Математическое моделирование

Система модельно-ориентированного программирования

Л.В. Круглов

Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский педагогический государственный университет», г. Москва, Россия

Аннотация. Задача исследования сложных многокомпонентных систем появляется в различных областях деятельности человека, и зачастую единственным возможным способом их анализа является имитационное моделирование. Такие системы могут быть заданы через описание своих составных частей (агентов), их поведения и взаимодействия. В данной работе рассматривается система программирования для алгоритмически полной «модельно-ориентированной» парадигмы, основывающейся на понятии «модели-компоненты» – сложной структуры с заданными характеристиками и поведением, семейство которых замкнуто относительно операции объединения в объемлющие структуры, называемые «моделями-комплексами». Рассматриваемая система программирования позволяет разрабатывать модели при помощи декларативного подхода, требует ограниченного применения других парадигм программирования для реализации деталей поведения агентов и предполагает естественным образом проводить параллельные вычисления.

Ключевые слова: *модельно-ориентированное программирование, модельный синтез, система программирования, сложные системы, имитационное моделирование.*

DOI: 10.14357/20790279230206

Введение

В связи с постоянным ростом сложности систем, применяемых в различных областях деятельности человека [1-3], моделирование играет все более значимую роль в процессе разработки и исследования их поведения, и во многих случаях симуляция является единственным способом проведения такого анализа. Отсюда следует сложность задачи формализации структуры и функционирования системы, а применение императивной объектно-ориентированной парадигмы при описании характеристик системы требует явной организации вычислений через вызовы методов объектов, что вносит дополнительную сложность в процесс разработки моделей и их отладки. Применение декларативного подхода, напротив, позволяет избежать явной организации вычислительного процесса при помощи описания структуры системы в терминах поведения и взаимодействия входящих в нее компонент [4]. Однако декларативная парадигма программирования достаточно плохо подходит для моделирования поведения агентов, являющихся некоторыми алгоритмами или функциональными зависимостями.

Вследствие этого, наиболее перспективными являются комбинированные подходы к модели-

рованию сложных систем, сочетающие декларативные средства формализации их структуры и функционирования и применения императивной или функциональной парадигмы для реализации отдельных элементов, отвечающих за поведение агентов системы. Такие совмещенные методы позволяют эффективно использовать преимущества каждой парадигмы и уменьшать влияние их недостатков.

Такой подход реализуется в рамках модельно-ориентированной парадигмы программирования [4], которая позволяет строить модели и разрабатывать программы из более агрегированных примитивов, называемых моделями-компонентами, и обладающих более высоким уровнем инкапсуляции по сравнению с объектно-ориентированной парадигмой. Синтезируемая из таких компонент система сама является моделью-компонентой, что позволяет применять единообразный процесс вычисления для широкого класса моделируемых систем. В рамках модельно-ориентированного подхода определение структуры системы и взаимодействия ее составных частей осуществляется при помощи некоторого декларативного языка, а для реализации функциональности агентов может применяться любая подходящая в каждом конкретном случае парадигма, что обеспечивает значительную гибкость для разработчиков моделей.

В данной работе показывается, что будучи отдельной парадигмой, модельно-ориентированное программирование является алгоритмически полным. Это означает, что данный подход может быть успешно применен к более широкому классу задач, чем моделирование сложных систем. Также в данной работе доказываются свойства корректности исполнения модельно-ориентированных программ, на основании которых была разработана система программирования, реализующая рассмотренные идеи совмещения различных парадигм программирования для проведения имитационных вычислений моделей сложных агентных систем.

1. Сравнение с другими подходами

Рассмотрим некоторые существующие подходы к задаче моделирования сложных систем.

Одним из них является унифицированный язык моделирования UML [5], который позволяет представлять структуру систем и проходящие в них процессы в графическом виде. В результате его применения получается объектно-ориентированная модель системы, для которой возможна генерация кода входящих в нее классов, отражающих структуру системы, но не взаимодействие ее компонентов, что объясняет ограниченную применимость UML.

Объектно-ориентированный анализ [6] в настоящее время широко применяется в индустрии программного обеспечения и имеет ряд схожих черт с модельно-ориентированным программированием. Так, оба подхода инкапсулируют свое состояние в базовой единице абстракции, которая включает в себя механизмы манипуляции своим состоянием. Однако в модельно-оринетированном программировании отстутвует возможность изменять состояние извне, из чего следует более высокий уровень инкапсуляции модельно-ориентированного подхода по сравнению с объектно-ориентированным, что упрощает восприятие структуры системы. Явная организация вычислений при применении объектно-ориентированного подхода позволяет добиваться большего параллелизма исполняния, чем при использовании модельно-ориентированного программирования, которое, с другой стороны, поддерживает менее гибкий, но автоматический параллелизм на уровне поведения компонент-системы с синхронизацией в главном процессе исполнения.

В рамках разработки создаются концептуальные модели, объединяющие данные и поведение систем и опускающие детали реализации с целью упрощения восприятия моделей [7]. В случае использования специальных языков описания систем (например, языка анализа и разработки архитектуры AADL [8]) также возможна частичная генерация программного кода модели. Однако данный подход в общем случае позволяет получить лишь неполное представление системы, требующее явной програмной реализации, либо строить системы из заранее подготовленного ограниченного набора элементов. Таким образом, данный подход также имеет ограниченную применимость.

