Ю. Б. Колесов
Ю. Б. Сениченков

МОДЕЛИРОВАНИЕ СИСТЕМ

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД

Рекомендовано Учебно-методическим объединением
по университетскому политехническому образованию в качестве
учебного пособия для студентов высших учебных заведений, обучающихся
по направлению подготовки 220100 – «Системный анализ и управление»

Санкт-Петербург
«БХВ-Петербург»
2012

УДК 681. 3.06(075.8)
ББК 32. 973.26-018.2я73
К60

Колесов, Ю. Б.

К60 Моделирование систем. Объектно-ориентированный подход. Учебное пособие / Ю. Б. Колесов, Ю. Б. Сениченков. — СПб.: БХВ-Петербург, 2012. — 192 с.: ил.

ISBN 5-94157-579-3

Учебное пособие содержит: краткое изложение языка UML — той его части, которая может быть использована как основа языка моделирования сложных динамических систем; описание и возможности предлагаемого авторами нового языка моделирования на базе гибридных автоматов, являющегося расширением UML; исторический обзор и примеры различных подходов к конструированию инструментов моделирования; объектно-ориентированный анализ сложных динамических систем. Книга является второй из трех книг, объединенных общим названием МОДЕЛИРОВАНИЕ СИСТЕМ.

Для студентов вычислительных специальностей технических вузов

УДК 681.3.06(075.8)
ББК 32.973.26-018.2я73

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Людмила Еремеевская*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Рецензенты:

Евгенев Г. Б., д. т. н., профессор кафедры «Компьютерные системы автоматизации производств» МГТУ им. Баумана

Ивановский Р. И., д. т. н., профессор кафедры «Распределенные вычисления и компьютерные сети» СПбГПУ

Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 15,48.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-579-3

© Колесов Ю. Б., Сениченков Ю. Б., 2006
© Оформление, издательство "БХВ-Петербург", 2006

Оглавление

Предисловие **7**

Глава 1. Объектно-ориентированный подход к моделированию **9**

Необходимость в унифицированном языке описания моделей 10
Классы, экземпляры и многокомпонентные системы 13
Использование UML на начальной стадии проектирования 17
Диаграммы классов 17
- Атрибуты 18
- Поведение 18
- Операции и методы 20
- Абстрактные и конкретные классы. Интерфейсы 20

Классы и отношения 23
- Ассоциация 23
- Обобщение 24
- Агрегация 25
- Наследование 26
- Полиморфизм 31

Поведение. Диаграммы состояний 31
Структурированные классификаторы 34
Компоненты 35
События и сигналы 36
Пакеты 39
Модель 40

Глава 2. Объектно-ориентированное моделирование сложных динамических систем на основе формализма гибридного автомата **43**

Активный класс и активный динамический объект 43
Пакеты и модель 51
Использование пассивных объектов 53

Переменные ... 54
Типы данных ... 55
Скалярные типы ... 56
Вещественный тип ... 56
Целые типы ... 56
Булев тип ... 57
Перечислимые типы ... 57
Символьные типы ... 57
Регулярные типы ... 58
Векторы ... 58
Матрицы ... 58
Массивы ... 59
Списки ... 59
Комбинированный тип (запись) ... 60
Явно определяемые типы ... 60
Сигналы ... 61
Автоматическое приведение типов ... 62
Система уравнений ... 62
Карта поведения ... 67
Состояния ... 67
Переходы ... 72
Структурная схема ... 80
Объекты ... 81
Связи ... 82
Регулярные структуры ... 85
Наследование классов ... 88
Добавление новых элементов описания ... 89
Переопределение унаследованных элементов ... 90
Полиморфизм ... 91
Параметризованные классы ... 91

Глава 3. Моделирование гибридных систем и объектно-ориентированный подход в различных пакетах ... 93

Моделирование гибридных систем в инструментальных средствах для "больших" ЭВМ ... 94
Язык SLAM II ... 94
Язык НЕДИС ... 96
Гибридные модели в современных инструментах моделирования ... 98
Моделирование гибридных систем в пакете Simulink ("блочное моделирование") ... 98
Моделирование гибридных систем на языке Modelica ("физическое моделирование") ... 122
Гибридное направление ... 127

Языки объектно-ориентированного моделирования 142
Simula-67 и НЕДИС 142
ObjectMath 143
Omola 143
Modelica 145
Инструменты "блочного моделирования" 147
Анализ существующих языков ООМ применительно к моделированию сложных динамических систем 147

Глава 4. Многообъектные модели 151

Глава 5. Объектно-ориентированное моделирование и объектно-ориентированный анализ 163

Сложная техническая система 163
Объектно-ориентированный анализ при разработке сложных технических систем 165
Объектно-ориентированное моделирование на последующих этапах разработки и сопровождения сложной технической системы 172
Системно-аналитическая модель как основа "сквозной" технологии проектирования 175

Литература 179

Дополнительная литература к главе 1 179
Дополнительная литература к главе 2 179
Дополнительная литература к главе 3 180
Дополнительная литература к главе 4 180
Дополнительная литература к главе 5 180

Предметный указатель 183

Предисловие

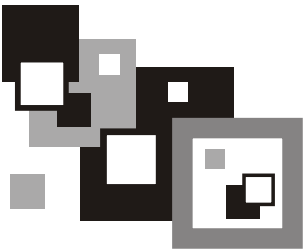
Вы держите в руках *том II* книги "Моделирование систем" — объектно-ориентированный подход. Объектно-ориентированное моделирование стало в последнее время основным инструментом проектирования сложных динамических систем.

Глава 1 книги посвящена рассмотрению UML, точнее того его подмножества, что непосредственно может быть использовано на ранней стадии проектирования сложных динамических систем. Язык UML де-факто стал стандартом моделирования сложных программных комплексов, и в частности может быть использован для построения моделей многокомпонентных иерархических систем.

Глава 2 представляет собой введение в объектно-ориентированный язык моделирования, построенный на гибридных автоматах — графическом представлении гибридных систем. Этот язык лежит в основе программного комплекса MvStudium (**http://www.mvstudium.com**).

Глава 3 содержит краткое описание существующих объектно-ориентированных языков моделирования и реализующих их программных комплексов.

В *главах 4* и *5* рассматриваются особенности проектирования многокомпонентных систем и обсуждаются возможности объектно-ориентированного анализа сложных систем.



ГЛАВА 1

Объектно-ориентированный подход к моделированию

"...сложность является неотъемлемой чертой универсального приложения, предназначенного для решения повседневных задач в реальном мире..."

Г. Буч, А. Якобсон, Дж. Рамбо [2]

Объектно-ориентированный подход к моделированию сложных динамических систем был предложен авторами языка SIMULA-67 [6]. Именно они стали использовать специальные модели — классы, описывающие сразу все множество близких по своим свойствам объектов, обладающих внутренней структурой и поведением. Имея такое обобщенное описание, можно уже выбирать конкретный элемент этого множества, необходимый для моделирования конкретного устройства, создавая экземпляр класса и наделяя его конкретными значениями параметров. Так возник новый подход к моделированию сложных многокомпонентных систем, названный *объектно-ориентированным подходом* (ООП), который, как оказалось, может быть применен и для проектирования программных комплексов. Он активно совершенствовался и применялся, прежде всего, при проектировании сложных программных комплексов и дискретных систем. Разрабатываемая программная система сначала моделируется с помощью специального объектно-ориентированного языка моделирования, а только затем превращается в реальный программный комплекс. Наиболее известным является *унифицированный язык моделирования* (Unified Modeling Language или сокращенно UML). Модификации объектно-ориентированной технологии применительно к моделированию сложных многокомпонентных динамических систем возникли сравнительно недавно и называются *объектно-ориентированным моделированием* (ООМ). Они представлены языком моделирования Modelica, реализованы в явном виде в пакетах AnyLogic, MvStudium, Ptolemy II, неявно — в Simulink, Stateflow и других компонентах пакета Matlab.

