**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ**

**(государственный университет)**

**ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД  
В ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКЕ  
И ИМИТАЦИОННОМ МОДЕЛИРОВАНИИ**

**Магистерская диссертация студента 436 группы ФАКИ**

**Евдокимова Алексея Витальевича**

**Научный руководитель: член-корреспондент РАН,  
доктор физико-математических наук, профессор  
Холодов Александр Сергеевич**

**Рецензент: доктор физико-математических наук, профессор Петров Игорь Борисович**

**Москва 2000**

**Оглавление**

1. Введение 3

1.1. Цель работы и её актуальность 3

1.2. Решаемые задачи и научная новизна работы 4

2. Объектно-ориентированная структура моделей 6

2.1. Постановка задачи и предпосылки для её решения 6

2.2. Объектно-ориентированная методология моделирования 12

2.3. Пример использования: моделирование организма человека 15

2.4. Резюме 20

3. Объектно-ориентированные численные методы 21

3.1. Постановка задачи и обзор существующих работ 21

3.2. Объектная интерпретация понятий вычислительной математики 22

3.3. Объектное представление конкретных численных методов 33

3.4. Пример использования: объектно-ориентированная библиотека численных методов для задач гидромеханики и массопереноса (теплопереноса) 43

3.5. Резюме 46

4. Многокомпонентные базы данных как средство поддержки методологии вычислительного эксперимента 47

4.1. Постановка задачи и предпосылки для её решения 47

4.2. Содержание многокомпонентного подхода к БД 54

4.3. Пример использования: архитектура базы данных обобщённой модели 61

4.4. Резюме 66

5. Заключение 67

# Введение

## Цель работы и её актуальность

При численном моделировании сложных систем возникает ряд проблем, которые требуют сочетания подходов, сложившихся в совершенно разных областях науки. Прежде всего, это касается огромного потенциала вычислительной математики, который для своей реализации сейчас всё сильнее требует привлечения компьютерных наук. В настоящее время появилась неприятная тенденция превращения вычислительной математики из прикладной дисциплины в чисто теоретическую, – точно так же, как это произошло с аналитической математикой, когда сложность решаемых задач превысила возможности расчётов на бумаге. Чем более эффективными и сложными становятся численные методы, тем больше вероятность, каждый из них будет применён ровно один раз – при подготовке соответствующей диссертации.

Интеллектуальные затраты на программную реализацию методов во многих случаях настолько превосходят затраты на их теоретическую разработку, что становится неочевидным, какую из этих двух частей одной работы считать научной, а какую – чисто практической. В связи с этим понятно, почему даже высокотехнологичные программные разработки сейчас всё реже основываются на современных численных методах, – выгоднее довести до уровня технологии простой и менее эффективный подход (например, подход, принятый в имитационном моделировании), чем сложный и более эффективный. Если специалисты по вычислительной математике не позаботятся о совместимости численных методов с компьютерными науками, то специалисты по компьютерным наукам (а тем более разработчики программ) позаботиться об этом будут не в состоянии.

Совместимость методов вычислительной математики и компьютерных наук имеет три основных аспекта. Во-первых, сложные численные методы оправданно использовать на практике, если они существуют не только в виде текстов научных статей и даже не только в виде кода программ (который по степени формализации не слишком отличается статей). Как показывает опыт создания вычислительных библиотек [1,2], методы должны быть представлены в виде строго формализованного кода, который можно напрямую использовать в прикладной программе. Во-вторых, применение достижений компьютерных наук имеет самостоятельное значение в вычислительной математике, где имеется очевидная тенденция к созданию гибридных методов на основе некоторого набора «элементарных» методов, а также к совместному решению задач всех известных (изученных по отдельности) типов. Имеются ввиду хорошо исследованные в компьютерных науках способы многократного использования одних и тех же методов при решении близких задач и при создании новых методов. Кроме того, так как численные методы обычно тесно связаны с реализующими их алгоритмами, совершенствование алгоритмов (являющееся предметом компьютерных наук) с неизбежностью способствует развитию вычислительной математики. Наконец, третий аспект совместимости вычислительной науки с информатикой касается развития самих компьютерных технологий для обеспечения потребностей численного моделирования. В частности, для хранения данных вычислительных моделей имеет смысл использовать более развитые подходы, чем приняты в современных системах управления базами данных, работающих с данными более простых предметных областей.

Таким образом, данная работа направлена преодоление разрыва, образовавшегося между традиционной вычислительной математикой и феноменально быстро развивающимися компьютерными науками. С одной стороны, этот разрыв препятствует применению достижений вычислительной математики к решению современных задач, которое немыслимо без компьютерных технологий. С другой стороны, использование некоторых идей компьютерных наук в области численных методов может стимулировать развитие вычислительной математики как таковой (а потребности численного моделирования могут стимулировать развитие компьютерных наук).

