Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Тамбовский государственный технический университет»

С. Ю. АЛЕКСЕЕВ

СТРУКТУРНО-ПАРАМЕТРИЧЕСКИЙ СИНТЕЗ ПРОГРАММНЫХ СИСТЕМ МОДЕЛИРОВАНИЯ ТЕХНОЛОГИЧЕСКОГО ОБОРУДОВАНИЯ ХИМИЧЕСКОЙ ПРОМЫШЛЕННОСТИ НА ОСНОВЕ АНАЛИТИЧЕСКИХ РЕШЕНИЙ ЗАДАЧ МАТЕМАТИЧЕСКОЙ ФИЗИКИ

Рекомендовано Научно-техническим советом
федерального государственного бюджетного образовательного
учреждения высшего образования «Тамбовский государственный
технический университет» в качестве монографии

Тамбов
♦ Издательский центр ФГБОУ ВО «ТГТУ» ♦
2019

УДК 004.94
ББК 32.972.1
А47

Р е ц е н з е н т ы :

Доктор технических наук, профессор,
директор по инновационному развитию АО «ТЗ «Ревтруд»
В. Е. Дидрих

Доктор технических наук, профессор,
заведующий кафедрой КИСМ ФГБОУ ВО «ТГТУ»
В. Г. Мокрозуб

Алексеев, С. Ю.

А47 Структурно-параметрический синтез программных систем моделирования технологического оборудования химической промышленности на основе аналитических решений задач математической физики : монография / С. Ю. Алексеев. – Тамбов : Издательский центр ФГБОУ ВО «ТГТУ», 2019. – 108 с. – 400 экз.

ISBN 978-5-8265-2074-1

Посвящена вопросам проектирования программного обеспечения, реализующего решения задач моделирования технических систем на основе аналитических решений уравнений математической физики.

Предназначена для студентов и аспирантов, научных работников в области инженерии программного обеспечения, профессиональных разработчиков программного обеспечения для технических систем.

УДК 004.94
ББК 32.972.1

ISBN 978-5-8265-2074-1

ВВЕДЕНИЕ

Роль вычислительной техники в обеспечении функционирования технологического оборудования химических производств постоянно возрастает так, что современное оборудование в большинстве своем содержит электронные компоненты и является результатом процессов автоматизированного проектирования.

В работе технологическое оборудование рассматривается как техническая система. Корректное функционирование каждой ее подсистемы зависит от функционирования других, составляющих подсистем. Законы функционирования технологического оборудования как технической системы не определяются просто законами функционирования отдельных его составных частей. Система в данном случае имеет интеграционные свойства, которые присущи ей только как целостной системе. Они проявляются только тогда, когда система рассматривается как единое целое – с точки зрения составляющих его элементов (подсистем) и связей между ними.

Постоянное расширение знаний о природе и механизмах физических процессов, реализуемых в технических системах рассматриваемого класса, расширяет их возможности и спектры решаемых ими задач. При этом усложняются сами технические системы. Это влечет за собой неизбежное увеличение объема и сложности задач, решаемых средствами вычислительной техники.

Спектр задач, решаемых средствами вычислительной техники в области технических систем, обширен. Это задачи исследования, моделирования, оптимизации, управления, поддержки принятия решений и дополненной реальности в технических системах. Они могут решаться на этапах проектирования технической системы или на этапах ее функционирования.

Если задачи решаются на этапах функционирования технической системы, их решения, как правило, реализуются встроенными вычислительными системами. Это специализированные средства вычислительной техники, имеющие сильно ограниченные по сравнению с бытовыми компьютерами вычислительные и функциональные мощности. Они являются неотъемлемой частью технической системы и функционируют, находясь в ее составе. Встраиваемые системы – это всегда системы с обратной связью, которые сами управляют или управляются событиями внешнего мира. В этих случаях на методы решения пере-

численных задач накладываются ограничения по требованиям к аппаратным вычислительным ресурсам и время, за которое должен быть получен результат.

Для задач, решаемых на этапах проектирования технических систем и их элементов, ограничения на методы решения по требованиям к вычислительным ресурсам и времени, за которое будет получен результат, присутствуют, но не являются жесткими.

Несмотря на различную специфику и особенности перечисленных задач, их объединяет то, что использование для их решения информации о закономерностях процессов, протекающих в технических системах, обеспечивает повышение качества их решения и, как следствие, повышение эффективности функционирования технических систем.

Традиционным и основным инструментом получения информации о закономерностях течения физических процессов на стадии проектирования технических систем являются методы математического моделирования. Наиболее полная информация о закономерностях и условиях протекания нестационарных процессов в технических системах может быть получена путем математического моделирования полей их целевых характеристик. Математическое моделирование обеспечивает формализованную модель представления технической системы, как части реального мира, в которой протекают моделируемые процессы. Расширение и эволюция методов исследований физических явлений добавляет информацию о природе протекающих процессов. Развитие знания об их природе выдвигает новые требования к качеству математического моделирования и, соответственно, усложняет модель. Таким образом, наблюдается постоянный рост сложности математических моделей и, как следствие, программ, которые их реализуют.

В современном программном обеспечении, ориентированном на решение научно-исследовательских и инженерно-технических задач, в настоящее время реализуются исключительно численные методы решения задач математической физики в частных производных. Это обусловлено возможностью получения приближенных решений для объектов сложной конфигурации и нелинейных математических постановок решаемых задач.

В то же время использование аналитических решений задач математической физики в частных производных позволяет многократно снизить объемы требуемых вычислений и их погрешности. Такой путь открывает возможности для создания программного обеспечения, гарантирующего высокую точность, надежность и достоверность

инженерно-технических расчетов, в том числе для систем реального времени.

Однако эффективная компьютерная реализация математических моделей на основе аналитических решений задач математической физики весьма специфична и требует создания комплекса соответствующих мероприятий и подходов, необходимых для разработки программного обеспечения.

Решения задач математического моделирования в этом случае рассматриваются как сочетание математических операторов и алгоритмов их использования, направленное на получение количественных характеристик моделируемых явлений и объектов. Несмотря на то, что решения, представленные в таком виде, полностью обеспечивают возможность расчета полей характеристик, использовать их вручную, без вычислительной техники, в большинстве случаев невозможно. Однократное выполнение расчета сопряжено с большим количеством промежуточных вычислений. На практике выполнение таких расчетов осуществляется многократно, для различных значений исходных данных.

Таким образом, для задач моделирования, решаемых в целях повышения качества технических систем, определяющее значение имеют как сами решения, так и их программная реализация. Программы могут представлять собой отдельные информационные системы, взаимодействующие с человеком, или функционировать в составе технических систем, взаимодействуя с их составными частями. Решения используются как основа программы, но основной результат обеспечивает вычислительный процесс. Пользователь или другие информационные системы взаимодействуют с программами, а не с алгоритмами и математическими операторами. Форма программной реализации алгоритмов и математических операторов так же, как и сами решения, определяют качество получаемых результатов моделирования: количество принятых допущений, точность, время получения результата и дальнейшие возможности его использования. Поэтому для задач моделирования принципиальную важность имеет программная реализация решений и ее качество, т.е. такие факторы, как структура программы, сложность ее потоков данных и управления, эффективность использования вычислительных ресурсов, время расчета, надежность, оценка достоверности исходных данных, возможность повторного использования кода.

Программное обеспечение абстрактно и нематериально. Оно не имеет физической природы и не подчиняется физическим законам. Отсутствие материального наполнения усложняет понимание программного обеспечения. Неформальный подход к построению программного обеспечения недостаточен на современном этапе. В этом случае на реализацию программных проектов могут уйти годы, их стоимость может многократно возрасти по сравнению с первоначальными оценками, а программное обеспечение будет ненадежным, сложным и медленным. Разработка новых и совершенствование существующих методов построения и функционирования программного обеспечения, ориентированного на использование аналитических решений задач математической физики в вычислительных алгоритмах, средств его исследования и анализа, является в настоящее время очень актуальной. Она обусловлена необходимостью соответствия используемых методов разработки и организации вычислений с результатами современных научных исследований в области программной инженерии, возможностями новых парадигм и языков программирования.

1. СОВРЕМЕННЫЕ МЕТОДЫ АРХИТЕКТУРНОГО ПРОЕКТИРОВАНИЯ ИНЖЕНЕРНО-ТЕХНИЧЕСКОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Существующие методы построения программного обеспечения являются универсальными, не учитывающими особенности какой-нибудь одной предметной области. Они направлены на сокращение трудоемкости процесса разработки и повышения качества любого программного обеспечения при увеличении его сложности и объема. Они решают проблему его сложности за счет декомпозиции на отдельные модули или фрагменты исходного кода. Программное обеспечение рассматривается как система взаимодействующих модулей.

Модули сочетают в себе полезную функциональность, направленную на решение задачи, и сервисную, направленную на обеспечение их работы. В программной системе появляются дополнительные сервисные модули и связи между ними, полностью предназначенные только для обеспечения работы программной системы. С точки зрения динамики работы программной системы – временные диаграммы работы ее модулей и взаимодействия между ними абсолютно не соответствуют временным диаграммам работы технической системы.

Сейчас компьютерное моделирование выполняется в несколько этапов (рис. 1.1).

Сначала определяются допущения, которые упрощают представление о технической системе и протекающих в ней процессах. Затем для полученного упрощенного представления технической системы строится математическая модель. В результате этого техническая система заменяется математической, с новыми сущностями и связями между ними, которые не совпадают с сущностями и связями исходной технической системы. Следующим этапом является решение полученной системы уравнений и его программная реализация. Он тоже видоизменяет сущности и связи между ними, которые присутствовали в математической системе, на лингвистические элементы конструкции языка программирования и фрагменты выполняемого кода, представляющие непрерывные процессы в дискретном виде. Можно сказать, что представление технической системы искажается в процессе компьютерного моделирования. Это происходит в большей или меньшей

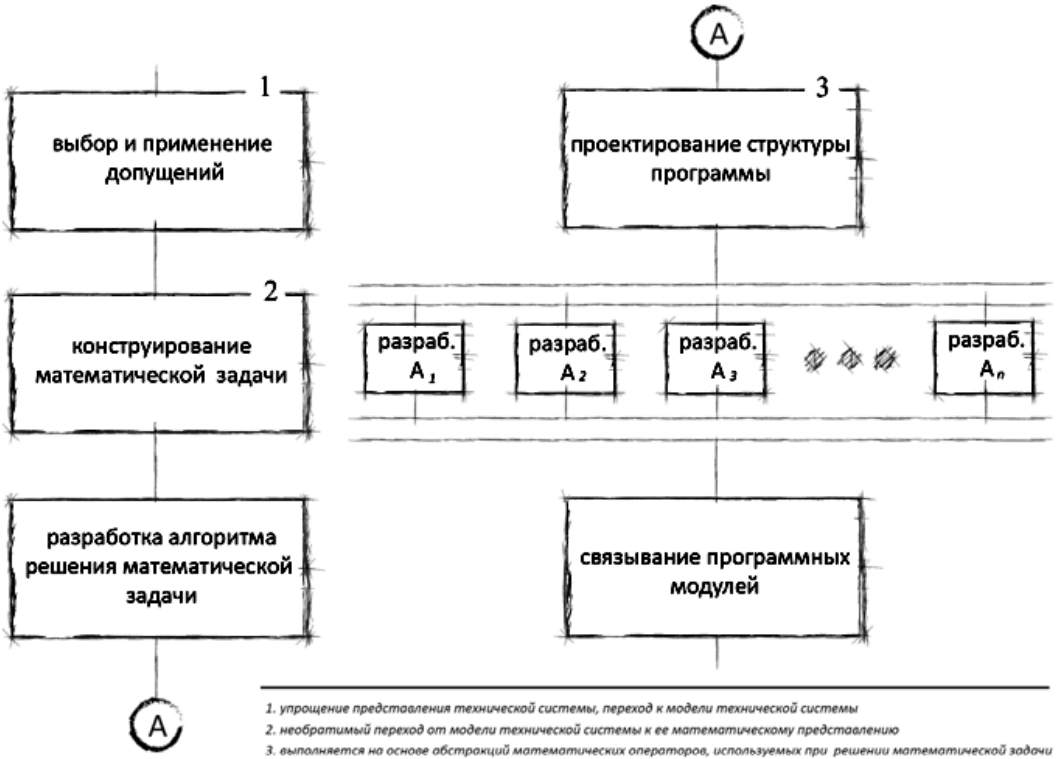


Рис. 1.1. Этапы построения компьютерной модели

степени на всех этапах компьютерного моделирования. Пользователь взаимодействуя с программой, видит программное представление технической системы, которое является результатом трех преобразований.

Общей особенностью современного программного обеспечения, реализующего решения задач моделирования является то, что оно предполагает конструирование и последующее решение математической задачи. Физика и природа моделируемых процессов рассматриваются только на этапе ее конструирования. Полученная задача решается дальше как самостоятельная при использовании программных реализаций только математических операторов. Программная система рассматривается только как средство реализации методов решения систем дифференциальных уравнений. Этот метод, когда фокус решения задачи концентрируется на решении систем уравнений, составляющих математическое описание протекающих процессов, сейчас широко применяется в программировании для различных прикладных областей. При этом в основном используется один поток вычисления, когда операции выполняются последовательно. Результатом решения являются значения отдельных функций, каждая из которых представляет поле целевой характеристики одного элемента технической системы. При такой организации моделирования динамика взаимодействия элементов технической системы рассматривается ограничено. Присутствует принципиальное несоответствие временных диаграмм работы программы временным диаграммам работы технической системы – через любые равные интервалы времени от начала работы технической системы и программы их состояния не совпадают.

На основе анализа современного программного обеспечения, ориентированного на решение научно-исследовательских и инженерно-технических задач на основе расчетов полей определяющих характеристик в элементах технических систем и потоках взаимодействующих с ними сред, выясняется, что в настоящее время реализуются исключительно численные методы решения соответствующих задач математической физики в частных производных. Это обусловлено возможностью получения приближенных решений для объектов сложной конфигурации и нелинейных математических постановок решаемых задач.

В то же время использование аналитических решений задач математической физики в частных производных (в линейных постановках для тел канонической формы) позволяет многократно снизить объемы требуемых вычислений и их погрешности (не говоря о прозрачности вычислительного процесса, физичности и наглядности промежуточных

результатов). Такой путь открывает возможности для создания программного обеспечения, гарантирующего высокую точность, надежность и достоверность инженерно-технических расчетов, в том числе для систем реального времени.

Однако эффективная компьютерная реализация математических моделей процессов тепло- и массопереноса на основе аналитических решений задач математической физики весьма специфична и требует создания комплекса соответствующих мероприятий и подходов, необходимых для разработки программного обеспечения.

Таким образом, разработка нового метода структурно-параметрического синтеза специализированных инженерных программных систем, ориентированных на использование аналитических решений задач математической физики в вычислительных алгоритмах, является безусловно актуальной задачей.

2. ПРОГРАММНОЕ ПРЕДСТАВЛЕНИЕ ТЕХНИЧЕСКИХ СИСТЕМ

Современный уровень развития теории и методов инженерии программных систем позволяет учесть специфику и расширить возможности использования аналитических решений задач математической физики. Это предполагает определение программного обеспечения в терминах системных связей и закономерностей функционирования элементов технической системы, где протекают моделируемые процессы, а не в терминах математической формулировки задачи и алгоритмов ее решения. В этом случае программное обеспечение реализует расчет характеристик процессов, протекающих в технических системах, на основе вычислений, которые соответствуют этим процессам структурно и динамически. Само программное обеспечение при этом представляется как программная система, структурно и динамически соответствующая технической системе. Другие свойства аналитических решений, которые являются их преимуществами в контексте модели абстракций технической системы, определяют целесообразность их использования:

– независимость объема вычислений от значений пространственных и временных координат – всегда, независимо от исходных данных, одинаковые время вычислений и объем вычислений, возможность планировать временные диаграммы работы программной системы;

– сокращение сложности вычислений полей характеристик, потоков энергии и вещества, средних и локальных значений характеристик процессов, балансных соотношений, дополнительных параметров;

– возможность создания универсальных программных абстракций и их многократного использования в рамках компонентно-ориентированных методов – вследствие возможности использования решений частных задач, полученных ранее;

– управление балансом вычислительной нагрузки на этапе проектирования программной системы ввиду возможности анализа и упрощения решений для характерных и предельных значений параметров процесса;

– повышение надежности асинхронного взаимодействия программных абстракций, больше возможностей в проектировании меха-

низмов их взаимодействия, отсутствие необходимости введения дополнительных вычислений для интерпретации полученных результатов – наглядность и «физичность» промежуточных и конечных расчетных результатов, сокращение затрат на отладку и тестирование программной системы.

2.1. СТРУКТУРА МЕТОДА СИНТЕЗА ПРОГРАММНЫХ АНАЛОГОВ ТЕХНИЧЕСКИХ СИСТЕМ

Структурное соответствие предполагает адекватность структуры программной системы структуре технической системы. Динамическое соответствие предполагает адекватность состояний программной системы состояниям технической системы в одинаковые моменты времени. Представление программы как структурной и динамической модели технической системы является общим универсальным принципом, позволяющим выполнять построение программ моделирования, основанных на использовании аналитических решений, для решения с их помощью широкого спектра прикладных задач различного назначения.

Модули программной системы функционально и структурно соответствует элементам технической системы. Временные диаграммы работы программной системы соответствуют временным диаграммам работы технической системы. Состояние программной системы в равные моменты времени от начала работы соответствует состоянию технической системы. Программная система становится структурной и динамической моделью технической системы.

Представляемый метод архитектурного проектирования имеет в основе своей совокупность методов программной инженерии. Ниже, на рис. 2.1, показана его структура.

1. Методы распределенных систем позволяют выполнять разработку систем, обладающих разной степенью децентрализации управления, параллелизма, автономности и физического распределения. Все этапы разработки и функционирования программного обеспечения рассматриваются с точки зрения взаимодействующих между собой агентов, которые являются абстракциями объектов предметной области.

2. Методы объектно-ориентированного программирования включают в себя описание семантики языка программирования, на котором реализуется программная система, системы типов, моделей памяти, распараллеливания вычислений, распределенных языков программирования, языков оперирования базами данных, а также вопросы безопасности на языковой основе.

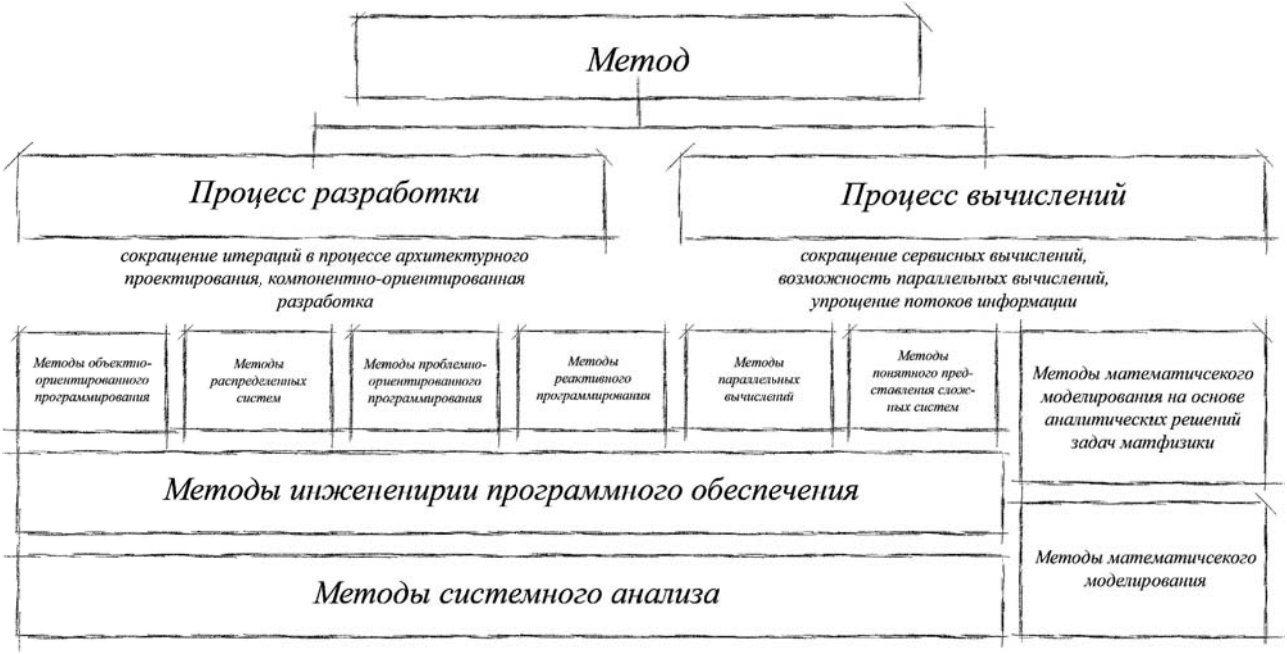


Рис. 2.1. Структура метода архитектурного проектирования

3. Методы параллельных вычислений обеспечивают построение программы на основе параллельно выполняющихся и взаимодействующих между собой процессов, распараллеливание и реинжиниринг последовательных алгоритмов и унаследованных приложений. Они описывают разработку, реализацию, анализ, профилирование и настройку параллельных приложений на современных платформах – многоядерной, распределенной или гибридной.

4. Методы предметно-ориентированного моделирования позволяют строить программные системы с использованием понятий, которые представляют объекты в предметной области, а не на основе концепций выбранного языка программирования. Тот факт, что формируются абстракции и семантика на основе объектов предметной области, позволяет разработчикам и пользователям воспринимать себя как непосредственно работающих с понятиями предметной области. Методы регламентируют одновременно разработку, внедрение и специфицирование системы. В ряде случаев конечные программные системы могут быть автоматически сгенерированы из этих спецификаций высокого уровня при использовании проблемно-ориентированных генераторов кода. Такая автоматизация возможна, так как язык моделирования и генераторы кода соответствуют требованиям узко определенной области.

5. Методы реактивного программирования и программирования на основе событий определяют поток выполнения программы последовательностью событий, возникающих внутри программной системы или в ее внешнем окружении. На основе этих методов эффективно реализуются механизмы, обеспечивающие активное состояние программного компонента, и функции обеспечения реакции на воздействия внешнего окружения. Это могут быть в общем виде сигналы от сенсоров, других программных компонентов или систем.

6. Методы построения понятных сложных систем определяют механизмы разработки, при которых сложная программа остается понятной человеку без дополнительных инструментальных средств. Явная сложность при моделировании крупномасштабных технических систем делает невозможным их полное понимание без помощи специализированных инструментов. Технологии модульного принципа позволяют создавать и применять мощные абстракции, что дает значительные преимущества в плане повторного использования и разделения проблем. Но те же самые абстракции в языках, моделях, на промежуточных уровнях разработки скрывают важные свойства программ-

ной системы. Это усугубляет проблему понимания поведения программной системы во время ее выполнения.

Композиция перечисленных методов позволяет конструировать программную систему как структурную и динамическую модель технической системы. Это свойство ограммной системы может быть использовано для расширения спектра получаемой в результате математического моделирования информации о состоянии каждого элемента технической системы в режиме реального времени, в исследовательских целях для определения сценариев работы технической системы в различных условиях.

2.2. ДЕКОМПОЗИЦИЯ ЗАДАЧИ МОДЕЛИРОВАНИЯ ТЕХНИЧЕСКИХ СИСТЕМ

Независимо от того, как построена техническая система, ее можно рассматривать как совокупность взаимодействующих подсистем, работа которых направлена на достижение одной или нескольких объединенных общей тематикой целей. Функционирование и взаимодействие подсистем рассматривается с точки зрения децентрализованного управления. Оно предполагает возможность одновременной работы подсистем и их взаимную коммуникацию без ограничений на время и порядок обмена информацией.

Основным принципом метода является расчет характеристик процессов, протекающих в технических системах на основе вычислений, структура и динамика которых соответствует структуре и динамике технической системы.

Сразу после принятия системы допущений техническая система декомпозируется на отдельные элементы. Для каждого полученного элемента технической системы конструируется программный аналог. Его можно рассматривать как программную абстракцию элемента технической системы, так как для него определяются структуры данных и функции так, чтобы он описывал свойства соответствующего ему элемента технической системы. Они представляются в виде функций времени и координат. Таким образом, отдельно для каждого элемента технической системы формулируется задача, решение которой описывает его состояние.

Можно перечислить еще несколько свойств, которые являются следствием разделения программного обеспечения на автономные абстракции с высокой степенью связности. Это возможность разрабаты-

вать абстракции независимо, соответствие подсистем по набору своих локальных состояний и функций элементам технической системы.

Обеспечение независимой разработки отдельных абстракций позволит снизить время разработки программной системы. Для программных систем, основанных на абстракциях математических операторов, которые были рассмотрены выше, разработка не всегда может выполняться параллельно. Это связано с тем, что блоки кода, реализующие шаги алгоритма решения общей задачи, сильно зацеплены между собой по общим областям данных и по управлению.

Повышение связности подсистем автоматически уменьшает их взаимное влияние, сокращает количество каналов обмена данными и управление в программной системе. Это приводит к снижению степени зацепления абстракций, упорядочиванию трафика данных и управления между ними, исключению сервисных сообщений, направленных на поддержание их работоспособности. Упорядочивание трафика позволяет упростить анализ временных диаграмм взаимодействия абстракций, снизить количество анализируемых коллизий, планировать временные диаграммы работы и взаимодействия компонентов на этапе проектирования. Это позволяет избежать введения дополнительных компонентов, выполняющих сервисные функции по обработке транзакций.