Еще одним важным способом решения задачи моделирования сложных агентных систем является агентно-ориентированное программирование [9, 10], в рамках которого система представляется как объединение взаимодействующих при помощи обмена сообщениями агентов, обладающих сложным «психическим» состоянием. Для реализации такого подхода необходима отдельная система программирования. Одна из ее реализаций, основывающаяся на агентно-ориентированном языке общего назначения SARL, описана в работе [11]. Как и в случае модельно-ориентированного подхода, базовая единица абстракции обладает своим состоянием и поведением, описываемыми на декларативном языке, а процесс исполнения

задается универсальной процедурой. Однако психическое состояние и поведение агентов является гораздо более ограниченным (ввиду наличия темпоральных способностей, обязательств и т.д.), чем состояние и поведение компонент модельно-ориентированного программирования. Кроме того, модельно-ориентированный подход позволяет объединять компоненты в более агрегированные сущности, что невозможно в рамках агентно-ориентированного программирования.

Таким образом, можно сделать вывод, что модельно-ориентированное программирование является шагом по увеличению уровня абстракции относительно объектно-ориентированного и агентно-ориентированного подходов. С учетом возможности применения различных парадигм для программирования поведения компонент это позволяет упростить анализ и разработку сложных систем, хоть и в некотрых случаях ведет к уменьшению гибкости по сравнению с другими подходами.

2. Модельно-ориентированное программирование

Сложная агентная система согласно определению, используемому в работе [12], может рассматриваться как «составной объект, части которого также могут быть системами, объединенными естественным образом в соответствии с определенными принципами или набором взаимосвязей». Это означает, что компоненты сложных систем сами могут являться сложными системами и что устройство базовых элементов системы (атомов, примитивов), их поведение и правила взаимодействия явно заданы. Такие компоненты системы, имеющие свое внутреннее состояние, определенное поведение и время от времени взаимодействующие между собой, называются агентами.

Реализуемый в данной работе подход основывается на понятиях «модели-компоненты» и «модельного синтеза», подробно описываемых в работах [4, 12]. Приведем основные принципы их устройства и функционирования, на которых основывается разработанная система модельно-ориентированного программирования в соответствии с работой [13].

Модель-компонента является базовой концепцией рассматриваемого подхода. Она может иметь некоторые свойства и поведение и рассматриваться как элементарный строительный блок моделируемой системы.

Каждая модель-компонента имеет набор внутренних и внешних характеристик. Внутренние характеристики являются ее именованным состоянием, а внешние характеристики – параметрами окружающей среды для модели-компоненты.

В отличии от объектно-ориентированного подхода процедуры манипуляции внутренними характеристиками модели-компоненты, называемые элементами, не могут быть вызваны снаружи самой модели-компоненты. Элементы организуются в независящие друг от друга процессы, состоящие из последовательности элементов и правил переключения текущих элементов. Независимость процессов означает, что множества вычисляемых их текущими элементами внутренних характеристик не пересекаются, откуда следует возможность их параллельного выполнения [14]. Программные реализации элементов называются методами.

Элементы модели-компоненты подразделяются на два класса: быстрые и медленные. Быстрыми элементами моделируются мгновенные изменения состояния модели, и они используются для вычисления дискретных характеристик анализируемой системы. Медленным элементам, напротив, требуется ненулевое модельное время для своего выполнения. Поэтому такие элементы используются для вычисления непрерывных характеристик моделируемой системы.

Переключения текущих элементов процессов осуществляются в результате срабатывания событий – точек синхронизации процессов, в которых требуется реакция системы на изменившиеся значения характеристик. Переключения элементов должны быть однозначными. В таком случае они будут представлять собой детерминированный конечный автомат для каждого процесса, в котором состояниями будут являться элементы модели-компоненты, а переходы между ними – определяться срабатыванием событий. Задача определения наличия нарушения однозначности переключения элементов для произвольной модели-компоненты является алгоритмически неразрешимой, что будет рассмотрено далее в Разделе 3. События также могут вычисляться параллельно [14].

Формально, событие – это функция $\Delta\tau = E(x, a)$ от внутренних характеристик x и внешних характеристик a модели-компоненты, которая возвращает в начале шага симуляции ожидаемое время до его срабатывания. Срабатывание события в момент времени t_i определяется уравнением:

$$E(x_i, a) = 0,$$

где x_i – вектор внутренних характеристик в момент времени t_i.

События управляют переключением элементов процессов модели-компоненты следующим образом: для каждой упорядоченной пары пере-

ключаемых элементов *{f, g}* должно существовать единственное событие $E_{f,g}$, при срабатывании которого будет осуществлено переключение этих элементов. Одновременное срабатывание некоторых двух событий $E_{f,g}$ и $E_{f,h}$ означает ошибку в проектировании модели.