## Решаемые задачи и научная новизна работы

Конечно, данная работа не претендует не описание всех областей взаимодействия вычислительной математики с компьютерными науками и, тем более, на окончательное решение всех возникающих при этом взаимодействии проблем. Со стороны компьютерных наук акцент сделан на объектно-ориентированный анализ, хотя в главе 4 затрагиваются также проблемы теории баз данных. С точки зрения вычислительной математики в главе 3 анализируются, прежде всего, общие характеристики численных методов, а конкретные алгоритмы, в их объектно-ориентированной формулировке, приводятся лишь в качестве иллюстрации эффективности общего подхода. Кроме этих двух дисциплин, в работе рассматривается область знания, которая обычно идентифицируется термином «имитационное моделирование в узком смысле» (в широком смысле имитационным моделированием можно называть любое моделирование на компьютере, включающее, в том числе, и численные методы). При этом конкретная имитационная модель (модель организма человека) в главе 2 также рассматривается только в качестве примера, а основное внимание уделено анализу возможностей сочетания имитационного моделирования и вычислительной математики на основе объектно-ориентированного подхода.

При достижении поставленной цели – сблизить некоторые подходы, принятые в вычислительной математике и информатике, – в работе решаются следующие задачи:

1. оптимизируется объектное представление элемента вычислительной модели и предлагаются расчётные алгоритмы, решающие проблемы такого представления;
2. анализируются свойства численных методов для объектных моделей;
3. разрабатывается подход к созданию объектно-ориентированной библиотеки численных методов, пригодной к быстрому изменению и расширению (а также к наглядному представлению);
4. создаётся принципиально новый подход к базам данных, предназначенный для хранения данных моделей и соответствующий методологии вычислительного эксперимента;
5. на основе этого подхода предлагается схема базы данных обобщённой модели, допускающая хранение не только числовых значений, но также структуры модели и численных методов.

Таким образом, основной задачей данной работы является не исследование какой-либо конкретной математической модели или численного метода расчёта этой модели, а создание теоретической основы для среды разработки широкого класса моделей. Под средой разработки понимается относительно универсальное инструментальное средство, позволяющее автоматизировать процесс создания и использования сложных моделей. Настоящая работа исследует возможности соединения в нем достоинств имитационного и математического моделирования на основе объектно-ориентированного подхода.

На защиту магистерской диссертации автором выносятся следующие положения:

1. Объектно-ориентированный подход (ООП) может служить основой для сочетания достоинств математического и имитационного моделирования.
2. На основе ООП можно создавать библиотеки численных методов, которые легче использовать в моделях и проще развивать, чем процедурные библиотеки.
3. ООП даёт возможность визуального редактирования моделей и библиотек.
4. Основанные на объектах численные методы инвариантны относительно типа задачи; эффективно рассчитывают неоднородные и распределённые системы.
5. Объектные алгоритмы могут быть более быстрыми, чем процедурные.
6. Для хранения данных моделей целесообразно использовать многокомпонентные СУБД, основанные на идеях ООП и соответствующие методологии моделирования.

# Объектно-ориентированная структура моделей

## Постановка задачи и предпосылки для её решения

### Объектно-ориентированный подход как основа для сочетания математического и имитационного моделирования

При создании моделей сложных систем часто возникает необходимость в применении мощного математического аппарата одновременно с использованием преимуществ имитационного моделирования. Под математическим аппаратом здесь понимается современные методы численного решения дифференциальных уравнений (как обыкновенных, так и в частных производных). Слово «имитационное моделирование» используется в узком смысле, прежде всего как способ формализации знаний экспертов определённой предметной области с последующим решением полученной системы дискретных или простейших дифференциальных уравнений.

Математические модели обычно игнорируют проблему оптимального представления знаний экспертов о моделируемой системе и сосредотачиваются на задаче обработки этих знаний, которые выражены в непонятной для экспертов математической форме. Ниже перечислены некоторые недостатки математического моделирования, следующие отсюда:

1. большое количество неизвестных исходных данных, приводящее к огромным затратам на отладочные вычислительные эксперименты;
2. проблема верификации моделей, связанная с тем, что выходные данные моделей не соответствуют структуре предметной области;
3. недостаточный учёт нелинейных взаимодействий в системе, для которых известны лишь эмпирические оценки;
4. сложность развития моделей, обусловленная, прежде всего, тем, что излишняя степень их формализации не позволяет работать с ними эксперту – не математику.

С другой стороны, в рамках имитационного подхода практически невозможно рассматривать распределённые модели. Кроме того, с ростом сложности даже «точечных» моделей размерность соответствующих им систем уравнений начинает превосходить разумные пределы, а свойственная имитационным моделям простота теряется.

Математический и имитационный подходы имеет смысл сочетать, к примеру, при комплексном моделировании организма человека, – именно эта задача будет использоваться для иллюстрации предмета данной главы в разделе 2.3. Для такой сложной системы, как человек, без современных методов вычислительной математики не обойтись при моделировании физических процессов, например, при расчёте переноса веществ кровеносной (дыхательной) системой и при расчёте скоростей течения крови (воздуха), на котором происходит этот перенос. Однако не менее важным является использование в моделях сложных систем неточных эмпирических зависимостей – регуляторных закономерностей в человеческом организме, – и для этого необходим имитационный подход.