Сначала осуществляется декомпозиция технической системы, исключение из нее тех составляющих, которые не используются в расчетах требуемых характеристик с точки зрения принятой системы допущений, определение для каждого ее элемента набора величин, характеризующих его состояние, которые необходимы для расчета требуемых характеристик. Декомпозиция задачи моделирования выполняется до уровня, при котором параметры всех процессов могут быть рассчитаны при использовании аналитических решений. При этом также возможно многократное повторение расчетов для определения зависимостей изменения параметров процессов во времени. Использование аналитических решений снижает количество выполняемых вычислительных операций так, что становится возможной работа модели в режиме реального времени, параллельно с процессами, протекающими в технической системе.

После декомпозиции технической системы осуществляется построение программной модели технической системы в адресном пространстве вычислительного комплекса. Для каждого элемента технической системы создается его программная абстракция. Она основывает-

ся на методах объектно-ориентированного проектирования и представляет собой вычислительную сущность, состоящую из набора переменных, описывающих ее состояние, и набора функций, выполняющих операции над ними. Функции обеспечивают поведение абстракции.

Абстракции связываются между собой так, чтобы они организовывали структуру, совпадающую со структурой технической системы. Это структурные связи, они обеспечивают изображение в программной системе конструкции технической системы. Дальше между абстракциями добавляются информационные связи, представляющие взаимодействие элементов технической системы (рис. 2.2).

Степень деления системы на составляющие, набор подсистем и временные диаграммы их работы определяется в соответствии с выбранной системой допущений. Каждая подсистема рассматривается как элемент технической системы, который обладает своим состоянием и поведением. Под поведением понимаются возможности взаимодействовать с окружающими подсистемами, сообщая им данные о своем состоянии, и изменять его под воздействием внешних для этого элемента факторов. Внешними факторами в данном случае являются воздействия других конструктивных элементов, составляющих техническую систему.

Каждому конструктивному элементу технической системы в программной системе соответствует свой программный элемент, который обладает возможностью автономно функционировать в отдельном адресном пространстве и потоке управления, сохранять и изменять свое состояние в рамках ограниченного множества и выполнять определенный набор операций. Она описывает состояние элемента технической системы в любой момент времени ее работы и реализует функции расчета этих состояний. Они рассчитываются в зависимости от поступающей информации об изменении состояния окружающих абстракций. Это соответствует картине реального процесса, протекающего в технической системе, – ее элементы также представляют собой законченные конструктивные единицы или сборки, обладающие возможностью существовать независимо, вне контекста технической системы, оказывают влияние друг на друга, в результате чего меняются их состояние и свойства.

Сценарий работы такой программной системы представляет собой совокупность временных диаграмм работы ее подсистем, и их взаимодействия. Он определяется сценарием работы моделируемой технической системы. Взаимное влияние элементов технической системы представляется обменом информацией между абстракциями.

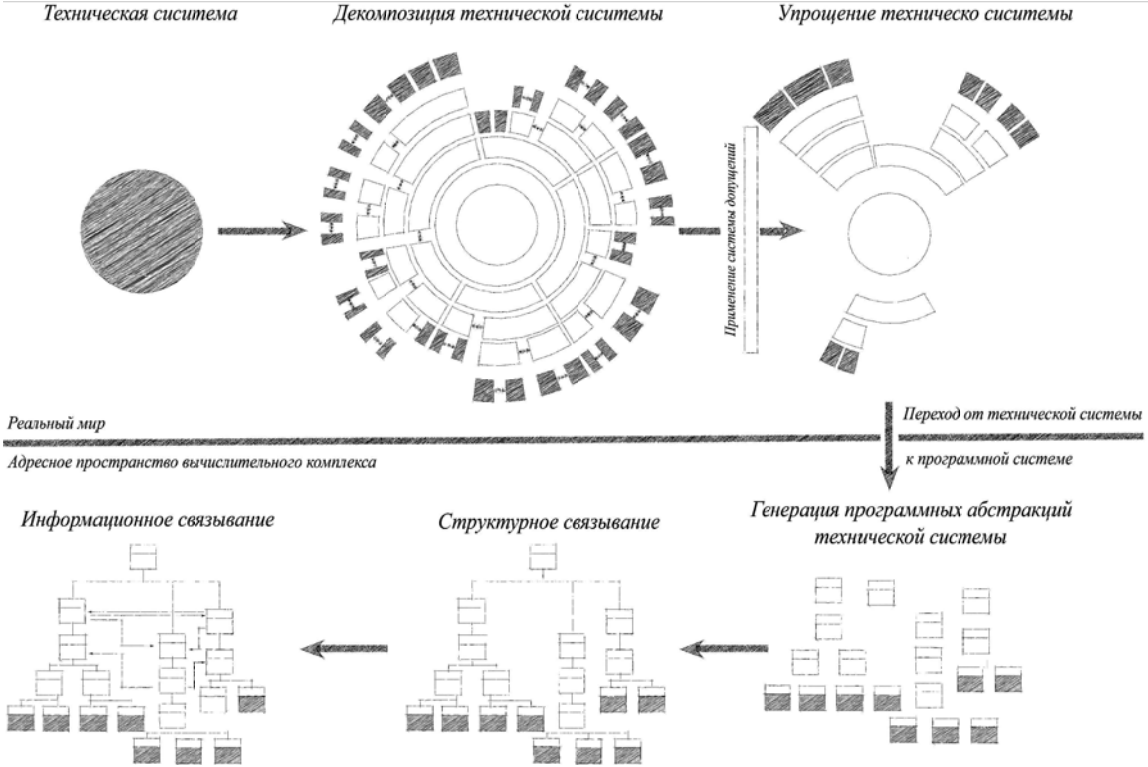


Рис. 2.2. Общая схема процесса построения программного изображения технической системы

При построении интерфейсов абстракций учитываются длительность взаимодействия элементов, информация о том, какие свойства элементов должны измениться при взаимодействии, количество параметров, характеризующих их состояния, сложность структур данных, необходимых для описания этих состояний.

Функционирование каждой абстракции зависит от функционирования других абстракций, составляющих общую систему. Функции программной системы определяют функции содержащихся в ней подсистем и их взаимодействие. Сама она рассматривается только как контейнер, в котором содержатся, функционируют и взаимодействуют абстракции.

На рисунке 2.2 представлена схема построения программной системы в соответствии с рассматриваемым методом. Согласно ему задача моделирования декомпозируется сразу на отдельные локальные подзадачи.

Принципиальным отличием процесса построения компьютерной модели на основе абстракций технической системы от традиционного, показанного на рис. 1.1, является декомпозиция не только программной реализация задачи, но и самой задача моделирования.

Математическое описание процессов, протекающих в технической системе, представляет собой совокупность математических описаний процессов, протекающих в каждом ее элементе. Моделирование работы технической системы осуществляется за счет моделирования работы ее отдельных подсистем и взаимосвязей между ними. Математическая модель разрабатывается не монолитно, целиком для всей системы, а для каждого объекта в отдельности.

Такое структурирование математического описания возможно при использовании аналитических решений систем уравнений, описывающих характеристики протекающих в элементах технической системы процессов. Использование аналитических решений сопряжено с рядом преимуществ – оно значительно снижает систематическую погрешность, повышает производительность и надежность работы программы, сокращает время получения результата.

Построение математической модели на основе аналитических решений может вызвать необходимость декомпозировать задачу дальше, до уровня, когда протекающие процессы могут быть описаны системами линейных дифференциальных уравнений. Такой подход требует разбиения области процесса на элементарные участки. Он предполагает тесное взаимодействие методов математического моделирова-

ния и методов программной инженерии, которые используются для описания типов и реализации взаимодействия элементарных участков.

Область процесса реализуется в виде последовательного набора объектов, которые являются программными аналогами элементарных участков. В каждом объекте определены особенности конструкции участка области процесса, которому он соответствует, и тип протекающих в ней процессов. В том случае, если конструкция области неизменна по длине (или в пространстве, для случая, когда процесс идет больше чем вдоль одной координаты), набор содержит однотипные объекты. Если конструкция или тип процессов изменяются в пространстве, набор содержит объекты различных типов. Здесь математическая модель формулируется и реализуется для элементарного участка. Стыковка элементарных участков, обеспечивающая моделирование процесса по всей области, осуществляется преимущественно методами программирования. Предлагаемый метод позволяет гибко описывать конструктивные особенности оборудования, учитывать изменения конструкции оборудования по длине области процесса, его деформации и изменения конструкции во времени.

Основная работа каждой абстракции направлена на вычисление значений содержащихся в ней переменных, описывающих ее состояние. Программные абстракции, как и элементы технической системы, в процессе работы постоянно взаимодействуют друг с другом, в результате чего изменяется их состояние. Это обеспечивает необходимость выполнения многократных расчетов состояния каждой абстракции. Исходными данными для расчета состояния каждой абстракции являются данные соседних, связанных с ней абстракций. Это соответствует изменению состояния элементов технической системы под воздействием на него связанных элементов.

С точки зрения работы программной системы такой подход позволит упростить логические операции по анализу типа объектов, снизить затраты вычислительных ресурсов на организацию связей между модулями, копирование, добавление, удаление и замену объектов, в зависимости от особенностей течения моделируемого процесса.

Для больших и сложных технических систем описанные выше подходы приходится комбинировать. Сначала программная система декомпозируется на объекты, которые являются абстракциями конструктивных единиц технической системы, затем для каждой конструктивной единицы определяются области течения процессов, которые необходимо моделировать. Полученные области разбиваются на эле-

ментарные участки. В программной системе каждая элементарная область представляется отдельной абстракцией, а вся область процесса – набором однотипных абстракций.

2.3. УРОВНИ ПРЕДСТАВЛЕНИЯ ПРОГРАММНЫХ АБСТРАКЦИЙ ЭЛЕМЕНТОВ ТЕХНИЧЕСКИХ СИСТЕМ

Программные абстракции рассматриваются на трех уровнях – уровень представления структуры, уровень представления динамики и концептуальный уровень (рис. 2.3). Каждый из этих уровеней определяет также и связывание абстракций.

На структурном уровне абстракции реализуют возможности представления свойств элементов технической системы и статических связей с другими абстракциями, отражающие конструкцию технической системы. Набор связанных между собой абстракций обеспечивает структурное представление технической системы. Эти возможности обеспечиваются за счет сочетания методов объектно-ориентированного программирования, проблемно-ориентированного программирования и представления сложных программных систем.

На динамическом уровне абстракции реализуют возможности представления процессов функционирования элементов технической системы. Синхронная и асинхронная работы абстракций и их взаимодействие обеспечивают динамическое представление технической системы. На этом уровне реализуются динамические связи – транспорт

3	*Концептуальный уровень*	*Методы математического моделирования*
2	*Динамический уровень* *моделирование взаимодействия элементов технической системы*	*Методы распределенных систем* *Методы параллельного вычисления* *Методы реактивного программирования* *Методы предметно-ориентированного моделирования* *Методы построения сложных систем*
1	*Структурный уровень* *моделирование конструкции технической системы*	*Методы объектно-ориентированного программирования* *Методы предметно-ориентированного моделирования* *Методы построения сложных систем*

Рис. 2.3. Уровни представления программных абстракций элементов технической системы

сообщений, несущих в себе информацию, которой обмениваются абстракции. Эти возможности реализуются сочетаниями методов распределенных систем, методов параллельного и реактивного программирования, предметно-ориентированного программирования и методов представления сложных программных систем.

Эти два уровня являются общими для всех программных систем моделирования технических систем. Они обеспечивают базовую функциональность системы и реализацию принципов структурного и динамического соответствия вычислений процессам, протекающим в технической системе.

Концептуальный уровень определяет смысловое содержание абстракции – какой элемент технической системы она представляет, моделирование каких процессов реализует, качество и точность моделирования. На этом уровне определяется содержание сообщений, которыми обмениваются абстракции. Смысловое содержание реализуется методами математического моделирования.

2.4. ОСОБЕННОСТИ ВЫЧИСЛЕНИЙ НА ОСНОВЕ ПРИНЦИПА ИХ СТРУКТУРНОГО И ДИНАМИЧЕСКОГО СООТВЕТСТВИЯ МОДЕЛИРУЕМЫМ ПРОЦЕССАМ В ТЕХНИЧЕСКИХ СИСТЕМАХ

Программные системы, спроектированные на основе модели, как абстракции технической системы, характеризуются высокой степенью связности, эффективным использованием вычислительных ресурсов, малым объемом вспомогательных вычислений и простоев, при ограничениях на вычислительную мощность комплекса и времени получения результатов расчетов. Такие свойства программных систем как моделирование структуры и динамики технической системы определяют основные возможности увеличения их эксплуатационных характеристик.

Свойство моделировать структуру технической системы обеспечивает представление модулей как абстракций технической системы, соответствие их локальных состояний и функций ее элементам. Это приводит к повышению связности компонентов и исключению из программной системы сервисных компонентов, направленных на обеспечение ее работы, снижению общего количества компонентов в программной системе. Снижение количества компонентов уменьшает их взаимное влияние, сокращает количество каналов обмена данными и управления в программной системе, объем сервисных функций, направленных на поддержание работоспособности системы, затраты

оперативной памяти. Для компонентов, работающих параллельно, сокращается интенсивность загрузки ресурсов процессора, для компонентов, выполняющихся последовательно, – время получения результата.

Эквивалентность структуры программной и технической систем обеспечивает присутствие в программной системе только тех элементов, которые являются абстракциями элементов технической системы и отсутствие других. Из программной системы исключаются все элементы, выполняющие сервисные вычисления, направленные на обеспечение работы программной системы – централизованные хранилища данных, менеджеры сообщений, транзакций и т.д. Использование программных элементов, выполняющих сервисные функции, характеризуется тем, что они связаны со всеми элементами программной системы. Их исключение значительно упрощает структуру программной системы (рис. 2.4).

Полностью сервисные вычисления исключить невозможно. В рассматриваемом случае они распределяются между абстракциями. Приближение абстракции к элементу технической системы согласно соответствиям, показанным на рис. 2.3, повышает ее связность – степень логической завершенности и ориентированности всех ее ресурсов на решение задач по расчету значений, определяющих ее состояние. Идеальным вариантом связности является связность элементов технической системы. Общее количество сервисных вычислений при распределении их по абстракциям снижается. Также количество сервисных вычислений снижается с повышением связности абстракции.

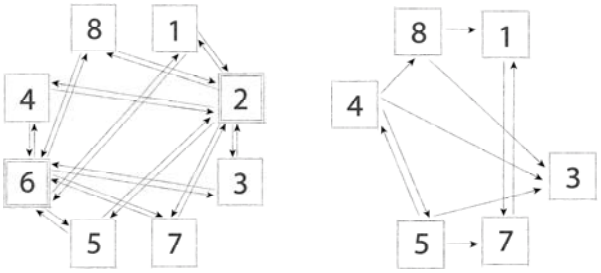
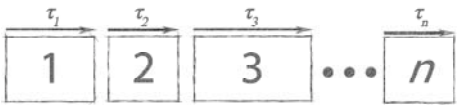


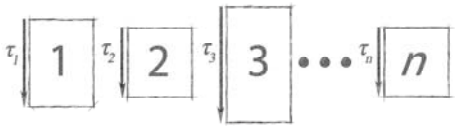
Рис. 2.4. Упрощение структуры программной системы при исключении сервисных компонентов (на примере программной системы, состоящей из 8-ми элементов, из которой исключаются сервисные элементы 6 и 2)

последовательная работа абстракций



$$\tau = \sum_{i=1}^{n} \tau_i$$

параллельная работа абстракций



$$\tau = \max \tau_i, \ i \in 1, ..., n$$

Рис. 2.5. Сокращение времени вычислений при организации параллельной работы абстракций

Эквивалентность динамики программной и технической систем, организация временных диаграмм работы абстракций так же, как временных диаграмм работы элементов технической системы, позволяет обеспечивать параллельные вычисления. Это повышает эффективность вычислений и сокращает время выполнения расчетов (рис. 2.5).

Динамическое и структурное соответствие программной и технической систем позволяет упорядочить и унифицировать обмен информацией между абстракциями. Содержание потоков информации в этом случае определяется как параметры воздействия элемента технической системы на соседние, с которыми он связан. Использование в основе работы абстракций аналитических методов расчетов их состояния всегда определяет содержание потоков информации как набор только физических значений. Такое их представление сокращает количество типов сообщений, которое необходимо обрабатывать в ходе своей работы абстракциям (рис. 2.6). Это позволяет дополнительно повысить эффективность внутренней работы самих абстракций – снижаются затраты вычислительных ресурсов на генерацию новых, получение и обработку сообщений, упрощаются внутренние потоки управления.

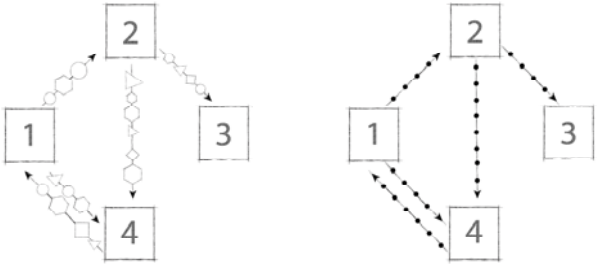


Рис. 2.6. Унификация количества передаваемых сообщений (на примере программной системы, состоящей из 4 абстракций; различными геометрическими фигурами показаны различные типы сообщений, различный размер фигур показывает различные размеры сообщений)

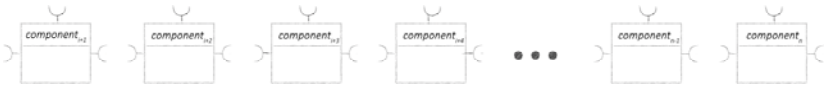
Для представления свойств абстракций используются элементарные типы или сложные структуры данных. Элементарные данные представляют одиночные значения, с их помощью описываются перечисленные выше свойства компонентов: геометрические параметры, физические характеристики и т.п. Если при течении моделируемых процессов физические характеристики сред изменяются, например, в зависимости от температуры, соответствующие свойства компонента представляются функциями.

Это приводит к снижению степени зацепления компонентов, уменьшению трафика обмена информацией между компонентами и в программной системе в целом. Уменьшение трафика позволяет повысить эффективность внутренней работы самих компонентов – снижаются затраты вычислительных ресурсов и времени на отправку, получение и обработку сообщений, упрощаются потоки управления внутри компонента, направленные на прерывание основной работы по вычислению характеристик процессов для обработки поступающих сообщений. Для распределенных вычислительных систем снижается количество потерянных сообщений, и, соответственно, затраты вычислительных ресурсов и времени на работу алгоритмов восстановления после ошибок. Сокращение трафика сообщений позволяет также планировать временные диаграммы работы и взаимодействия компонентов на этапе проектирования программной системы, что позволяет избежать введения дополнительных компонентов, выполняющих сервисные функции по обработке транзакций.

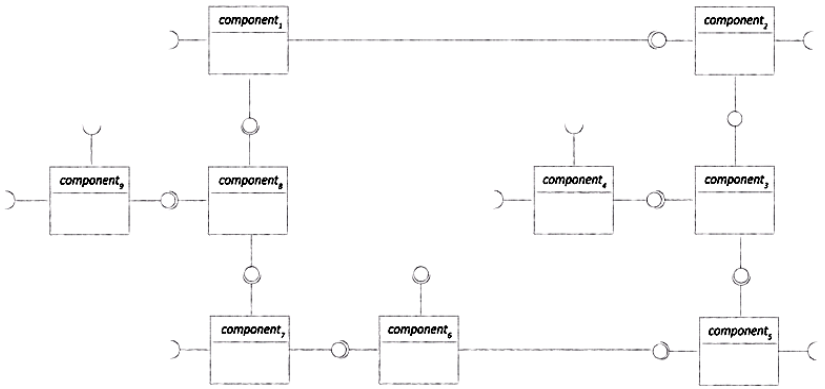
2.5. ИСПОЛЬЗОВАНИЕ МЕТОДОВ КОМПОНЕНТНО-ОРИЕНТИРОВАННОЙ РАЗРАБОТКИ ПРОГРАММНЫХ СИСТЕМ МОДЕЛИРОВАНИЯ ТЕХНИЧЕСКИХ СИСТЕМ

Свойством, которое является следствием высокой связности получаемых на основе модели программных абстракций, является принципиальная возможность использовать для построения программных систем моделирования компонентно-ориентированную архитектуру.

В рассматриваемом случае компонентно-ориентированный подход сопоставим с принципом *Lego*, обеспечивает недорогой и высококачественный стандарт быстрого изготовления программного обеспечения. Он имеет особенно высокий потенциал многократного использования. Компонент реализуется в виде объектного бинарного кода или байт-кода так, чтобы на его использование не накладывались ограничения синтаксиса и семантики используемых языков программирования.



а)



б)

Рис. 2.7. Построение программной системы на основе компонентов:
а – доступный набор компонентов;
б – построенная из них программная система

Использование компонento-ориентированного подхода возможно, если программные элементы определены как независимые модули, обеспечивающие полный набор функций и свойств, необходимых для решения одной локальной задачи. Тогда они могут рассматриваться как отдельные строительные элементы программной системы. Совместное использование перечисленных выше методов позволяет определять компоненты как программные аналоги элементов технической системы, где протекают моделируемые процессы. Роль каждого компонента в программной системе соответствует роли его аналога в технической системе. Соответственно тому, как элементы технической системы определяют течение моделируемых процессов, программные компоненты участвуют в решении задачи моделирования.

Свойства и методы компонента обеспечивают возможность сохранения его состояния и его функциональность, направленные на расчет для текущего момента времени характеристик соответствующего элемента технической системы и протекающих в ней процессов. Обработчики событий позволяют компоненту реагировать на внешние события, связанные с поведением других компонентов или оборудования.

Использование компонентно-ориентированного подхода позволит быстро макетировать программные системы, реализующие решение требуемой задачи моделирования. По мере накопления в библиотеке компонентов время разработки программных систем моделирования будет снижаться при сохраняющихся ограничениях на скорость выполнения расчетов, точность полученных результатов, надежность работы программной системы. Это позволит сэкономить время и ресурсы, затрачиваемые сейчас на изготовление прототипов и макетных образцов. Новые, реализованные в виде компонентов решения локальных задач, сохраняются и добавляются в библиотеку, которая постоянно расширяется.

При разработке программных систем моделирования на основе рассматриваемой модели увеличивается скорость разработки и одновременно повышается качество полученных проектных решений.

При описании множества состояний абстракций на основе аналитических решений задач математической физики задача моделирования может быть декомпозирована ниже уровня отдельных конструктивных элементов технической системы. Область течения процессов каждого конструктивного элемента представляется в виде совокупности последовательных локальных участков, для которых возможно получение аналитического решения. Каждый локальный участок рассматривается

как отдельный элемент и для него в программной системе присутствует соответствующая абстракция. Такая организация не вступает в противоречие с принципом, что количество компонентов программной системы не должно превышать количество элементов технической системы, включенных в ее модель, и роли этих компонентов должны согласовываться. Все компоненты программной системы напрямую работают только с контейнером, для них существует только он один, который описывает всю область течения процесса в целом. Это незначительно усложняет устройство одной абстракции программной системы, не оказывая никакого влияния на конструкцию, функциональность, логику работы остальных абстракций, составляющих программную систему. Также это не оказывает никакого влияния на каналы обмена информацией между ними.

Для каждого элемента технической системы, включенного в модель, определяются характеристики, которые описывают его состояние как следствие протекающих в нем процессов. На основе определенных характеристик для каждого программного компонента формируется множество свойств, которые используются для описания изменения состояния программного компонента в течение времени работы программной системы. Изменения состояния программного компонента соответствуют изменениям состояния элемента технического комплекса.

Для представления свойств компонентов используются элементарные или сложные структуры данных. Элементарные данные представляют одиночные значения, с их помощью описываются перечисленные выше свойства компонентов: геометрические параметры, физические характеристики и т.п. Если при течении моделируемых процессов физические характеристики сред изменяются, например, в зависимости от температуры, соответствующие свойства компонента представляются функциями. Основными характеристиками течения процессов в техническом комплексе являются поля температур, концентраций, давлений и скоростей. Они представляются сложными математическими операторами, описать которые элементарными данными практически невозможно. Для их представления используются специализированные программные компоненты, включающие в себя сложные структуры данных, и функции, реализующие соответствующие математические операторы. Эти компоненты входят в состав основных компонентов, составляющих скелет программной системы, наряду с элементарными данными.

Такая организация так же, как и в случае с составными абстракциями, не увеличивает число основных программных компонентов и не изменяет их ролей. Она всего лишь регламентирует способ организации внутреннего устройства компонентов. В этом случае так же, как в случае с составными абстракциями, отсутствует влияние на конструкцию программной системы, логику работы отдельных абстракций и коммуникации между ними.

Таким образом, программные абстракции получают возможность сохранять подробную информацию о множестве своих уникальных состояний в виде полей целевых характеристик, являющихся функциями трех координат и времени.

Свойства элементов технического комплекса изменяются в ходе протекающих процессов и их взаимодействия между собой. Также и в программной системе ее поведение и изменение свойств абстракций осуществляется в результате взаимодействия компонентов. Для изменения свойств программных компонентов в общем виде используются сообщения между абстракциями.

