УДК 519.87

Ю. И. Бродский, А. Н. Мягков

ДЕКЛАРАТИВНОЕ И ИМПЕРАТИВНОЕ ПРОГРАММИРОВАНИЕ В ИМИТАЦИОННОМ МОДЕЛИРОВАНИИ СЛОЖНЫХ МНОГОКОМПОНЕНТНЫХ СИСТЕМ

Изложены подходы к моделированию таких сложных систем, про которые хорошо известно, из каких компонент они состоят, какие функции эти компоненты выполняют, по каким правилам взаимодействуют между собой. Проблема моделирования, причем весьма непростая, заключается в воспроизведении поведения и оценке возможностей такой системы в целом. Рассмотрены вопросы эффективности применения различных парадигм программирования для решения задачи синтеза многокомпонентной системы.

E-mail: yury_brodsky@mail.ru

***Ключевые слова*:** *имитационное моделирование, сложные системы, парадигмы программирования, поведение системы, объектно-ориентированное программирование, декларативное программирование.*

Под императивным программированием понимают распространенный подход к написанию программ на языках программирования типа FORTRAN, семейства C или Java, согласно которому программа представляет собой последовательность инструкций-приказов, выполняемых на компьютере. Использование императивного программирования позволяет описать последовательность действий, достаточную (на взгляд разработчика) для получения результата поставленной задачи. При этом не предполагается, что результат заранее известен, и скорее всего цель императивного программирования состоит в получении этого результата (например, численное решение системы уравнений в частных производных или изучение поведения сложной системы в имитационном эксперименте с ее моделью). Более того, на стадии отладки программ полученный результат, вообще говоря, удивляет и поражает своих создателей.

При декларативном программировании, наоборот, описывают, каким должен быть известный заранее желаемый результат, например, статическая страничка HTML, документ в системе LaTeX или DVD-проект в системе авторинга Scenarist. При этом выбор последовательности действий, приводящей в системе декларативного про-

граммирования к описанному результату, как правило, оставляют на усмотрение соответствующего компилятора или интерпретатора (о чем свидетельствует иногда заметная разница представления одного и того же документа HTML в различных браузерах).

Под парадигмой программирования, согласно определению Памелы Зейв, понимают набор представлений о некотором классе программных систем, допускающих реализацию с помощью этой парадигмы набора представлений о способе программирования [1].

Помимо упомянутых выше императивной и декларативной парадигм, существуют и другие весьма интересные парадигмы программирования, например функциональная, однако наша основная цель — обсуждение путей решения вполне конкретной прикладной задачи, а не изучение теории программирования.

Задача синтеза многокомпонентной системы. Эта задача, по крайней мере на первый взгляд, хорошо укладывается в императивную парадигму программирования. В самом деле, отталкиваясь от имеющихся знаний об отдельных компонентах сложной системы, мы беремся построить синтез всей системы, основанный на воспроизведении известного поведения каждой ее компоненты, и при этом собираемся наблюдать и изучать поведение системы в целом, не известное нам заранее. В основе синтеза многокомпонентной системы лежит идея, высказанная в работах Н.П. Бусленко, — дать каждой из компонент, про которые нам все известно, максимально проявить себя, учитывая при этом и все межкомпонентные связи [2].

Основным инструментом реализации проектов императивного программирования являются объектноориентированные языки программирования. В свое время эти языки возникли в значительной мере как ответ на запросы исследователей, занимавшихся имитационным моделированием. Одним из самых первых объектноориентированных языков можно считать язык Симула-67. Структура сложной системы, состоящей из большого числа экземпляров различных типов компонент, хорошо описывается иерархией классов объектноориентированного программирования (ООП). У ООП множество известных всем достоинств, которые и сделали этот подход бесспорным лидером в программировании последних 25 лет. Однако применительно к поставленной задаче ООП имеет существенные недостатки.

Главный из них (конечно, для класса задач, о которых идет речь в настоящей работе) — отсутствие у объекта поведения. Объекту присущи характеристики, его методы, а поведения — умения давать ответы на стандартные запросы внутренней и окружающей среды — у него нет. Безусловно, ООП в соединении с соответствующим программным обеспечением (ПО) промежуточного уровня позволяет объектам даже в распределенных системах взаимодействовать между

собой, наблюдать характеристики и других объектов и пользоваться их методами, однако все это делается в некотором смысле «вручную». По сути, поведение коллектива программистов, приступающих к созданию сложного проекта и вооруженных набором библиотек полезных классов, можно уподобить человеку, имеющему несколько компьютеров, каждый из которых оснащен большим набором полезных прикладных программ, но из системных программ — только загрузчик, и поставили перед ним задачу сделать из этого распределенную систему. Понятно, что современный программист возмутился бы страшно: «Так не бывает! Где моя сетевая операционная система? Где, наконец, ПО промежуточного уровня?» Однако при создании сложной системы средствами ООП дело обстоит именно так: к сложности содержательных вычислений предметной области добавляется еще и сложность организации поведения объектов, а также их связей и взаимодействий между собой.