Конечно, было бы хорошо соединить эффективность математических методов с такими свойствами имитационных моделей, как наглядность, близость к предметной области и, главное, простота развития. Однако проблема заключается в отсутствии общепринятой основы для построения комплексных моделей, которые бы сочетали достоинства обоих подходов. Эта основа должна быть одинаково удобной как для имитационного представления структуры моделируемой системы, так и для её формализации и обработки математическими методами.

Процедурно-ориентированные библиотеки вычислительных алгоритмов на низкоуровневых языках типа FORTRAN и C [3] не подходят в качестве такой основы хотя бы потому, что задачу для них нельзя представить наглядно и следует формулировать на языке уравнений. Кроме того, какими бы универсальными ни были процедуры библиотек, часто требуется расширять их возможности (или, наоборот, сужать эти возможности для упрощения использования, для уменьшения затрат машинного времени и т.п.). Однако если библиотека реализована на процедурном (структурном) языке программирования, для незначительного изменения в ней какого-либо численного метода необходимо существенно корректировать его код. При этом неизвестно, будут ли после такой корректировки правильно работать программы, использовавшие данный численный метод в его первоначальном варианте. Ещё большие трудности возникают, когда появляются новые версии библиотеки – после этого нужно либо провести заново все необходимые изменения в коде, либо пользоваться старыми, но зато хорошо адаптированными к решаемым задачам, версиями численных методов.

В противоположность процедурным библиотекам вычислительной математики, наглядность и возможность развития свойственна средствам имитационного моделирования, подобным встроенному в математический пакет MatLab средству Simulink [10] и ведущим начало от высокоуровневых языков моделирования типа Dynamo и Stella. Однако эти средства пригодны лишь для не слишком сложных в математическом смысле задач (по причинам, описанным в разделе 2.1.3), и принятый в них подход не может служить основой для сочетания имитации с современной вычислительной математикой. В частности, он не подходит к решению уравнений в частных производных.

Данная глава (глава 2) имеет целью показать, что в качестве основы для совместной реализации математического и имитационного способов моделирования очень хорошо подходит объектно-ориентированный подход (ООП). А именно, что построенные на объектно-ориентированной основе модели

1. как имитационные модели, наглядны, пригодны к быстрому расширению и учёту большого числа разнородных закономерностей;
2. как математические модели, могут эффективно решать сложные, в том числе пространственно-распределённые задачи.

После того, как будут обоснованы эти утверждения и будет построена объектно-ориентированная структура обобщённой модели, получает смысл проведённая в главе 3 адаптация численных методов для работы с такими структурами и исследование их свойств.

|  |  |
| --- | --- |
| **Комплексные модели** | |
|  | |
| **Математические модели** | **Имитационные модели** |
|  |  |
| Моделирование сложных, в том числе распределённых, систем | Наглядность структуры  Возможность быстрого развития |
|  | |
| Привязка процедур к данным, которые они обрабатывают | Агрегация  Наследование |
|  |  |
| **Объектно-ориентированный подход** | |
|  | |
| **Рис. 2.1.** ООП как основа для сочетанияматематического и имитационного моделирования | |

### Понятия ООП в применении к моделированию

Любая объектная модель базируется на понятии объекта как совокупности данных, к которым привязаны некоторые процедуры их обработки (методы). При этом каждое данное из этой совокупности также, в свою очередь, может являться объектом (понятие агрегации) (см. рис. 2.2). Если в процедурной модели процедуры манипулируют пассивно хранящимися данными, то в объектной сами данные – объекты – взаимодействуют между собой. Множество похожих объектов с одним и тем же набором свойств и методов образует класс, который, собственно, и является минимальным блоком программы, аналогом процедуры.

Согласно общепринятой терминологии, объектно-ориентированная модель отличается от просто объектной возможностью наследования классов объектов. При наследовании существующего класса (суперкласса) получается новый класс, называемый подклассом (см. рис. 2.2). Объекты подкласса имеют те же свойства, и могут участвовать в тех же взаимодействиях, что и объекты суперкласса. При этом подкласс может иметь новые свойства и новые методы, а также изменять методы своего суперкласса. Наследование облегчает изменение и расширение объектно-ориентированных систем.

|  |  |
| --- | --- |
| Наследование классов  Агрегация объектов |  |
| **Рис. 2.2.** Понятия объектно-ориентированного подхода | |

В таблице 2.1 перечисленные выше понятия рассматриваются подробнее, причём акцент делается на их применение в моделировании.