Формально, модель-компоненту можно определить как алгоритм *F*, вычисляющий значения внутренних характеристик на каждом шаге процесса симуляции [15]:

$$x_{i+1} = F(x_i, a, \Delta t),$$

где x_i – вектор внутренних характеристик в момент времени t_i, a – вектор внешних характеристик, а $\Delta t = t_{i+1} - t_i$ – следующий интервал модельного времени. Из данного определения следует существование универсальной процедуры исполнения модели-компоненты.

Наличие такой процедуры означает, что система является замкнутой: предполагается, что разработчик обладает достаточной информацией о системе для построения модели, которая сможет воспроизвести ее динамику (то есть значения ее внутренних характеристик) в любой момент модельного времени некоторого временного отрезка при условии, что известно ее начальное состояние [4].

Поведение модели-компоненты определяется ее характеристиками и множеством имеющих к ним доступ элементов и изменяется в результате срабатывания событий. Универсальная процедура исполнения модели-компоненты реализует ее поведение и однозначно определяет порядок имитационных вычислений (симуляции). Она задается следующим алгоритмом [4, 13]:

1. Вычислить события для текущих методов.
2. Выполнить переключение текущих методов согласно вычисленным событиям.
3. Вычислить следующий шаг модельного времени.
4. Выполнить текущие методы.
5. Перейти к Шагу 1.

Шагом (итерацией) симуляции будем называть последовательность Шагов 1-4 данного алгоритма.

Модель-комплекс. Несколько моделей-компонент могут быть объединены в рамках более высокоуровневой сущности, называемой модель-комплекс, в которой некоторые модели-компоненты могут явно моделировать внешние характеристики некоторых других моделей-компонент. Семейство математических объектов «модель-компонента» является замкнутым относительно операции объединения в рамках «моделей-комплексов» [4]. Это означает, что модель-комплекс сама является моделью-компонентой, что позволяет разработчикам реализовывать многокомпонентные модели систем со сколь угодно большим уровнем вложенности и гарантирует применимость универсальной процедуры исполнения для построенных моделей-комплексов.

Для задания модели-комплекса достаточно указать:

1) включаемые модели-компоненты и их количество;
2) коммутацию внутренних и внешних характеристик моделей-компонент – соответствие некоторых внутренних характеристик одних экземпляров моделей-компонент внешним характеристикам других экземпляров моделей-компонент;
3) отождествление внешних характеристик моделей-компонент – соответствие некоторых внешних характеристик одних экземпляров моделей-компонент внешним характеристикам других экземпляров моделей-компонент.

Важно отметить, что при объединении моделей-компонент в процессе модельного синтеза в модель-комплекс однозначность вычисления внутренних характеристик может быть нарушена, что будет означать наличие ошибки в проектировании модели. Задача определения наличия нарушения однозначности вычисления внутренних характеристик для произвольной модели-компоненты является алгоритмически неразрешимой, что будет показано далее в Разделе 3.

Модельно-ориентированное программирование. На основании рассмотренных выше концепций можно ввести понятие модельно-ориентированного программирования (или модельно-ориентированной парадигмы программирования) как реализации моделей сложных агентных систем при помощи моделей-компонент и моделей-комплексов, исполнение которых определяется универсальной процедурой исполнения модели-компоненты.

Модельно-ориентированный подход позволяет разделить процесс программирования на два основных шага:

1. Декларативное программирование моделей-компонент и моделей-комплексов.
2. Функциональное или императивное программирование реализаций элементов и событий моделей-компонент (или при необходимости применение некоторой другой подходящей парадигмы программирования).

Таким образом, структура и взаимодействие агентов сложных систем программируется при помощи декларативных средств, а детали их поведения реализуются при помощи подходящей в ка-

ждом конкретном случае парадигмы программирования, что позволяет сократить время разработки и уменьшить количество ошибок в разработанных программах [13].

3. Алгоритмическая полнота

Далее рассмотрим утверждение об алгоритмической полноте модельно-ориентированной парадигмы программирования [13]. Доказательство будет проведено при помощи реализации машины Тьюринга с полубесконечной лентой [16] (имеющей такую же вычислительную мощность, что и обычная машина Тьюринга) при помощи модельно-ориентированного подхода.

Теорема (об алгоритмической полноте). Модельно-ориентированная парадигма программирования алгоритмически полна.

Доказательство. Построим машину Тьюринга с полубесконечной лентой на основе понятия модели-компоненты следующим образом:

1. Множество значений $Q = \{q_0, q_1, ..., q_m\}$ внутренней характеристики x_q некоторой модели-компоненты представляет собой множество состояний машины Тьюринга.
2. Значение q_0 характеристики x_q представляет начальное состояние.
3. Множество значений $T \subseteq Q$ характеристики x_q представляет множество конечных состояний.
4. Внутренние характеристики $X = \{x_1, x_2, ...\}$ представляют полубесконечную ленту.
5. Номер внутренней характеристики $c \in \{1, 2, ...\}$ обозначает текущую ячейку полубесконечной ленты, $x_c \in X$.
6. Номер внутренней характеристики $k \in \{1, 2, ...\}$ обозначает начальную ячейку полубесконечной ленты, $x_k \in X$.
7. Множество S допустимых значений внутренних характеристик X представляет символы алфавита ленты.
8. Значение b внутренних характеристик $b \in S$ является пустым символом.
9. Множество $S' \subseteq S \setminus \{b\}$ значений внутренних характеристик X представляет множество допустимых символов начального слова.
10. Множество внутренних характеристик $X' = \{x_i, ..., x_j\} \subset X$ представляет инициализированные начальным словом ячейки ленты, внутренние характеристики $x_b \in X \setminus X'$ имеют пустой символ в качестве начального.
11. Функция перехода представляется быстрым элементом $f_t = f(x_q, x_c, X)$, который изменяет значение текущей ячейки в соответствии с текущим состоянием x_q и текущим значением внутренней характеристики x_c, обновляет значение текущего состояния и увеличивает на единицу, уменьшает на единицу либо оставляет без изменений номер текущей характеристики c.
12. Исполнение элемента f, представляющего функцию перехода, инициируется срабатыванием пустого события $E_{f,f}$ в начале каждого шага симуляции.
13. Исполнение модели-компоненты останавливается, если значение характеристики x_q, представляющей текущее состояние, входит в множество конечных состояний, то есть при $x_q = q_t \in T$.

В результате указанного построения была получена семерка $<S, b, S', Q, T, q_0, F>$, представляющая машину Тьюринга с полубесконечной лентой, реализованную на основе модели-компоненты, и описан порядок ее исполнения и инициализации исходных данных. Теорема доказана.

Алгоритмическая полнота модельно-ориентированной парадигмы программирования показывает, что применимость данного подхода не ограничивается моделированием агентных систем, но что он также позволяет реализовать любую вычислимую функцию [16] (то есть реализуемую машиной Тьюринга).

4. Корректность модельно-ориентированных программ

В данном разделе будет рассмотрен ряд свойств модельно-ориентированной парадигмы программирования, который определяет выразительные возможности системы модельно-ориентированного программирования и ограничивает функциональные возможности этапов компиляции и исполнения программ. Для формализации таких ограничений необходимо ввести понятие модельно-ориентированной программы и ее состояния.

Определение. Под *модельно-ориентированной программой* будем понимать пятерку:

$$M = <I, O, P, E, S>,$$

где I – множество внутренних характеристик, O – множество внешних характеристик, P – множество процессов, E – множество событий, S – детерминированные конечные автоматы, описывающие переключения методов каждого процесса.

Определение. Под состоянием *модельно-ориентированной программы* будем понимать тройку:

$$M = <I_c, P_c, T_c>,$$

где I_c – множество фактических значений внутренних характеристик I, P_c – набор элементов процес-

сов P, каждый из которых является текущим в своем процессе, T_c – текущее модельное время.

Определение. *Состоянием с корректным вычислением событий* будем называть состояние, при выполнении шага симуляции, на котором не происходит одновременного наступления события ни для одного процесса. *Состоянием с корректным вычислением элементов* будем называть состояние, для которого у любой пары текущих элементов множества вычисляемых внутренних характеристик не пересекаются.

Далее рассмотрим два фундаментальных свойства модельно-ориентированных программ, ограничивающих теоретическую возможность определения корректности вычисления событий и элементов.

Теорема (о корректности вычисления событий). Задача определения состояния, в котором нарушается корректность вычисления событий модельно-ориентированной программы, является алгоритмически неразрешимой.

Доказательство. Проведем доказательство от противного. Допустим, что существует такой алгоритм $A(I, O, m_1, m_2)$, который по некоторому представлению двух методов, реализующих события модельно-ориентированной программы и являющихся некоторыми произвольными машинами Тьюринга, определяет, могут ли они на некоторых входных данных завершиться с одинаковым результатом. Пусть данный алгоритм возвращает единицу в случае положительного ответа и ноль – иначе.

Далее рассмотрим некоторую произвольную программу p и некоторые входные данные I_p к ней, представленные в качестве внутренних характеристик, и применим алгоритм A: $A(I_p, Ø, p, p)$. В результате работы алгоритма получим единицу, если программа p завершается на входных данных I_p, и получим ноль, если не завершается. Таким образом, мы пришли к противоречию, так как при наличии алгоритма A становится возможным решить проблему останова [17]. Теорема доказана.

Теорема (о корректности вычисления элементов). Задача определения состояния, в котором нарушается корректность вычисления элементов модельно-ориентированной программы, является алгоритмически неразрешимой.

Доказательство. Аналогично проведем доказательство от противного. Допустим, что существует такой алгоритм $A(I, O, P, E, S)$, который для заданной модельно-ориентированной программы $M<I, O, P, E, S>$ определяет, может ли она при исполнении на некоторых начальных значениях характеристик I и O перейти в состояние, в котором некоторые два элемента $p_1, p_2 \in P$ возвращают значения для пересекающихся множеств внутренних характеристик. Пусть данный алгоритм возвращает единицу в случае положительного ответа и ноль – иначе.