3. СТРУКТУРА МАТЕМАТИЧЕСКИХ ОПЕРАТОРОВ, ОПРЕДЕЛЯЮЩИХ СОДЕРЖАНИЕ ПРОГРАММНЫХ АБСТРАКЦИЙ ЭЛЕМЕНТОВ ТЕХНИЧЕСКИХ СИСТЕМ

Методология математического моделирования химико-технологических процессов на основе аналитических решений задач математической физики в частных производных, разработанная профессором Е. Н. Туголуковым, нашла широкое применение в инженерной и научно-исследовательской практике. Структура и формы математических операторов, составляющих аналитические решения задач математической физики, описывающих свойства программных абстракций элементов технической системы, рассмотрены ниже на примере решения одномерной задачи нестационарной теплопроводности для тел канонической формы.

Несмотря на возможность непосредственного решения задачи с неоднородными граничными условиями, в случае получения решения для практической реализации на компьютере целесообразнее получать решение с выделением стационарной составляющей температурного поля. При этом достигается лучшая сходимость рядов, составляющих решение нестационарной составляющей задачи теплопроводности, что практически приводит к снижению объема вычислений и вычислительных погрешностей.

При решении задач теплопроводности в многослойных телах с неоднородными граничными условиями предпочтительно решение искать в виде алгебраической суммы решений стационарной задачи теплопроводности с неоднородными граничными условиями и нестационарной задачи теплопроводности с однородными граничными условиями. Кроме того, решение стационарных задач теплопроводности в ряде случаев имеет самостоятельное значение.

Ниже приведены постановка и решение одномерной задачи нестационарной теплопроводности с произвольным начальным условием, неоднородными граничными условиями и распределенным внутренним источником тепла.

Ниже приведена постановка задачи и показано решение с произвольным начальным условием, неоднородными граничными условиями и распределенным внутренним источником тепла.

$$\frac{\partial t_i(r_i,\tau)}{\partial \tau} = a_i^2 \left(\frac{\partial^2 t_i(r_i,\tau)}{\partial r_i^2} + A_{k,i} \frac{\partial t_i(r_i,\tau)}{\partial r_i} \right) + Q_i\ (r_i,\tau),$$

$$i = 1,2,...,N,\ R_{i-1} \le r_i \le R_i,\ k = 0,1,2,\ \tau > 0, \tag{3.1}$$

$$t_i(r_i,0) = f_i\ (r_i), \tag{3.2}$$

$$\lambda_1 \frac{\partial t_1(R_0,\tau)}{\partial r_1} - \alpha_1 \left(t_1(R_0,\tau) - t_{c1}(\tau)\right) = 0, \tag{3.3}$$

$$\lambda_N \frac{\partial t_N(R_N,\tau)}{\partial r_N} + \alpha_N \left(t_N(R_N,\tau) - t_{cN}(\tau)\right) = 0, \tag{3.4}$$

$$t_j(R_j,\tau) = t_{j+1}(R_j,\tau),\ \lambda_j \frac{\partial t_j(R_j,\tau)}{\partial r_j} = \lambda_{j+1} \frac{\partial t_{j+1}(R_j,\tau)}{\partial r_{j+1}},\quad j = 1,2,...,N-1. \tag{3.5}$$

где r – пространственная координата; τ – время; N – число слоев многослойной области; $t_i(r_i,\ \tau)$ – температурное поле i-й области; $Q_i(r_i, \tau)$ – функция внутреннего источника тепла i-й области; λ_i, a_i^2 – соответственно коэффициенты теплопроводности и температуропроводности i-й области; $A_{k,i}$ – коэффициенты уравнения, определяемые видом координат:

- для декартовой системы координат $k = 0$, $A_{0,i} = 0$;
- для цилиндрической системы координат $k = 1$, $A_{1,i} = 1/r_i$;
- для сферической системы координат $k = 2$, $A_{2,i} = 2/ r_i$;

α_1, α_N – коэффициенты конвективной теплоотдачи от внешних поверхностей в окружающую среду; $f_i\ (r_i)$ – начальное распределение температуры в i-й области; $t_{c1}(\tau)$, $t_{cN}(\tau)$ – температуры окружающей среды как функции времени; R_{i-1}, R_i – координаты границ i-й области.

Решение задачи (3.1) – (3.5) имеет вид

$$t_i(r_i,\tau) = \sum_{n=1}^{\infty} \frac{U(\mu_n,\tau) W_i(r_i,\mu_n)}{S_n}, \tag{3.6}$$

где

$$U(\mu_n,\tau) = \exp\left(-\mu_n^2\ \tau\right)\left(U(\mu_n,0) + \int_0^{\tau} \left(G(\mu_n,\tau) + F(\mu_n,\tau)\right) \exp\left(\mu_n^2\ \tau\right) d\tau \right), \tag{3.7}$$

$$U(\mu_n, 0) = \sum_{m=1}^{N} \frac{\lambda_m}{a_m^2} \int_{R_{m-1}}^{R_m} \rho(r_m) f_m(r_m) W_m(r_m, \mu_n) dr_m , \qquad (3.8)$$

$$G(\mu_n, \tau) = \sum_{m=1}^{N} \frac{\lambda_m}{a_m^2} \int_{R_{m-1}}^{R_m} \rho(r_m) Q_m(r_m, \tau) W_m(r_m, \mu_n) dr_m , \qquad (3.9)$$

$$F(\mu_n, \tau) = \frac{\alpha_N}{\lambda_N} W(R_N, \mu_n) t_{cN}(\tau) + \frac{\alpha_1}{\lambda_1} W(R_0, \mu_n) t_{c1}(\tau), \qquad (3.10)$$

$$N \frac{\lambda_m}{a_m^2} \int_{R_{m-1}}^{R_m} \rho(r_m) W_m^2(r_m, \mu_n) dr_m \quad . \qquad (3.11)$$

В декартовой системе координат:

$$W_i(r_i, \mu_n) = C_{i,n} \sin\left(\frac{\mu_n r_i}{a_i} + \varphi_{i,n} \right), \qquad (3.12)$$

$$C_{1,n} = 1, \quad C_{j+1,n} = C_{j,n} \frac{\sin\left(\frac{\mu_n R_j}{a_j} + \varphi_{j,n} \right)}{\sin(\varphi_{j+1,n})}, \quad j = 1, 2, ..., N-1, \qquad (3.13)$$

$$\varphi_{1,n} = \operatorname{arctg}\left(\frac{\lambda_1 \mu_n}{\alpha_1 a_1} \right), \qquad (3.14)$$

$$\varphi_{j+1,n} = \operatorname{arctg}\left(\frac{\lambda_{j+1} a_j}{\lambda_j a_{j+1}} \operatorname{tg}\left(\frac{\mu_n R_j}{a_j} + \varphi_{j,n} \right) \right), \quad j = 1, 2, ..., N-1, \qquad (3.15)$$

μ_n – n-й положительный корень уравнения

$$\frac{\lambda_N \mu}{a_N} \cos\left(\frac{\mu R_N}{a_N} + \varphi_N \right) + \alpha_N \sin\left(\frac{\mu R_N}{a_N} + \varphi_N \right) = 0, \qquad (3.16)$$

$$\rho(r_m) = 1. \qquad (3.17)$$

В цилиндрической системе координат:

$$W_i(r_i,\mu_n)=C_{i,n}\,J_0\left(\frac{\mu_n r_i}{a_i}\right)+D_{i,n}\,Y_0\left(\frac{\mu_n r_i}{a_i}\right), \tag{3.18}$$

$$C_{1,n}=1,\quad D_{1,i}=-\frac{\frac{\lambda_1\,\mu_i}{a_1}J_1\left(\frac{\mu_i}{a_1}R_0\right)+\alpha_1 J_0\left(\frac{\mu_i}{a_1}R_0\right)}{\alpha_1 Y_0\left(\frac{\mu_i}{a_1}R_0\right)+\frac{\lambda_1\mu_i}{a_1}Y_1\left(\frac{\mu_i}{a_1}R_0\right)}, \tag{3.19}$$

$$C_{j+1,i}=\frac{C_{j,i}J_0\left(\frac{\mu_i}{a_j}R_j\right)+D_{j,i}Y_0\left(\frac{\mu_i}{a_j}R_j\right)-D_{j+1,i}Y_0\left(\frac{\mu_i}{a_{j+1}}R_j\right)}{Y_0\left(\frac{\mu_i}{a_{j+1}}R_j\right)},$$

$$j=1,2,...,N-1, \tag{3.20}$$

$$D_{j+1,i}=\frac{\frac{\lambda_j a_{j+1}}{\lambda_{j+1}a_j}J_0\left(\frac{\mu_i}{a_{j+1}}R_j\right)\left(C_{j,i}J_1\left(\frac{\mu_i}{a_j}R_j\right)+D_{j,i}Y_1\left(\frac{\mu_i}{a_j}R_j\right)\right)-}{J_0\left(\frac{\mu_i}{a_{j+1}}R_j\right)Y_1\left(\frac{\mu_i}{a_{j+1}}R_j\right)-}\rightarrow$$

$$\rightarrow\frac{-J_1\left(\frac{\mu_i}{a_{j+1}}R_j\right)\left(C_{j,i}J_0\left(\frac{\mu_i}{a_j}R_j\right)+D_{j,i}Y_0\left(\frac{\mu_i}{a_j}R_j\right)\right)}{-J_1\left(\frac{\mu_i}{a_{j+1}}R_j\right)Y_0\left(\frac{\mu_i}{a_{j+1}}R_j\right)},$$

$$j=1,2,...,N-1, \tag{3.21}$$

μ_n – n-й положительный корень уравнения

$$C_{N,n}\left(J_0\left(\frac{\mu R_N}{a_N}\right)-\frac{\mu\lambda_N}{a_N\alpha_N}J_1\left(\frac{\mu R_N}{a_N}\right)\right)+$$

$$+D_{N,n}\left(Y_0\left(\frac{\mu R_N}{a_N}\right)-\frac{\mu\lambda_N}{a_N\alpha_N}Y_1\left(\frac{\mu R_N}{a_N}\right)\right)=0, \tag{3.22}$$

$J_0(z), J_1(z), Y_0(z), Y_1(z)$ – функции Бесселя,

$$\rho(r_m)=r_m. \tag{3.23}$$

В сферической системе координат:

$$W_i(r_i, \mu_n) = \frac{C_{i,n}}{r_i} \sin\left(\frac{\mu_n r_i}{a_i} + \varphi_{i,n} \right), \tag{3.24}$$

$$C_{1,n} = 1, \quad \varphi_{1,n} = -\frac{\mu_n R_0}{a_1} + \operatorname{arctg}\left(\frac{\lambda_1 R_0 \mu_n}{a_1(\alpha_1 R_0 + \lambda_1)} \right), \tag{3.25}$$

$$\varphi_{m+1,n} = -\frac{\mu_n R_m}{a_{m+1}} + \operatorname{arctg}\left(\frac{a_{m+1}}{\lambda_{m+1}\mu_n} \left(\frac{\lambda_{m+1} - \lambda_m}{R_m} + \frac{\mu_n \lambda_m}{a_m} \operatorname{tg}\left(\frac{\mu_n}{a_m} R_m + \varphi_{m,n} \right) \right) \right),$$
$$m = 1, 2, ..., N-1, \tag{3.26}$$

$$C_{m+1,n} = C_{m,n} \frac{\sin\left(\frac{\mu_n}{a_m} R_m + \varphi_{m,n} \right)}{\sin\left(\frac{\mu_n}{a_{m+1}} R_m + \varphi_{m+1,n} \right)}, \quad m = 1, 2, ..., N-1, \tag{3.27}$$

μ_n – n-й положительный корень уравнения

$$\frac{\mu}{a_N} \cos\left(\frac{\mu R_N}{a_N} + \varphi_N \right) + \left(\frac{\alpha_N}{\lambda_N} - \frac{1}{R_N} \right) \sin\left(\frac{\mu R_N}{a_N} + \varphi_N \right) = 0, \tag{3.28}$$

$$\rho(r_m) = r_m^2. \tag{3.29}$$

Для того чтобы грамотно реализовать в программных абстракциях структуру соответствующих решений, необходимо использовать сложные абстрактные типы данных – с помощью встроенных типов данных они представлены быть не могут. Это определяет сложную структуру программных абстракций, представляющих элементы технических систем. Она включает абстракции математических конструкций, присутствующих в аналитическом решении соответствующей задачи.

Таким образом, в программной системе присутствуют два уровня разбиения. Первый соответствует разбиению технической системы на элементы, состояние которых может быть описано задачами, имеющими аналитические решения. Второй дальше разбивает каждую абстракцию на части, соответствующие частям математических конструкций, представляющих аналитическое решение.

4. ПРОГРАММНОЕ ПРЕДСТАВЛЕНИЕ СТРУКТУРЫ ТЕХНИЧЕСКИХ СИСТЕМ

Конструктивно техническая система может быть представлена в виде иерархической структуры. Такая форма представляет декомпозицию технической системы в пространстве и позволяет однозначно описать конструктивные зависимости ее элементов и отчасти их взаимное местоположение. Она определяет место объекта в общей структуре, допускает описание составных объектов и связей между объектами.

Особенностью иерархических структур является то, что они предполагают одновременное существование всех своих элементов. При моделировании структуры технических систем это свойство используется для обеспечения целостности ее представления. В технических системах в ходе течения в них процессов элементы могут претерпевать качественные изменения. Иерархические структуры также позволяют исключать элементы и добавлять на место исключенных новые. При этом всегда сохраняется их целостность представления технической системы в любой момент времени ее функционирования. Моделирование случаев, когда в результате работы технической системы изменяется ее конструкция, возможно за счет исключения из структуры старых и добавления новых элементов. В случае моделирования изменения конструкции технической системы возможность элементов древовидных иерархических структур обеспечивать контроль своих подчиненных элементов обеспечивает целостность структуры технической системы.

В технических системах элементы могут конструктивно изменяться или исчезать. Из иерархических структур также можно исключать элементы или менять их тип, но при этом всегда благодаря этому свойству сохраняется их целостность, соответствующая состоянию технической системы.

Можно выделить основные типы иерархических структур по силе связей элементов в них – с сильными, слабыми связями, смешанные и многоэшелонные.

Иерархические структуры с сильными связями (древовидные иерархические структуры) – структуры, в которых каждый элемент нижнего уровня подчиняется только одному элементу верхнего уровня.

Иерархические структуры со слабыми связями – это структуры, где каждый элемент нижнего уровня подчиняется нескольким элементам вышележащего уровня.

Смешанные иерархические структуры – это структуры, в которых присутствуют вертикальные и горизонтальные связи между элементами.

Многоэшелонные иерархические структуры – сложные структуры с различными видами отношений элементов в рамках уровня и различной степенью вмешательства элементов вышележащего уровня в организацию отношений между элементами нижележащего. Основной отличительной особенностью таких структур является определенный уровень независимости элементов низших уровней от элементов верхних уровней.

Структура технической системы может быть представлена с помощью любого из четырех описанных вариантов. Наиболее предпочтительным является первый вариант – иерархические древовидные структуры. Он позволяет лаконично и четко, используя минимальное количество взаимосвязей, описать структуру технической системы (рис. 4.1). Практический опыт показывает, что с помощью древовидных структур могут быть представлены большинство технических систем. Некоторые отношения элементов в технических системах могут быть рассмотрены с точки зрения иерархических структур со слабыми связями, в которых один элемент подчинен нескольким родительским элементам. Это увеличивает количество связей между элементами, которые необходимо реализовывать, и усложняет отношения элементов структуры.

В подавляющем большинстве случаев такой вариант может быть описан при незначительном изменении уровня декомпозиции технической системы с помощью древовидной структуры.

При описании структуры на основе иерархической древовидной структуры возможно возникновение элементов, которые направлены на сохранение целостности иерархической структуры. На рисунке, в иерархической структуре, технический комплекс представлен как отдельный элемент. С физической точки зрения он представляет собой набор взаимодействующих подсистем, которые также представлены в программной системе объектами. Эти объекты однозначно описывают состояние подсистем в каждый момент времени. Функции элемента, описывающего техническую систему, в этом случае формальны, они направлены только на сохранение целостности иерархической структуры. Этот элемент не выполняет никакой вычислительной нагрузки и на этапе выполнения программной системы он себя никак не проявляет.

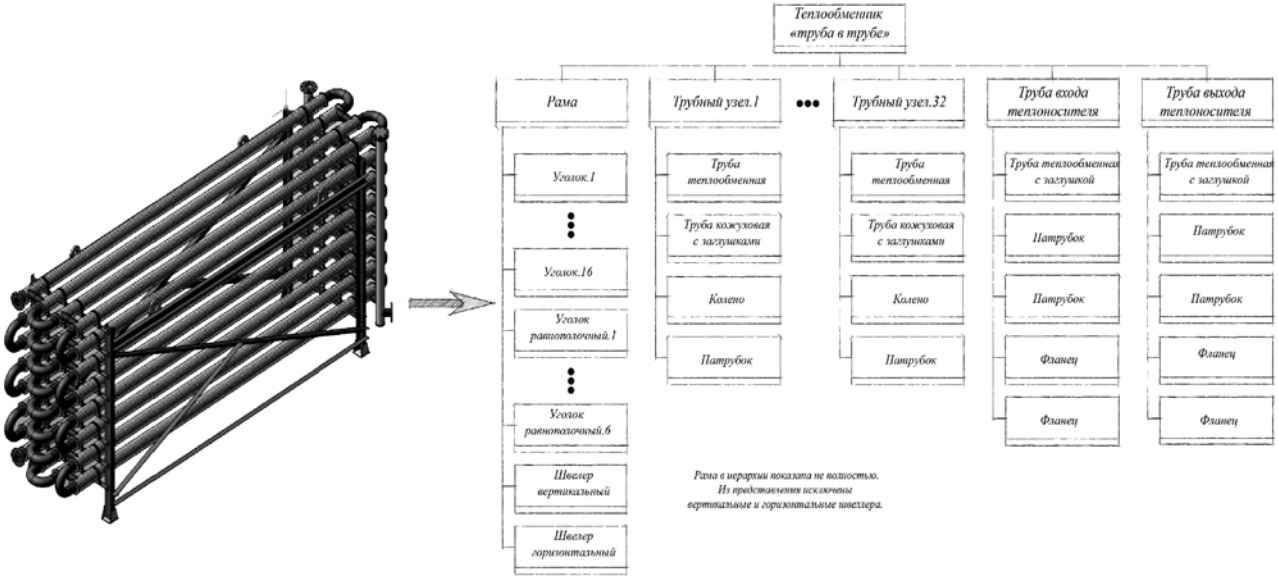


Рис. 4.1. Представление структуры технической системы в виде древовидной иерархии

Аналогичная ситуация может наблюдаться и на нижних уровнях иерархии. Это не противоречит принципу, что каждый элемент в программной системе должен соответствовать своему аналогу в технической системе.

В иерархическую систему включаются только конструктивные элементы технических систем. На самых нижних ее уровнях появляются элементы, представляющие абстракции математических операторов. Представление структуры технической системы в виде древовидной иерархии, в которую включены только конструктивные элементы и абстракции математических операторов, обеспечивает выполнение принципа о времени жизни элементов технической системы. Все элементы структуры имеют то же самое время жизни, что и сама структура. Также и в технических системах – все элементы функционируют только в ее составе. Они имеют возможность существовать отдельно, вне контекста технической системы, но при этом они не функционируют. Аналогично тому, как функционирующие элементы не могут существовать вне контекста технической системы, так и свойства этих элементов не могут существовать отдельно от элементов. С другой стороны, элементы технической системы всегда обладают набором свойств. Всегда время жизни свойств элементов и самих элементов равно. Ниже, на рис. 4.2, показано распределение данных, представляющих свойства элементов, и вычислений между элементами программной системы.

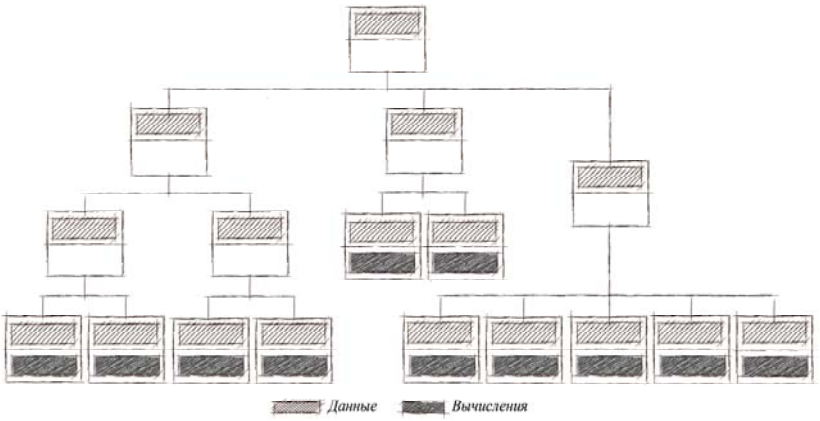


Рис. 4.2. Распределение вычислений и данных в программном представлении структуры технической системы

Объединение в иерархической структуре элементов технической системы и их состояний в целостностное представление множеств состояний для каждого элемента позволяет однозначно моделировать структуру технической системы с возможностью сохранения и изменения множества ее состояний. На уровне структуры определяются и осуществляются основные функции контроля за временем жизни объектов. Такое представление технической системы в полной мере соответствует формам структурных и конструктивных отношений ее элементов.

Полученное представление технической системы в виде древовидной иерархической структуры является основой для структуры программного обеспечения, моделирующего работу технической системы. Оно обеспечивает возможность построения программных абстракций технических систем, моделирующих их структуру, является основой для обеспечения возможности моделирования одновременно протекающих взаимосвязанных процессов и обеспечивает одновременную работу оборудования.

Элементы технической системы рассматриваются с точки зрения множества их состояний и поведения. Поэтому как структура программной абстракции технической системы, так и ее отдельные составные части, могут быть рассмотрены с точки зрения методов объектно-ориентированного проектирования. Связность, зацепление, достаточность, полнота и примитивность являются основными понятиями, характеризующими эффективность реализации структуры программных абстракций.

Центральным понятием теории объектно-ориентированного проектирования является объект, размещенный в памяти экземпляра абстрактного типа данных, описывающего возможные состояния и функции соответствующей конструктивной единицы технического комплекса. Объекты взаимодействуют посредством передачи сообщений, которые формируются вызовами функций объекта – получателя сообщения. Время, когда объекты не обрабатывают сообщения, они пассивно располагаются в оперативной памяти, не занимая при этом вычислительных ресурсов.

Представление технической системы в виде древовидной иерархической структуры, элементы которой функционируют на основе децентрализованной модели управления, предполагает использование двух типов отношений – связи и ассоциация. Для представления между элементами используются связи (рис. 4.3). Посредством связей одна абстракция запрашивает действие у другой и получает его результат.

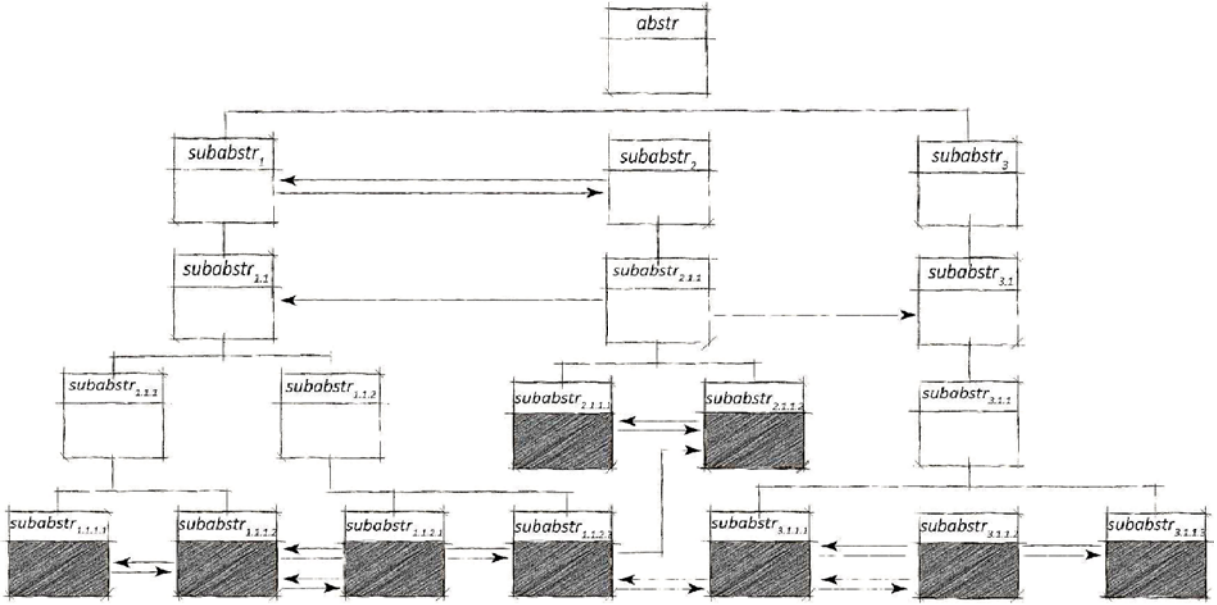


Рис. 4.3. Структурные отношения между элементами программной системы

В программной системе связи обозначают равноправные отношения между абстракциями.