**Таблица 2.1**

Использование объектно-ориентированного подхода в задачах моделирования.

|  |  |
| --- | --- |
| *Суть ООП состоит в следующем:*   1. Программа представляет собой совокупность объектов, объединяющих данные и методы их обработки.    1. Смысл объекта (поведение) отделён от его реализации (структуры) за счёт того, что методы могут не содержать кода, а данные имеют определённую область видимости.    2. Данные объектов (поля) также могут быть объектами. 2. Каждый объект относится к определённому типу (классу), т.е. является его экземпляром. 3. Тип (класс) может наследовать часть данных и методов от других типов (классов), то есть быть их подтипом.   *Примечание* к пункту 1.2: в одном и том же поле могут храниться объекты определённого типа и всех его подтипов. | *Суть объектно-ориентированной*  *модели состоит в следующем:*   1. Модель представляет собой совокупность элементов, объединяющих параметры модели и методы их расчёта.    1. Смысл элемента (набор параметров и основные методы их обработки) отделён от его реализации (от вспомогательных данных и методов).    2. Данные элементов (параметры, ссылки на соседние элементы и т.д.) также могут быть объектами. 2. Каждый элемент относится к определённому типу (классу), и вычислительные алгоритмы обычно разделены на несколько методов разных классов. 3. Тип (класс) может наследовать часть данных и методов от других типов (классов), причём данные и аргументы методов могут быть не только определённого типа, но и всех его подтипов |
| *ООП имеет следующие преимущества*  *перед другими подходами:*   1. компактность программ, уменьшающая их стоимость и время их разработки; 2. возможность повторного использования классов, программ и проектов; 3. эффективное разделение работы между дизайнерами и программистами; 4. быстрая адаптация программ к изменяющимся задачам и требованиям; 5. уменьшение риска разработки сложных программ за счёт интеграции и тестирования их компонентов на протяжении всего времени разработки, а не на последнем её этапе. 6. ориентация на человеческое восприятие мира, позволяющая программировать даже без понимания деталей работы компьютера. | *Объектно-ориентированные модели*  *имеют следующие преимущества*  *перед процедурно-ориентированными:*   1. компактность модели; 2. возможность повторного использования частей модели; 3. эффективное разделение работы над численными методами, содержанием и наглядным представлением модели; 4. быстрая адаптация модели к изменению структуры и численных методов; 5. уменьшение риска разработки сложных моделей за счёт интеграции и тестирования их частей на протяжении всего времени разработки, а не на последнем её этапе; 6. ориентация на предметную область модели, позволяющая моделировать без глубокого знания численных методов и, возможно, без умения программировать |

### Существующие объектные средства моделирования

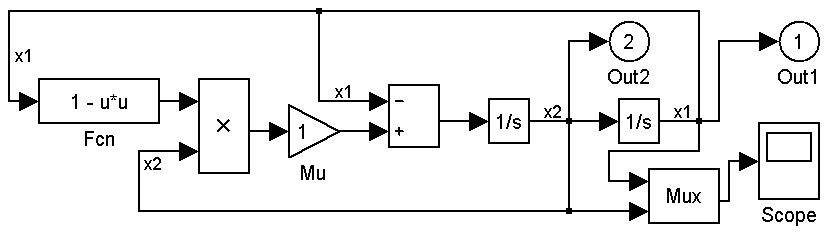
Таким образом, любая более-менее сложная математическая или имитационная модель является удобным предметом для применения объектно-ориентированного подхода. Конечно, этот факт был давно замечен и использован в большом количестве программных средств для моделирования [7,8]. Однако в большинстве таких средств объектно-ориентированным является только внешнее представление элементов модели, которое перед началом расчётов компилируется в код процедурно-ориентированного языка программирования или просто преобразуется в параметры готового кода. Например, блоки многих имитационных моделей преобразуются в переменные системы обыкновенных дифференциальных уравнений, а связи между ними – в функции её правых частей.

Такой подход, безусловно, решает часть проблем стандартных моделей, связанных с их недостаточной наглядностью и с негибкостью по отношению к изменениям структуры. В частности, в моделях появляются чётко ограниченные блоки, которые можно быстро удалять, добавлять и использовать в совершенно других моделях. Тем не менее, изменение самих блоков ввиду их процедурно-ориентированной реализации весьма затруднительно, а методы расчёта таких моделей вообще не изменяемы (не имеют объектно-ориентированного представления даже на внешнем уровне). Кроме того, трансляция модели из графически интерпретируемых объектов в параметры процедур является достаточно сложной задачей, и этот факт тормозит развитие методов математической обработки модели (для каждого нового метода необходимо также реализовывать алгоритмы трансляции, а иногда и вводить новые объекты).

Существенно меньше распространены средства моделирования, в которых не только сама структура модели является объектно-ориентированной, но и работающие с ней численные методы. В этом направлении существуют два основных подхода. Первый обычно реализуется в задачах CFD (вычислительной газо- и гидродинамики) и других разделов механики и электродинамики сплошных сред [6]. Считается, что объектная методология имеет два основных преимущества по отношению к таким задачам – приспособленность к параллельным вычислениям и к решению задач со сложной геометрией. При описании параллельных вычислений в качестве объектов рассматриваются процессоры, обменивающиеся между собой данными в ходе расчётов, а при описании геометрии – однородные области интегрирования (часто примитивной формы), взаимодействующие между собой на границах. Для реализации этих преимуществ достаточно простейшей объектной модели, даже без наследования. Как следствие, проблему изменения и расширения численных методов данная трактовка понятия объекта не решает.