Необходимость в унифицированном языке описания моделей

Традиционная технология проектирования сложных систем, не предполагающая предварительного компьютерного моделирования, включает в себя следующие основные этапы: формулировку требований к будущей системе, разработку на их основе проектной документации, создание опытного образца, его испытание на соответствие требованиям и сопровождение промышленного образца.

В идеальном случае полная функциональная спецификация, разработанная системными аналитиками или автоматически с помощью компьютерных технологий, уже является проектным решением, остается малое — построить физическую модель по этой спецификации, которая и будет проектируемой системой. К сожалению, это не так. Создание реальной системы, соответствующей разработанным функциональным спецификациям, требует творческих, не формализуемых инженерных решений и ручного труда.

Даже, казалось бы, полностью формализованная "мягкая часть" современных технических систем — встроенное программное обеспечение — чаще всего выполняется на специальных встроенных ЭВМ или микропроцессорах с особыми каналами обмена, специальными операционными системами и другими особенностями, предполагающими "ручную" доводку разработанного программного обеспечения. Без этого "ручного", плохо формализуемого, этапа трудно обеспечить специальные требования надежной работы системы при заданных температурах, вибрациях, перегрузках, уровнях излучения, ограничениях на объем и вес. В *главе 5* обсуждаются условия, при которых возможна автоматическая генерация встроенного программного обеспечения по функциональной спецификации, и эти условия оказываются еще очень далекими от сегодняшних реалий.

В процессе разработки сложной технической системы наряду с моделями высокого уровня используются так же и модели низкого уровня абстракции, трудно воспринимаемые человеком, трудно формализуемые и требующие ручного труда, при переходе от модели к реальной системе.

На этапе стендовой отработки макета системы часто используется так называемое полунатурное моделирование, когда отдельные блоки системы представлены либо реальными образцами, либо их имитаторами. Теоретически в качестве имитаторов нужно использовать соответствующие функциональные спецификации, однако это не всегда удается, и приходится разрабатывать для них специальные модели.

Наконец, на этапе сопровождения после внесения изменений моделирование должно полностью повторяться, начиная с того уровня, на котором были внесены изменения. На практике, конечно, это выполняется далеко не всегда.

Благодаря возможностям современной вычислительной техники переходить к изготовлению опытного образца стали только после создания и испытания на соответствие предъявляемым требованиям компьютерной модели. Такую технологию можно назвать проектированием на основе моделирования, и она позволяет использовать преимущества компьютерного моделирования уже на самых ранних этапах разработки. Вычислительные эксперименты с моделью еще не существующей системы позволяют проводить более тщательный анализ предлагаемых решений, чем при традиционном проектировании, когда большинство ошибок и несоответствий требованиям обнаруживается на этапе экспериментов с опытным образцом. Описание модели на языке моделирования — это строгое описание ее функциональных свойств. Модель становится единой формальной функциональной спецификацией проектируемой системы, частью проектной документации, доступной всем разработчикам. В цепочке технологических операций появляется новый этап — построение и исследование формальных спецификаций. Этап построения и исследования функциональных спецификаций на основе объектно-ориентированной технологии называется *объектно-ориентированным анализом* и рассматривается в *главе 5*. Сегодня предпочтение отдают проектированию на основе моделирования и средам моделирования, автоматизирующим как можно больше этапов разработки.

Программных сред, поддерживающих технологию проектирования, основанную на моделировании, и способных по заданному описанию системы строить модель, существует много. Отсутствие в них возможностей поддерживать тот или иной этап разработки принимается как данное. Разработчики используют конкретную среду моделирования и дополнительно устанавливают свои правила совместной работы отдельных коллективов. В результате получается своеобразная, во многом неформальная, надстройка над средой моделирования.

В то же время почти для всех существующих сред моделирования характерны следующие недостатки:

- изолированность, не позволяющая использовать компоненты и библиотеки различных сред в одном проекте, приводящая к многократному дублированию одних и тех же моделей на различных языках моделирования. И дело здесь не столько в конкуренции и борьбе за потребителя, в результате чего нужные компоненты оказываются недоступными, сколько в трудности правильно воспроизвести поведение "чужого" компонента в

"своей" среде. Проблема "обмена" моделями между средами до сих пор еще не нашла удовлетворительного решения;

- отсутствие полного набора средств исследования модели и, как следствие, необходимость строить, исследовать и визуализировать поведение отдельных компонентов или всей модели в других средах, где нужные инструменты анализа существуют, или дублировать их.

В этих условиях ценной является любая возможность унификации, поэтому представляется, что в качестве основы для современного объектно-ориентированного языка моделирования сложных динамических систем должен использоваться унифицированный язык моделирования. Этим языком может быть UML — объектно-ориентированный язык прототипирования сложных программных комплексов и вычислительных систем [2]. Язык UML заслуживает внимания по нескольким причинам.

Во-первых, авторы UML являются ведущими специалистами и язык зафиксировал все основные достижения объектно-ориентированного подхода к проектированию программных систем, став фактическим стандартом. Поэтому создателям специальных языков ООМ имеет смысл использовать UML как своего рода метаязык.

Во-вторых, часть конструкций UML может быть использована непосредственно для описания структуры проекта, а часть — в качестве основы для расширения.

По мере введения различных понятий и конструкций UML, будет показано, как можно использовать их непосредственно в языках моделирования сложных динамических систем.

В то же время авторы UML считают, что "UML — язык дискретного моделирования, и он не предназначен для разработки непрерывных систем, встречающихся в физике и механике". Это справедливо, если под непрерывной физической системой понимать традиционную математическую модель в виде изолированной системы, описываемой обыкновенными дифференциальными уравнениями или уравнениями в частных производных, не меняющей своей формы во времени. В случае же многокомпонентных систем, меняющихся под воздействием дискретных событий, многие конструкции языка UML могут быть использованы для описания моделей таких систем.

Классы, экземпляры и многокомпонентные системы

> "При работе с задачами и системами, состоящими из большого количества элементов, возможность разложения на составные части приобретает первостепенное значение. Внимание человека должно сосредотачиваться; чтобы мыслить точно и связно, необходимо по возможности сокращать количество одновременно охватываемых понятий. Разлагая большую задачу на части, можно получить задачи-компоненты с таким небольшим количеством деталей, чтобы каждую из этих задач можно было обдумывать и анализировать в целом. Разложение на составные части абсолютно необходимо, если в анализе и программировании задачи принимает участие несколько человек".
>
> *У.-И. Дал, Б. Мюрхауг, К. Июгорд [6]*

Понятие *класса* как множества сущностей (объектов), имеющих одинаковое функциональное назначение, структуру и поведение и отличающихся значением параметров, является ключевым для проектирования многокомпонентных систем. С понятием класса неразрывно связано понятие *экземпляра класса*, т. е. конкретного объекта из множества всех объектов того же самого класса с уникальными значениями параметров.