Далее рассмотрим некоторую произвольную программу p и некоторые входные данные I_p' к ней, представленные в качестве внутренних характеристик. Построим модельно-ориентированную программу $M_p<I_p, Ø, P_p, E_p, S_p>$ следующим образом:

1) множество внутренних характеристик I_p соответствует входным параметрам программы p, множество внешних характеристик пусто;
2) множество процессов P_p состоит из двух элементов $\{p_1, p_2\}$: $p_i = \{a_i, b\}$, причем элементы a_i реализуются тривиальной программой, всегда возвращающей значение ноль для характеристики под номером i, а элемент b всегда возвращает ноль для первой характеристики;
3) множество событий E_p состоит из одного элемента e_p, реализуемого программой p', которая запускает программу p и возвращает единственное число на основе значений выходных параметров программы p;
4) множество конечных автоматов процессов S_p состоит из двух одинаковых элементов s: переключение $a \rightarrow b$ по событию e_p.

Применим алгоритм A к построенной программе M_p для некоторых начальных значений внутренних характеристик I_p': $A(M_p)$. По построению оба процесса при переходе по событию e_p (то есть в результате успешного исполнения программы p на входных данных I_p') будут иметь элемент b в качестве нового текущего, возвращающий значение первой характеристики, и алгоритм A решает задачу обнаружения факта такого перехода. Следовательно, в результате работы алгоритма получим единицу, если программа p завершается на входных данных I_p', и получим ноль, если не завершается. Таким образом, мы пришли к противоречию, так как при наличии алгоритма A становится возможным решить проблему останова. Теорема доказана.

С точки зрения разработки системы модельно-ориентированного программирования данные фундаментальные утверждения означают, что по записи модельно-ориентированной программы на этапе компиляции невозможно определить ее корректность, что влечет за собой необходимость как наличия проверок времени исполнения, обеспечивающихся системой программирования, так и дополнительного контроля со стороны разработчиков моделей за процессом их симуляции.

Далее рассмотрим ряд свойств модельно-ориентированных программ, связанных непосред-

ственно с обработкой их формального представления. Так как такое формальное представление выполняется как запись на некотором языке, то проверка данных свойств может быть выполнена на этапе компиляции программы.

Определение. *Транзитивным отождествлением внешних характеристик* компонент модели-комплекса будем называть последовательность отождествлений $c_i = c_j$, $c_j = c_k$, …, $c_p = c_q$, $c_m \in O$.

Определение. *Корректным набором отождествлений внешних характеристик* компонент модели-комплекса будем называть набор, в котором транзитивные отождествления не образуют циклов.

Теорема (о корректности отождествлений в модели-комплексе). Задача проверки корректности отождествления внешних характеристик компонент модели-комплекса алгоритмически разрешима и имеет временную сложность $O(m + n)$, где m – это количество отождествляемых характеристик, а n – количество отождествлений.

Доказательство. Представим отождествления внешних характеристик как направленный граф $G(V, E)$, в котором множество вершин V совпадает со множеством отождествляемых характеристик, а множество ребер E состоит из пар (c_i, c_j) для каждого отождествления $c_i = c_j$. Тогда для определения наличия циклов в транзитивных отождествлениях необходимо и достаточно определить наличие цикла в направленном графе G, например при помощи поиска в глубину [18], откуда следует временная сложность $O(n + m)$. Теорема доказана.

Определение. *Корректным набором коммутаций внешних и внутренних характеристик* будем называть такой набор, в котором с отождествляемыми внешними характеристиками коммутируется одна и та же внутренняя характеристика.

Теорема (о корректности коммутаций в модели-комплексе). Задача проверки корректности коммутации внешних и внутренних характеристик компонент модели-комплекса алгоритмически разрешима и имеет временную сложность $O(m + n + q)$, где m – количество отождествляемых характеристик, n – количество отождествлений, q – количество коммутаций.

Доказательство. Обойдем граф отождествления внешних характеристик G (после установления факта отсутствия в нем циклов) вглубь по обратному направлению ребер для каждой листовой вершины c_{leaf}. При этом для каждого перехода к следующей вершине c_j в соответствующем отождествлении $c_i = c_j$ заменим характеристику c_j на c_{leaf}. Так как во множестве отождествлений допускается только одно вхождение характеристики c_i с левой стороны тождеств вида $c_i = c_j$, то у каждой вершины c_i графа G будет не более одного потомка, и, следовательно, указанная замена внешней характеристики будет выполнена единственный раз. Далее для каждой коммутации $c_i = q_k$ (где q_k – некоторая внутренняя характеристика) при наличии замены внешней характеристики на некоторую другую внешнюю характеристику c_{leaf} или выполнения $c_i = c_{leaf}$ запишем q_k в качестве коммутации для c_{leaf}, причем если для c_{leaf} ранее уже была определена некоторая другая коммутация $q_m \neq q_k$, то фиксируется факт нарушения корректности коммутации, так как в таком случае для внешней характеристики c_{leaf} существует более одной коммутации с различными внутренними характеристиками. Замены внешних характеристик на листовые и далее запись коммутаций листовых внешних характеристик на внутренние может быть организована через массивы длин n и m. Отсюда следует временная сложность рассмотренного подхода $O(n + m + q)$. Теорема доказана.