Поведение компоненты сложной системы есть функциональный аналог операционной системы в области системного ПО и функциональный аналог бытовой культуры в социальных системах, т. е. способность давать стандартные ответы на стандартные запросы внутренней и окружающей среды. При имитационном моделировании сложных систем очень важно уметь моделировать поведение компонент. Как упоминалось ранее, именно из их поведения естественно синтезировать поведение системы в целом.

В истории развития информатики известно несколько подходов к объектному программированию, в силу которых они обладают поведением. Эти подходы зародились в среде исследователей, занимавшихся проблемами искусственного интеллекта, и известны как акторная модель [3] и агентное программирование [4, 5]. Несмотря на то что авторы настоящей работы согласны с главным посылом этих подходов — важностью моделирования поведения агентов системы — детали описания этого поведения [3—5], по их мнению, слишком окрашены спецификой проблематики искусственного интеллекта, т. е. недостаточно универсальны. Так, у Шохема [4, 5] поведение — это поведение ментальное, о состоянии которого рассуждают в категориях «убеждения», «обязанности», «способности» и т. д. Акторная модель Хьюитта [3] интересна своей ориентированностью на параллельные вычисления, однако асинхронный обмен акторов сигналами-сообщениями и порождение новых акторов представляются нам избыточными конструкциями, затрудняющими реализацию подобных систем [6].

Проблема моделирования заключается в том, что, согласившись на наличие поведения у компонент системы, нужно учиться описывать и реализовывать это поведение. Такое поведение, с одной стороны, достаточно сложное, так как представляет собой поведение эле-

ментарной сложной системы. Заранее оно неизвестно, поскольку может зависеть от внешних по отношению к рассматриваемой компоненте условий, на которые она должна уметь должным образом реагировать. Следовательно, такие поведенческие моменты необходимо вычислять, и алгоритмы их вычисления должны быть описаны на императивных языках программирования. С другой стороны, кое-что о поведении компоненты нам всегда заранее известно, так как она в силу исходных предположений работы имеет в предметной области моделирования достаточно хорошо изученный прообраз. Поэтому обычно можно предварительно перечислить весь набор ее элементарных действий и какое из них может перейти в другое, под действием каких обстоятельств, поскольку так устроен прообраз этой компоненты в предметной области моделирования. Именно это знание, коррелирующее с пониманием устройства в предметной области как прообраза отдельной компоненты, так и всей многокомпонентной модели, мы имеем заранее (еще до построения модели), и оно не зависит от хода имитационных экспериментов. Следовательно, знание об устройстве модели сложной системы вполне можно выразить на декларативном языке программирования.

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

Предлагаемое разделение описаний на декларативное и императивное оказывается весьма технологичным: декларативную и императивные части можно отлаживать независимо одна от другой. Самая сложную императивная составляющая программы распадается на ряд независимых между собой законченных элементарных алгоритмов. При этом она редуцируется настолько, что чаще всего возможности ООП оказываются слабо востребованными: алгоритмически цельную программу с фиксированным набором входных и выходных параметров чаще всего нетрудно реализовать даже на языке FORTRAN. И это существенно облегчает отладку системы. Вся объектная ориентированность, идущая от предметной области моделирования, оказывается в декларативном описании модели.

Одной из первых сред программирования, воплотивших рассмотренную выше концепцию разделения описания сложной системы на декларативную и императивную составляющие, была разработанная в ВЦ АН СССР инструментальная система имитационного моделирования MISS (Multilingual Instrumental Simulation System). Концеп-

ция программирования, лежащая в ее основе, предложена в работах [7, 8], а полное описание среды моделирования на уровне руководства пользователя — в [9]. В системе MISS в среде MS-DOS реализована система программирования на специальном декларативном языке описания сложных систем, интегрированная с базой данных, системой поддержки выполнения модели и системой презентации результатов моделирования. В качестве императивных языков программирования, на которых написаны вычислительные алгоритмы, разрешены две версии языка MODULA-2, а также языки C и C++ в Борландовской реализации. До этого моделирующее сообщество в основном создавало языки моделирования императивного типа.