Разбиение реальной или проектируемой системы на компоненты, построение классов, соответствующих этим компонентам, установление связей между компонентами, аналогичных связям между компонентами моделируемой реальной или проектируемой системы, и построение модели из экземпляров классов с учетом существующих связей — это ключевые моменты технологии объектно-ориентированного моделирования. При объектно-ориентированном подходе структура модели соответствует структуре реального объекта.

Использование *экземпляров класса* позволяет собирать новую модель из типовых компонентов, подстраивая их каждый раз под конкретные условия функционирования, а также использовать построенную модель в качестве компонента других моделей (рис. 1.1). Так создаются модели и новые библиотечные компоненты в пакете Simulink. Поэтому любой инструмент моделирования, позволяющий строить модели из стандартных компонентов, явно или неявно использует технологию ООМ. В то же время инструменты, использующие все возможности ООМ, позволяют пользователю не только самому определять новые элементарные компоненты (создавать новые классы, не являющиеся механической композицией уже существующих), но и применять наследование и полиморфизм для модификации уже разработанных классов.

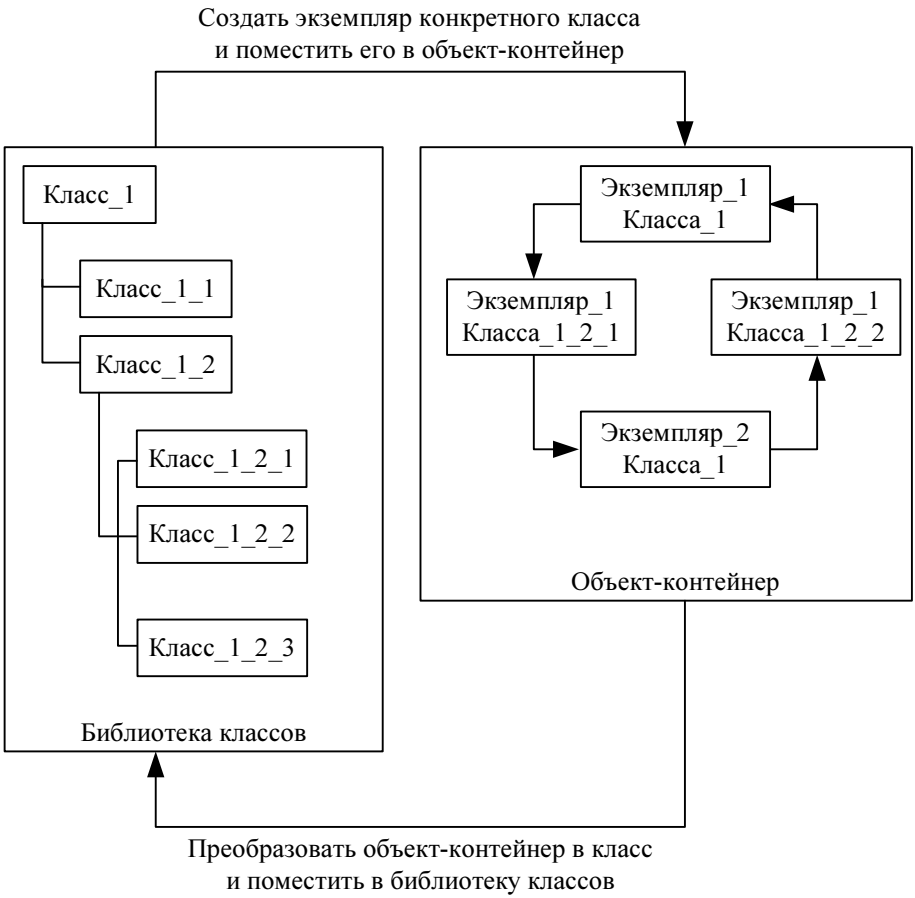


Рис. 1.1. Создание нового компонента с помощью готовых компонентов

В UML выделяются "активные" и "пассивные" объекты и соответственно "активные" и "пассивные" классы, экземплярами которых они являются. *Активным объектом*, согласно UML, называется "объект, который владеет потоком управления и может инициировать управляющие воздействия" [2]. *Пассивным объектом* называется "объект, у которого нет собственного потока управления. Все его операции выполняются под управлением потока, прикрепленного к активному объекту" [2]. Применительно к дискретным системам эти определения вполне достаточны, однако применительно к ООМ сложных динамических систем они представляются несколько узкими. Поэтому заменим "поток управления" на "процесс, развивающийся во времени" и "управляющие воздействия" — на "воздействия на другие объекты". В *гла-*

ве 2 для компонентов сложных динамических систем (СДС) вводится специальное понятие "активного динамического объекта".

Компоненты модели СДС могут быть как активными, так и пассивными объектами. Пассивным объектом удобно считать совокупность данных и чисто алгоритмических операций (например, объект "список" с набором операций добавления, удаления и модификации).

Объектно-ориентированный подход позволяет скрывать детали реализации компонента и делать "видимыми" только те переменные и операции, которые нужны другим компонентам (инкапсуляция данных). В активных объектах данные и поведение — законы, определяющие изменение переменных состояния во времени, скрыты от наблюдателя, видны обычно только отдельные переменные состояния, значения которых изменяются во времени, и операции, управляющие поведением. В ОПП мы последовательно вызываем операции (методы) других программных объектов, для того чтобы изменить данные этих объектов или получить нужный результат, необходимый нам как промежуточный в процессе "своих" вычислений, а в ООМ мы преимущественно используем результаты параллельных действий, проводимых другими объектами, или управляем их поведением с помощью сигналов.

Объектно-ориентированный подход позволяет использовать ранее созданные классы для проектирования новых классов, дополняя и видоизменяя отдельные свойства уже разработанных классов. В результате возникает иерархия классов, в которой новые классы автоматически включают в себя данные и операции своих предшественников и отражают в своем описании в явном виде только новую информацию. Отношение между классами в такой иерархии называется *обобщением*. Новые классы обобщают свойства уже существующих классов, ранее созданных. Механизм, с помощью которого создаются новые классы, включающие в себя свойства предшественников, называется *наследованием*.

Под обобщением также понимают и правила использования экземпляров одних классов (созданных позже) вместо экземпляров других классов (созданных ранее), если новые классы наследуют свойства старых, дополняя их. Действительно, в этом случае все новые классы включают без изменений все свойства своих предшественников и умеют делать все то, что умели делать их предки. Таким образом, экземпляры новых классов всегда можно использовать вместо экземпляров их предшественников.

Обобщение может быть связано и с переопределением свойств предшественников. В случае переопределения операций классов появляется возможность получить так называемые *полиморфные операции*. Упрощенно говоря, каждый новый класс в иерархии видоизменяет некоторое общее для всех классов

действие (как минимум, общим для этих действий является имя), реализуя его по-своему. И вертолет, и самолет с вертикальной посадкой умеют летать и садиться в произвольном месте, но реализуют это разными способами. И тот, и другой можно определить как потомок класса "летательный аппарат" с операциями "взлет", "полет", "посадка". И если иерархия выстроена так:

"летательный аппарат" ← вертолет ← самолет с вертикальной посадкой,

то конкретный самолет всегда можно использовать вместо конкретного вертолета. С помощью полиморфных операций можно передать специфические черты общих для родственных классов операций, приписав каждому классу в иерархии собственную модификацию общей для всех операции. При этом мы можем не уточнять, что "взлет", "полет", "посадка" осуществляется вертолетом или самолетом — это и так становится ясным из контекста, где указывается вид используемого летательного аппарата.