Обычно связи возникают между абстракциями одного уровня. Связи могут быть как одно-, так и двусторонними. Односторонние связи используются, когда необходимо представить воздействие одного элемента технической системы на другой. Например, это может быть воздействие горячего теплоносителя на стенки канала. Двусторонние связи обычно используются, когда необходимо представить взаимодействие элементов технической системы.

В зависимости от конструкции технической системы могут возникать связи между уровнями. В зависимости от конструкции технической системы, они не обязательно могут существовать между элементами, имеющими одного родителя, также допускается возникновение связей через уровни. Связи определяются взаимной видимостью абстракций в программной системе, которая в свою очередь определяется конструкцией технической системы, структурными отношениями ее элементов и связями, через которые осуществляется их взаимное влияние.

Например, в теплообменном аппарате могут располагаться системы трубок, теплоносители. На рисунке представлен объект, представляющий элементарный участок теплообмена в кожухотрубчатом теплообменнике. Фрагменты потоков теплоносителей находятся внутри теплообменника. Поэтому объект, который представляет в программной системе фрагмент теплообменника, включает в себя два атрибута, которые представляют объекты теплоносителей. В данном случае объект ячейки теплообмена – целое, а теплоносители – его части. Другими словами, теплоноситель – часть состояния теплообменной ячейки. Исходя из ячейки теплообмена, можно найти один или другой теплоноситель. Однако по теплоносителю нельзя найти содержащий его объект (называемый также его контейнером), если только сведения о нем не являются частью состояния теплоносителя.

Иерархическая структурная модель определяет видимость только подчиненных объектов. Объекты нижнего уровня видны объектам верхнего уровня. При этом видимость объектов через уровни иерархии не обеспечивается.

Видимость элементов программной системы определяется взаимным влиянием элементов технической системы, которое в свою очередь регламентируется моделью взаимодействия элементов технической системы. Поэтому видимость объектов может быть абсолютно различной и выходить за рамки, предоставляемые иерархической структурой.

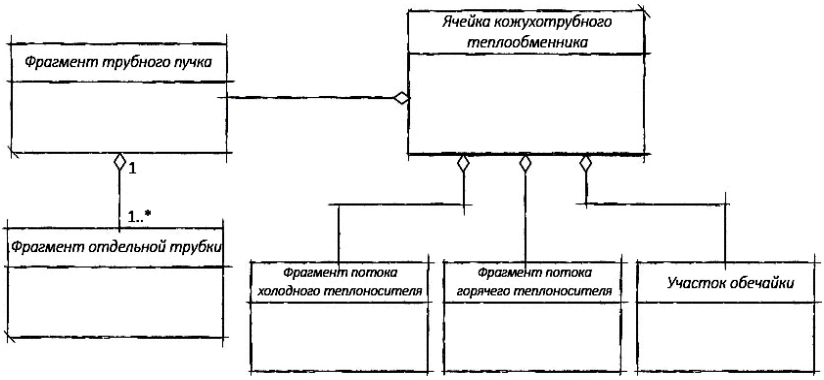


Рис. 4.4. Пример агрегации объектов

Для обеспечения видимости объектов конструкция каждой абстракции предусматривает ссылки на то множество объектов, на которое она в соответствии с моделью взаимодействия и функционирования элементов технической системы оказывает влияние. Это может быть прямая ссылка на абстракцию, расположенную в памяти, адрес или дескриптор объекта, для того, чтобы иметь возможность отсылать к нему сообщения.

Моделирование взаимной видимости подсистем технического комплекса на основе анализа их взаимного влияния позволяет минимизировать количество связей между объектами программной системы. Это в свою очередь упрощает ее структуру и соответственно уменьшает вычислительную нагрузку и трафик за счет уменьшения количества обрабатываемых сообщений.

4.1. ПРЕДСТАВЛЕНИЕ ПРОГРАММНЫХ АБСТРАКЦИЙ ЭЛЕМЕНТОВ ТЕХНИЧЕСКИХ СИСТЕМ НА СТРУКТУРНОМ УРОВНЕ

Каждая абстракция конструируется на основе объектно-ориентированных методов. Она является размещенным в оперативной памяти экземпляром определенного разработчиком абстрактного типа данных. Согласно терминологии объектно-ориентированного проектирования абстрактные типы данных представляют классы абстракций. Они определяют поведение и содержание абстракций, их реакцию на внешние сообщения и воздействия, которые они могут оказывать на

другие объекты. Основная нагрузка на формирование функциональности абстракции, ее места и роли в программной системе, возможного множества ее состояний приходится на описание типа данных, экземпляром которого абстракция является.

Ниже в рамках главы рассмотрены механизмы получения функциональности абстракций, соответствующей структурному уровню. Для представления отношений классов и абстракций в этой главе использована нотация UML.

4.2. СВЯЗНОСТЬ ПРОГРАММНЫХ АБСТРАКЦИЙ ТЕХНИЧЕСКИХ СИСТЕМ

Построение программных элементов как абстракций элементов технической системы позволяет повысить степень того, насколько отдельная абстракция образует логически законченную программную единицу (связность). Предельным видом связности принимается связность элемента технической системы.

Связность программных элементов необходимо максимально повышать. При построении их как абстракций технической системы это делать проще, так как в этом случае перед глазами разработчика присутствует образец в виде элемента технической системы, который необходимо повторить. Это осуществляется посредством определения наборов, структур данных и методов, оперирующих ими, таким образом, чтобы поведение состояний элемента в программной системе и возможности представления множества его состояний максимально соответствовали поведению соответствующего ему физического элемента в технической системе.

Методами объектно-ориентированного программирования определяется несколько видов связности – по совмещению, логическая, временная, коммуникационная, последовательная, функциональная, связность данных. Для реализации задач моделирования на основе аналитических решений не все из перечисленных видов связности можно использовать.

Связность данных возникает, когда процедуры модуля обращаются к одним и тем же данным модуля. Это показывает то, что все процедуры направлены на решение общих проблем и все процедуры реализуют поведение модуля, в результате которого изменяется его состояние. Этот вид связности полностью соответствует специфике абстракций элементов технической системы – все действия над элементом технической системы приводят к изменению его состояния.

Функциональная связность и связность данных желательны – они позволяют отразить специфику реализации задач моделирования. При их наличии все элементы абстракции связаны выполнением единой задачи. Задача в данном случае может быть сформулирована как расчет законов изменения состояний элементов технической системы во времени и пространстве. Для решения этой задачи используются несколько функций, размещенных внутри программного элемента. Работа каждой их этих функций направлена на выполнение действий с данными, представляющими состояние программного элемента. Можно сказать, что в контексте реализации задач моделирования связность по данным и функциональная связность являются двумя аспектами одного вида связности.

Эти два типа связности достаточны для того, чтобы реализовать задачи моделирования. Следующие типы связности могут использоваться, но очень редко и в специфических случаях.

Последовательная связность возникает, если элементы модуля должны активизироваться в определенном порядке. Эта связность часто является следствием попытки избежать зацепления по управлению. Обычно такая связность может быть исключена, если поднять уровень абстрагирования. Для задач моделирования такой тип связности может встречаться редко, но если он встречается, то это, значит, представление программы выходит за рамки физического представления и надо попытаться его заменить на связность по данным или функциональную.

Коммуникационная связность возникает, когда элементы (или функции объекта) объединены в модуль, поскольку они имеют доступ к одним и тем же устройствам ввода/вывода. Она может встречаться не в самих задачах моделирования, а в задачах, которые используют их результаты. Это может быть реалистичное графическое представление результатов моделирования. Например, данные в эту подсистему поступают в канал или через сокет, например. Но несмотря на то, что предпосылки могут возникать, этого типа связности необходимо избегать и механизмы доступа реализовывать в программных элементах, имеющих аналог в технической системе и отражающих физические сущности моделируемой системы. В этом случае сохраняется свойство программной системы быть структурным аналогом технической системы. Следствием этого является:

– исключение дополнительных сервисных программных элементов, обслуживающих каналы передачи данных и транзакции;

– исключение необходимости искусственно менять расписание работы программной системы так, чтобы учесть в нем циклы работы выделенного модуля;

– исключение дополнительных коммуникационных связей;

– появление возможности учитывать особенности трансляции данных с учетом их специфики в конкретном программном объекте.

Логическая связность, возникает, когда имеется связь между элементами модуля, но нет фактического соединения ни по данным, ни по управлению. Примером ее является библиотека математических функций, где каждая функция реализована отдельно, без связи с другими.

Временная связность возникает, когда элементы объединяются вместе, так как все они должны использоваться примерно одновременно. Модуль инициализации программы является типичным примером.

Связность по совмещению означает, что элементы модуля группируются без видимой причины, часто как результат произвольного «разбиения на модули» большой программы.

При построении программной системы как структурного аналога технической системы последние три вида связности должны быть исключены. В том случае, если они появляются, это необходимо рассматривать как сигнал к тому, чтобы выполнить декомпозицию еще раз, и избежать их.

Следствием связности являются еще две характеристики структуры программной абстракции – достаточность и полнота.

Достаточность характеризует наличие в программном элементе всего необходимого для реализации логичного и эффективного поведения, соответствующего физической сущности его аналога. Предельным уровнем достаточности, аналогично тому, как он определялся для связности, является достаточность физического элемента технической системы. Каждый из них обладает полным набором свойств и возможностей функционировать в рамках технической системы, изменять свое состояние под воздействием других ее элементов и воздействовать на них. Нарушение требования достаточности обнаруживается очень быстро – как только создается клиент, использующий абстракцию.

Под полнотой подразумевается наличие в интерфейсной части класса всех характеристик абстракции. Идея достаточности предъявляет к интерфейсу минимальные требования, а идея полноты охватывает все аспекты программного элемента как абстракции элемента технической системы.

4.3. ЗАЦЕПЛЕНИЕ ПРОГРАММНЫХ АБСТРАКЦИЙ ТЕХНИЧЕСКИХ СИСТЕМ

Зацепление характеризует взаимозависимость компонентов программы. Можно выделить несколько основных видов зацепления между программными элементами – зацепление по внутренним данным, по глобальным данным, при управлении, по параметрам. Несмотря на то, что зацепление модулей минимизируется, полностью от него избавиться невозможно и оно всегда присутствует в программной системе. В программных системах, реализующих задачи моделирования, целесообразно использовать два вида зацепления – по параметрам и по глобальным данным.

Зацепление по параметрам происходит, когда один программный элемент пользуется услугами и процедурами другого элемента и единственная связь между ними осуществляется через входные и выходные параметры. Этот вид зацепления может быть использован для описания всех видов взаимодействия моделируемых процессов. В качестве примера можно рассмотреть случай, когда теплоноситель прогревает стенку. Элемент, представляющий в программной системе теплоноситель, вызывает процедуру элемента, который представляет стенку. В вызове метода в качестве параметра передается количество теплоты, которое передано стенке с учетом температуры теплоносителя, коэффициента теплоотдачи, теплофизических свойств теплоносителя. Процедура осуществляет расчет температуры стенки, с учетом ее теплофизических свойств и тепловых потоков, направленных от стенки, к объектам, с ней соприкасающимся. Элемент, представляющий теплоноситель не располагает информацией о состоянии элемента, описывающего свойства стенки, и наоборот. Обмен информацией осуществляется через процедуру, представляющую в программной системе их взаимодействие. Это полностью соответствует картине физического процесса – когда теплоноситель и стенка существуют автономно и независимо друг от друга, а их состояние изменяется вследствие взаимодействия.

Показанный на примере процесса теплообмена вариант зацепления элементов через параметры может быть реализован для других процессов. Этот вариант зацепления позволяет четко описать в программной системе механизмы взаимодействия между элементами технического комплекса, обеспечить минимизацию дополнительных (например, связанных с решением математических задач моделирования) связей между модулями, четко формализовать вид и структуру передаваемой между модулями информации.

Альтернативным вариантом решения рассмотренного примера с теплоносителем может быть, если ее решить через зацепление по внутренним данным. Оно происходит, когда один из элементов изменяет локальные данные в другом элементе. Эта деятельность противоречит всякому физическому смыслу. То есть в этом случае, например, теплоноситель может изменять напрямую, а не через взаимодействие температуру стенки, а также обладает информацией о соприкасающихся элементах. Следствием того, что использование этого метода противоречит физической картине взаимодействия процессов, является то, что он затрудняет понимание и осознание смысла программы, и ее следует избегать, где только возможно. В том случае, если такое зацепление возникает между модулями в процессе проектирования программной системы, необходимо вернуться на предыдущий шаг и выполнить декомпозирование еще раз. При проектировании программной системы на основе модели взаимодействия процессов появление такого вида зацепления исключается.

Зацепление по глобальным данным может быть использовано для того, чтобы отразить свойства иерархически организованных процессов, когда процессы на нижних уровнях иерархии имеют доступ к информации процессов верхних уровней иерархии.

На практике различают два вида глобальных переменных – когда глобальные данные видны всем модулям программной системы или когда они ограничены областью видимости. В первом случае в качестве глобальных данных могут выступать условия окружающей среды по отношению в моделируемой системе процессов, температура, влажность, физические и теплофизические характеристики среды. В этом случае моделируется тот факт, что все элементы или процессы имеют контакт со средой и имеют доступ к ее характеристикам. Во втором случае – когда переменные ограничены областью видимости, он является частным случаем первого. Область видимости ограничивается обычно модулем. Здесь в качестве примера можно привести обечайку аппарата, которая собрана из нескольких конструктивных элементов. В модуль обечайки входят другие подчиненные конструктивные элементы, выполненные из того же материала. Для подчиненных элементов данные о материале не сохраняются. Это приводит к дублированию данных и они сохраняются один раз в модуле, описывающем всю обечайку, в которую также входят другие конструктивные элементы. В этом случае модули, описывающие конструктивные элементы обечайки, имеют доступ к ее глобальным данным.

Зацепления по параметрам и по глобальным данным являются типичными для задач моделирования. Их появление не является сигналом к тому, что проектирование системы идет в неправильном направлении. Такие виды зацепления упрощают понимание программы, обеспечивают статическую проверку типов на этапе разработки, разгружая тем самым этап выполнения программы.

Зацепление по внутренним данным и зацепление по управлению только нагромождают программную систему, противоречат физической картине процесса, что является причиной возникновения дополнительных взаимозависимостей модулей, не всегда очевидных.

Зацепление по управлению происходит, когда один модуль должен выполнить некие операции в порядке, который определяется другим модулем. Наличие зацепления по управлению показывает, что модуль выполнен на более низком уровне абстрагирования, чем требуется. Это может дать толчок к тому, чтобы на этапе проектирования повторить итерацию по декомпозиции всей программной системы. Поэтому всегда при декомпозиции с учетом модели взаимодействия элементов технического комплекса необходимо опираться на то, чтобы не встречались зацепления по управлению.

В общем случае появление двух последних видов зацепления является сигналом того, что программа должна быть переделана и декомпозиция выполнена еще раз.

Таким образом, только два вида могут быть использованы для построения программных систем моделирования. Их по возможности тоже необходимо стараться минимизировать. Например, количество параметров, которые передаются в процедуру, тоже не может быть слишком большим. Некоторые источники рекомендуют ограничивать количество передаваемых параметров 5. В задачах моделирования это не всегда возможно с точки зрения представления взаимосвязей моделируемых процессов. Формальный подход к уменьшению параметров – объединение их в структуры по какому-либо признаку или вообще без признака является в данном случае источником ошибок и неоднозначностей. Таких ситуаций следует однозначно избегать. Самым правильном является ход, когда при большом количестве параметров, передаваемых в процедуру, выполняется анализ – почему это происходит в сторону функций, которые реализует вызываемая процедура. Может быть, ее стоит разбить на две или наоборот объединить в себе несколько процедур. При этом свободу творчества необходимо ограничивать принципом, исключающим зацепление по управлению.

4.4. ВЗАИМОСВЯЗЬ СВЯЗНОСТИ И ЗАЦЕПЛЕНИЯ ПРОГРАММНЫХ АБСТРАКЦИЙ ТЕХНИЧЕСКИХ СИСТЕМ

Связность программных элементов находится в обратнопропорциональной связи между их зацеплением. Чем выше связность элемента, тем в большей степени он автономен при решении своих задач, и, как следствие этого, он меньше зацеплен с окружающими абстракциями. Элементы должны быть зацеплены минимально. Предельный случай зацепления программных элементов определяется зацеплением элементов технической системы.

Каждый программный элемент должен иметь конкретную цель, а его методы должны обеспечивать ее достижение. В этом случае элемент считается связным. Элементы с высокой степенью связности, взаимодействуя между собой, не требуют выполнения дополнительных вычислений. Повышение связности модулей снижает их зацепление, и наоборот.

Использование структурного соответствия программной и технической систем обеспечивает четкое выделение цели работы модуля и однозначно определяет его роль в программной системе. Основной целью программных элементов является расчет характеристик процессов, протекающих в физическом комплексе. Поэтому для каждого элемента цель формулируется в виде набора процессов, за расчёт характеристик которых он отвечает. Также использование модели взаимодействия позволяет однозначно определить формы взаимодействия с другими элементами, требования к точности расчетов и времени, за которое должны быть получены результаты. Вокруг цели программного элемента объединяются структуры данных, представляющие характеристики процессов и исходные данные для расчета, процедуры, выполняющие расчеты. Таким образом, обеспечивается построение модуля с высокой степенью внутренней связности.

4.5. ВЗАИМОСВЯЗЬ ПРОГРАММНЫХ АБСТРАКЦИЙ И АБСТРАКТНЫХ ТИПОВ ДАННЫХ

Каждая абстракция является размещенным в оперативной памяти экземпляром абстрактного типа данных. Опираясь на терминологию объектно-ориентированного проектирования, которая используется для представления структурного моделирования, абстрактные типы данных представляют классы объектов. В большинстве практических случаев классы статичны, т.е. все их особенности и содержание определяются

на этапе разработки программы Объекты, напротив, в процессе выполнения программы многократно создаются и уничтожаются.

Одним из способов моделирования материальных потоков в химико-технологических схемах или в аппаратах является представление их в виде последовательного набора порций, где каждая порция характеризуется неизменными по длине свойствами. Классы, которые описывают свойства порций потока, создаются на этапе реализации модели (разработки программы). Порции потоков, в которых рассматриваются различные процессы, описываются отдельными классами. Например, порция потока, движущегося в теплообменном аппарате, описывается классом *A*, порция потока, находящегося в массообменном аппарате, – классом *B*. Трактовка этих классов по самому их определению статична. Объекты этих классов динамичны – они постоянно создаются, удаляются и изменяются. Поступление или выход потока в технологическую схему или в аппарат характеризуется непрерывным во времени поступлением новых или удалением порций. В программной системе это моделируется генерацией или удалением новых объектов, описывающих порции потока. При переменных расходах количество генерируемых или удаляемых в единицу времени объектов также меняется. Моделирование устройств, которые разветвляют или объединяют потоки, также связано с генерацией или удалением новых объектов. Если в предыдущем случае количество объектов в системе оставалось неизменным, то при моделировании разветвлений или объединений потоков общее количество объектов в системе может многократно увеличиваться или уменьшаться.

4.6. ИСПОЛЬЗОВАНИЕ ОТНОШЕНИЙ ТИПОВ ДАННЫХ ДЛЯ ОПИСАНИЯ ЭЛЕМЕНТОВ ТЕХНИЧЕСКИХ СИСТЕМ

Существует несколько общих методов построения абстрактных типов данных – ассоциация, наследование, агрегация, использование, инстанцирование, метакласс. Этих способов достаточно для того, чтобы описать в их отношениях конструктивные особенности технической системы и структуру протекающих в ней процессов.

4.6.1. АССОЦИАЦИЯ

Ассоциация описывает отношения целого и части, приводящие к соответствующей иерархии объектов, так, что от целого (агрегата) можно прийти к его частям (атрибутам). Ассоциация по умолчанию

не объясняет как типы общаются друг с другом, указывая только семантическую зависимость и какие роли классы играют друг для друга. Ассоциация определяет только участников, их роли и мощность отношений между ними. Ассоциация означает физическое вхождение одного объекта в другой – например в теплообменном аппарате могут располагаться системы трубок, теплоносители. Ассоциация по умолчанию не объясняет как типы общаются друг с другом, указывая только семантическую зависимость и какие роли классы играют друг для друга. Ассоциация определяет только участников, их роли и мощность отношений между ними.

4.6.2. НАСЛЕДОВАНИЕ

Под наследованием понимается возможность доступа представителей дочернего класса (подкласса) к данным и методам родительского класса. В качестве примера можно рассмотреть класс теплоносителей. Теплоносители характеризуются определенными свойствами – теплоемкость, теплопроводность, и определенными функциями – расчет коэффициентов теплоотдачи, расчет своих теплофизических свойств в зависимости от текущей температуры. Эти функции и свойства не являются уникальными, для одного выделенного теплоносителя они общие – для воздуха, воды и т.д., когда они используются в качестве теплоносителей. Таким образом, связываются определенные свойства и поведение с общей категорией теплоноситель, и вода, воздух, глицерин, используемые в качестве теплоносителей, являются частным случаем, поведение для данного подкласса определяется автоматически.

Наследование транзитивно, так что класс может наследовать черты надклассов, отстоящих от него на несколько уровней. Например, если класс элементарного участка теплообмена в кожухотрубчатых теплообменных аппаратах является подклассом элементарного участка теплообменного аппарата, а тот в свою очередь является подклассом элементарного участка аппарата химической технологии, то класс участка теплообмена наследует в себе свойства как элементарного участка теплообменного аппарата, так и элементарного участка аппарата химической технологии. Эти утверждения не всегда верны, но они хорошо иллюстрируют идеализированный подход к наследованию, который формализован в принципе подстановки. Согласно ему, если есть два класса A и B, такие, что класс B является подклассом класса A (возможно, находясь в иерархии на несколько ступеней), то мы должны

иметь возможность подставить представителя класса B вместо представителя класса A в любой ситуации, причем без видимого эффекта.

Применение принципа подстановки наглядно можно проиллюстрировать на следующем примере. Рассмотрим кожухотрубчатый теплообменный аппарат, в котором процесс теплообмена сопровождается фазовыми превращениями. Поступающий в аппарат теплоноситель имеет температуру выше точки росы, он сначала охлаждается, а потом из него конденсируется влага.

При моделировании аппарат рассматривается как совокупность элементарных участков теплообмена и конденсации. В программной системе теплообменный аппарат представлен массивом объектов, каждый из которых представляет ячейки теплообмена или конденсации. Их абстракции получены в результате наследования от абстракции, описывающей элементарный участок области процесса. Она содержит в себе информацию о длине участка и геометрических параметрах аппарата. При наследовании в абстракции ячеек теплообмена (HeatCell) и конденсации (CondCell) добавляются данные и функции, позволяющие рассчитать параметры процессов. Расчет состояния аппарата и теплоносителей осуществляется в результате последовательного расчета всех ячеек. Переход к следующей ячейке осуществляется за счет того, что объект текущей ячейки, состояние которой рассчитано, вызывает метод (функцию), который инициирует расчет состояния следующей ячейки. Массив не располагает информацией о том, чем является следующая ячейка – участком теплообмена или конденсации (рис. 4.5).

Он работает с ней как с элементарным участком и не имеет информации о том, какому типу и какой из ячеек теплообмена или конденсации он посылает команду. Для того чтобы гарантировать, что вызывая метод для инициализации расчета, объект получит нужный результат, этот метод определяется в базовом классе.

Реализация этого метода учитывает особенности процессов конденсации и теплообмена, поэтому она выполнена уже в подклассах. Во время выполнения расчета объект ячейки, инициируя расчет следующего объекта, оперирует значениями, которые фактически представляют собой объекты порожденных подклассов. В одном случае сообщение отсылается ячейке теплообмена, в другом – конденсации. Для того чтобы эта процедура работала верно, необходимо, чтобы функциональные возможности каждого из подклассов соответствовали ожидаемым функциональным обязанностям, определенным для родительского класса; т.е. здесь подклассы должны быть также и подтипами.

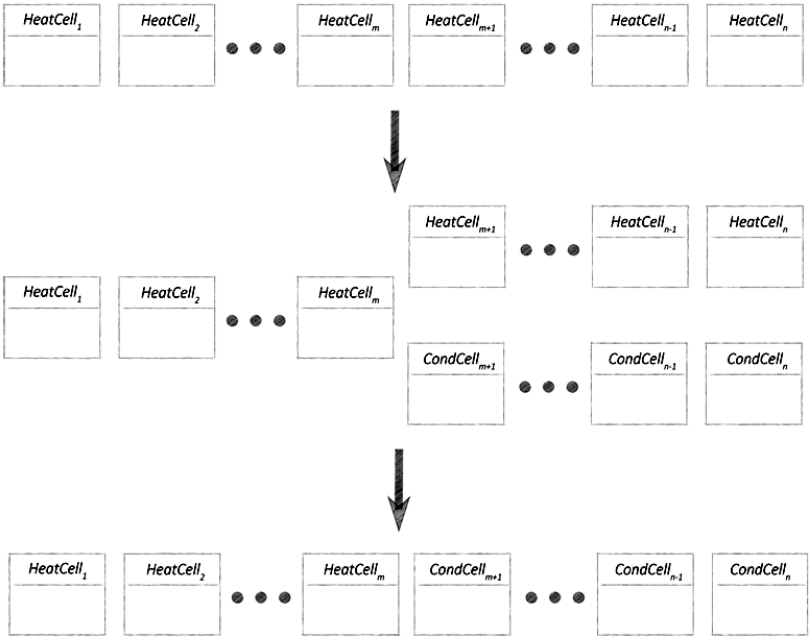


Рис. 4.5. Пример организации и взаимодействия объектов для расчета состояния кожухотрубчатого теплообменного аппарата

Ниже рассмотрены их особенности применительно к программным системам моделирования, в основе которых лежат аналитические решения задач математической физики.