Впоследствии идея разделения описания сложной системы на декларативную и императивную части применялась неоднократно. Так, в спецификации архитектуры верхнего уровня (High Level Architecture — HLA) [10] — средстве создания сложных распределенных моделей — устройство системы, ее компонент и связей между ними описывается специальными шаблонами. В коммерчески успешной отечественной инструментальной системе имитационного моделирования AnyLogic [11] в качестве декларативного языка описания сложной системы применяется интерактивная графическая система с отображением на экране пиктограмм компонент будущей системы и построением связей между ними. В качестве императивного языка программирования для системы AnyLogic используется язык Java. Наконец, появился и даже стал международным стандартом язык моделирования объектных систем UML [12], однако авторам настоящей статьи он не симпатичен в силу своей громоздкости и, порой, за счет этого — неоднозначности. В качестве результатов компиляции транслятор UML может выдавать заготовки модулей для императивных языков.

В настоящее время в ВЦ РАН продолжает развиваться концепция разделения описания сложной системы на декларативную и императивную составляющие; работает макет инструментальной системы распределенного моделирования. Некоторое затруднение вызывает название этой концепции — такие названия, как агент, актор и даже компонента уже заняты, и в программистском сообществе под ними понимают вполне определенные и отличные от описанного выше подходы. Сами разработчики концепции в работах [7—9, 13—15] называли свой подход объектным или объектно-событийным, что тоже не совсем верно, так как не отражает главных его моментов — наличия поведения у компоненты и декомпозицию описания сложной системы на декларативную и императивную составляющие.

Концепция описания модели [6] базируется на понятии компоненты. Компонента — в некотором смысле «элементарная» сложная система. Основой конструкции компоненты служит объект объектно-

го анализа в том смысле, что описываемая ниже компонента есть некий тип или класс моделей, а заполняться данными и запускаться на счет будут экземпляры этого класса. Однако главное отличие компоненты от объекта в том, что компонента — это объект с поведением. Опишем устройство компоненты.

1. Характеристики. Компонента, как и объект, имеет характеристики. Эти характеристики разобьем на внутренние и внешние. К внутренним относятся характеристики, которые компонента моделирует, к внешним — информация о внешнем мире.

2. Процессы. Функциональность компоненты удобно структурировать следующим образом. Считается, что компонента реализует один или несколько параллельно выполняющихся процессов. Процесс состоит в последовательном чередовании элементов — алгоритмически элементарных методов. Если какой-либо процесс какую-то часть модельного времени не выполняется, удобно считать, что в это время он выполняет «пустой» элемент, не изменяющий никаких характеристик компоненты.

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

По отношению к модельному времени некоторые элементы (сосредоточенные или быстрые) выполняются мгновенно. Быстрыми элементами можно моделировать дискретные характеристики системы. Выполнение других элементов занимает определенное время. Если при этом элемент для любого промежутка времени $\Delta\tau$, не превосходящего стандартный шаг моделирования Δt, выдает некий осмысленный результат, такой элемент называют распределенным или медленным. Распределенные элементы — естественное средство вычисления непрерывных характеристик модели.

Может оказаться, что выполнение элемента занимает определенное модельное время, но результат его действия наступает лишь в конце, после полного выполнения элемента, т. е. никаких промежуточных результатов за время, меньшее полного времени выполнения, нет. Такие элементы называют условно-распределенными. Вообще говоря, условно-распределенные элементы можно не рассматривать как отдельный класс, а моделировать парой: пустой распределенный элемент, за которым идет сосредоточенный, выдающий результат.

Частью предлагаемой концепции является жесткая дисциплина работы методов с фазовыми переменными модели: каждый метод имеет право изменять только «свои» переменные. Эта дисциплина основана на принятии предположения о детерминированности и однозначности имитационных вычислений. В рамках предлагаемой

концепции конфликт доступа, возникающий, когда методы A и B вычисляют одну и ту же характеристику $X = X_A$ и $X = X_B$, может быть разрешен, например, введением метода C, который, получая на входе в качестве параметров X_A и X_B, вычисляет на их основании искомую характеристику X, устраняя тем самым не только конфликт доступа к ней, но и, очевидно, возникающую неоднозначность вычисления упомянутой характеристики. Принятая дисциплина доступа к характеристикам позволяет вызывать параллельно те методы, которые в модельном времени выполняются одновременно.