Экземпляры классов можно создавать в начале вычислительного эксперимента и использовать их на протяжении всего процесса моделирования, а можно создавать, когда в них возникает потребность, и уничтожать, как только они становятся ненужными. Возможность динамически создавать и уничтожать экземпляры классов позволяет создавать *модели с переменным числом компонентов*.

Модели с переменным во времени числом компонентов возникают во многих приложениях, и в частности широко используются при моделировании систем массового обслуживания.

ООМ целесообразно использовать, когда:

- среди компонентов модели много однотипных и характерных для данной прикладной области, отличающихся только значениями параметров;
- имеется возможность так разбить моделируемый объект на компоненты, что классы, им соответствующие, образуют библиотеки, имеющие древовидную структуру;
- система имеет переменную структуру, в которой число компонентов неизвестно заранее, и они могут появляться и исчезать в зависимости от наступления различных событий;
- библиотеки классов, создаваемые для одного проекта, могут быть многократно использованы непосредственно или после модификации в других проектах.

Использование UML на начальной стадии проектирования

> "UML должен быть достаточно выразительным, чтобы работать со всеми концепциями, возникающими в современной системе, такими, например, как параллелизм и распределенность, а также с такими механизмами создания программного обеспечения, как инкапсуляция и компоненты".
>
> "UML запутан, неточен и сложен".
>
> *Г. Буч, А. Якобсон, Дж. Рамбо [2]*

UML предоставляет пользователю большой набор средств графического описания структуры и поведения сложных систем. Выбор конкретного способа представления модели с помощью графических элементов языка определяется пользователем.

Предполагается, что модель проектируемой системы создается сначала в самом общем виде и постепенно уточняется. На первом этапе обсуждаются список возможных вариантов использования будущей системы, статическая структура модели (классы и их отношения), распределяются виды работ между участниками разработки. На втором этапе детализируется структура классов, уточняется, из каких компонентов они состоят и как эти компоненты связаны между собой, конкретизируется динамическое поведение системы. На третьем этапе выбирается конкретная конфигурация вычислительной системы, на которой будет реализована модель, и выделяются ресурсы отдельным компонентам. Нас в дальнейшем будут интересовать только два первых этапа.

Для описания моделей сложных динамических объектов, в отличие от моделей сложных программных комплексов, достаточно небольшого подмножества из всего арсенала средств, предлагаемых UML, и уточнения некоторых понятий, касающихся поведенческих аспектов моделируемой системы.

Диаграммы классов

Класс — это языковая конструкция, служащая для описания множества объектов, обладающих общими свойствами. Такие конструкции языка создатели UML называют классификаторами, т. к. они наделяют создаваемый элемент модели свойствами, по которым его можно отличить от других элементов.

С классом в UML связаны понятия *атрибута, операции, метода и поведения* — общих свойств, присущих моделируемому объекту. Связи между классами и свойства классов отображаются на диаграммах классов.

Атрибуты

Под *атрибутом* понимается описание поименованного элемента заданного типа, для которого указана область значений. Какие именно типы можно использовать, определяется конкретным языком моделирования. При создании экземпляра класса его значение из указанного диапазона должно быть конкретизировано. Исключение составляют атрибуты, общие для всего класса в целом, их значения остаются постоянными для любого экземпляра класса и присваиваются им при инициализации самого класса (иногда их называют статическими или атрибутами класса). Описание атрибутов подчиняется определенным синтаксическим правилам и размещается в секции описания атрибутов. В частности, при описании статических атрибутов соответствующая строка в секции атрибутов подчеркивается.

Обязательным элементом описания атрибута является только его имя. Имя должно быть уникальным внутри класса и может иметь различные области видимости: + (публичная), # (защищенная), - (приватная), ~ (пакетная) видимость — приведенные символы используются как элементы языка описания атрибутов. В качестве значения видимости по умолчанию выбирается приватная видимость, делающая переменные класса невидимыми извне. В общем случае имя атрибута представляется составным именем, указывающим класс, в котором он объявляется, или "путь" к этому атрибуту в общем "пространстве" имен модели, что учитывает вложенность описаний. Любой атрибут может иметь конкретное предопределенное значение, которое он получит в момент создания экземпляра класса.

Поведение

Поведение отражает динамические свойства классов и подразделяется на *взаимодействие*, *смену состояний* и *деятельность*. И соответственно, поведение может быть представлено тремя способами: в виде диаграмм последовательности и коммуникации, диаграмм состояний и диаграмм деятельности.

Взаимодействие, представляемое диаграммами последовательности и коммуникации, — это обмен сообщений между компонентами модели. Диаграммы последовательности и коммуникации описывают участников взаимодействия (источники передачи и получателей сообщений), последовательности обмена сообщениями между ними, линии жизни участников (экземпляры классов могут создаваться и уничтожаться в любой момент модельного времени), а также накладывают временные ограничения на время передачи и приема сообщений.

Назначение диаграмм последовательности и коммуникации — указать участников и возможные сценарии обмена сообщениями между ними. Достаточно

очевидно, что такие диаграммы нужны разработчику на самом раннем этапе проектирования функциональной схемы, а в дальнейшем могут служить только для проверки правильности разработанных алгоритмов посылки и обработки полученных сообщений.

Смена состояний является важным признаком проявления динамики. Под состоянием в UML понимаются "условия или ситуация в жизненном цикле объекта, во время которой он удовлетворяет некоторому условию, выполняет собственную деятельность или ожидает какого-либо события [2]".

Напомним, что при моделировании сложных динамических систем лучше употреблять термин "гиперсостояния", т. к. речь может идти об объединении множества состояний, удовлетворяющих некоторому условию.

О гиперсостояниях идет речь в картах состояний Харела. В гибридных системах гиперсостояния системы ассоциируются с длительными процессами, происходящими в моделируемых объектах. Например, мы говорим, что система отопления находится в нормальном состоянии (явное гиперсостояние, т. к. речь идет о длительной работе системы управления), если в комнате поддерживается заданная температура. Если система отопления не работает и ожидает сигнала, чтобы начать выполнение предписанных заданий: начать или прекратить автоматическую поддержку температуры, перейти к аварийному режиму поддержания температуры — это тоже состояние системы, состояние ожидания. Наконец, система отопления может выполнять предписанные длительные действия (проверять оборудование перед началом отопления), только после завершения которых она может перейти в состояние ожидания. В гибридных системах гиперсостояниям обычно предписываются уравнения, параметры и форма которых неизменны.

Деятельность в UML трактуется как управление всей системой. Управление в динамической системе может осуществляться непрерывно (например, управление непрерывной системой по положению и скорости с помощью ПИД-регулятора) и дискретно (с помощью встроенных цифровых вычислителей в контуре управления непрерывным устройством, предполагающее квантование непрерывного входного и выходного управляющих сигналов). С точки зрения современных сложных динамических систем, это достаточно простые виды управления. На управление можно смотреть и шире. В этом смысле деятельность связана с алгоритмами управления, или с анализирующими и поддерживающими характеристики системы в заданных границах, или с оптимизирующими ее поведение в соответствии с заданными критериями, или приводящими систему к выбранной цели. В последнем случае управление выражается в предписании, какие действия необходимо выполнить для достижения заданной цели. Как описать процессы управления, являются ли эти процессы последовательными или параллельными, в какой по-

следовательности их следует выполнять. Именно это и указывается на диаграммах деятельности.