4.6.2.1. Порождение подклассов для специализации (порождение подтипов)

При порождении подкласса для специализации новый класс является более конкретной, частной формой родительского класса, но удовлетворяет спецификациям родителя во всех существенных моментах. Для этой формы полностью выполняется принцип подстановки. Вместе со следующей категорией (наследование для спецификации) специализация является наиболее идеальной формой наследования, к которой должна стремиться хорошая программа.

Класс (*Apparat*) предоставляет общие операции с аппаратами химической технологии – определение положения в пространстве, опре-

деление температуры окружающей среды. Специализированный подкласс массообменных аппаратов (*MassApparat*) наследует общие операции с аппаратами химической технологии и дополнительно обеспечивает средства для расчета концентраций целевых компонентов (рис. 4.6).

Класс массообменных аппаратов удовлетворяет всем свойствам, ожидаемым от аппаратов в общем виде, и, следовательно, является подтипом класса Apparat в дополнение к тому, что он является его подклассом.

Нестационарный профиль температуры в плоской стенке представляется в виде суммы решений стационарной и нестационарной задач. С точки зрения математики это операторы с набором определенных свойств. Их свойства определяются соответствующими классами (рис. 4.7). Например, стационарная составляющая, представляемая полиномом, определена в классе A, нестационарная, в виде суммы членов ряда – в классе B. Стационарный профиль температур представляется в виде суммы гиперболических косинусов и синусов (класс C). Нестационарный профиль температур по длине области процесса представляется в виде произведения экспоненциальных функций (класс D). Соответственно и классы A, B, C, реализуют различные операции. Все эти классы могут быть объединены в одну иерархию, описывающую математические операторы.

Профили температур являются одной из характеристик конструктивных единиц оборудования. В рассматриваемом примере в первых двух случаях профили температур характеризуют состояние стенки, разделяющей потоки теплоносителей. В третьем случае профиль температур характеризует состояние области течения процесса. Для того чтобы реализовать классы различных профилей температур в стенках и в области течения процесса, необходимо учитывать как свойства математических операторов, представленных в классах A, B, C, D, так и характеристики элементов оборудования. Элементы оборудования

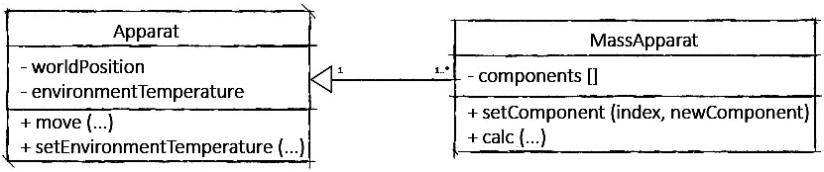


**Рис. 4.6. Получение класса,
представляющего характеристики массообменных аппаратов**

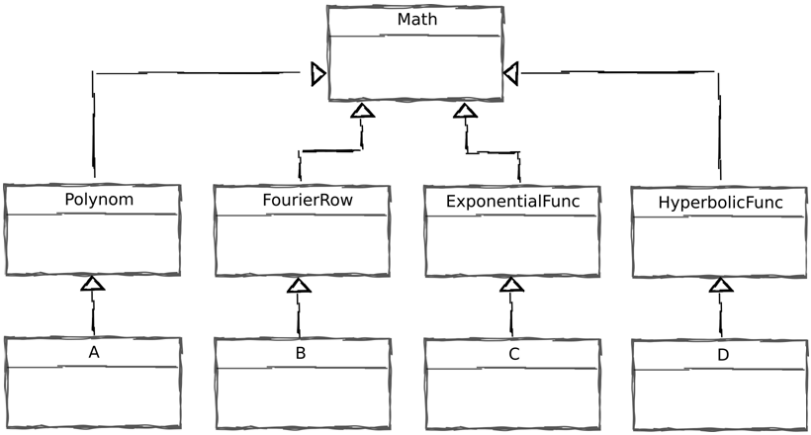


Рис. 4.7. Иерархия наследования математических операторов

также представляются классами. Ввиду того, что эти классы не описывают свойства математических операторов, их нельзя отнести к иерархии, представленной на рис. 4.7. Соответственно, эти классы образуют новые иерархии (рис. 4.8) – одна, которая описывает свойства систем на основе аналитических решений канонических задач, вторая – которая описывает свойства областей процесса.

4.6.2.2. Порождение подкласса для спецификации

Родительский класс описывает поведение, которое реализуется в дочернем классе, но оставлено нереализованным в родительском. Наследование можно использовать в случае, если необходимо гарантировать поддержку классами определенного общего интерфейса, т.е. реализацию ими одних и тех же методов. Родительский класс может комбинировать реализованные операции и объявление действий, осуществление которых выполняется дочерним классом. Отличия в интерфейсе между родительским и дочерним классами в данном случае отсутствуют – последний просто обеспечивает выполнение описанного, но не реализованного в родительском классе поведения. Это можно рассматривать как случай порождения специализирующего подкласса, в котором подклассы являются неусовершенствованием существующего типа, реализацией неполной, абстрактной спецификации. В таких случаях родительский класс иногда называют абстрактно специфицированным классом.

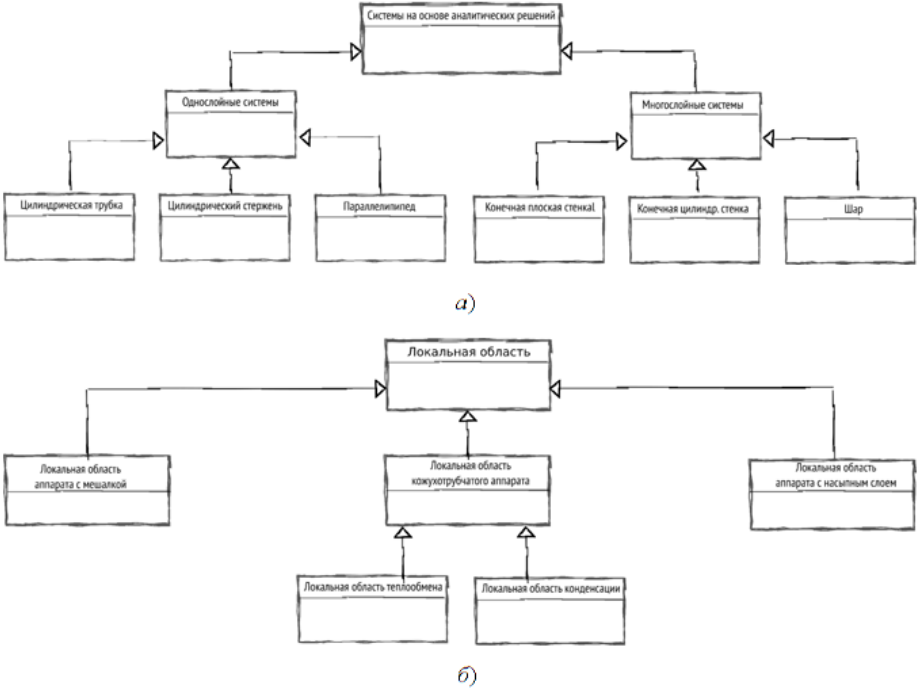


Рис. 4.8. Пример иерархии классов:
а – описывающей свойства систем на основе аналитических решений для тел канонической формы;
б – локальных областей процесса

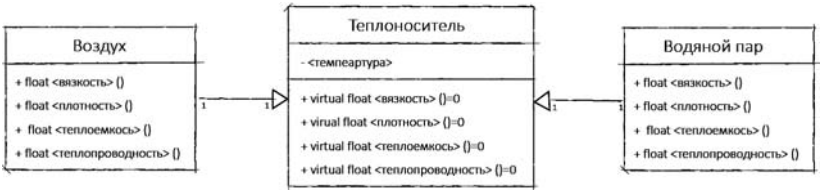


Рис. 4.9. Получение классов теплоносителей с помощью механизма наследования

В общем случае порождение подклассов для спецификации распознается по тому, что фактическое поведение не определено в родительском классе – оно только описано и будет реализовано в дочернем классе.

Такой вариант наследования может быть использован при порождении классов теплоносителей (наборов классов) (рис. 4.9).

Таким образом, каждый теплоноситель предоставляет функции расчета таких своих параметров, как плотности, теплопроводности, вязкости, теплоемкости от температуры. Для разных теплоносителей это будут разные зависимости. В базовом классе определены отложенные методы, которые в дочерних классах реализуются в зависимости от того, свойства какого теплоносителя описывает подкласс. Использование наследование для спецификации позволяет еще на этапе кодирования и компилирования программы выявить ошибки вызова функций объектов класса.

4.6.2.3. Порождение подкласса для обобщения

Наследование при порождении подклассов для обобщения расширяет родительский класс для создания объекта более общего типа. Дочерний класс модифицирует или переопределяет некоторые методы родительского класса в целях получения объекта более общей категории. Наследование для обобщения может применяться, когда построение программной системы выполняется на основе существующих классов, которые мы не хотим или не можем изменять. Порождение подкласса для обобщения характеризуется тем, что оно основывается в первую очередь на значениях данных и в меньшей степени на функциональном поведении.

В качестве примера можно рассмотреть создания абстракции аппарата. Например, определен класс теплообменных аппаратов (Heat-

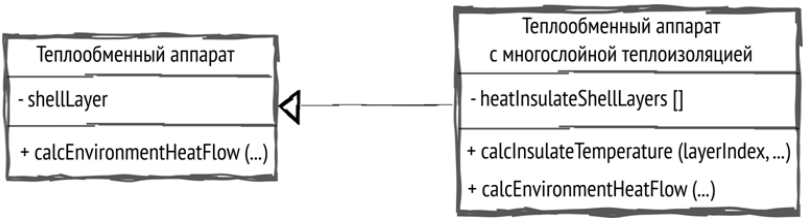


Рис. 4.10. Порождение класса с теплоизолированной обечайкой

Apparat) и необходимо создать тип аппаратов с дополнительным слоем внешней теплоизоляции. Дочерний класс будет отличаться наличием дополнительных полей переменных, описывающих характеристики дополнительного слоя теплоизоляции (рис. 4.10). Также будет переопределена наследуемая процедура расчета тепловых потерь через обечайку аппарата.

В общем случае рекомендуется избегать порождения подкласса для обобщения, используя вместо этого перевернутую иерархию типов и наследование для специализации. При этом для программных систем моделирования на основе аналитических решений существует ряд случаев, когда наследование для обобщения очень удобно и органично. Например, есть класс, описывающий температурный профиль в однослойной стенке. От него порождается класс, который описывает температурный профиль в многослойной стенке. В новом классе добавляются переменные, которые описывают профиль температур в каждом слое. Новых функций в дочерний класс не добавляется, а реализация тех функций, которые отвечают за расчет профиля температур в слоях, переопределяется. Таким образом, в результате получается общий класс, который описывает профиль температур в многослойной стенке. Случай, когда стенка состоит из одного слоя, для порожденного класса является частным, в то время как для родительского класса он является основным и единственным.

4.6.2.4. Порождение подкласса для расширения

Дочерний класс добавляет новые функциональные возможности к родительскому классу, но не меняет наследуемое поведение. В отличие от порождения подкласса для обобщения, которое модифицирует существующие функциональные возможности объекта, порождение

для расширения добавляет классу совершенно новые свойства. Порождение для обобщения переопределяет по крайней мере один метод родителя, а функциональные возможности подкласса основываются на родительских. Расширение просто добавляет новые методы к родительским, в результате чего функциональные возможности подкласса расширяются и повторяют существующие методы родителя лишь частично.

В качестве примера порождения подкласса для расширения можно привести класс, описывающий область процесса как упорядоченный набор элементарных участков. Набор ячеек может быть реализован на основе стандартных классов вектора (*Vector*) или односвязного списка (*List*). В случае необходимости от них могут быть поражены подклассы, которые предоставляют дополнительные методы для операций с ячейками, например такие, как поиск ячейки в массиве, замена ячеек, несуществующие в базовом классе специфические методы перебора ячеек. Такие операции имеют смысл для подкласса, но не для родительского класса.

Поскольку функциональные возможности родителя остаются нетронутыми и доступными, порождение подкласса для расширения не противоречит принципу подстановки, и, следовательно, такие подклассы всегда будут подтипами.

4.6.2.5. Порождение подкласса для ограничения

Порождение для ограничения используется в случае, когда возможности подкласса должны быть ограничены по сравнению с родительским. Так же, как и при обобщении, необходимость использования порождения для ограничения чаще всего возникает, когда класс строится на основе существующей иерархии, которая не может быть изменена.

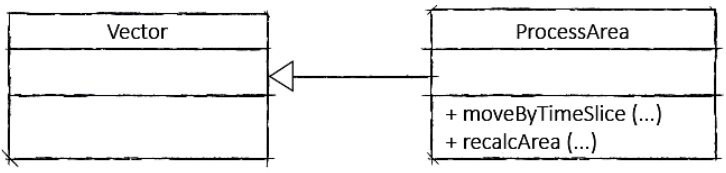


Рис. 4.11. Порождение класса области процесса, представленной набором ячеек

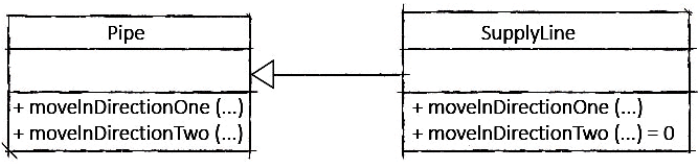


Рис. 4.12. Порождение класса линии подачи потоков в аппарат

В качестве примера можно привести класс, который представляет трубопровод, по которому движется жидкость или газ. Поток в нем представляется набором последовательно располагающихся элементарных объемов. Класс трубопровода обладает функциональностью, которая позволяет обеспечивать сдвиг ячеек в двух направлениях. Это обеспечивает моделирование движения потока. В том случае, если необходимо только одно направление, от класса порождается дочерний класс, где неиспользуемые методы заглушены или модифицированы так, чтобы при использовании они не приводили к ошибкам в работе программной системы. Здесь переопределение методов ограничивает возможности класса (рис. 4.12).

4.6.2.6. Порождение подкласса для варьирования

Порождение подкласса для варьирования применяется, когда два класса имеют сходную реализацию, но не имеют никакой видимой иерархической связи между абстрактными понятиями, ими представляемыми. При этой форме наследования дочерний и родительский классы являются вариациями на одну тему, и связь «класс–подкласс» произвольна.

Например, классы, описывающие свойства различных аппаратов, могут использовать одинаковые решения задач математической физики. При этом аппараты по своему назначению в технологической схеме используются различно и нет никаких оснований для того, чтобы классы этих аппаратов объединялись в одну иерархию.

Возникновение такого отношения классов является косвенным показателем того, они были рассмотрены не с той точки зрения и ее необходимо скорректировать. Если получившиеся отношения не корректировать, то с большой вероятностью это может привести к неоправданному увеличению связей между объектами и количества передаваемых между объектами сообщений, параллельному выполнению одинаковых операций различными объектами.

4.6.2.7. Порождение подкласса в целях конструирования

При наследовании для конструирования дочерний класс использует методы, предоставляемые родительским классом, но не является подтипом родительского класса. Принцип подстановки нарушается из-за реализации и переопределения методов родительского класса.

Часто класс наследует почти все функциональное поведение родительского класса, изменяя только имена методов или определенным образом модифицируя аргументы. Это может происходить даже в том случае, когда новому и родительскому классам не удается сохранить между собой отношение «быть экземпляром».

В качестве примера можно привести два различных случая. Наиболее близким к этому случаю является случай, когда программная система распределена по нескольким узлам вычислительной компьютерной сети. При обмене данными один программный элемент отсылает сообщение другому по сети. Родительский класс обеспечивает запись неструктурированных двоичных данных в порт или сокет. Подкласс строится для каждой структуры данных, которая будет передаваться. Он реализует процедуру передачи для определенного типа данных, которая использует методы родительского класса для непосредственного направления данных в сокет, а затем адресату.

Второй случай – когда родительский класс выполняет операцию расчета параметров процессов на элементарном участке. Мы порождаем другой класс, который описывает другой процесс на этом элементарном участке. Например, необходимо создать класс, который описывает процессы массопереноса на элементарном участке. Массообменные константы зависят от температуры. Поэтому целесообразно было бы сначала найти температуру на этом элементарном участке, затем определить актуальное состояние массообменных констант, после чего находить количество перенесенного вещества на выделенном участке. Здесь целесообразно, имея класс, описывающий процесс теплообмена на элементарном участке, наследовать и получить класс, описывающий процесс массообмена (рис. 4.13). В класс добавляются новые методы, необходимые для расчета массообменных процессов, а методы пересчета состояния элементарного участка переопределяются так, что метод родительского класса вызывается в дочернем, а затем выполняются другие действия.

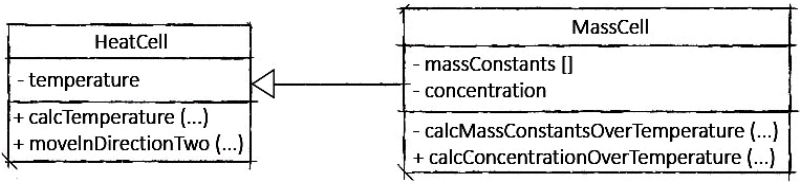


Рис. 4.13. Пример порождения класса локальной области массообмена от класса локальной области теплообмена

В обоих случаях дочерний класс не является более специализированной формой родительского класса, подстановка представителей дочернего класса туда, где используются представители родительского класса, заведомо приведет к ошибке.

В языках со статическими типами данных скептически относятся к порождению подклассов для конструирования, поскольку оно часто напрямую нарушает принцип подстановки (появляются подклассы, не являющиеся подтипами). С другой стороны, будучи быстрым и легким способом построения новых абстракций, оно широко используется в языках с динамическими типами данных. В библиотеке стандартных программ языка *Smalltalk* можно найти множество примеров порождения подклассов в целях конструирования.

4.6.2.8. Порождение подкласса для комбинирования (множественное наследование)

Выше были рассмотрены формы наследования от одного родителя. Основное преимущество одиночного наследования заключается в том, что функциональные возможности родителя наследуются всеми объектами, и при этом гарантируется, что каждый объект обладает необходимым уровнем функциональности.

Ситуация, в которой класс имеет только одного родителя, является типичной и используемой в большинстве случаев. Однако могут возникать ситуации, когда использование одиночного наследования может вызывать избыточные отношения классов и, как следствие, абстракций. При реализации нового класса, описывающего свойства стенки со стационарным профилем температур, невозможно удовлетворить всем условиям с помощью иерархии одиночного наследования (рис. 4.14). В этом случае целесообразно и уместно использовать множественное наследование.

Рис. 4.14. Пример иерархии классов, описывающих свойства математических операторов

Профиль температур в плоской стенке вычисляется как сумма стационарной составляющей, представляемой классом Polynom, и нестационарной, представляемой классом FourierRow, показанных на рис. 4.7. Поэтому класс WallTempProfile, представляющий температурный в плоской стенке, может быть унаследован от двух классов – Polynom и FourierRow.

4.6.2.9. Наследование через общих предков

Проблема возникает, если использовать два класса, имеющие общего родителя. Например, есть два класса, описывающие ячейку теплообмена (HeatCell) и конденсации (CondCell), которые унаследованы оба от одного общего класса, представляющего ячейку (Cell). Необходимо создать класс, который сможет представлять ячейку теплообмена и конденсации (HeatCondCell) в различные моменты времени.

Такой вариант целесообразно использовать в том случае, когда невозможно заранее знать, как ячейка будет работать, и информация о характере процессов, протекающих в ней, появляется в момент вызова метода, пересчитывающего его состояние.

В этом случае переименование методов позволяет принять решение по поводу организации функций, определенных одновременно в классах теплообменной (HeatCell) и конденсационной (CondCell) ячеек.

Сложность возникает в случае, когда от общего родителя наследуются еще свойства. Здесь для надежной работы методов, которые объявлены в классе родителя (Cell) и переопределены в дочерних (HeatCell и CondCell), необходимо обеспечить для каждого отдельные копии данных. При реализации класса HeatCondCell возникает необхо-

димость изменения кода этих методов для того, чтобы определить механизм использования копий данных, унаследованных от общего родителя. Для методов, которые присутствуют в родительском классе и не переопределяются в дочерних, этого не требуется. Например, метод, который определяет положение ячейки в пространстве, одинаково реализуется в базовом классе Cell, не переопределяется в классах HeatCell и CondCell, может быть без изменений реализован в классе HeatCondCell.

Этот пример был рассмотрен без привязки к языкам программирования и особенностям реализации компиляторов. Некоторые языки программирования прямо или косвенно предоставляют механизмы решения этой проблемы на уровне генерации кода.

4.6.3. АГРЕГАЦИЯ

Отношение агрегации между классами аналогично отношению агрегации между их экземплярами. Агрегация предполагает, что один класс полностью включается в другой как его составная часть. Если между объектами используется отношение «целое/часть», то их классы должны находиться между собой в отношении агрегации.

Такое соотношение используется для описания сложных подсистем, составных физических объектов и конструктивных единиц оборудования. Можно выделить два типа агрегации – включение по значению и по ссылке. С помощью этих способов агрегации можно моделировать широкий спектр объектов технической системы и их отношений.

Включение по значению используется тогда, когда один объект входит в другой как часть. При разворачивании этих отношений в виде графа мы получаем иерархическую структуру. Например, теплообменный аппарат состоит из обечайки, в которой расположены наборы трубок, и прикрепленных к ней опор. Аналогично физической модели, можно создать класс, представляющий теплообменный аппарат (HeatApparat), который будет в себя включать классы обечайки (Ring) и опор (Support). Класс обечайки будет в себя включать массив классов (Pipes), представляющих трубку (Pipe).

Агрегация по ссылке имеет место в тех случаях, когда один объект моделируемой системы входит в другой как часть, но при этом в контексте решаемой задачи может существовать отдельно от него. Например, объект создан в рамках процесса, отличного процесса,

в котором создается объект, содержащий его, или в нем протекают процессы, которые относятся не только к рассматриваемому объекту, этот объект является общим для двух и более различных объектов. В рассмотренном примере класс HeatApparat по-прежнему означает целое, но его часть, экземпляр класса Support, содержится косвенно. Теперь эти объекты существуют отдельно друг от друга – появилась возможность создавать и уничтожать экземпляры классов независимо. Использование такого отношения позволит динамически, на этапе выполнения программы (вычислительных процедур), моделировать различные варианты крепления теплообменного аппарата.

Чтобы избежать структурной зависимости через ссылки, необходимо придерживаться договоренности относительно создания и уничтожения объектов, ссылки на которые могут содержаться в разных местах. Агрегация является направленной, как и всякое отношение «целое/часть». Объект Support входит в объект HeatApparat, и не наоборот. Физическое вхождение одного в другое нельзя «зациклить», а указатели – можно за счет того, что каждый из двух объектов может содержать указатель на другой.

Наиболее распространенным случаем использования агрегации по ссылке является случай, когда необходимо смоделировать работу нескольких физических объектов, имеющих общие конструктивные элементы. Физически существует один экземпляр этого общего конструктивного элемента, но он входит в различные объекты и влияет на моделируемые процессы, происходящие в них. Экземпляр класса общего конструктивного элемента в программной системе создается отдельно, а экземпляры классов физических объектов, в которые он входит, включают его как ссылку. При этом моделируется сразу взаимосвязь аппаратов и появляется возможность избежать дублирования значений, которое возникает в данном случае при использовании физического включения. Оно потребовало бы создания одинаковых экземпляров класса общего конструктивного элемента для каждого экземпляра класса физического объекта, куда входит конструктивный элемент. При этом с большой вероятностью будут возникать сложности с изменением состояния для каждого объекта, входящего в другие как часть. Решение этих сложностей потребует дополнительных вычислительных ресурсов и не будет эффективным.

Использование вариантов агрегации по значению и по ссылке показано на рис. 4.15, на примере общего (без привязки к типу аппарата) представления структуры локальной области теплообмена.