Как и всякий метод, каждый элемент имеет входные и возвращаемые параметры. Концепция моделирования предполагает возможность распределенного вычисления элементов, т. е. элемент, вообще говоря, может быть найден разработчиком компоненты в Интернете, поэтому его параметры таковы, какими их сделал его автор, и, скорее всего, никак не согласованы с характеристиками компоненты, которые реализовал ее разработчик. Поэтому при описании компоненты необходимо уделять внимание описанию коммутаций входных параметров элементов с внутренними и внешними характеристиками компоненты и выходных параметров элементов — с ее внутренними характеристиками.

4. События. Событие — это то, что нельзя пропустить при моделировании динамики системы — точки синхронизации различных ее функциональностей, представляемых процессами, т. е. моменты времени, в которые 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 — метод, входными параметрами которого является подмножество внутренних и внешних характеристик компоненты, а выходной параметр один — прогнозируемое время до наступления этого события. Если это время равно нулю, значит, событие уже наступило. События управляют чередованием элементов в процессе.

Если для каждой упорядоченной пары элементов процесса $\{A, B\}$ возможен переход, ему обязательно соответствует 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 $E_{\{A, C\}}$ свидетельствует лишь о том, что

разработчик модели при ее проектировании упустил из рассмотрения этот случай, которому, быть может, должен соответствовать переход $\{A, D\}$. Возможно, тем, кто знаком с архитектурой компьютера или событийно-ориентированными языками программирования, данное определение событий покажется странным и вызовет такие вопросы, как, где здесь нечто подобное обработчикам событий или прерываний? Дело в том, что в моделировании (если, конечно, не брать моделирование полунатурное, когда к компьютерному комплексу подключают реальное изделие) чаще всего наступление события должна определить сама модель, даже если это событие неожиданное, например случайное, в нужный момент модель вполне детерминированно должна запустить генератор случайных чисел, чтобы выяснить, не произошло ли оно. А подобному порядку действий вполне адекватно отвечает изложенная выше концепция событий.

Выполнение компоненты. Компонента представляет собой элементарную, но тем не менее полноправную модель сложной системы. Поэтому ее можно запустить на выполнение имитационного эксперимента, если, конечно, имеются начальные данные и способ нахождения внешних характеристик компоненты в моменты наступления событий.

Правила запуска компоненты на выполнение следующие. Во-первых, задают стандартный шаг моделирования Δt. Во-вторых, полагают, что в начале шага моделирования известны текущие элементы всех процессов и все внутренние и внешние характеристики модели. Далее с помощью специализированной программы:

1) вычисляются события, связанные с текущими элементами процессов. Если есть наступившие события, проверяется, нет ли переходов к быстрым (сосредоточенным) элементам, если они есть — выполняются быстрые элементы (они становятся текущими), затем осуществляется возврат к началу п. 1; если нет переходов к быстрым элементам, совершаются переходы к новым медленным (распределенным) элементам, затем возврат к п. 1;

2) если нет наступивших событий, из всех прогнозируемых событий выбирается ближайшее с шагом $\Delta\tau$;

3) если стандартный шаг моделирования не превосходит прогнозируемого времени до ближайшего события, $\Delta t \le \Delta\tau$, рассчитываются текущие распределенные элементы со стандартным шагом Δt. В противном случае, они вычисляются с шагом $\Delta\tau$ до ближайшего спрогнозированного события;

4) процесс вычислений возвращается к п. 1.

Приведенные правила выполнения компоненты могут вызывать такие вопросы, как не может ли выполнение быстрых элементов в п. 1, а также уменьшение шага времени в п. 3 привести к зацикliва-

нию программы за счет возникновения точек накопления системных событий (подробно см. в [6]. К сожалению, полностью гарантировать отсутствие зацикливания можно лишь для непрерывных систем, описываемых дифференциальными уравнениями с правой частью, удовлетворяющей условию Липшица по совокупности переменных. Для более широкого класса моделей [6] можно доказать (неконструктивно) существование нециклящегося набора событий.

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

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

какие компоненты и в каком количестве экземпляров в него входят;

коммутацию компонент внутри комплекса, если она имеет место, т. е. внутренние переменные каких компонент являются внешними переменными и каких именно компонент комплекса.

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

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

внутренние переменные комплекса — объединение внутренних переменных всех его компонент;

процессы комплекса — объединение всех процессов его компонент;

методы комплекса — объединение всех методов его компонент;

события комплекса — объединение всех событий его компонент;

внешние переменные комплекса — объединение всех внешних переменных его компонент, из которого исключаются все те переменные, которые моделируются явно какими-либо компонентами.

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

Для описания состава и устройства сложной системы разработан специальный декларативный язык ЯОКК (язык описания комплексов и компонент), который развит в сторону упрощения языка MISS [9] и описан в [6, 13].