С точки зрения имитационного моделирования важным преимуществом использования объектов является возможность наглядного представления структуры модели и её визуального редактирования (то есть изменения без программирования). Обычно такая возможность реализуется путём построения модели из отдельных параметров, которые можно связывать между собой с помощью разных функций. Функции и параметры являются объектами многочисленных типов (классов). В частности, функции могут быть заданы таблично или с помощью формулы, могут зависеть не только от значений параметров, но и от их производных, а также зависеть от них с запаздыванием. В наиболее развитых системах имитационного моделирования [7,9,10] среди этих функций имеются достаточно сложные дифференциальные и дискретные операторы, что формально позволяет графически описать любую систему уравнений. К сожалению, такой системой не могут пользоваться эксперты предметной области, далёкие от математики. Например, для решения с её помощью дифференциального уравнения нужно понимать смысл преобразования Лапласа, а представление этого уравнения довольно громоздко (см. рис. 2.3).

**Уравнение Ван дер Поля**



**Рис. 2.3.** Пример объектного представления параметров и функций модели (Simulink).

## Объектно-ориентированная методология моделирования

### Построение моделей из элементов

С точки зрения объектно-ориентированного анализа, упомянутые выше области интегрирования слишком велики и сложны, чтобы их рассматривать как элементарные объекты модели, и методы этих объектов всё равно придётся организовывать в процедурно-ориентированном стиле. В частности, класс области интегрирования должен содержать, по крайней мере, несколько массивов для сеточных функций, соответствующие значения которых (элементы массивов с одинаковым индексом) связаны между собой гораздо сильнее, чем элементы одного массива. С другой стороны, отдельные параметры и функции, наоборот, слишком малы как объекты, так как для создания из них хоть сколько-нибудь сложной модели требуются тысячи и миллионы таких объектов. Напрашивается вывод, что в качестве элемента модели следует рассматривать объект промежуточного размера между областью и параметром, и этот объект ниже называется просто элементом.

Элемент должен представлять собой набор семантически связанных параметров, к которым привязаны определённые алгоритмы вычислительной математики, а точнее – те части алгоритмов, которые рассчитывают именно эти параметры. При формировании объектно-ориентированной библиотеки численных методов математику достаточно запрограммировать некоторое количество классов элементов, которые соответствуют либо типичным математическим объектам (к примеру, системе обыкновенных дифференциальных уравнений), либо типичным объектам рассматриваемой предметной области. Например, в гидромеханике или физиологии человека такими типичными объектами являются трубки (сосуды), резервуары, обменники и т.д. После того, как созданы классы элементов, из них, как из конструктора, эксперт по данной предметной области может строить разнообразные модели безо всякого программирования (см. раздел 2.3.2).

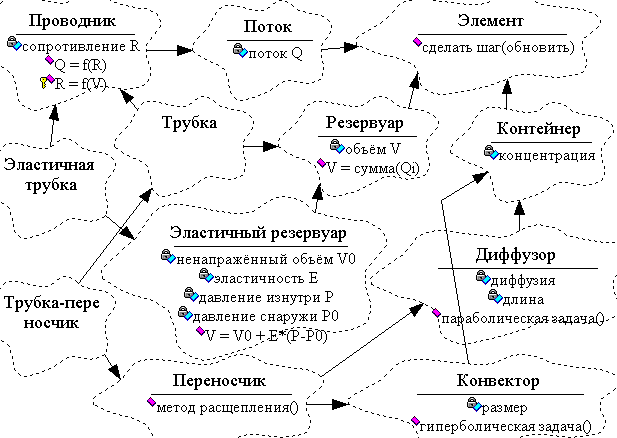
Элементы рассчитывают значения своих параметров на основе связей с другими элементами. При этом связи являются чисто декларативными, а не процедурными, благодаря чему эксперт создаёт их, не задумываясь о том, с помощью каких математических методов эти связи выражаются. Конечно, для отдельных нетипичных соотношений между параметрами элементов можно также использовать обычные функциональные связи, поскольку в силу нетипичности для них нецелесообразно программировать специальные классы элементов.

### Создание типов элементов на основе наследования

На первый взгляд может показаться, что система моделирования, основанная на заранее запрограммированных классах элементов, не является такой гибкой, как существующие средства имитационного моделирования. Однако объектно-ориентированный подход позволяет обойти эту проблему за счёт наследования классов элементов. Если один класс элемента наследуется от другого, то он использует имеющиеся у того параметры и методы их расчёта, при этом добавляя какие-либо свои параметры и методы.

Например, пусть исходно был запрограммирован класс проводника, то есть элемента, который характеризуется сопротивлением и (по)током через него и предназначен для расчёта электрической цепи или, что то же самое, гидродинамической системы жёстких трубок. Если требуется учесть растяжимость трубок, достаточно унаследовать свойства и методы класса проводника, добавив к ним свойство объёма и процедуру расчёта сопротивления трубки через этот объём. (см. рис. 2.4) Далее класс растяжимой трубки (то есть сосуда, если пользоваться терминологией предметной области – физиологии) можно использовать в качестве родительского класса для эластичных, пластичных и других сосудов, различающихся моделью растяжения.