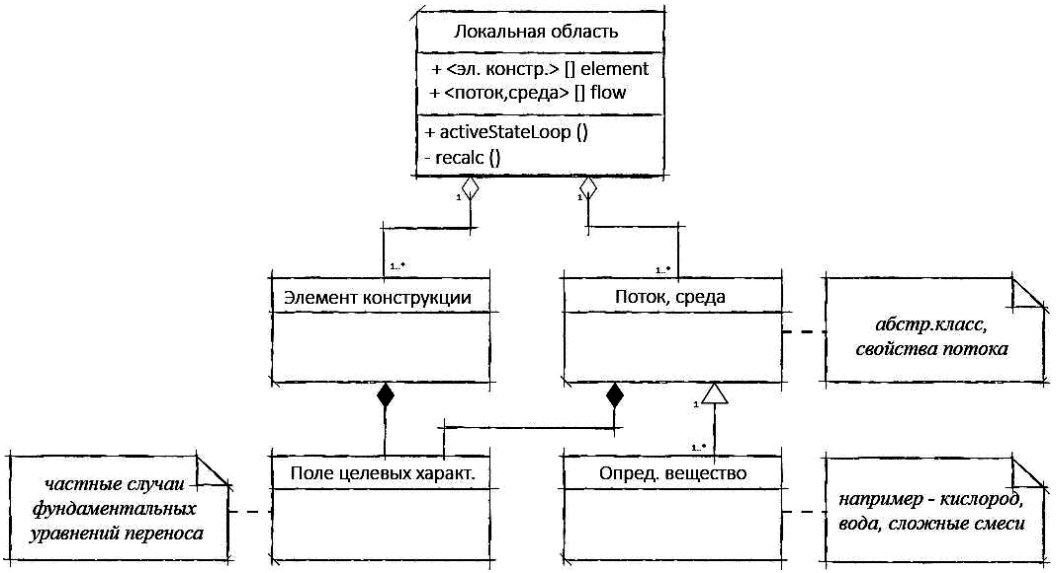


Рис. 4.15. Использование вариантов агрегации по значению и по ссылке для представления локальной области процесса теплообмена

Необходимо отметить, что далеко не все языки программирования предоставляют механизмы для такого гибкого управления памятью, которое позволяет связывать объекты по значению и по ссылке. Так, для широко используемой сейчас Java существует набор операций, которые выполняются для абстрактных типов данных только по ссылке. Выходом из положений в данном случае может быть разыменование переменных, чтобы подчеркнуть их физическую сущность. Но при этом моделировании на уровне конструкций языка программирования таких отношений не происходит.

Ассоциация часто может быть заменена циклической агрегацией или циклическими отношениями использования. Однако чаще ассоциация, которая по определению предполагает двунаправленность, во время проектирования превращается в простую агрегацию или отношение использования, выявляя ограничения на направление ассоциации.

4.6.4. ИСПОЛЬЗОВАНИЕ

Отношение использования между классами соответствует равноправной связи между их экземплярами. Это то, во что превращается ассоциация, если оказывается, что одна из ее сторон пользуется услугами другой. В большинстве случаев использование происходит через сигнатуры интерфейсных функций. Можно представить, что при пересчете состояния ячейки класс Cell использует в теле функции run вызов класса, содержащего в себе параметры окружающей среды. Отношения целого и части здесь не работают, так как этот объект класса, описывающего окружающую среду, не входит в объект Cell, а только используется. В типичном случае такое отношение использования проявляет себя, если в реализации какой-либо операции происходит объявление локального объекта используемого класса. Строгое отношение использования иногда несколько ограничительно, поскольку клиент имеет доступ только к открытой части интерфейса сервера.

4.6.5. ИНСТАНЦИРОВАНИЕ

Инстанцирование определяет создание класса, от которого потом создается объект. В практических целях широко используется инстанцирование параметризованных классов. Они представляют собой шаблон для построения других классов. Шаблон может быть параметризован другими классами, объектами или операциями. Параметризован-

ный класс должен быть инстанцирован перед созданием экземпляров. С помощью шаблонов можно применять один общий класс для формирования классов различных физических объектов. В данном случае физический объект рассматривается как совокупность ячеек. При передаче в качестве параметра в контейнер класса, представляющего теплообменную ячейку HeatCell, получается теплообменный аппарат. Если в качестве параметра указан класс ячейки массообменного аппарата MassCell, то получается массообменный аппарат. Таким образом, для ячеек различных типов может быть использован один контейнерный класс.

Различие между контейнерными классами может возникать только в случаях, когда отличаются способы организации ячеек. Для описания упорядоченного набора, линейно расположенного в пространстве, необходим один контейнер, для описания двух- или трехмерных наборов – свои контейнерные классы. Комбинация наследования и инстанцирования параметризованных классов позволяет собирать классы из различных ячеек, моделируя различные конструкции и положения физических объектов.

Параметризованный класс не может иметь экземпляров, пока он не будет инстанцирован. Можно рассмотреть два различных случая создания различных аппаратов – теплообменного (HeatApparat) и массообменного (MassApparat). Объекты HeatApparat и MassApparat – это экземпляры совершенно различных классов, которые даже не имеют общего суперкласса. Тем не менее они получены из одного параметризованного класса Apparat, который описывает способ организации набора тепло- или массообменных ячеек и операции над ними.

В случае, если необходимо собрать контейнер из ячеек различных типов, в качестве параметра указывается указатель на базовый класс, от которого они наследуются. Благодаря этому, любые объекты подклассов ячеек, помещенные в контейнер, не будут «срезаться» и сохранят свое полиморфное поведение.

В общем случае существуют четыре основных способа создавать параметризованные классы. Первый – использовать макроопределения. Так было в раннем C++ и существует в C, но макросы находятся вне семантики языка, при каждом инстанцировании создается новая копия программного кода. Поэтому их целесообразно использовать только для небольших проектов. Второй способ основывается на позднем связывании и наследовании, как это делается в Smalltalk, Ruby. При таком подходе можно строить только неоднородные контейнерные классы,

так как в языках нет средства, обеспечивающего создание нужного класса элементов контейнера. Каждый элемент трактуется как экземпляр некоторого удаленного базового класса. Третий способ реализован в языках семейства Object Pascal, которые имеют и мощную систему контроля типов, и наследование, но не поддерживают никакой разновидности параметризованных классов. В этом случае приходится создавать обобщенные контейнеры, как в Smalltalk или Ruby, но использовать явную проверку типа объекта, прежде чем помещать его в контейнер. Наконец, есть собственно параметризованные классы, впервые появившиеся в CLU и присутствующие в C++ и Eiffel.

4.6.6. МЕТАКЛАССЫ

В контексте рассматриваемой задачи использование метаклассов аналогично случаям, рассмотренным выше, может иметь широкое прикладное значение. Это может быть автоматизированная среда быстрой разработки или визуального программирования задач моделирования. Здесь концепция может быть применена как к классам аппаратов, так и к классам профилей температур. Так же метаклассы могут быть использованы при создании специализированного языка программирования, предназначенного для решения задач моделирования. Использование специализированного языка программирования позволит сократить время разработки программ без потери их качества, упростит реализацию программных элементов, реализующих динамическую генерацию моделей физических систем.

5. ПРОГРАММНОЕ ПРЕДСТАВЛЕНИЕ ФУНКЦИОНИРОВАНИЯ ТЕХНИЧЕСКИХ СИСТЕМ

В технических системах, во время их работы, протекают взаимосвязанные физические и химические процессы. Они могут протекать одновременно или последовательно (после того, как заканчивается один, запускается другой), в одной или разных областях пространства. Области пространства ограничиваются аппаратами или конструктивными единицами технического комплекса. Совокупность этих процессов, их распределение во времени и пространстве определяет поведение технологического комплекса, результаты его работы. Например, работа химико-технологической схемы обеспечивается совместной работой входящих в нее аппаратов. В каждом из них протекает ряд взаимосвязанных процессов. Изменение параметров одного из них влечет за собой изменение характеристик получаемых продуктов, может вызвать изменение работы всего технического комплекса. В качестве примера выполнения параллельных процессов в рамках одного аппарата можно рассматривать работу кожухотрубчатого теплообменного аппарата или реактора с мешалкой. В теплообменнике движутся теплоносители – это один процесс и происходит перенос тепла – другой процесс. Эти процессы имеют разную природу, но связаны между собой и происходят одновременно. Так же и в реакторе одновременно происходят процессы перемешивания и химических превращений, теплообмена.

5.1. ПРЕДСТАВЛЕНИЕ ПРОЦЕССОВ В ТЕХНИЧЕСКИХ СИСТЕМАХ НА ОСНОВЕ ДЕЦЕНТРАЛИЗОВАННОГО УПРАВЛЕНИЯ

Использование децентрализованных архитектур дает возможность обеспечивать выполнение параллельных вычислений. Так же это позволяет сопоставлять во времени выполнение расчетов программной системой и течение физических процессов в технической системе. Посредством независимой работы программных элементов определяются поведение и результаты программной системы. Полученные элементы программной системы обеспечивают группировку данных и методов, описывающих состояние и поведение физических объектов рассматриваемой технической системы. При этом отдельно осуществ-

ляется моделирование физических объектов технической системы и ее структуры в целом.

Программная система, реализующая решение задачи моделирования основывается на асинхронных по своему характеру, допускающих прерывание работы и выполняющихся параллельно действиях, каждое из которых может быть не детерминировано и происходить в случайные и непредсказуемые моменты времени.

Основной задачей, решаемой каждым программным элементом (абстракцией), является расчет параметров соответствующего оригинала технического комплекса. Расчет выполняется многократно, на протяжении всего времени работы программной системы, определяя тем самым состояния объекта во времени. Он выполняется на основе данных о состоянии объекта в предыдущий момент времени и данных, поступающих к нему от других объектов. Структура таких информационных связей между программными элементами определяется взаимодействием соответствующих элементов технической системы.

Вычисления организуются в виде нескольких потоков выполнения, в каждом из которых решается конкретная подзадача. Каждый объект реализует как минимум один поток выполнения. В них вычисляются промежуточные для всей программной системы результаты для последующего слияния и формирования заключительной части вывода. Децентрализованные архитектуры позволяют организовать процесс вычислений параметров взаимосвязанных процессов так, чтобы они выполнялись параллельно (рис. 5.1). Количество потоков вычисления при этом не ограничивается. Таким образом, при использовании децентрализованных архитектур можно смоделировать процесс работы технического комплекса за счет организации потоков вычисления так, чтобы каждый поток рассчитывал отдельно параметры одного процесса.

В этом случае задача моделирования процессов разбивается на независимые подзадачи, которые могут выполняться одновременно. Подобная параллельная организация работы позволяет существенно повысить эффективность решения всей задачи. Параллельный процесс вычислений характеристик процессов можно организовывать двумя различными способами.

Абстракции представляют собой активные сущности – они могут отдельно выполнять действия независимо от других объектов. Возможность взаимодействия объектов обеспечивается тем, что, выполняя действия, он обладает возможностью принимать сообщения от других

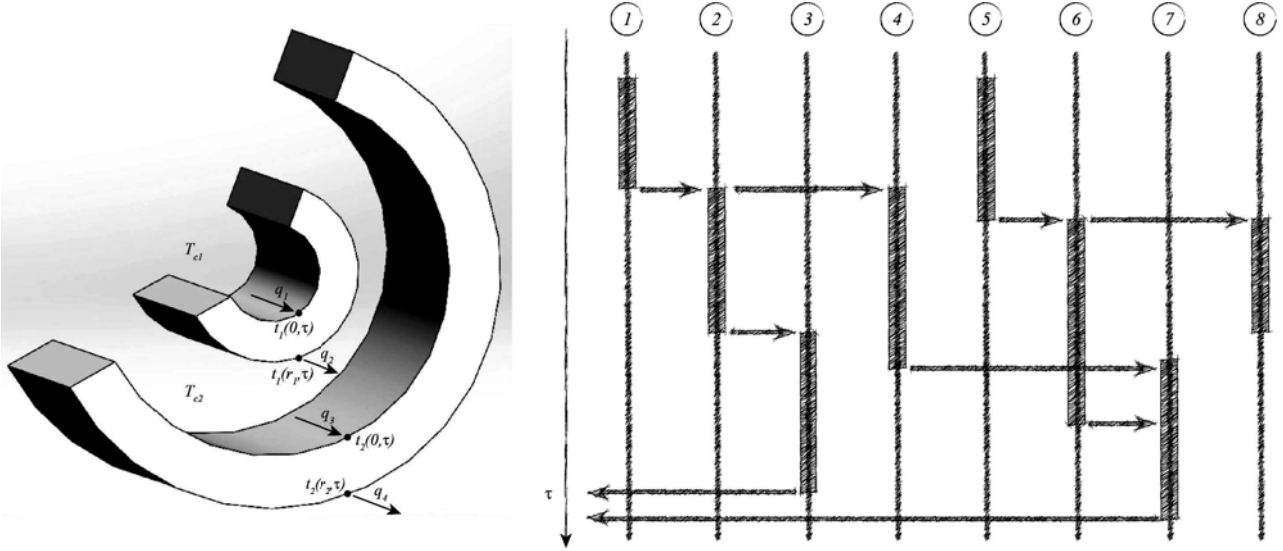


Рис. 5.1. Параллельное вычисление характеристик процессов, которые протекают одновременно (на примере фрагмента теплообменника, горячий теплоноситель течет в трубном пространстве):

1 – расчет поля температур стенки трубки; *2* – расчет теплового потока q_1 от горячего теплоносителя к стенке трубки; *3* – расчет температуры теплоносителя T_{c1}; *4* – расчет теплового потока q_2 от стенки трубки к холодному теплоносителю; *5* – расчет поля температур стенки обечайки; *6* – теплового потока q_3 от холодного теплоносителя к стенке обечайки; *7* – расчет температуры холодного теплоносителя T_{c2}; *8* – расчет тепловых потерь q_4

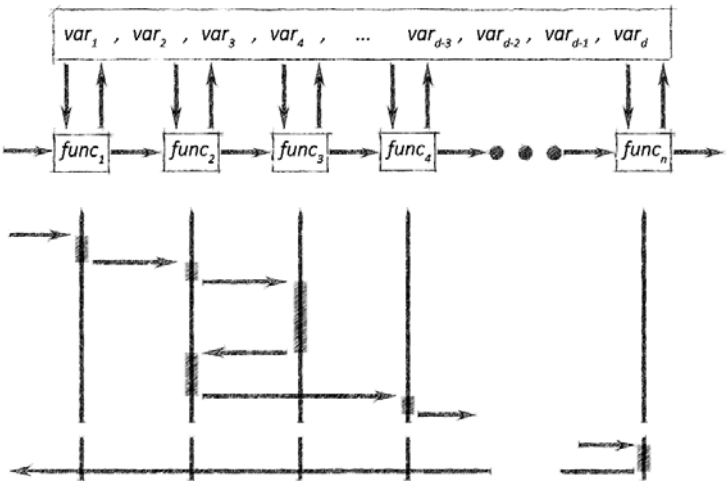
объектов и выполнять действия, в зависимости от получаемой в сообщениях информации. Такая работа объекта теоретически может выполняться бесконечно долго, но обычно ее продолжительность соответствует длительности процессов, которые протекают в их физических аналогах.

Такая архитектура позволяет использовать методы параллельных вычислений для того, чтобы обеспечивать скорость обработки информации, равную скорости течения моделируемых физических процессов в техническом комплексе. В том случае, если использование аппаратных мощностей недостаточно для обеспечения требуемой скорости вычислений, такая архитектура может быть легко, без изменений использована для распределенных систем.

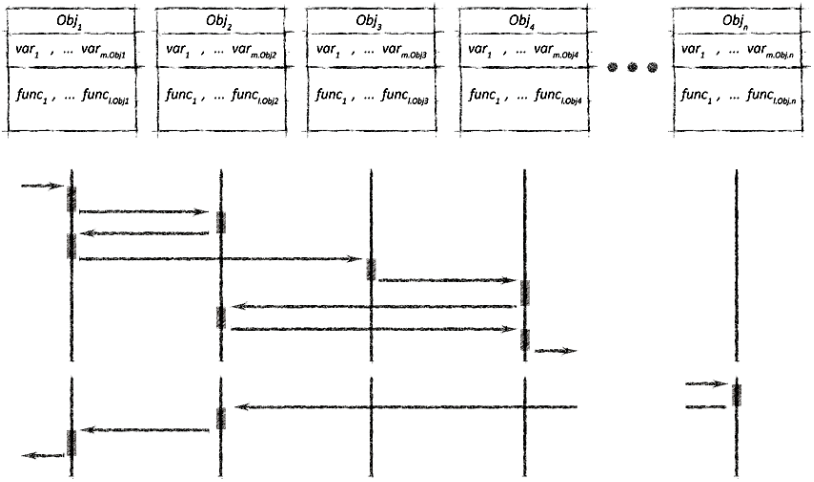
5.2. НЕДОСТАТКИ МЕТОДОВ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ ПРИ МОДЕЛИРОВАНИИ ДИНАМИКИ ТЕХНИЧЕСКИХ СИСТЕМ

Теория объектно-ориентированного проектирования регламентирует конструирование и использование объектов на уровне исходных текстов программы. С точки зрения потока управления программы передача сообщения объекту эквивалентна вызову функции. Фрагменты кода программы выполняются также последовательно, как при процедурном программировании. Возможность одновременного выполнения двух и более фрагментов программы без использования дополнительных инструментальных средств отсутствует. Это свойство объектов является принципиальным недостатком при их использовании для построения абстракций элементов технических систем. С точки зрения разработки они удобны и решают ту задачу, для решения которой они предназначены – разделить программу на независимые компоненты, которые можно независимо разрабатывать. Однако объекты никак не влияют на ход выполнения программы. Коммуникации в объектно-ориентированных техниках предполагают передачу сообщений объектам. На самом деле с точки зрения реализации и семантики языка это все тот же вызов публичных функций. Поэтому объекты позволяют только сократить сложность и упростить процесс разработки. Но их использование никак не влияет на ход выполнения программы.

Современными объектно-ориентированными языками предоставляются в среднем равные возможности по организации условных операторов и циклов, которые могут передавать управление



а)



б)

Рис. 5.2. Последовательное выполнение вычислений при организации программной системы на основе:
а – методов функциональной декомпозиции;
б – методов объектно-ориентированного программирования

в рамках одного вычислительного процесса тому или иному фрагменту кода. Управление ходом программы осуществляется в процессе выполнения условных операторов и операторов цикла. Оно не может быть осуществлено в любой момент времени, а только в определенные моменты, когда последовательность выполнения операторов программы дойдет до места, где расположен условный оператор или оператор цикла, и предусмотрено принятие решения о ходе выполнения программы.

5.3. ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ НА ОСНОВЕ АКТОРОВ И АГЕНТОВ

Одним из направлений эволюции концепции объектов являются концепции акторов и агентов. Они обладают свойством параллельного выполнения и коммуникации. Это обеспечивает возможность их использования для моделирования динамики структурной единицы технической системы.

Актор является программным модулем, выполняющим действия в ответ на поступающие сообщения. Он объединяет в себе данные и функции, т.е. помимо возможности действия, актор обладает возможностью сохранять и изменять свое состояние. Вид действия определяется содержанием и формой поступающих сообщений. В периоды времени, когда все сообщения обработаны, а новые еще не поступили, актор находится в состоянии активного ожидания сообщений, не выполняя никаких дополнительных действий. Для акторов определены три основных вида действия – выбор способа реакции на сообщение, генерировать новые сообщения и акторы. Никаких ограничений на последовательность этих действий не накладывается, они все могут выполняться параллельно.

Модель акторов успешно используется в качестве основы для реализации работы и моделирования широкого спектра параллельных программных систем. К их числу относятся электронная почта, веб-сервисы, сетевые серверы, многопользовательские интерактивные системы, инструменты для тестирования, управления и слежения.

Модель акторов является основой для модели вычислений на основе агентов, которая формализовано значительно меньше. Многие идеи, введенные в модели акторов, в настоящее время находят также применение в многоагентных системах. Агенты расширяют функциональность акторов возможностями автономного поведения и целе-

образования. В общем случае агент представляет собой программу, которая характеризуется наличием автономного поведения, благодаря которому он взаимодействует с внешней сложной и динамично развивающейся средой. Принципиальным отличием их функциональности от функциональности акторов является возможность их автономного поведения вне зависимости от сообщений, которые они принимают. Агенты за счет своих возможностей автономного поведения и целеобразования предполагают определенную долю интеллекта в своих действиях. Взаимодействие подразумевает восприятие динамики среды, выполнение действий, изменяющих ее, возможность интерпретации наблюдаемых явлений, решения задач, вывода заключений и определения действий. Родственные и производные определения агентов используют термины интеллектуальных агентов (обладающих аспектами искусственного интеллекта и возможностями к обучению и рассуждению), автономных агентов (способных изменять способ достижения своих целей), распределенных агентов (выполняющих действия на физически различных компьютерах), многоагентных систем (распределенные агенты, которые не имеют возможности достижения цели в одиночку и, следовательно, должны общаться) и мобильных агентов (агентов, которые могут переместить свое выполнение на другие процессоры).

Построение абстракций на основе агентов обеспечивало бы возможность автономного поведения и интеллектуального целеобразования для каждой абстракции. Это не соответствует принципу структурного и динамического соответствия программной и технической систем. Рассматриваемая модель оперирует неодушевленными элементами технической системы, которые принципиально лишены возможности целеобразования и интеллектуального поведения. Определение агентов, используемых для конструирования абстракций технических систем, должно сужать рамки и специализировать традиционные определения. Являясь основой абстракций неодушевленных элементов технической системы, агенты в этом случае должны также исключать возможности интеллектуального поведения и целеобразования.

Существуют альтернативные модели параллельных вычислений, но они обладают рядом особенностей, делающих их использование для реализации вычислений на основе модели абстракций технической системы громоздкими. Основной такой особенностью является их плохая или полная несовместимость с моделью объектов, которая используется для моделирования структуры технической системы.

Они с затруднением могут быть использованы для того, чтобы дополнить и развить функциональность объектов до возможности выполнения ими параллельных вычислений.

Возможно комбинирование двух этих моделей вычислений и избирательное использование агентов совместно с акторами. Это не противоречит модели абстракций технической системы, снижает уровень избыточной функциональности, но оставляет дальнейшие возможности для его снижения.

5.4. ПРЕДСТАВЛЕНИЕ ПРОГРАММНЫХ АБСТРАКЦИЙ ЭЛЕМЕНТОВ ТЕХНИЧЕСКИХ СИСТЕМ НА ДИНАМИЧЕСКОМ УРОВНЕ

Модель акторов наиболее близка к тому понятию программных абстракций, которое рассматривается в рамках метода. Она представляет собой математическую модель параллельных вычислений. Эта модель параллельных вычислений имеет большую историю, хорошо изучена и формализована. Для нее разработана операционная семантика, система аксиоматических законов, детонационная семантика (семантика переходов). Акторы обладают минимальным набором необходимых свойств для того, чтобы моделировать параллельную работу и динамику взаимодействия элементов технической системы. При этом использование модели акторов не совсем подходит в качестве основы функционирования абстракций элементов технической системы, в силу того, что их функциональность выходит за рамки модели акторов:

– функционирование абстракций в контексте иерархической централизованной структуры;

– абстракции в отличие от акторов могут инициировать действия в программной системе;

– абстракции имеют ограничения на обработку сообщений.

Рассматриваемый метод архитектурного проектирования не рассматривает расширение и дополнение модели акторов, хотя принципиально это возможно, и с незначительными доработками и дополнениями она может быть использована. Так же он не рассматривает сужение модели агентов. Функционирование абстракций реализуется на основе активных объектов, которые в свою очередь конструируются на основе объектов, используемых на этапе построения структуры (рис. 5.3).

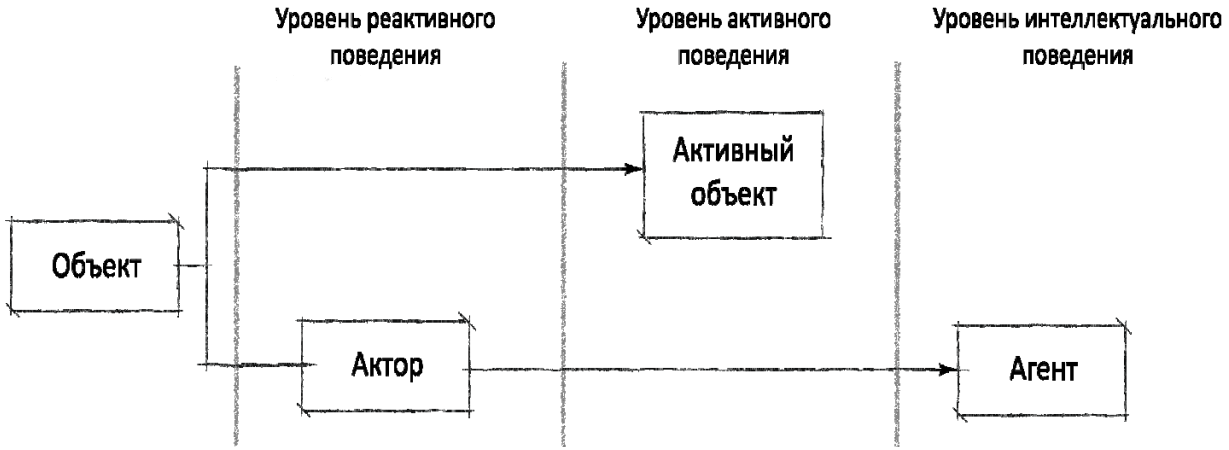


Рис. 5.3. Возможные представления динамического уровня абстракций

Объекты дополняются наличием механизмов, обеспечивающих параллелизм вычислений, прямой асинхронный обмен сообщениями с учетом особенностей, которые требует методология. Эта функциональность в самом базовом варианте обычно представляет собой процедуру, которая запускается одновременно с созданием активного объекта и время работы которой совпадает с временем его жизни. Она обеспечивает прием сообщений и запуск дочерних процессов, которые обрабатывают входящие сообщения и выполняют расчеты (рис. 5.4).

Процедура, обеспечивающая активное и аллертное состояние активного объекта, обеспечивает обработку поступающих сообщений. Для того чтобы сохранить принцип асинхронного обмена данными, инициируется новый процесс или поток выполнения. Эти потоки и процессы могут потом дальше дробиться, порождая новые процессы и потоки. Запуск процедуры осуществляется одновременно с созданием экземпляра актора.

Такой вариант конструирования активного объекта является универсальным и не зависит от языка программирования. Он может быть реализован на любом языке, независимо от его назначения и специализации.