Отметим, что применение декларативного программирования в имитационном моделировании не ограничивается описанием устройства сложной системы. Если система и в самом деле достаточно сложная, всегда возникает вопрос рациональной организации ее данных, т. е. проектирования и описания базы данных. Общепринятым средством в этом случае служит язык SQL, который применительно к задаче описания баз данных является декларативным, хотя может быть и императивным, если речь идет о выборках и обновлении данных.

Еще одна область имитационного моделирования, в которой могут применяться декларативные описания, — область подготовки презентаций результатов моделирования. Языки описания подобных презентаций неизвестны, однако необходимость в них явно имеется. При этом самое простое, но тем не менее весьма полезное, что можно сделать, — по таблице-выборке из базы данных строить диаграммы, как в MS Excel или в OpenOffice Calc. Возможно также применение популярных в последнее время геоинформационных систем — отображение пиктограммами на интересующей карте динамики перемещения моделируемых объектов. И наконец, использование простейшей анимации в стиле Flash-морфинг форм и перемещение объектов с возможным изменением размеров. Более серьезная анимация пока остается прерогативой компьютерных игр.

Основной вывод — моделирование многокомпонентной системы является сложным процессом не только вследствие имитационных вычислений (несомненно, эти вычисления могут быть весьма сложны), но и императивных, на долю которых остается не более четверти сложности всего проекта. Не менее сложны организация данных модели, устройства модели (состав, связи, поведение компонент), а также организация отображения результатов моделирования. Последние три задачи обычно допускают декларативное описание. Возможно, все это коррелирует с тем, что в развитых экономиках, кото-

рые, несомненно, являются сложными системами, непосредственных производителей материальных не слишком много — все остальные заняты чем-то другим.

Работа выполнена при финансовой поддержке РФФИ (*грант № 10-07-00176*) *и РГНФ* (*грант № 12-06-00932*).

СПИСОК ЛИТЕРАТУРЫ

1. Zave P. A compositional approach to multiparadigm programming. IEEE Software, 6(5): 15—25, September 1989.
2. Бусленко Н. П. Моделирование сложных систем. М.: Наука, 1978.
3. Hewitt C. Viewing Control Structures as Patterns of Passing Messages Journal of Artificial Intelligence. June 1977.
4. Shoham Y. Agent-oriented programming // Artificial Intelligence. Vol. 60, 1993. P. 51–92.
5. Shoham Y. MULTIAGENT SYSTEMS: Algorithmic, Game-Theoretic, and Logical Foundations Cambridge: Cambridge University Press, 2010.
6. Бродский Ю. И. Распределенное имитационное моделирование сложных систем. М.: ВЦ РАН, 2010.
7. Бродский Ю. И., Лебедев В. Ю., Огарышев В. Ф., Павловский Ю. Н., Савин Г. И. Общие проблемы моделирования сложных организационно-технических систем // Вопросы кибернетики. Проблемы математического моделирования и экспертные системы. М.: Научный совет АН СССР по комплексной проблеме «Кибернетика», 1990. С. 42–48.
8. Бродский Ю. И., Лебедев В. Ю. Инструментальная система для построения имитационных моделей хорошо структурированных организационно-технических комплексов // Вопросы кибернетики. Проблемы математического моделирования и экспертные системы, М.: Научный совет АН СССР по комплексной проблеме «Кибернетика», 1990. С. 49–64.
9. Бродский Ю. И., ЛебедевВ. Ю. Инструментальная система имитации MISS. М.: ВЦ АН СССР, 1991.
10. Kuhl F., Weatherly R., Dahmann J. Creating Computer Simulation Systems: An Introduction to the High Level Architecture NY: Prentice Hall PTR, 1999.
11. Осоргин А. Е. AnyLogic 6. Лабораторный практикум. Самара: ПГК, 2011.
12. Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя. 2-е изд. М.; СПб.: ДМК Пресс, Питер, 2004.
13. Brodsky Y. Simulation Software // System Analysis and Modeling of Intergrated Word Systems. V. 1, Oxford: EOLSS Publishers Co. Ltd., 2009. P. 287–298.
14. Brodsky Y., Tokarev V. Fundamentals of simulation for complex systems. // System Analysis and Modeling of Integrated World Systems. V. 1, Oxford: EOLSS Publishers Co. Ltd., 2009. P. 235–250.
15. Бродский Ю. И., Павловский Ю. Н. Разработка инструментальной системы распределенного имитационного моделирования // Информационные технологии и вычислительные системы. 2009. № 4. С. 9–21.

Статья поступила в редакцию 03.07.2012.