5. Система программирования

Далее рассмотрим разработанную в рамках данной работы систему модельно-ориентированного программирования, теоретической основой которой являются понятия модели-компоненты и модели-комплексы и рассмотренные свойства корректности. Данная система программирования состоит из следующих шести модулей:

- компилятор;
- исполнитель;
- отладчик;
- база данных экспериментов;
- сервер методов;
- подсистема визуализации.

Структура системы модельно-ориентированного программирования и направления взаимодействия ее компонентов показаны на рис. 1. Основные этапы ее работы можно описать следующими шагами:

1. Исходная программа передается компилятору.
2. Затем целевая компонента из скомпилированной программы передается на запуск модулю исполнения (исполнителю).
3. Исполнитель выполняет итерации универсальной процедуры исполнения модели-компоненты.
4. Для удаленного вызова методов исполнитель взаимодействует с одним или несколькими серверами методов.

5. Команды от отладчика поступают в исполнитель, который их выполняет и возвращает результат обратно отладчику.
6. Состояния программы записываются исполнителем в базу данных экспериментов.
7. При необходимости исполнитель оповещает подсистему визуализации о факте обновления состояния программы.

Рассмотрим подробнее основную функциональность модулей системы модельно-ориентированного программирования.

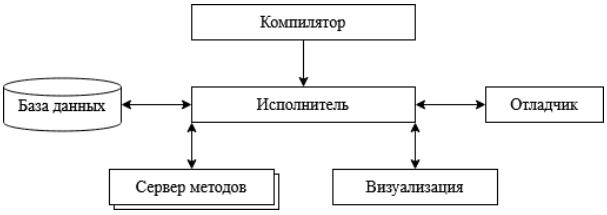

Рис. 1. Структура системы модельно-ориентированного программирования

Компилятор. Исходная программа, содержащая описание моделей-компонент и моделей-комплексов, передается компилятору декларативного «языка компонент и комплексов», основывающегося на работах по модельному синтезу [4, 19]. Компилятор является одним из основных модулей рассматриваемой системы программирования и выполняет следующие задачи:

- синтаксический и семантический анализ кода;
- проверка корректности программы (в том числе корректность отождествлений и коммутаций) и сообщение об ошибках;
- перевод сущностей во внутреннее представление;
- синтез моделей-комплексов;
- генерация представления полученных моделей-компонент на языке компонент и комплексов.

Внутреннее представление используется на этапах проверки корректности кода, синтеза моделей компонент, генерации результирующего кода и далее при симуляции модели.

Исполнитель. Далее полученная программа на языке компонент и комплексов, а также информация о начальных данных передается второму основному модулю системы модельно-ориентированного программирования – исполнителю, который будет выполнять симуляцию указанной модели-компоненты. Главными задачами исполнителя являются:

1) выполнение универсальный процедуры исполнения модели-компоненты;
2) контроль корректности вычисления событий и элементов;
3) запись данных об экспериментах (состояний программы) в базу данных.

В случае обнаружения нарушения свойств корректности пользователю сообщается об ошибке и исполнение останавливается. Для реализации указанной функциональности исполнителю необходимо взаимодействовать со вспомогательными подсистемами: базой данных экспериментов и серверами методов исполнения. При необходимости также осуществляется взаимодействие с модулем визуализации (с целью его уведомления об изменении состояния программы) и отладчиком (для выполнения его команд и возвращения результатов).

База данных экспериментов используется для хранения результатов запусков симуляции моделей. Данная информация сохраняется исполнителем после завершения очередного шага симуляции, что позволяет останавливать и возобновлять симуляцию различных моделей на любом необходимом шаге симуляции.

Сервер методов. Для вычисления элементов и событий используется сервер методов (один или более). Его основными задачами является:

- хранение и выдача информации об их параметрах и способах запуска;
- хранение исполняемого кода методов;
- удаленный запуск методов на исполнение.

Отладчик. Еще одним важным модулем рассматриваемой системы модельно-ориентированного программирования является отладчик, который интерпретирует получаемые команды от пользователя и трансформирует их в запрос для исполнителя. Основная функциональность отладчика включает в себя:

1) пошаговое исполнение итераций симуляции, элементов и событий;
2) исполнение до заданной итерации и заданного модельного времени;
3) чтение и модификация значений внутренних характеристик;
4) чтение значений внешних характеристик.

В качестве дополнительной функциональности можно указать возможность останова исполнения на произвольном корректном состоянии с последующим возобновлением исполнения или выполнением отладочных команд.