Вычисления состояний элементов технической системы на основе сконструированных рассмотренным образом активных объектов преимущественно реактивны. Основная масса объектов, составляющих программную систему, действия не инициирует. Это в большей степени по сравнению с моделью акторов соответствует тому представлению о технической системе, которое положено в основу рассматриваемого метода архитектурного проектирования – согласно ему техническая система состоит из взаимодействующих между собой элементов. Часть этих элементов не продуцирует своих собственных воздействий и является также реактивной. Часть элементов обладают способностью самостоятельно, без внешних воздействий, в соответствии со своими внутренними законами, изменять свое состояние.

В отличие от модели акторов, которую необходимо было бы для реализации такого представления технической системы дополнять сервисами или агентами, здесь отсутствует необходимость смешения разных моделей вычислений. Простое сочетание моделей акторов, агентов или сервисов делает недействительными гарантии, которые обычно предоставляет каждая модель в отдельности, тем самым нарушая предположения разработчиков. При этом они часто объединяются поразному, и семантика комбинации не всегда четко определена.

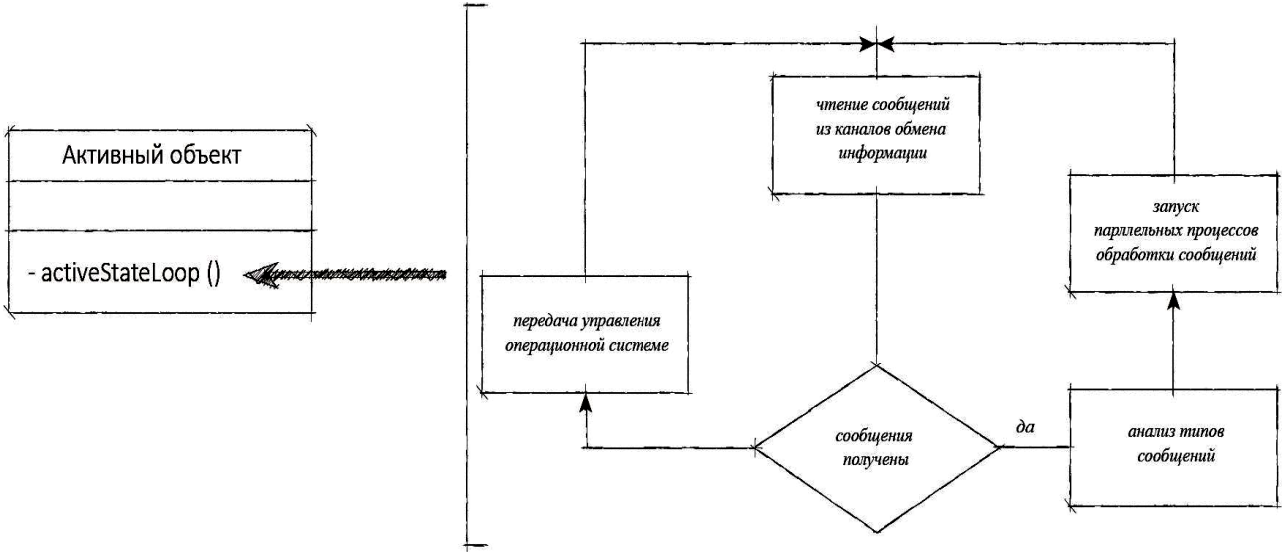


Рис. 5.4. Конструирование активного объекта

Использование модели вычислений на основе активных объектов обеспечивает такие ее характеристики, как параллелизм вычислений внутри одной абстракции и между ними, возможность динамического создания абстракций, возможность взаимодействия не только через прямой асинхронный обмен сообщениями без каких-либо ограничений на порядок прибытия сообщений, но также и использование синхронного режима взаимодействия.

Активные объекты, так же как и объекты, являются экземплярами абстрактных типов данных, которые описывают возможные состояния и функции соответствующей конструктивной единицы технической системы. Они характеризуются наличием механизмов, обеспечивающих параллелизм вычислений и прямой асинхронный обмен сообщениями без ограничений на расписание движения сообщений. Для объектов, рассмотренных выше, используемых для представления структуры технической системы, как было показано выше, исключается возможность их параллельной работы. Для активных объектов такая возможность обеспечивается. Это предоставляет возможности параллельного функционирования построенных на основе активных объектов абстракций технической системы, организации параллельных процессов вычисления внутри каждой абстракции и балансирования вычислительной нагрузки между ними таким образом, чтобы обеспечить требуемое время выполнения расчетов. Также как и в случае моделирования структуры, при этом обеспечивается возможность реализации части функций на уровне отношений типов данных и повышается связность абстракций. Благодаря этому уменьшается их зацепление, упрощается логика абстракций и всей программной системы (например, уменьшается количество логических операторов и операций выбора), сокращается объем сервисных вычислений и исходного кода.

5.5. СОВПАДЕНИЕ СОСТОЯНИЙ ТЕХНИЧЕСКОЙ И ПРОГРАММНОЙ СИСТЕМ ВО ВРЕМЕНИ

Принцип динамического соответствия программной и технической систем позволяет обеспечивать выравнивание времени работы программной системы относительно времени работы технической системы. При этом состояния программной системы и технической системы через равные промежутки от начала их работы совпадают.

Состояние каждой программной абстракции описывается нестационарным полем целевых характеристик (температура, давление, ско-

рости, концентрации). Решение нестационарной задачи позволяет получить информацию о времени, за которое элементы технической системы изменяют свое состояние. Полученная информация может быть спроецирована на программную систему. Это становится возможным, если время работы технической и программных систем представить в виде совокупности временных интервалов. На границах интервалов состояние технической и программных систем совпадают.

Для этого функциональность абстракций дополняется операторами, которые информируют операционную систему о том, что программной системе необходимо больше вычислительных ресурсов или, наоборот, они ей сейчас не требуются. На основе этой информации операционная система предоставляет абстракции больше или меньше вычислительных ресурсов, в том объеме, чтобы выполнить расчет состояния за требуемое время.

Возможен случай, когда отсутствует принципиальная возможность получить больше вычислительных ресурсов и выполнить расчет нового состояния абстракции за требуемое время. Тогда интервал времени, за которое изменилось состояние элемента технической системы увеличивается до значения, которое потребуется программной абстракции, чтобы рассчитать свое состояние. Можно определить минимальный временной отрезок, для которого осуществляется пересчет состояний, как время, необходимое абстракции, чтобы выполнить расчет своего состояния. Использование аналитических решений позволяет выполнять расчет состояния для момента времени любой длины без потери точности. Таким образом, здесь может быть выполнено выравнивание времени за счет определения временного интервала, для которого необходимо выполнять расчет состояния как интервала, для которого может быть выполнен расчет нового состояния абстракции. Увеличение временного интервала снизит количество моментов времени, когда состояния совпадают (рис. 5.5).

Такой вариант регулирования скорости работы программы позволяет изменять длину временных интервалов в процессе работы программной системы. Использование временных интервалов переменной длины добавит гибкости в управлении ресурсами вычислительных комплексов и направлении их на нужды программной системы.

Этот подход невозможно точно реализовать на этапе проектирования, так как неизвестно расчетное время и закономерности изменения характеристики процессов. Для его реализации в абстракциях пре-

дусматриваются операторы, которые измеряют время выполнения фрагментов кода. Полученная информация определяет временные интервалы расчетов состояний элементов. Особенностью аналитических решений является значение функции для любых значений аргументов при равной нагрузке и времени его получения.

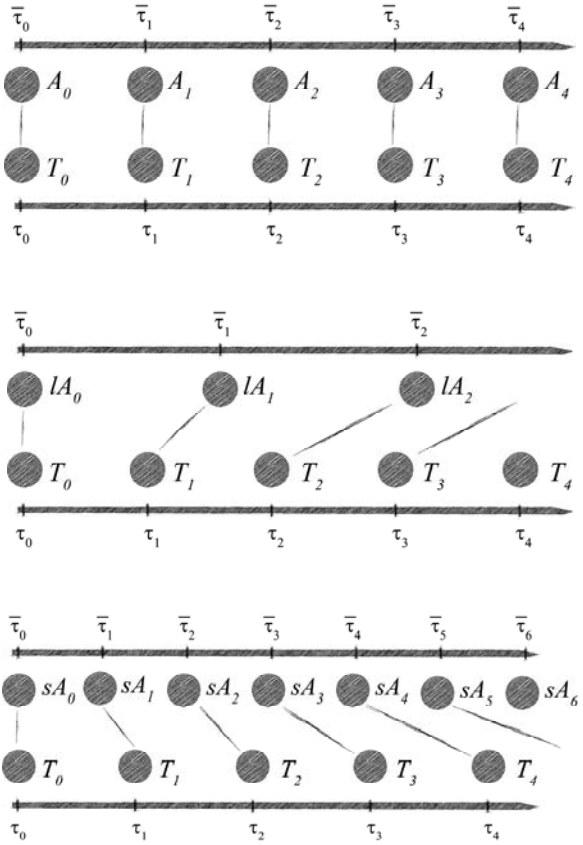


Рис. 5.5. Соответствие скорости вычислений скоростям течения моделируемых процессов:

$T_0 – T_4$ – состояния технической системы;

$\tau_0 – \tau_4$ – моменты времени на временной оси работы технической системы;

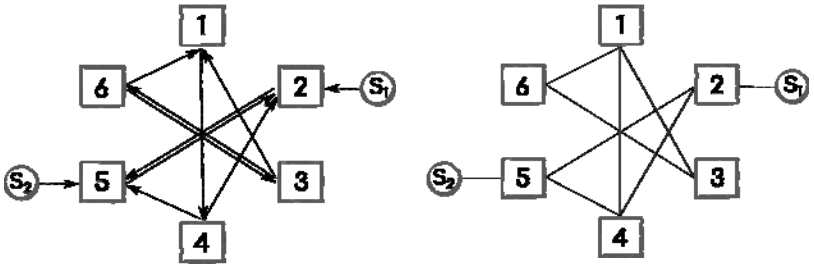
$A_0 – A_4$, $lA_0 – lA_2$, $sA_0 – sA_6$ – состояния программной системы;

$\overline{\tau}_0 – \overline{\tau}_6$ – моменты времени на временной оси работы программной системы

Работа программной системы синхронно с технической системой может иметь место при решении задач информационной поддержки технологических процессов. В них результаты моделирования используются для дополнения информации о технологическом процессе, предоставляемой средствами контроля. В этом случае программная система использует данные, поступающие от сенсоров, как исходные данные для решения задачи моделирования. Результатом ее работы является дополнительная информация о состоянии технической системы.

Сенсоры, функционируя в составе оборудования технической системы, измеряют его параметры и передают данные программной системе. Для нее сенсоры являются устройствами ввода данных. Ограничения на количество сенсоров, подключенных к программной системе отсутствуют. Каждый сенсор подключается к соответствующей абстракции технической системы. Данные, поступающие от сенсора, обеспечивают изменение значения одной или нескольких ее характеристик и запуск циклов расчета новых состояний (рис. 5.6).

Функционирование программной системы на цифровой аппаратной платформе определяет то, что данные поступают порциями, с определенной периодичностью. Сенсоры являются в данном случае генераторами потоков данных. Для системы, чья динамика выполнена на рассмотренных выше активных объектах и носит реактивных характер, генерируемые сенсорами потоки данных являются причиной запуска



а) *б*)

Рис. 5.6. Работа программной системы под управлением потока данных от сенсоров:
а – сенсоры S_1 и S_2 подключены к абстракциям, каналы связи между абстракциями установлены, но данные от сенсоров не поступают, и программная система не работает; *б* – данные от сенсоров поступают и это является причиной работы программной системы

циклов расчета нового состояния технической системы по вновь поступившим исходным данным. Каждая порция данных, поступающая от сенсоров, характеризует состояние оборудования технической системы в текущий момент времени. Решение задачи моделирования и пересчет предыдущего состояния для этих исходных данных обеспечит новое состояние программной системы, соответствующее состоянию технической системы для этого же момента времени. Последовательное решение задачи моделирования для каждой порции данных, поступающих от сенсоров, обеспечит множество состояний программной системы во времени. Эти состояния будут соответствовать состояниям технической системы и ее элементов.

Сенсоры выполняют роль промежуточного слоя и обеспечивают переход между реальным миром, где функционирует техническая система, и адресным пространством вычислительного комплекса, где функционирует программная система.

С одной стороны, сенсоры привязаны к физическому оборудованию, с другой – генерируют потоки исходных, необходимых для работы программной системы. Поступление исходных данных в программную систему инициирует в ней циклы расчетов. Обработка одной порции исходных данных – решение задачи моделирования и изменение состояния элементов программной системы – должна быть выполнена до момента поступления новой порции данных. Параллелизм, который присущ модели вычислений на основе активных объектов, обеспечивает возможность управления временем решения задачи моделирования.

Датчики могут быть заменены программными эмуляторами. По отношению к программе они ведут себя как сенсоры, но в действительности они не привязаны к технологическому оборудованию. В этом случае связь с технической системой, а может быть и сама техническая система отсутствуют. Программная система функционирует автономно. Так же как и в рассмотренном выше случае, действия в программной системе инициируются потоками данных, поставляемых программными эмуляторами сенсоров. Здесь генерация данных осуществляется в соответствии с заранее определенными законами изменения исходных данных во времени. Она осуществляется также с определенной периодичностью. Привязка к реальному времени осуществляется соответствием момента времени, в который осуществляется генерация данных, моменту времени, для которого она осуществляется. Так как связь с технической системой отсутствует, порции данных могут генерироваться без привязки к реальному времени. Увеличение или

уменьшение скорости генерации исходных данных позволяет моделировать увеличение или уменьшение скорости работы технической системы.

Увеличение скорости своей работы относительно скорости работы технической системы может использоваться при решении задач прогнозирования. Замедление скорости работы может использоваться для изучения и наглядного представления быстротекущих процессов.

5.6. УПРАВЛЕНИЕ ВЫЧИСЛИТЕЛЬНОЙ НАГРУЗКОЙ

Состояние элементов технической системы описывается сложными системами уравнений. Время выполнения программы, реализующей алгоритм расчета значений решения системы уравнений для заданных координат в пространстве и момента времени, может быть достаточно велико. Моменты времени, когда должно быть получено состояние технической системы, могут не соответствовать требованиям решаемой с помощью программной системы задачи. В этом случае интервалы необходимо сокращать. Как средство сокращения времени получения результата рассматривается и используется распараллеливание вычислений.

В идеальном случае программа, разделенная на n потоков выполнения, будет выполнена в n раз быстрее. Это возможно при условии, что каждый поток выполняется на своем отдельном процессоре. Несмотря на свою очевидность, такой идеальный случай встречается редко по ряду причин. Для достижения требуемой производительности простого определения параллелизма недостаточно: необходимо внимательно управлять перемещением данных, взаимодействием потоков и процессов, а также тщательно выбирать нужную степень грануляции параллелизма.

Распараллеливание вычислений может осуществляться до того момента, пока специфика решаемой задачи позволяет определять n независимых потоков выполнения. Параллельное вычисление обычно влечет за собой накладные расходы, которые отсутствуют при последовательных вычислениях.

Производительность параллельных вычислений определяется двумя аспектами – объемом операций, которые выполняются в единицу времени, и временем, которое требуется для выполнения операции (латентность). Несмотря на то, что два этих понятия на первый взгляд схожи, они по-разному влияют на производительность вычислительного процесса.

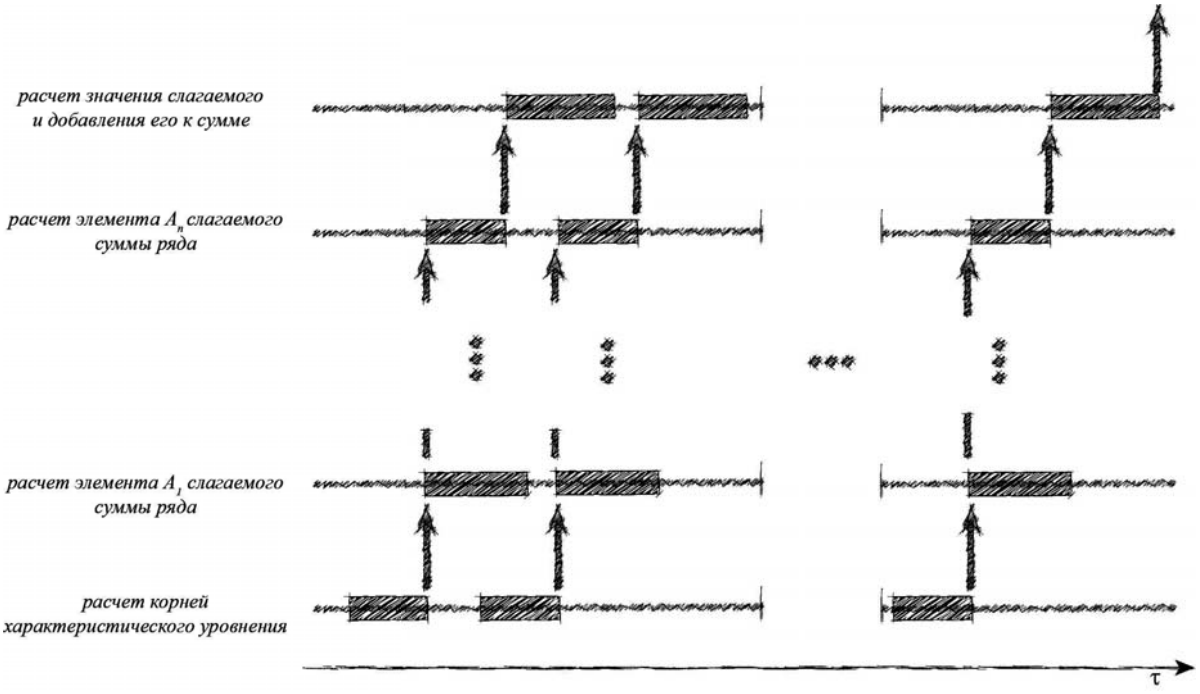


Рис. 5.7. Возможность распараллеливания алгоритма реализации аналитических решений задач математической физики

Снижение латентности в большинстве общих случаев находится в компетенции аппаратных методов, применяющих параллелизм. Использование аналитических решений задач математической физики в частных производных обеспечивает возможность высокой степени распараллеливания вычислений (рис. 5.7). Определение состояния элемента технической системы при их использовании может быть разделено на параллельно выполняющиеся потоки до уровня составных частей математических операторов, выполняющих расчет полей. Отдельные потоки вычислений могут быть определены для их отдельных элементов, также отдельно может быть осуществлен поиск собственных чисел задачи. Параллельно может выполняться решение характеристического уравнения, расчет на основе найденных собственных чисел значений членов ряда и вычисление их суммы.

В результате получаются небольшие процессы, расписание работы которых известно еще на стадии проектирования абстракции. Это позволяет реализовать их взаимодействие без дополнительных затрат вычислительных ресурсов.

Учет особенностей компьютерной реализации и структуры аналитических решений задач математической физики позволяют получить высокую степень распараллеливания, представить процесс вычислений как работу очень маленьких задач и практически исключить проблему латентности без привлечения аппаратных методов и изменения качественных и количественных характеристик аппаратных средств.

5.7. ОСОБЕННОСТИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ НА ОСНОВЕ АБСТРАКЦИЙ ЭЛЕМЕНТОВ ТЕХНИЧЕСКИХ СИСТЕМ

Все затраты ресурсов, возникающие при параллельных решениях, но отсутствующие в последовательных, рассматриваются как накладные расходы. Накладные расходы возникают при создании потоков и процессов для одновременного исполнения, а также при их уничтожении. Работа с динамической памятью – выделение, инициализация – требует больших затрат, поэтому процессы имеют более высокие накладные расходы, чем потоки. После создания первого процесса создание всех последующих потоков и процессов связано с накладными расходами, которые отсутствуют при последовательных вычислениях. Ниже рассмотрены источники накладных расходов при параллельных вычислениях.

Синхронизация представляет собой форму накладных расходов, возникающих, когда один поток или процесс ожидает завершения события в другом потоке или процессе. Например, поток может ожидать, пока другой поток вычислит значение или освободит ресурс. Потоку может понадобиться установка и снятие блокировки для области данных для предотвращения транзакций. Установка блокировок, их снятие относится к накладным расходам. Здесь объем накладных расходов растет пропорционально количеству установленных блокировок.

Обмен информацией между процессами или потоками можно рассматривать как один из видов их синхронизации. Он может быть организован несколькими способами.

Один процесс может ожидать информацию от другого процесса и для этого приостанавливать свое выполнение. В идеальном случае все процессы должны работать непрерывно, без простоев. В соответствии с принципом динамического соответствия вычислений процессам в технической системе, простои могут допускаться в том случае, если они соответствуют простоям процессов в технической системе. В большинстве случаев этого не происходит, и на практике время простоя определяется факторами синхронизации и взаимодействия параллельных процессов. Распространенной причиной является ожидание процессом внешнего события – данных от сенсоров, данных или управления от других работающих процессов. Для программных систем моделирования ожидание данных от сенсоров является случаем, когда простой соответствует простою процессов в технической системе. Еще одной причиной простоя является неравномерное распределение работы между процессами.

Альтернативой ему может служить вариант, когда обеспечивается непрерывная работа коммуницирующих процессов. Абстракция посылает запрос на получение данных в тот момент, когда у нее возникает в этом необходимость. Абстракция, которая отсылает данные, делает это тогда, когда ей необходимо, например, информировать другие процессы об изменении состояния его родительской абстракции. В вычислениях на основе взаимодействия абстракций технической системы возможны оба этих случая. Основным условием их корректной работы должна быть гарантия со стороны абстракций, что они всегда смогут принять любое сообщение без ограничений на время доставки. Эта гарантия обеспечивается тем, что модель взаимодействия построена на основе активных объектов.

Система моделирования не является системой, управляемой пользователем или спонтанными внешними событиями, в которых время и частоту возникновения событий прогнозировать очень сложно. Она в основном сосредоточена на самой себе и работе своих модулей. Вмешательства в расписание ее работы в принципе возможны в виде воздействия от других систем. Основным, единственным и самым главным вмешательством является порция исходных данных, которая поступает в систему моделирования и которую система моделирования обрабатывает, рассчитывая параметры процессов. Иногда возможны отдельные воздействия на объекты технического комплекса, что отражается в модулях системы. Они изменяют параметры отдельных модулей, эти процессы в технических системах происходят параллельно.

Задачи моделирования характеризуются тем, что на момент их программной реализации объем вычислений, количество параллельно работающих процессов, распределение вычислительной нагрузки и расписание коммуникаций между ними известны. На момент реализации известны все механизмы взаимодействия конструктивных единиц и потоков технического комплекса, виды и структура математических зависимостей, которые будут реализованы, приблизительно известна вычислительная нагрузка на процессы. Это позволяет принимать меры по выравниванию вычислительной нагрузки между процессами (рис. 5.8) на этапе разработки программы и обеспечивать планирование процессов так, чтобы учитывать взаимное расписание их работы. Уже на этапе проектирования программной системы необходимо выбрать такую степень распараллеливания процессов, чтобы выровнять латентность абстракций и исключить бездействие некоторых из них в ожидании исходных данных для расчетов. При этом уменьшается

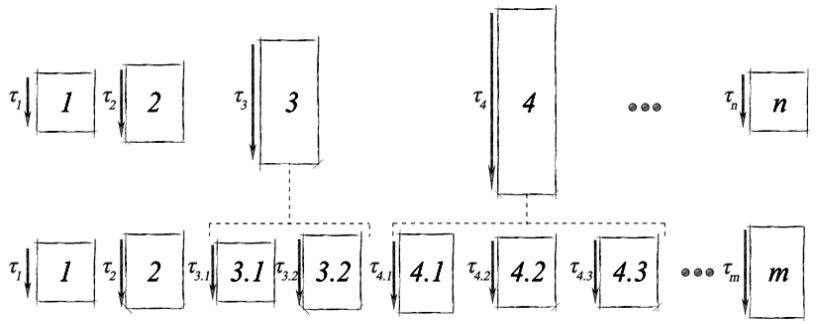


Рис. 5.8. Балансировка вычислительной нагрузки

объем вычислений, выполняемый параллельными сервисными потоками, в лучшем случае они совсем исключаются.

Построение программной системы на основе взаимодействующих абстракций элементов технической системы исключает общий канал обмена информацией и возможность конкуренции между несколькими параллельными абстракциями за доступ к нему. При его использовании вероятна ситуация, когда несколько процессов конкурируют за доступ к общему каналу обмена данными. Такая конкуренция дополнительно сопровождается периодической проверкой статуса канала конкурирующим потоком. Это увеличивает нагрузку на общий канал обмена сообщениями. Также практически исключает возможность состязания за ресурсы памяти. Эта архитектура не предполагает общего хранилища данных. Вся информация, которой оперирует программная система, распределена между ее абстракциями. Доступ к ресурсам общей памяти возникает только в одном случае – это доступ абстракции к переменным, описывающим характеристики внешней среды и являющимися глобальными по отношению к программной системе.