А теперь предположим, что требуется решать не только эллиптическую задачу расчёта потоков и давлений в системе трубок, но и параболическую задачу переноса веществ или тепла в этой системе. Понятно, что методы расчёта конвективного, диффузионного и конвективно-диффузионного переноса должны существовать в специальных элементах-переносчиках независимо от элементов, связанных с моделированием механики сосудов. (см. рис. 2.4) Чтобы использовать эти методы для задачи переноса в сосудах, достаточно создать элемент, наследующийся от сосуда и от переносчика одновременно, причём никакого нового программного кода создавать не требуется. В объектно-ориентированных языках, не поддерживающих множественное наследование, то же самое достигается путём одиночного наследования от одного класса (например, от сосуда) и агрегации экземпляра другого класса (например, переносчика).



**Рис. 2.4.** Пример иерархии наследования классов модели (в объектной нотации Г. Буча).

Таким образом, наследование даёт возможность быстрого развития не только моделей, но и заложенных в них численных методов. При этом упростить и формализовать изменение кода программной реализации методов можно настолько, чтобы осуществлять его с помощью визуального редактора. Данная возможность отсутствует как в случае трактовки области интегрирования в качестве основного объекта модели (визуальное редактирование её методов не слишком отличается от обычного и не может быть формализовано), так и в случае трактовки в этом качестве параметра или функции (к ним не могут быть привязаны «интеллектуальные» методы, поэтому в них нечего редактировать).

## Пример использования: моделирование организма человека

Как было отмечено выше, ООП особенно необходим при совместном использовании разнородных методологий (например, математического и имитационного моделирования) и при описании неоднородных (например, дискретно-непрерывных) систем. Примером такой неоднородной системы, требующей привлечения как современных методов вычислительной математики, так и неточных эмпирических зависимостей, является комплексная модель организма человека.

Безусловно, речь не идет о модели, полностью описывающей все происходящие в организме процессы (даже если предположить, что для всех них могут быть созданы адекватные модели и найдены все необходимые исходные данные, то для компьютерной реализации не хватит ресурсов современной вычислительной техники). Однако многие задачи (например, прогнозирование последствий приёма человеком лекарств) всё-таки требуют одновременного рассмотрения большинства функциональных систем организма. При этом для одних систем (например, для кровообращения) необходимо детальное рассмотрение их анатомической структуры на базе распределённых моделей, для других (например, для выделительной системы) подходят обыкновенные дифференциальные уравнения, а для третьих (например, для нервной регуляции) достаточно использования дискретных моделей и эмпирических зависимостей. Поскольку элементы всех этих систем взаимодействуют между собой, без их объектного представления создать комплексную модель организма практически невозможно.

Ниже данное утверждение иллюстрируется при описании некоторых блоков квазистационарной модели организма, название которой отражает тот факт, что она не рассматривает быстротекущие и колебательные процессы, сосредотачиваясь на расчёте равновесных или осреднённых по времени значениях параметров.

### Описание некоторых блоков модели и проблем их расчёта

Основой комплексной модели организма человека является модель кровеносной системы: во-первых, она служит посредником между всеми другими системами, осуществляя транспорт веществ в организме, во-вторых, параметры кровообращения являются основными показателями состояния человека. Поскольку кровеносное русло включает сосуды существенно различных размеров, их совместное моделирование требует комбинации различных математических подходов. Достаточно изотропную сеть мелких сосудов корректнее описывать моделью сплошной среды, то есть с помощью дифференциальных уравнений в частных производных. Это позволяет не вводить для таких сосудов лишних анатомических характеристик типа их размеров, сводя их свойства к одному параметру непрерывной задачи – проницаемости ткани по отношению к крови. С другой стороны, модель макроскопических сосудов сводится к системе с сосредоточенными параметрами, гидродинамическая часть которой рассчитывается по алгебраическим уравнениям (Кирхгоффа), а транспортная часть – по обыкновенным дифференциальным уравнениям. Проблема в том, что при попытке учесть в рамках такой дискретной модели большое число сосудистых поколений размерность сосредоточенной системы превысит размерность системы сеточных уравнений, аппроксимирующей непрерывную задачу в мелких сосудах.

Следовательно, даже для моделирования кровеносной системы как таковой (т. е. без её связей с другими системами организма) возникает задача совместного расчёта разнородных (дискретных и непрерывных) моделей. Как показано в [11], чисто математические способы решения проблемы связывания дискретной модели кровообращения с непрерывной моделью приводят к громоздким выкладкам, какими бы хитроумными они ни были (а их процедурно-ориентированная реализация требует не менее громоздких алгоритмов). Поэтому значительно более эффективной представляется объектная формулировка обеих частей модели, унифицирующая представление дискретных элементов и узлов пространственной сетки.