Непосредственно диаграммы деятельности не нашли применения в современных языках моделирования — исключение составляют языки моделирования, использующие сети Петри.

И деятельность, как разновидность глобального поведения всей системы, и смена состояний могут быть представлены одним и тем же графическим способом — диаграммой состояний. Такая диаграмма по существу является "дискретной", т. е. классической картой состояния, отражающей, что и в какой последовательности следует выполнять для реализации управления, с указанием того, какие процессы можно выполнять параллельно. Язык UML допускает использование диаграмм состояний для описания деятельности, и мы воспользуемся этим, чтобы не описывать подробно собственно диаграмму деятельности.

Операции и методы

Помимо секции атрибутов, описание любого класса содержит секцию операций. Авторы UML отождествляют операцию с любой "вычислительной услугой", которую один экземпляр класса предоставляет другим экземплярам, или использует сам в процессе работы. Эта услуга, так же как атрибуты класса, может иметь различную видимость. Имя операции является единственным синтаксическим элементом описания операции, хотя в описании может использоваться и список входных и выходных параметров, с указанием их типов.

Описание операций класса помещается в секцию операций класса. Конкретная реализация операции называется *методом*. При описании операций им можно и не сопоставлять метод, тогда они называются *абстрактными*. Имена абстрактных операций выделяются курсивом.

Обычно операции осуществляются над конкретными данными, приписанными экземпляру, однако операции могут использоваться для преобразования данных, приписанных всему классу. К операциям, свойственным классу в целом, можно отнести операции над статическими данными класса, выполняемыми в момент инициализации класса.

Абстрактные и конкретные классы. Интерфейсы

Класс изображается прямоугольником, имеющим три секции — секцию имени, секцию атрибутов и секцию операций (рис. 1.2, *а*, *б*). Обязательной среди этих секций является секция имени.

С помощью классов могут порождаться новые сущности — экземпляры классов, в которых конкретизируются значения их атрибутов (прямые экземпляры). Такие классы называются *конкретными* (рис. 1.2, *б*). Классы бывают и *абстрактными* (рис. 1.2, *а*). Их используют для описания общей структуры всех его потомков — конкретных классов. Абстрактные классы не могут иметь прямых экземпляров. Прямые экземпляры потомков абстрактного класса называются *косвенными экземплярами абстрактного класса.*

На раннем этапе проектирования иногда достаточно знать, какие операции класса будут видны извне (рис. 1.2, *в*), иными словами, какими операциями данного класса могут воспользоваться другие классы. Для описания таких операций в UML предусмотрены интерфейсы. *Интерфейс* — это описатель видимых снаружи операций класса, или по-другому описатель услуг, предоставляемых классом. Один и тот же класс может иметь несколько интерфейсов, описывающих различные услуги. Интерфейсы бывают обеспечиваемыми и требуемыми. *Обеспечиваемым* называется интерфейс, который предоставляет класс; *требуемый интерфейс* — это тот набор операций, который необходим классу от других классов для выполнения своих задач. Интерфейс можно рассматривать как абстрактный класс, содержащий только абстрактные операции. Интерфейс, как и класс, является классификатором. Отношение обобщения применимо к интерфейсу, таким образом строя интерфейсы, можно применять механизм наследования.

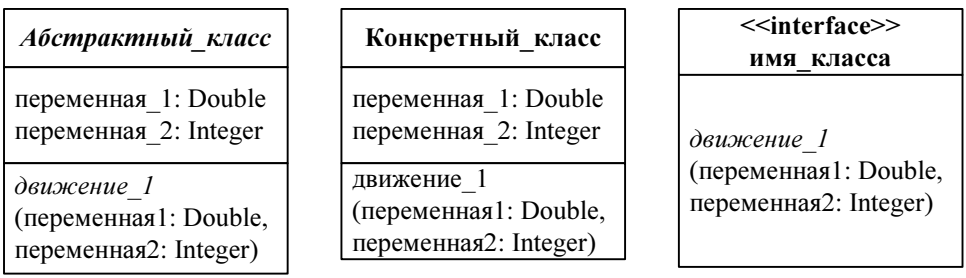


Рис. 1.2. Языковые конструкции: *а* — абстрактный класс; *б* — конкретный класс; *в* — интерфейс

Как уже отмечалось, классы могут быть как активными, так и пассивными. *Активные классы* изображаются особым образом (рис. 1.3). В этой главе не столь существенно, является ли класс пассивным или активным, поэтому мы не будем усложнять рисунки, подчеркивая графически активность или пассивность рассматриваемого класса.

На рис. 1.4 показаны последовательные этапы разработки: интерфейса, абстрактного и конкретных классов для построения некоторой механической

Рис. 1.3. Изображение активного класса

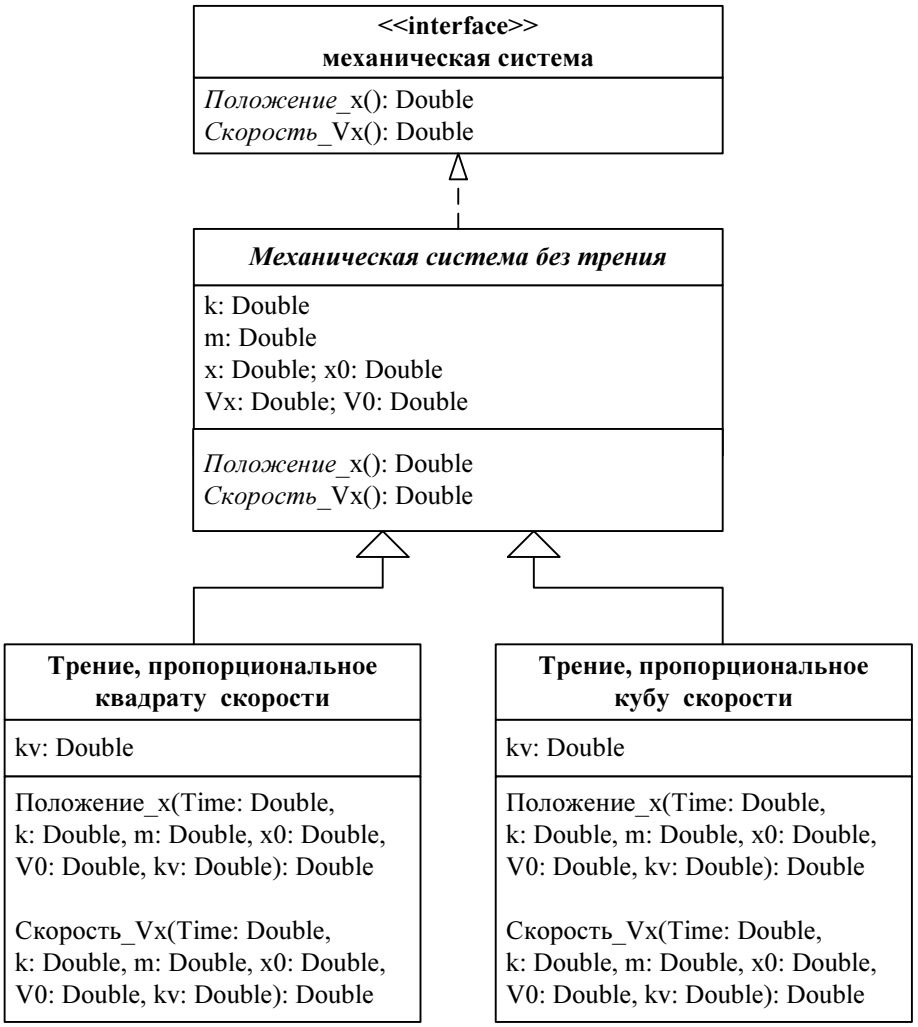


Рис. 1.4. Построение модели с помощью интерфейса и различного вида классов

модели, у которой законы движения различным образом учитывают трение. Сам интерфейс указывает только на то, что конкретный класс может предоставить законы, по которым вычисляются положение и скорость объекта. У класса "механическая система" имеются две абстрактные операции. Классы "трение, пропорциональное квадрату скорости" и "трение, пропорциональное кубу скорости" наследуют свойства класса "механическая система без трения" и по-разному реализуют требуемые операции. Таким образом, оба класса предоставляют требуемый интерфейс, например, для класса, который в дальнейшем будет строить графики зависимости положения и скорости от времени.