Использование аналитических решений позволяет обеспечить высокий уровень распараллеливания задачи – до отдельных математических операторов, составляющих оператор для расчета полей целевых характеристик. Без использования параллельных вычислений поиск собственных чисел задачи, расчет на их основе коэффициентов членов ряда, значений членов ряда осуществляется последовательно, и результаты промежуточных операций необходимо сохранять. Выполнение всех этих операций может быть организовано параллельно. При этом отсутствует необходимость сохранения промежуточных результатов. Полученные значения собственных чисел могут быть сразу же использованы для вычисления значений членов ряда, которые сразу же могут суммироваться. В этом случае реализация аналитических решений не требует больших объемов памяти. Можно сказать, что даже наоборот, позволяет сократить их, сосредотачивая вычислительные ресурсы в сфере вычислений. Поэтому даже в случае, когда программная система функционирует на специализированных контроллерах встраиваемых систем, этот фактор не снижает производительность при выполнении большого количества мелких операций.

В некоторых случаях искусственное расширение объема используемой памяти может повысить степень параллелизма и скорость вычисления математических операторов, составляющих аналитическое решение. Такая реализация возможна для абстракций на нижних

уровнях иерархического представления технической системы, чьи характеристики описываются аналитическими решениями. Требования к памяти не всегда растут с такой же скоростью, как требования, предъявляемые к вычислениям.

Использование дополнительных объемов памяти снижает связность абстракции по данным. Такое снижение связности одного модуля может оказать негативное влияние на всю систему и ограничивает возможности повторного использования кода. Например, для одной задачи целесообразно появление одного набора переменных, для другой задачи – другого, и использование первого набора для нее может быть неэффективно. Поэтому в общих случаях такой вариант лучше исключать вообще, и проектировать абстракцию без введения дополнительных объемов памяти.

При реализации конкретной прикладной задачи можно выполнить адаптацию внутренних механизмов, реализующих функциональность абстракции, к условиям задачи. В зависимости от того, на каком вычислительном оборудовании будет выполняться программная система (например, это может быть микроконтроллер с ограниченными вычислительными ресурсами), и того, насколько необходимо будет снижать при этом время получения результата, дополнительные объемы памяти могут быть использованы. Этот метод не нарушает целостность программной системы, но снижает связность абстракций, которая проявляется при повторном использовании кода. Изменения будут локальны, всегда будут находиться рамках одной конкретной абстракции, и для ее внешнего окружения будут незаметны. Такая адаптация сопряжена с необходимостью проведения дополнительных исследований времени расчета полей целевых характеристик, описывающих состояние абстракции.

В общем случае существуют четыре причины, по которым увеличение степени параллелизма не вызывает пропорционального снижения времени вычислений – накладные расходы, отсутствующие при последовательных вычислениях, простаивающие процессоры, состязание за ресурсы, нераспараллеливаемые вычисления. Все остальные причины являются частными случаями этих четырех факторов. В случае рассматриваемого класса программных систем основной причиной являются накладные расходы. Выше показано, что остальные три причины присутствуют в небольшом количестве частных случаев.

Накладные расходы складываются из накладных расходов на обеспечение параллельной работы абстракций и накладных расходов,

связанных с передачей сообщений между ними. Количество абстракций, работающих параллельно, каналов их взаимодействия и трафик определяются моделью абстракций технической системы. Поэтому специальных методов, которые позволяют уменьшать накладные расходы, не предусматривается.

В рамках отдельных абстракций, которые решают локальную математическую задачу, количество параллельно работающих процессов и каналов их взаимодействия определяется структурой математических операторов и алгоритмами их решения. Накладные расходы при этом незначительны. Это объясняется в первую очередь тем, что само использование аналитических решений характеризуется незначительным объемом вычислений. Без распараллеливания локальная задача может быть решена в течение короткого временного интервала. То количество параллельных процессов, которое позволяет определить структура математических операторов и алгоритмы их решения, не приводит к определяющей роли накладных расходов.

5.8. ПОКАЗАТЕЛИ ПРОИЗВОДИТЕЛЬНОСТИ ВЫЧИСЛЕНИЙ НА ОСНОВЕ АБСТРАКЦИЙ ЭЛЕМЕНТОВ ТЕХНИЧЕСКИХ СИСТЕМ

В контексте параллельных вычислений производительность является комплексным явлением, так как необходимо учитывать множество аспектов, в том числе использование памяти, время обработки, накладные расходы. Поэтому простое использование системы показателей производительности предполагает целый ряд тонкостей. Самым наглядным показателем является время выполнения, также называемое временем задержки. Это время, прошедшее от точки, в которой первый процесс начинает выполнять программу, до момента, когда последний процесс завершает выполнение. В задачах моделирования это время определяется как время между поступлением на вход абстракции исходных данных и получением для них значения функции, которая представляется аналитическим решением.

Дополнительно к времени выполнения комплексным критерием, показывающим качество временных диаграмм выполнения программы, являются интервалы между взаимодействием системы моделирования с другими системами. Эти интервалы показывают время простоя вычислительных ресурсов. Таким образом, программная система закончила работу и для текущих исходных данных получен результат –

состояние системы процессов с заданной точностью. Система останавливает свою работу до момента поступления новых исходных данных. В это время другие подсистемы, программные или аппаратные могут использовать этот результат и использовать освободившиеся вычислительные ресурсы. Это может продолжаться до того момента, пока не придут новые исходные данные. Это совокупный критерий, который показывает время выполнения и частоту поступления исходных данных.

Отдельно время выполнения может быть не очень показательно. Например, возможны случаи, когда время выполнения мало, но частота поступления исходных данных для расчета тоже велика, и время простоя после получения результата незначительно. В другом случае – у программной системы время выполнения большое и программная система с точки зрения критерия времени выполнения плохая, но она работает с редко поступающими исходными данными и она нас вполне устраивает. Таким образом, целесообразно использование совокупного критерия, характеризующего время простоя и время работы программной системы. Использование этого совокупного критерия – время получения результата от частоты поступления исходных данных – является более показательным. Он характеризует качество реализации задачи и возможность ее использования. Чем больше время простоя между получением результата и поступлением новой порции данных, тем больше возможность использования программной системы для решения прикладных задач.

6. ПРОГРАММНОЕ ПРЕДСТАВЛЕНИЕ ВЗАИМОСВЯЗЕЙ ЭЛЕМЕНТОВ ТЕХНИЧЕСКОЙ СИСТЕМЫ

Объекты, физические или программные, представленные в виде несвязанного набора, не представляют интереса. Только в процессе их взаимодействия реализуются функции системы. Способность программных элементов взаимодействовать между собой позволяет представлять в программной системе взаимосвязанные элементы технической системы и их влияние друг на друга. Децентрализованная модель функционирования программных систем определяет два вида взаимодействия между программными модулями – синхронное и асинхронное.

6.1. СВЯЗЫВАНИЕ ПРОГРАММНЫХ АБСТРАКЦИЙ С ПОМОЩЬЮ СООБЩЕНИЙ

Асинхронное взаимодействие не имеет никаких ограничений на время возникновения взаимодействия и его длительность и соответствует принципам работы элементов технической системы. Достоинством асинхронного взаимодействия является отсутствие потерь времени на ожидание синхронизирующего сигнала или готовности взаимодействующих модулей обрабатывать информацию. При этом в программной системе присутствует также и синхронное взаимодействие. С его помощью элементы, свойства которых меняются по заранее известному закону, информируют другие элементы о изменении своего состояния через определенные промежутки времени. В этих элементах изменение состояния возникает не под воздействием внешних факторов, а на основе внутренней логики. Так представляется воздействие элементов технической системы, состояние которых определяется не на основе взаимодействия с другими элементами, а на основе течения внутренних процессов или внешних по отношению к технической системе факторов. Например, так может быть представлена подача теплоносителей, температура которых изменяется во времени по заданному закону, или воздействие на обечайку теплообменного аппарата изменяющейся температуры окружающей среды.

Связи реализуют себя в рассматриваемых программных системах через сообщения. В общем случае сообщение представляет собой пакет данных, направленный от одного программного модуля к другому. Концепция сообщений является высокоуровневой версией датаграмм.

Сообщения могут быть больше по размерам и могут быть сделаны надежными, стойкими, безопасными и при необходимости могут быть охвачены механизмом транзакций. В такого рода сообщениях используются сложные структуры заголовков. В них прописывается получатель, код сообщения, контрольные суммы и т.д. (рис. 6.1). Структура сообщений может быть различной и варьироваться в зависимости от протокола, в рамках которого происходит обмен данными.

Такие сообщения могут использоваться как для обмена сетевыми сообщениями, так и для обмена между модулями программы, работающими на одном компьютере. Для некоторых систем, функционирующих на одном компьютере для организации потокового ввода данных или вывода, имеет смысл использовать тривиальную коммуникацию через файл. В любом случае сообщения вместе с полезной информацией содержат в себе служебную информацию, необходимую для доставки сообщения получателю и идентификации им полученного сообщения.

Взаимодействие рассматривается как композиция однонаправленных воздействий элементов технической системы друг на друга. Например, взаимодействие двух элементов технической системы рассматривается как пара воздействий, направленных от одного элемента и назад. Это необходимо для того, чтобы для каждого случая, индивидуально иметь возможность определять необходимость двунаправленного взаимодействия. В том случае, если это не нужно, конструируется только однонаправленный канал обмена сообщениями. Он рассматривается как атомарная программная единица обмена информацией. Воздействие одного элемента технической системы может распространяться одновременно на несколько ее элементов.

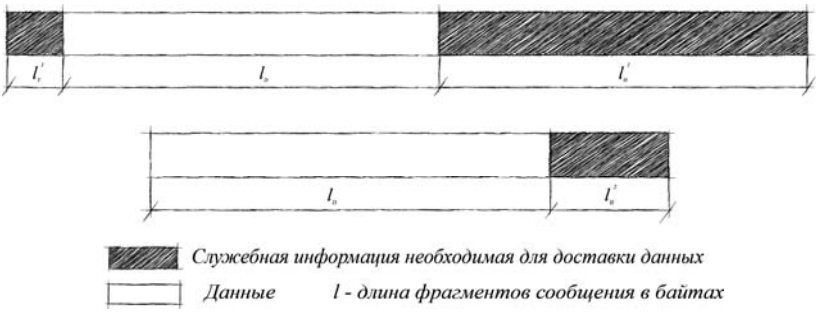


Рис. 6.1. Структура сообщений

С точки зрения принципа структурного и динамического соответствия вычислений процессам в технической системе представления взаимодействия элементов как композиции воздействий их друг на друга целесообразно. Это декомпозиция абстракции одной абстракции и получение на ее основе двух с более высокой степенью связности.

Сообщения являются абстракциями воздействия (рис. 6.2). Они могут отсылаться одной или сразу нескольким абстракциям. Сообщения передаются по структурным связям между программными абстракциями. Они устанавливаются между абстракциями при представлении структуры технической системы. Структурные связи статичны – они не содержат в себе информации о направлении воздействия элементов, одно- или двунаправленным оно является. На основе этих связей формируются каналы транспорта сообщений.

Такая организация обеспечивает отсылку сообщений только тем абстракциям, которым они нужны. Каналы передачи сообщений определяются взаимными конструктивными связями элементов технической системы. В решения локальных задач моделирования, реализуемых абстракциями, входят значения величин, характеризующих состояния соседних элементов, тех, с которыми связан рассматриваемый элемент. Эти значения, характеризующие состояние и его изменения соседних элементов, передаются в сообщениях. С точки зрения моделирования элементы связываются через граничные условия. Таким образом, определяется содержание сообщений (рис. 6.3).

Частота взаимодействия абстракций измеряется количеством сообщений между ними в единицу времени. Использование существующих моделей разработки может значительно увеличить объемы информации, передаваемые между абстракциями, частоту взаимодей-

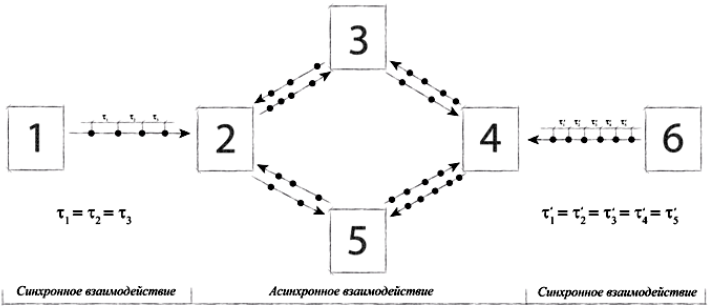


Рис. 6.2. Представление взаимодействия программных элементов

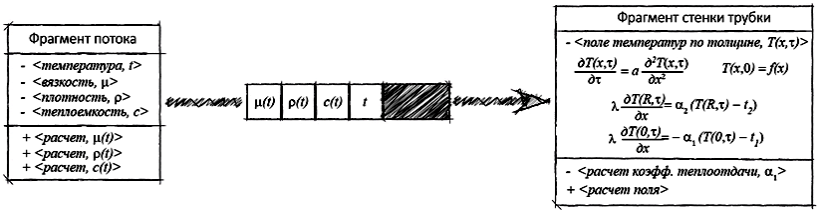


Рис. 6.3. Содержание сообщения на примере взаимодействия потока теплоносителя со стенкой канала

ствия и накладные расходы. Разработка на основе модели абстракций технической системы повышает уровень связности модулей и, следовательно, сокращает степень их зацепления, объемы передаваемой между абстракциями информации и частоту взаимодействия. Использование этой модели исключает возникновение сервисных сообщений. В программной системе остаются только те сообщения, которые соответствуют взаимодействиям элементов в технической системе. Доля полезной информации в содержании этих сообщений возрастает. Здесь исключается необходимость в каждой абстракции предусматривать возможности анализа всего многообразия сообщений, циркулирующих в системе. Аналогично тому, как в технической системе взаимодействие возникает только между ее элементами, так и в программной системе сообщения возникают между абстракциями.

6.2. ТИПЫ СООБЩЕНИЙ ДЛЯ ОБМЕНА ИНФОРМАЦИЕЙ МЕЖДУ АБСТРАКЦИЯМИ ЭЛЕМЕНТОВ ТЕХНИЧЕСКОЙ СИСТЕМЫ

Взаимодействие элементов технической системы длится непрерывно, в течение всего времени ее работы. Изменение во времени количественных характеристик взаимодействия элементов может быть представлено гладкой функцией. Ввиду того, что программная система реализуется на цифровых вычислительных машинах, представление взаимодействия в ней дискретно.

Выравнивание длительности сообщения до длительности взаимодействия элементов технической системы, которое в большинстве случаев совпадает со временем ее работы, обеспечивает постоянное взаимодействие абстракций и возможность определения параметров взаимодействия элементов технической системы во времени. Аналогично

тому, как при взаимодействии элементов технической системы, с течением времени меняются количественные характеристики взаимодействия, сообщения также должны обеспечивать доставку новых данных с требуемой периодичностью.

Это может быть достигнуто посредством последовательных сообщений, отсылаемых с определенной периодичностью. Сообщения, составляющие эту последовательность, рассматриваются как количественные характеристики этого взаимодействия. Каждое сообщение передает информацию о состоянии абстракции в момент времени, соответствующий отправке сообщения.

Обычно каждое сообщение несет в себе полноценный заголовок, обеспечивающий корректную его доставку и целостность. В этом случае присутствует избыточность данных и их дублирование. В каждом сообщении передается часть информации, которая является одинаковой для всех сообщений одной последовательности. Проектирование каналов транспорта сообщений на основе структурных связей между абстракциями, выполненных в соответствии с принципами модели абстракций технической системы, позволяет исключить дублирование информации и сократить трафик. Для описания каждого воздействия формируется отдельный канал передачи сообщений. Каждый канал обмена сообщениями формируется на основе структурных связей между абстракциями. Таким образом, автоматически учитывается принцип, что взаимодействие между элементами технической системы происходит в соответствии с их конструктивными связями. Общие данные, которые должны содержаться в заголовках сообщений и обеспечивать его корректную доставку и целостность, представляются на уровне канала. Все каналы обмена сообщениями статичны и неизменны в течение всего времени работы программной системы, как и конструктивные связи между элементами технической системы. Изменения могут возникнуть только в случае изменения конструкции технической системы в процессе ее работы. Каждый канал формируется только между двумя абстракциями и реализует транспорт сообщений только в одну сторону, от одной абстракции к другой. Каналы обеспечивают транспорт сообщений только от одной абстракции к другой. Конец канала на стороне получателя можно рассматривать как абстракцию границы элемента технической системы, на который оказывается воздействие, а данные, поступающие к ней в сообщениях, – как условия на границе. По одному каналу всегда циркулирует очень ограниченное количество типов сообщений, чаще всего вообще один.

Поэтому нет необходимости идентифицировать поступающие сообщения, обеспечение его целостности гарантирует канал. Это позволяет совсем отказаться от заголовков сообщений, направляя получателю чистые данные. Таким образом, протокол обмена сообщениями реализуется на уровне структуры программной системы.

Построение сообщений в соответствии с принципом структурного и динамического соответствия программной и технической систем значительно упрощает алгоритмы работы абстракций. Исключаются вычислительные операции, выполняемые абстракциями, направленные на анализ сообщений. При обмене обычными сообщениями такие ситуации возникают регулярно при поступлении каждого сообщения. Получатель анализирует его и в зависимости от типа сообщения переключается в соответствующий режим работы. При большой скорости поступления сообщений объем вычислений, связанных с анализом типов поступающих сообщений, может конкурировать с объемом полезных вычислений, направленных непосредственно на решение задачи. Переключение режимов работы абстракций также сопряжено с большим объемом дополнительных вычислений, связанных в основном с необходимостью создания и инициализации переменных, используемых абстракцией в требуемом режиме. Этот тип вычислений также может конкурировать по объему с полезными. В случае организаций сообщений как абстракций воздействия абстракции изначально выводятся на режим работы, который неизменно продолжается столько же времени, сколько функционирует программная система. Это исключает необходимость частого анализа поступающих сообщений и частого переключения режимов работы абстракций.

Абстракция отсылает информацию о своем состоянии тем абстракциям, с которыми она связана. Периодичность генерации сообщений соответствует времени, которое затрачивается на расчет ее нового состояния. При этом пересчет профиля целевых характеристик, который характеризует состояние абстракции, может осуществляться или нет. Расчет нового состояния профиля не выполняется, если с поступившими данными он изменяет профиль в пределах требуемой точности.

Простое вычисление нового состояния абстракции на основе аналитических решений не требует большого времени и больших вычислительных ресурсов. Поэтому вычисление состояния абстракции может осуществляться каждый раз, как только приходит ей сообщение об изменении состояния соседних, связанных с ней абстракций. Если абстракция связана больше чем с одной абстракцией, то здесь возникает

неопределенность, если информация о состоянии приходит от разных абстракций, приходит в разное время. Есть два варианта. Первый – абстракция для расчета своего нового состояния ждет, когда от всех связанных абстракций придет информация о их актуальном состоянии. Второй вариант, когда она не ждет этого и запускает цикла расчета на основе новой информации от одной абстракции и старой информации от других абстракций. Метод выбирается в зависимости от требований к реализации решаемой задачи.

Первый вариант, когда абстракция ждет информации от всех, взаимодействующих с ней абстракций, предпочтительней. Как только все сообщения пришли, запускается цикл расчета состояния. После расчета состояния цикл повторяется снова. Такой алгоритм для рассматриваемого класса задач работоспособен, потому что все абстракции имеют приблизительно равное время расчета своего состояния. Те абстракции, которые не меняют своего состояния со временем, но обеспечивают воздействие на другие абстракции, генерируют сообщения о воздействии в соответствии с внутренним таймером.

Для каждой абстракции реализуется единый цикл, который ожидает сообщения от связанных абстракций. Так как условием запуска расчета является поступление от всех абстракций, цикл один. Он ожидает, когда по всем каналам в абстракцию будут доставлены сообщения. После того как все сообщения получены, запускается цикл расчета. Запускать расчет в параллельном потоке нет необходимости, так как в любом случае, пока не будет рассчитано новое состояние, расчет не будет проводиться. Распараллеливание необходимо в данном случае только для ускорения процесса расчетов.

Альтернативным может быть вариант, когда сообщения от воздействующих абстракций не поставляют данные о своем состоянии. Они присылают управляющие сообщения, которые говорят связанным абстракция о том, что их состояние изменилось. Остальные абстракции после этого начинают расчет своего состояния. Необходимые для расчета данные они сами запрашивают у тех абстракций, с которыми они связаны в тот момент времени, когда они нужны. В тот момент, когда абстракции необходимы данные, которыми располагает другая, связанная с ней абстракция (например, для расчета температуры потока может потребоваться температура стенки, которая рассчитывается параллельно другой абстракцией), генерирует сообщение, которое запрашивает необходимые данные. Этот тип обмена информацией, в отличие от рассмотренного выше, всегда предполагает ответ на

сообщение. Сообщения, которые несут в себе информацию о воздействии, никогда не требуют ответа. Эти сообщения всегда требуют ответ. Он всегда следует последовательно за запросом. Этот вариант предполагает усложнение системы сообщений, изменение структуры каналов транспорта сообщений, которая позволит с натяжкой рассматривать сообщение как абстракцию воздействия.

ЗАКЛЮЧЕНИЕ

Рассмотренный метод позволяет строить программные системы в терминах технической системы, области ее использования и назначения, учитывать на уровне структуры программной системы особенности их функционирования. Тот факт, что формируются абстракции и семантика на основе объектов технического комплекса и предметной области, позволяет разработчикам, и особенно пользователям, воспринимать себя как непосредственно работающих с элементами предметной области. Это определяет основные преимущества методологии – повышение эффективности процесса разработки приложений, возможности более тесного вовлечения экспертов прикладной области в процесс разработки, повышение эффективности функционирования программной системы и вычислений, возможность строить приложения широкого спектра использования – автономные приложения для настольных компьютеров, интерактивные приложения на основе транзакций, встроенные системы, системы пакетной обработки, системы виртуальной/дополненной реальности, системы для моделирования и симуляции, системы сбора данных, web-приложения и т.д.

При этом вычисления не реализуют в прямом виде алгоритм решения общей математической задачи. Вместо этого реализуется сценарий работы программной системы, основанный на функционировании ее отдельных составляющих и связей между ними. Каждая составляющая отдельно решает локальную математическую задачу, описывающую состояние отдельного элемента технической системы.

Дальнейшее развитие методологии возможно в направлении повышения степени формализации процесса проектирования и функционирования программных систем до уровня, когда возможно использование автоматизированных инструментов разработки, быстрого проектирования и макетирования. При высокой степени формализации процесса проектирования возможна разработка специализированного проблемно-ориентированного языка программирования для создания программных абстракций технических систем.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. **Туголуков, Е. Н.** Математическое моделирование технологического оборудования многоассортиментных химических производств : монография / Е. Н. Туголуков. – М. : Изд-во «Машиностроение», 2004. – 100 с.
2. **Sommerville, I.** Software Engineering / I. Sommerville. – 10th Edition. – Pearson Education Limited, 2016.
3. **Booch, G.** Object-oriented analysis and design with applications / G. Booch. – Thrid Edition. – Pearson Education Inc., 2007.
4. **Mall, R.** Fundamentals of software engineering / R. Mall. – PHI Learning Pvt. Ltd., 2018.
5. **Budd, T.** Understanding Object-oriented programming with Java / T. Budd. – Pearson Education, 2002.
6. **Braude, E. J.** Software engineering: modern approaches / E. J. Braude, M. E. Bernstein. – Waveland Press, 2016.
7. **Parallel** programming: concepts and practice / B. Schmidt et al. – Morgan Kaufmann, 2017.
8. **Kirk, D. B.** Programming massively parallel processors: a hands-on approach / D. B. Kirk, W. H. Wen-Mei. – Morgan kaufmann, 2016.
9. **Walton, P.** Integrated software reuse: management and techniques / P. Walton, N. Maiden (ed.). – Routledge, 2018.
10. **Design** patterns: Entwurfsmuster als Elemente wiederverwendbarer objektorientierter Software / E. Gamma et al. – MITP-Verlags GmbH & Co. KG, 2015.
11. **Richards, M.** Software architecture patterns / M. Richards. – O'Reilly Media, Incorporated, 2015.

СОДЕРЖАНИЕ

Научное издание

АЛЕКСЕЕВ Сергей Юрьевич

СТРУКТУРНО-ПАРАМЕТРИЧЕСКИЙ СИНТЕЗ ПРОГРАММНЫХ СИСТЕМ МОДЕЛИРОВАНИЯ ТЕХНОЛОГИЧЕСКОГО ОБОРУДОВАНИЯ ХИМИЧЕСКОЙ ПРОМЫШЛЕННОСТИ НА ОСНОВЕ АНАЛИТИЧЕСКИХ РЕШЕНИЙ ЗАДАЧ МАТЕМАТИЧЕСКОЙ ФИЗИКИ

Монография

Редактор Л. В. К о м б а р о в а

Инженер по компьютерному макетированию М. Н. Р ы ж к о в а

ISBN 978-5-8265-2074-1

Подписано в печать 22.08.2019.
Дата выхода в свет 30.09.2019.
Формат 60×84/16. 6,28 усл. печ. л.
Тираж 400 экз. (1-й з-д 50). Заказ № 84

Издательский центр ФГБОУ ВО «ТГТУ»
392000, г. Тамбов, ул. Советская, д. 106, к. 14.
Телефон (4752) 63-81-08.
E-mail: izdatelstvo@admin.tstu.ru

Отпечатано в типографии ФГБОУ ВО «ТГТУ»
392008, г. Тамбов, ул. Мичуринская, д. 112А
Телефон (4752) 63-07-46
E-mail: tipo_tstu68@mail.ru