Такой же вывод можно сделать при рассмотрении задачи совместного расчёта параметров сердца и сосудистой системы. Конечно, работа сердца описывается гораздо более сложными уравнениями (см. [11], а также табл. 3 приложения), чем закон пропорциональности между потоком и градиентом давления, характерный для сосудов, поэтому, казалось бы, нельзя рассчитывать их единообразно. Однако если решать задачу отдельно для обоих желудочков сердца при фиксированных свойствах сосудов (интегральном сопротивлении кругов кровообращения) и отдельно – для сосудов при фиксированных параметрах желудочков (артериальном давлении и потоке), то возникает, по крайней мере, два вложенных итерационных процесса (один из них – например, процесс решения задачи в сосудах – должен сходиться на каждой итерации другого процесса – например, процесса расчёта сердечных параметров). Очевидно, эффективность таких расчётов очень низка, и её можно повысить, если объединить итерации схем сердечных и сосудистых уравнений в один процесс. Это возможно только за счёт объектно-ориентированной трактовки элементов сосудистой системы и сердца и их наследования от общего суперкласса (см. раздел 3.4.2). Одновременно такой подход увеличивает гибкость модели: в процедурно-ориентированный алгоритм из двух вложенных процессов весьма затруднительно вставить расчёт дополнительных активных элементов кровеносной системы (например, желудочков сердца, мышечного, дыхательного или других «насосов»).

Ещё более очевидные проблемы, пригодные к решению с помощью объектно-ориентированного подхода, возникают при моделировании транспорта веществ в организме. Вещества могут попадать в организм через дыхательную или пищеварительную систему (в первом случае – за счет конвекции и диффузии, во втором – с участием мышц органов желудочно-кишечного тракта), проникать в кровь (диффузионным путём или за счёт всасывания), транспортироваться кровеносной системой, испытывая по дороге разнообразные превращения и абсорбируясь выделительной системой; наконец, вещества могут разными способами потребляться в органах и тканях. Уравнения для всех этих процессов довольно похожи, однако разные пространственные размерности соответствующих моделей (для пищеварения и выделения – точечные модели, для кровообращения и дыхания – ветвящиеся одномерные и квазитрёхмерные) не позволяют использовать для их расчёта одни и те же процедуры. Особенно сложно при процедурном подходе формализовать условия на границах раздела систем, число типов которых намного больше, чем число самих систем. Однако если представлять организм человека в виде совокупности связанных между собой элементов, то при разработке их методов не нужно заботиться о том, с какой элемент (какой системе принадлежит, является ли «граничным») связан с данным элемент. Кроме того, размерность системы фактически не влияет на объектные методы моделирования (подробнее это описано в разделе 3.2.3), а за счёт наследования алгоритмы расчётов сходных физических процессов в разных функциональных системах организма можно реализовывать только один раз.

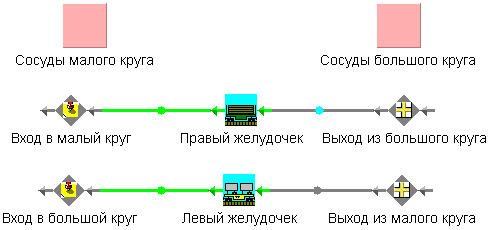
Таким образом, даже поверхностный обзор некоторых блоков модели организма человека показывает, что для её реализации необходим объектно-ориентированный подход, – прежде всего, ввиду комплексного характера модели, неоднородности её элементов и сложности связей между ними.

### Объектно-ориентированное представление структуры модели

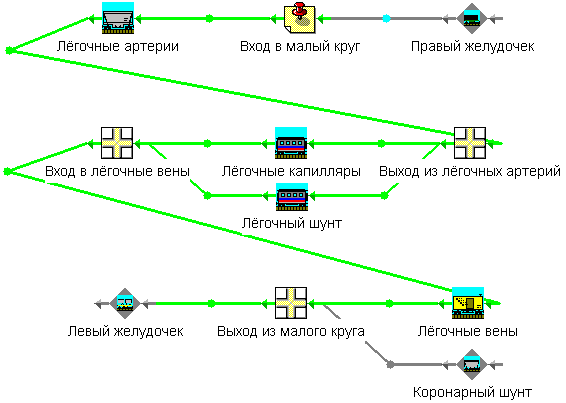
В данном разделе в виде схем представлены результаты объектно-ориентированного анализа некоторых функциональных систем организма человека (более подробно они описаны в [12]). Их модели представляют собой совокупность элементов – объектов различных классов, – соединённых между собой направленными связями, смысл которых описан в разделе 2.2.1. Структура самих элементов подробно приведена в разделе 3.4.1 при рассмотрении объектно-ориентированной библиотеки численных методов, на которой основана данная модель. Классы элементов обозначены на схемах символическими изображениями, соответствие с которыми определяется таблицей 2.2. Каждая схема соответствует некоторой подсистеме организма. Если изображение какого-либо элемента уменьшено по сравнению с остальными и помещено в ромбическую рамку, то он относится к другой подсистеме, являясь связующим звеном между ней и подсистемой рассматриваемой схемы.