На рис. 1.4 использована стрелка, нарисованная пунктиром. Она указывает на то, что оба конкретных класса реализуют один и тот же интерфейс. Можно сказать, что конструирование абстрактных и конкретных классов зависит от предложенного интерфейса. Пунктирная стрелка на рис. 1.4, восходящая к интерфейсу, как раз и указывает на отношение зависимости между интерфейсом и классом.

Классы и отношения

Классификаторы могут находиться между собой в различных отношениях— ассоциации, зависимости, обобщения, реализации, использования. Основными отношениями для классов можно считать отношения ассоциации, обобщения, агрегации и композиции. Интерфейсы могут находиться в отношении обобщения. Таким образом, для создания и классов, и интерфейсов можно использовать механизм наследования.

Ассоциация

Ассоциация классов — это отношение, указывающее, что между экземплярами классов существует логическая связь. Ассоциации являются важным элементом диаграмм классов. У ассоциации может быть имя, помогающее понять, что подразумевается под логической связью. Наиболее простой вид ассоциации — бинарная ассоциация, или упорядоченная пара (Имя класса_1, Имя класса_2). Например, ассоциация (Станция слежения, Объект слежения) с именем `следит` указывает на ведущую роль и назначение станции слежения. В данном случае, помещенное на первое место имя класса "Станция слежения" указывает, что при рассмотрении этой пары главную роль в конкретной модели играет именно этот класс. Бинарная ассоциация изображается сплошной линией (рис. 1.5), соединяющей два изображения классов. У концов этой линии (полюсов ассоциации) можно указать множественность экземпляров, участвующих в ассоциации. В нашем примере участвует один экземпляр

"Станции слежения", но с ним может быть связано произвольное число экземпляров класса "Объект слежения". На возможное отсутствие объектов слежения указывает значение ноль для нижней границы числа экземпляров, за которыми может следить станция слежения, и произвольное число — для верхней границы. Связь между экземплярами не обязательно должна быть постоянной — она может возникать и исчезать в процессе работы модели.

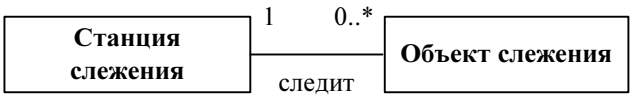


Рис. 1.5. Ассоциация с указанием множественности

С помощью различных видов ассоциации можно указать на различные виды связей между экземплярами: например, если элемент связан сразу с двумя экземплярами, то можно уточнить, может ли он быть связан одновременно с одним экземпляром, или только с одним из них.

Построение ассоциаций между классами чрезвычайно важно на первоначальном этапе построения будущей модели и может найти отражение в комментариях в таких реально используемых графических элементах языков моделирования, как функциональные схемы.

Обобщение

> "Миф о том, что наследование — это всегда хорошо, является проклятием объектно-ориентированных технологий с момента их зарождения".
>
> *Г. Буч, А. Якобсон, Дж. Рамбо [2]*

Отношение обобщения, как мы уже видели, является отношением, с помощью которого строится, например, дерево классов. В корень дерева помещается предок, или *суперкласс*, а далее следуют его потомки — *подклассы*. Напомним, что если класс в некоторой иерархии является абстрактным, то его имя записывается курсивом (рис. 1.6, *а*, *б* показывают два возможных способа представления дерева классов, связанных отношением обобщения). Дерево классов может использоваться и на этапе предварительного рассмотрения модели, и как графический элемент языка моделирования. Вспомните, как подобные графические конструкции используются для описания иерархически организованных файлов.

Отношение обобщения может быть применено и к другим компонентам, имеющим иерархическую структуру, к уже рассматривавшимся интерфейсам.

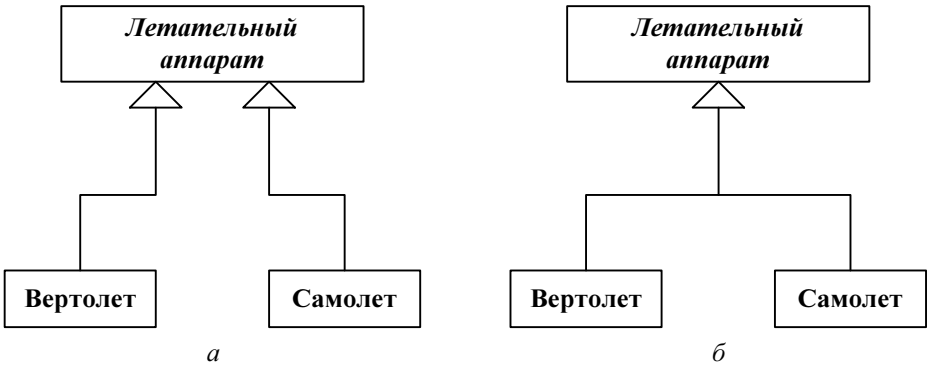


Рис. 1.6. Обобщение. Два допустимых способа представления дерева классов: *а* — соединительные линии независимы; *б* — соединительные линии объединены

На диаграмме классов можно дополнительно указать, что данное дерево классов полностью разработано и не будет в дальнейшем расширяться — {complete}, или требует доработки {incomplete}.

Агрегация

Отношение агрегации между классами является частным случаем отношения ассоциации (рис. 1.7), и указывает на то, что один класс в качестве составляющих может содержать другие классы. Это отношение "часть — целое", и оно используется для того, чтобы показать, что класс имеет внутренние компоненты и, возможно, иерархическую структуру.

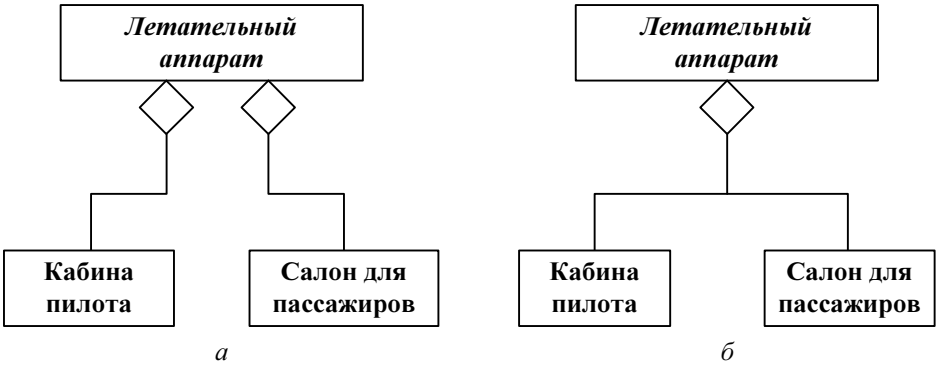


Рис. 1.7. Агрегация. Два допустимых способа представления структуры класса: *а* — соединительные линии независимы; *б* — соединительные линии объединены

Частным случаем агрегации является отношение композиции (рис. 1.8), указывающее, что класс не только имеет структуру, но и представляет собой

единое целое. Создавая такой класс, мы создаем все его компоненты одновременно, уничтожая — уничтожаем все его составляющие.

