <https://zerocoder.ru/> (дата обращения: 29.07.2022).
4. Шрайбер Т. Дж. Моделирование на GPSS. М.: Машиностроение, 1980, 592 с.
5. Форрестер Дж. Мировая динамика. М.: Физматгиз. 1978. 168 с.
6. Brodsky Yu.I. Simulation Software // System Analysis and Modeling of Integrated World Systems. Oxford: EOLSS Publishers Co. Ltd., 2009. P. 287-298.
7. Бродский Ю.И. Структурная теория сложных систем. Модельный синтез. // Математическое моделирование и численные методы, 2022. №3(35), С. 105-122.
8. Бродский Ю.И. Роды структур Н. Бурбаки в задаче синтеза имитационных моделей сложных систем и модельно-ориентированное программирование // Журнал Вычислительной математики и математической физики, 2015, Т. 55, № 1, С. 153–164. DOI:10.7868/S0044466915010056
9. Date C.J., Darwen H. A Guide to the SQL standard: a users guide to the standard database language SQL, 4th ed., Addison Wesley, 1997. 414 p.
10. Бродский Ю.И., Лебедев В.Ю. Инструментальная система имитации MISS. М.: ВЦ АН СССР, 1991, 180 с.
11. Kuhl F., Weatherly R., Dahmann J. Creating Computer Simulation Systems: An Introduction to the High Level Architecture NY: Prentice Hall PTR, 1999. 212 p.
12. The Slice Language <https://doc.zeroc.com/ice/latest/the-slice-language> (доступ 18 октября 2023).
13. Хантер Д., Рафтер Д., Фаусетт Д., и др. XML. Работа с XML, 4-е издание М.: «Диалектика», 2009. 1344 с.
14. Фаулер М. UML. Основы, 3 изд. Пер. с англ. СПб: Символ-Плюс, 2004. 192 с.
15. Павловский Ю.Н., Смирнова Т.Г. Введение в геометрическую теорию декомпозиции. М.: ФАЗИС, ВЦ РАН, 2006. 169 с.
16. Карпов В.Е., Коньков К.А. Основы операционных систем: курс лекций. М.: Физматкнига, 2019. 328 с.
17. Круглов Л.В. Система модельно-ориентированного программирования // Труды Института системного анализа Российской академии наук. 2023. Т. 73, № 2. С. 59-69. DOI:10.14357/20790279230206.

Бродский Юрий Игоревич Федеральный исследовательский центр «Информатика и управление» РАН, Москва, Россия. Ведущий научный сотрудник, доктор физ.-мат. наук. Область научных интересов: структурная теория сложных систем, имитационное моделирование сложных систем. E-mail: yury_brodsky@mail.ru

Круглов Леонид Вячеславович Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский педагогический государственный университет», Москва, Россия. Младший научный сотрудник, аспирант. Область научных интересов: теоретическая информатика, моделирование, информационные технологии. E-mail: leonid.kruglov.cmc@gmail.com

Structural Theory of Complex Systems. Model-Oriented Programming

Yu. I. Brodsky¹, L. V. Kruglov^{II}

¹Federal Research Center “Computer Science and Control” of the Russian Academy of Sciences, Moscow, Russia

^{II} Moscow Pedagogical State University, Moscow, Russia

Abstract. We propose a formal definition of the complex system computer model as a species of structure in the sense of N. Bourbaki – the “model” species of structure. The class of mathematical objects defined by this species of structure has the following two properties: a complex created by combining mathematical objects of the “model” species of structure according to the certain rules is itself a mathematical object of the same species of structure. The process of computation organization is same for all

mathematical objects of this species of structure and therefore can be implemented by a single universal program. These two properties make it possible to build an end-to-end technology for the development, description, synthesis and software implementation of complex systems models - Model-oriented programming.

Keywords: complex systems, structural theory, model synthesis model-oriented programming.

DOI 10.14357/20718632230409

EDN VTCHQU

References

1. Knuth D. The Art of Computer Programming, vol.1. Fundamental Algorithms. 3rd ed. Addison-Wesley. 1997. 672 p.
2. Booch G., Rumbaugh J., Jacobson I. UML. User Guide, 2-ed., Addison-Wesley, 2005.
3. <https://zerocoder.ru/>, last accessed 2022/06/06.
4. Schriber T.J. Simulation using GPSS. New York, Wiley, 1974. 534 p.
5. Forrester J. World Dynamics. Wright-Allen Press. 2nd ed. 1973. 123 p.
6. Brodsky, Yu. I. Simulation Software, // System Analysis and Modeling of Integrated World Systems. Oxford: EOLSS Publishers Co. Ltd., 2009. P. 287-298.
7. Brodsky Yu.I. Strukturnaya teoriya slozhnykh system. Model'nyj sintez. [Structural theory of complex systems. Model synthesis.] Matematicheskoe modelirivanie i chislennyje metody [Mathematical modeling and computational methods], 2022, no. 3(35), pp. 105-122.
8. Brodsky Yu.I. Bourbaki's Structure Theory in the Problem of Complex Systems Simulation Models Synthesis and Model-Oriented Programming // Computational Mathematics and Mathematical Physics Vol. 55 No. 1, 2015. P. 148-159. DOI:10.1134/S0965542515010054
9. Date C.J., Darwen H. A Guide to the SQL standard: a users guide to the standard database language SQL, 4th ed., Addison Wesley, 1997. 414 p.
10. Brodsky Yu.I., Lebedev V.Yu., Instrumental'naya sistema imitatsionnogo modelirovaniya MISS [Instrumental Simulation System MISS], Moscow: CC AS of the USSR, 1991, 180 p. <http://www.ras.ru/ph/0005/VDJBWF5N.pdf> (in Russian)
11. Kuhl F., Weatherly R., Dahmann J. Creating Computer Simulation Systems: An Introduction to the High Level Architecture NY: Prentice Hall PTR, 1999. 212 p.
12. The Slice Language <https://doc.zeroc.com/ice/latest/the-slice-language> (accessed 18 October, 2023).
13. Hunter D., Rafter J., Fawcett J., et al. Beginning XML 4th ed. Indianapolis: Wiley, 2007.
14. Fowler M. UML Distilled 3-ed., Addison-Wesley, 2004, 192 p.
15. Pavlovsky Yu.N., Smirnova T.G. Vvedenie v geometricheskuyu teoriyu dekompozitsii. [Introduction to the Geometric Decomposition theory] Moscow: FAZIS, CC RAS, 2006. 169 p. (in Russian).
16. Karpov V.E., Kon'kov K.A. Osnovy operatsionnykh sistem: kurs lektsiy. [Basim of Operating systems: course of lectures] Moscow: Fizmatkniga, 2019. 328 p. (in Russian)
17. Kруглов Л.В., Sistema model'no-orientirovannogo programirovaniya [Model-oriented programming system] // Trudy Instituta Sistemnogo Analiza Rossijskoj Akademii Nauk [Proceedings of ISA RAS], 2023. Vol. 73, No. 2, pp. 59-69. DOI:10.14357/20790279230206. (in Russian).

Brodsky Yury I. Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences, Leading Research Scholar, Doctor of Mathematics. Research interests – structural theory of complex systems, simulation of complex systems.

Kruglov Leonid V. Moscow Pedagogical State University, Scientific Center of High-Priority Studies and Academic and Teaching Personnel Training, 1/1 Malaya Pirogovskaya str., Moscow, 119435, Russia. E-mail: leonid.kruglov.cmc@gmail.com