Визуализация. Подсистема визуализации не является обязательным компонентом описываемой системы программирования, однако наличие возможности отображения хода вычислений может быть крайне полезна во многих случаях. Особенность функционирования данного модуля заключается в том, что программа

```
TYPE Coordinates {
   double x, y;
}
METHOD Move SLOW {
   ADDRESS "localhost";
   INPUT { Coordinates v, c, t; }
   OUTPUT { Coordinates c; }
}
METHOD Select FAST {
   ADDRESS "localhost";
   INPUT { Coordinates c, t(5); }
   OUTPUT { Coordinates t; }
}
METHOD Reached EVENT {
   ADDRESS "localhost";
   INPUT { Coordinates c, t; }
}

COMPLEX Agents {
   COMPONENTS { Dot(3); }
}
```

```
COMPONENT Agent {
   PARAM { Coordinates targets(5); }
   PHASE { Coordinates v, cur, next; }
   ELEMENTS { p1:(move: Move), (select: Select); }
   EVENTS { (reached: Reached); }
   SWITCHES { move, select: reached; select, move; }
   COMMUTATION {
      INPUT {
         move.v=PHASE.v;
         move.c=PHASE.cur;
         move.t=PHASE.next;
         select.t=PARAM.targets;
         reached.c=PHASE.cur;
         reached.t=PARAM.next;
      }
      OUTPUT {
         move.c=PHASE.cur;
         select.t=PHASE.next;
      }
   }
}
```

Листинг 1. Модель движения агентов на плоскости

визуализации зависит только от разработчика исполняемой модельно-ориентированной программы, то есть ответственность за отображение данных лежит на разработчике модели. Поэтому единственной задачей данной подсистемы является уведомление пользовательской программы визуализации о появлении новых данных и ожидание ответа от нее о возможности продолжения симуляции.

Пример симуляции. Рассмотрим пример простой модели (Листинг 1) на языке компонент и комплексов [19], интерпретируемой описанной системой программирования. Модель описывает движение трех агентов на плоскости между набором целевых точек. Поведение отдельных агентов моделируется экземплярами компоненты *Agent*, которые объединяются в комплекс *Agents*. Для обновления текущих координат путешественника используется медленный метод *Move*, достижение очередной точки определяется событием *Reached*, выбор следующей точки для посещения осуществляется при помощи быстрого метода *Select*. Программные реализации этих методов предполагаются доступными на локальном сервере методов. Для запуска симуляции данной модели необходимо скомпилировать модель-комплекс *Agents* и задать исходные параметры каждого агента и шаг модельного времени, после чего исполнитель начнет выполнять универсальную процедуру исполнения модели-компоненты.

Заключение

В данной работе были рассмотрены основные принципы функционирования системы модельно-ориентированного программирования. Были доказаны фундаментальные свойства модельно-ориентированной парадигмы программирования, которые составляют теоретическую основу разработанной системы программирования и ограничивают функциональные возможности этапов компиляции и исполнения модельно-ориентированных программ. Также были рассмотрены основные этапы функционирования разработанной системы программирования, взаимосвязь ее модулей и их основные задачи.

Дальнейшим направлением исследований будет определение ограничений существующей реализации системы программирования с целью повышения эффективности ее работы и удобства разработки моделей пользователями.

Литература

1. *Turnbull L. et al.* Connectivity and complex systems: learning from a multi-disciplinary perspective //Applied Network Science. 2018. Т. 3. №. 1. P. 1-49.
2. *Butler S., O'Dwyer J.P.* Cooperation and stability for complex systems in resource-limited environments //Theoretical ecology. 2020. Т. 13. №. 2. P. 239-250.
3. *Hu Y., Parhizkar T., Mosleh A.* Guided simulation for dynamic probabilistic risk assessment of complex systems: concept, method, and application //Reliability Engineering & System Safety. 2022. Т. 217. P. 108047.
4. *Бродский Ю.И.* Модельный синтез и модельно-ориентированное программирование //Москва: ВЦ РАН. 2013.

5. *D'souza D.F., Wills A.C.* Objects, components, and frameworks with UML: the catalysis approach. Addison-Wesley Longman Publishing Co., Inc., 1998.
6. *Booch G. et al.* Object-oriented analysis and design with applications //ACM SIGSOFT software engineering notes. 2008. Т. 33. №. 5. P. 29-29.
7. *Selic B.* The pragmatics of model-driven development //IEEE software. 2003. Т. 20. №. 5. P. 19-25.
8. *Feiler P.H., Gluch D.P., Hudak J.J.* The architecture analysis & design language (AADL): An introduction. Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst. 2006.
9. *Shoham Y.* Agent-oriented programming //Artificial intelligence. 1993. Т. 60. №. 1. P. 51-92.
10. *Shoham Y.* An overview of agent-oriented programming //Software agents. 1997. Т. 4. P. 271-290.
11. *Rodriguez S., Gaud N., Galland S.* SARL: a general-purpose agent-oriented programming language //2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT). IEEE, 2014. Т. 3. P. 103-110.
12. *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.
13. *Kruglov L., Brodsky Y.* Model-Oriented Programming //Proceedings of CBU in Natural Sciences and ICT. 2021. Т. 2. P. 63-67.
14. *Бродский Ю.И., Круглов Л.В.* О структурном подходе к концептуальному моделированию широкого класса крупномасштабных систем // Управление развитием крупномасштабных систем (MLSD'2021). 2021. С. 375-387.
15. *Бродский Ю.И., Павловский Ю.Н.* Разработка инструментальной системы распределенного имитационного моделирования //Информационные технологии и вычислительные системы. 2009. №. 4. С. 9-21.
16. *Goldin D.Q. et al.* Turing machines, transition systems, and interaction //Information and computation. 2004. Т. 194. №. 2. P. 101-128.
17. *Burkholder L.* The halting problem //ACM SIGACT News. 1987. Т. 18. №. 3. P. 48-60.
18. *Tarjan R.* Depth-first search and linear graph algorithms //SIAM journal on computing. 1972. Т. 1. №. 2. P. 146-160.
19. *Brodsky Y., Kruglov L.V.* Model-Oriented Programming as a Consequence of the Structural Theory of Multi-Component Complex Systems // International Journal of Education and Information Technologies. 2021. Т. 15. P. 1-12.