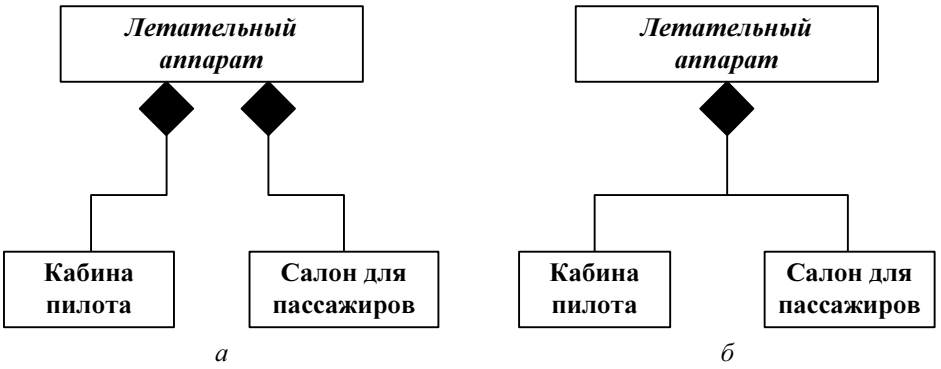


Рис. 1.8. Композиция. Два допустимых способа представления композиции: *а* — соединительные линии независимы; *б* — соединительные линии объединены

Наследование

Отношение обобщения непосредственно связано с различными способами построения новых классов на базе существующих.

Наследование — это механизм, позволяющий на базе имеющихся классов строить новые, используя уже готовые классы, как часть новых. Новые классы обобщают существующие, дополняя их новыми свойствами. Строить новые классы можно различными путями.

- **Построение новых классов с помощью композиции.**

  Весьма распространенным является прием, когда пользователю доступен только экземпляр класса-контейнера, куда можно помещать экземпляры существующих классов, настраивать их параметры и связывать между собой (см. рис. 1.1). При этом существует возможность сохранить созданный компонент в виде элемента пользовательской библиотеки (нового класса). Помещая экземпляр вновь созданного класса в новый контейнер и объединяя его с экземплярами других классов, можно построить новый компонент и иерархию классов. Этот подход чрезвычайно удобен, если заранее известно, что новое устройство можно собрать, используя только существующие компоненты.

- **Наследование путем добавления новых свойств.**

  Можно строить новые классы, наделяя существующие классы новыми свойствами (добавляя переменные, структурные компоненты и операции,

т. е. все те элементы, что составляют класс в конкретном языке программирования или моделирования) и сохраняя все уже имеющиеся. Предшественник нового класса просто встраивается в него как готовый элемент, без каких-либо изменений (рис. 1.9).

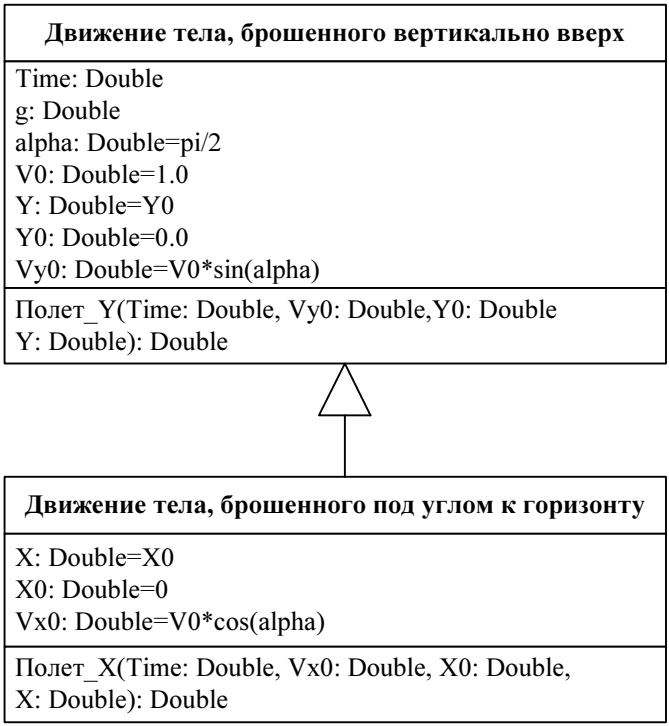
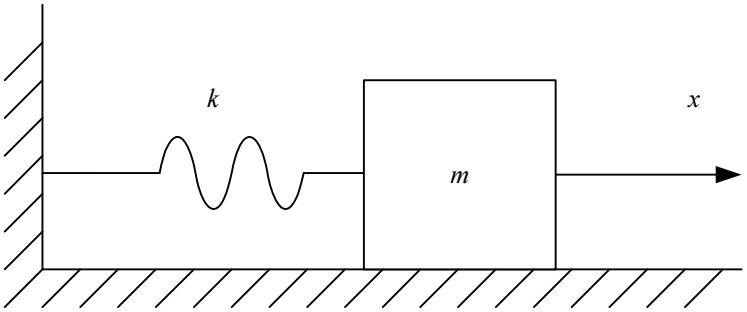


Рис. 1.9. Наследование добавлением новых свойств

На рис. 1.9 показаны два класса: "Движение тела, брошенного вертикально вверх" и "Движение тела, брошенного под углом к горизонту". Второй класс наследует операцию вычисления высоты тела и все переменные состояния первого класса и добавляет только операцию вычисления дальности полета как функции от времени и начальных условий. Столь же просто добавить в эту иерархию и еще более сложную модель. Сейчас выбранная нами система координат неподвижна. Свяжем эту систему координат с движущимся прямолинейно и с постоянной скоростью новым объектом, например, поместим наш шарик в самолет. Используя модель "Движение тела, брошенного под углом к горизонту" как базовую, не представляет труда описать полет шарика в самолете в системе координат, связанной с поверхностью Земли.