Круглов Леонид Вячеславович. Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский педагогический государственный университет», Учебно-научный центр приоритетных исследований и проблем подготовки научно-педагогических кадров, г. Москва, Россия. Младший научный сотрудник, аспирант. Количество печатных работ: 9. Область научных интересов: теоретическая информатика, моделирование, информационные технологии.
E-mail: leonid.kruglov.cmc@gmail.com

Model-oriented programming system

L.V. Kruglov
Moscow Pedagogical State University, Moscow, Russia

Abstract. The problem of analysis of complex multiple component systems arises in various areas of science and engineering. Modeling and simulation is often the only available approach to it. Such systems can be described in terms of their structure, behavior and interaction of its components (agents). In this work, a programming system for Turing complete "model-oriented" paradigm is proposed, which is based on the concept of "model-component" – a complex structure with fixed characteristics and behavior and without external methods. The set of model-components is closed under the union operation into a containing structure called "model-complex". The proposed programming system allows to develop models using declarative approach, requires limited application of other paradigms for agent behavior programming and naturally allows parallel computations.

Keywords: *model-oriented programming, model synthesis, programming system, complex systems, simulation modeling.*
DOI: 10.14357/20790279230206

References

1. *Turnbull Laura et al.* Connectivity and complex systems: learning from a multi-disciplinary perspective. Applied Network Science 3.1 (2018): 1-49.
2. *Butler Stacey and James P. O'Dwyer.* Cooperation and stability for complex systems in resource-limited environments. Theoretical ecology 13.2 (2020): 239-250.
3. *Hu Yunwei, Tarannom Parhizkar and Ali Mosleh.* Guided simulation for dynamic probabilistic risk assessment of complex systems: concept, method, and application. Reliability Engineering & System Safety 217 (2022): 108047.
4. *Brodsky Yu.I.* Model'nyj sintez i model'no-orientirovannoe programmirovanie [Model synthesis and model-oriented programming]. Moscow: CC RAS, 2013.
5. *D'souza, Desmond F. and Alan Cameron Wills.* Objects, components, and frameworks with UML: the catalysis approach. Addison-Wesley Longman Publishing Co., Inc., 1998.
6. *Booch Grady et al.* Object-oriented analysis and design with applications. ACM SIGSOFT software engineering notes 33.5 (2008): 29-29.
7. *Selic Bran.* The pragmatics of model-driven development. IEEE software 20.5 (2003): 19-25.
8. *Feiler Peter H., David P. Gluch and John J. Hudak.* The architecture analysis & design language (AADL): An introduction. Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2006.
9. *Shoham Yoav.* Agent-oriented programming. Artificial intelligence 60.1 (1993): 51-92.
10. *Shoham Yoav*. An overview of agent-oriented programming. Software agents 4 (1997): 271-290.
11. *Rodriguez Sebastian, Nicolas Gaud and Stéphane Galland*. SARL: a general-purpose agent-oriented programming language. 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT). Vol. 3. IEEE, 2014.
12. *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 55.1 (2015): 148-159.
13. *Kruglov Leonid and Yury Brodsky.* Model-Oriented Programming. Proceedings of CBU in Natural Sciences and ICT 2 (2021): 63-67.
14. *Brodsky Yu.I., Kruglov L.V.* O strukturnom podhode k konceptual'nomu modelirovaniyu shirokogo klassa krupnomasshtabnyh sistem [About structural approach to conceptual modeling of a wide class of large-scale systems]. Management of Large-Scale System Development (MLSD'2021). 2021.
15. *Brodsky Yu.I., Pavlovsky Yu.N.* Razrabotka instrumental'noj sistemy raspredelennogo imitacionnogo modelirovaniya [Development of instrumental system of distributed simulation modeling]. Journal of Information Technologies and Computing Systems 4 (2009): 9-21.
16. *Goldin Dina Q. et al.* Turing machines, transition systems, and interaction. Information and computation 194.2 (2004): 101-128.
17. *Burkholder Leslie.* The halting problem. ACM SIGACT News 18.3 (1987): 48-60.
18. *Tarjan Robert.* "Depth-first search and linear graph algorithms." SIAM journal on computing 1.2 (1972): 146-160.
19. *Brodsky Yu.I., Kruglov L.V.* Model-Oriented Programming as a Consequence of the Structural Theory of Multi-Component Complex Systems. International Journal of Education and Information Technologies 15 (2021): 1-12.

Kruglov L.V. Moscow Pedagogical State University, Scientific Center of High-Priority Studies and Academic and Teaching Personnel Training, Moscow, Russia. E-mail: leonid.kruglov.cmc@gmail.com.