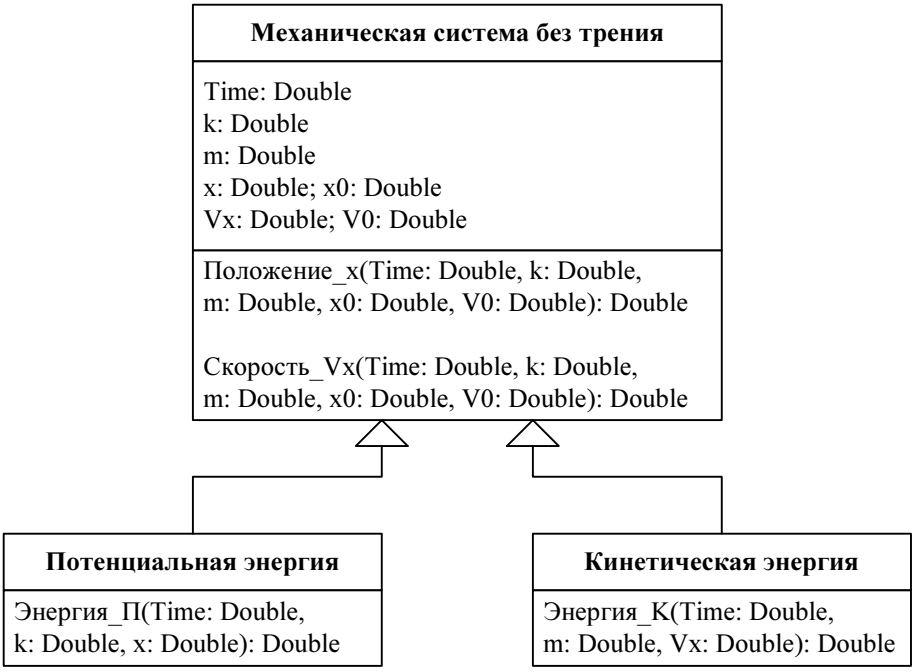


а

$$m\frac{d^2x}{dt^2} + k \cdot x = 0$$

$$T = \frac{1}{2} \cdot \left(\frac{dx}{dt}\right)^2$$

$$V = \int_0^x k \cdot x \cdot dx = \frac{1}{2} \cdot k \cdot x^2$$



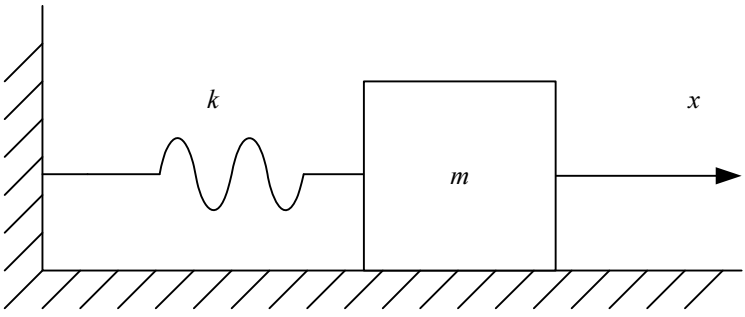
б

Рис. 1.10. Добавление новых свойств в исходный класс: *а* — механическая система; *б* — представление математической модели в виде дерева классов

Если все классы от предка до последнего потомка оказываются вложенными друг в друга, тогда они образуют одну-единственную ветвь дерева (рис. 1.9). Если новые классы, порожденные от одного и того же предшественника, отличаются между собой различными элементами, каждому из них добавлено свое, новое уникальное свойство, то возникает дерево классов. На рис. 1.10 показана простейшая механическая система, в которой класс-предок вычисляет только положение и скорость груза, движущегося за счет сжатия пружины, а его потомки могут вычислить дополнительно кинетическую и потенциальную энергию груза.

В ООМ дерево классов можно строить так, чтобы возник набор последовательно усложняющихся моделей одного и того же реального объекта. Надо признаться, что наборы последовательно усложняющихся моделей встречаются реже, чем наборы, в которых используют уже существующий класс просто как заготовку для нового класса. В последнем случае свойства предков и потомков могут существенно различаться. Но даже этот последний прием используется весьма ограниченно — деревья чаще оказываются "кустами" с небольшим числом уровней.

Рассмотренный вид наследования называется *обычным*, или *одиночным*, *наследованием*. В то же время можно представить себе ситуацию, когда потомок может наследовать свойства нескольких классов, принадлежащих



a

$$m\frac{d^2x}{dt^2} + k \cdot x = 0$$

$$m\frac{d^2x}{dt^2} + k \cdot x + kv \cdot \left(\frac{dx}{dt}\right)^2 = 0$$

$$m\frac{d^2x}{dt^2} + k \cdot x + kv \cdot \left(\frac{dx}{dt}\right)^3 = 0$$

Рис. 1.11. *(Часть 1 из 2.)* Переопределение операций исходного класса:
а — механическая система

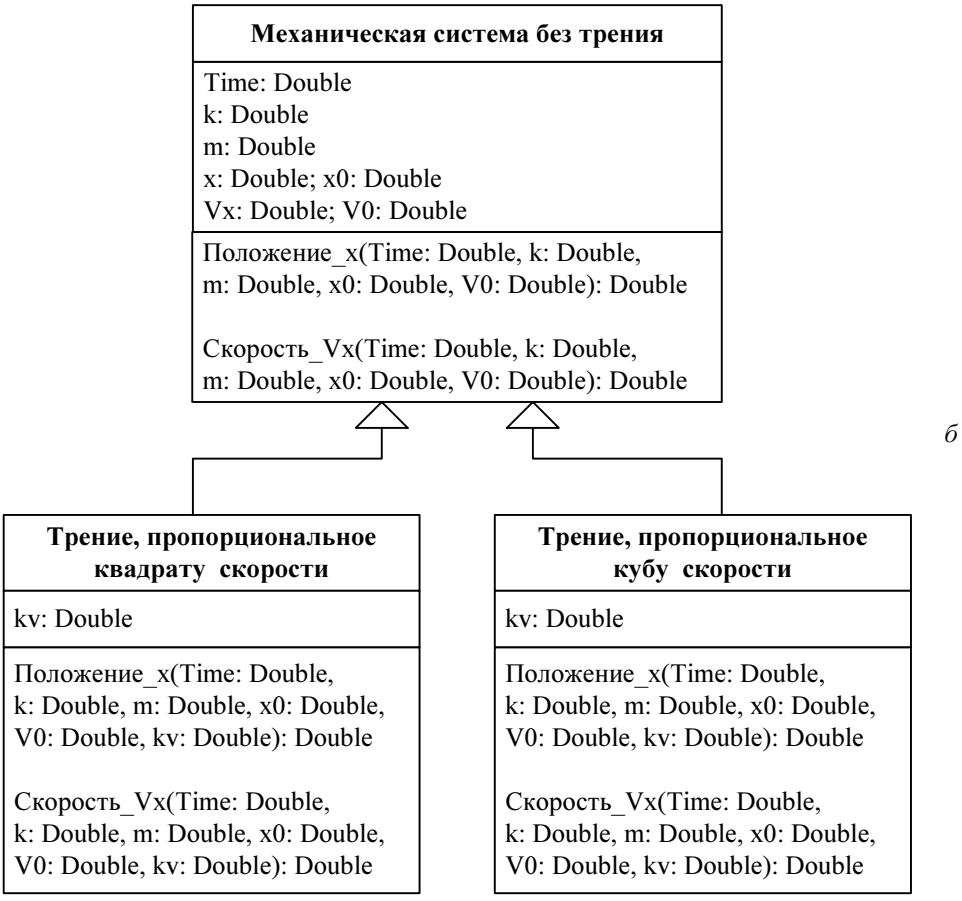


Рис. 1.11. *(Часть 2 из 2.)* Переопределение операций исходного класса: *б* — дерево классов-моделей

разным деревьям, и тогда наследование называется *множественным*. Множественное наследование — потенциально опасный механизм создания новых сущностей, т. к. у классов-родителей могут оказаться одинаковые по сигнатуре, но разные по семантике функции.

- **Наследование путем переопределения существующих свойств.**

  Во многих случаях потомки отличаются от предков тем, что они изменяют (переопределяют) свойства своих родителей. Обычно речь идет о переопределении операций или структуры. Например, в нашем примере простой механической системы предком может быть класс, моделирующий колебания груза в среде без трения, а его потомки могут учитывать различные виды трения (рис. 1.11).