Следует заметить, что рисунки 2.5-2.7 не являются теоретическими схемами структуры модели. Они созданы с помощью специального графического редактора расчётной программы, то есть изображённые на рисунках объекты (элементы) создаются при инициализации модели и в ходе моделирования выполняют соответствующие их классам численные алгоритмы. При создании элементов используется, помимо приведённой структурной информации и декларативных связей, также количественная информация и функциональные связи. Функциональные связи (зависимости между параметрами элементов) также имеют прямое отношение к рассматриваемой здесь структуре модели и вводятся в неё с помощью графических редакторов, однако за недостатком места они не приводятся.

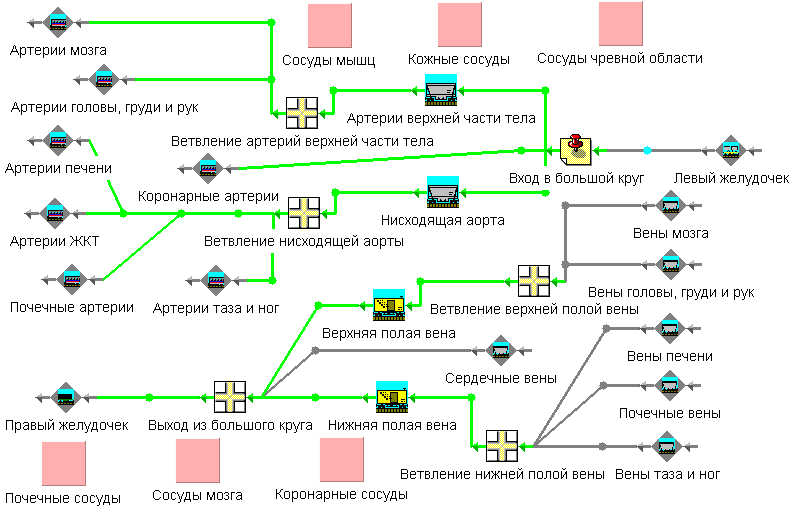
По той же причине ниже описывается лишь структура некоторых блоков модели (относящихся, прежде всего, к кровеносной системе), что достаточно для иллюстрации принятого подхода к моделированию. Распределённые блоки модели на схемах также отсутствуют, но лишь потому, что к настоящему времени они ещё не реализован интерфейс, позволяющий наглядно представлять сложные области интегрирования и связывать их с другими элементами модели.



**Рис. 2.5.** Общая структура модели кровеносной системы.



**Рис. 2.6.** Структура модели малого круга кровообращения.



**Рис. 2.7.** Структура модели большого круга кровообращения.

**Таблица 2.2**

Условные обозначения некоторых классов элементов модели организма человека на рис. 2.5-2.7. Номера соответствуют таблицам 1-4 приложения.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *№* | *Элемент* | *Рис.* | *№* | *Элемент* | *Рис.* |
| 27 | Точка конвекции |  | 31 | Градиент-конвектор |  |
| 28 | Узел конвекции |  | 32 | Насос-конвектор |  |
| 29 | Конвектор |  | 33 | Сосуд-конвектор |  |
| 30 | Проводник-конвектор |  | 34 | Эластичный конвектор |  |

## Резюме

В данной главе рассмотрены проблемы математического и имитационного моделирования, которые, как показано, целесообразно решать на основе объектно-ориентированного представления структуры моделей. Путём анализа подходов, принятых в существующих объектных средствах моделирования, выбрана оптимальная трактовка понятия объекта – элемента вычислительных моделей. Эта трактовка позволяет быстро создавать, легко развивать и наглядно представлять не только сами модели, но и методы расчёта, на которых эти модели основаны. Предложенный общий подход проиллюстрирован на примере объектно-ориентированной структуры конкретной модели организма человека. Данная глава является подготовительной по отношению к главе 3 и не претендует на большую научную новизну, лишь уточняя понятия и методики, имеющиеся в литературных источниках и существующих средствах имитационного моделирования.

# Объектно-ориентированные численные методы

## Постановка задачи и обзор существующих работ

Конечно, математические свойства численных методов не зависят от способа их реализации, и поэтому предметом данной главы отнюдь не является улучшение этих свойств путём их программирования на объектно-ориентированном (а не на процедурном) языке. Объектно-ориентированный подход в вычислительной математике нужен не столько сам по себе, сколько ради применения сложных численных методов к имитационным моделям, необходимость чего детально описана в разделе 2.1.1. Вычислительные алгоритмы на объектно-ориентированной основе являются более гибкими, пригодными к визуальной разработке и повторному использованию в рамках некоторой библиотеки. Эти достоинства, которые свойственны имитационному подходу к моделированию, ориентированному на наглядность и простоту развития моделей, были обсуждены в главе 2. Ниже обсуждается другое, менее очевидное, преимущество ООП – удобство моделирования систем, которые сложны не в имитационном, а в математическом смысле.

Проблемы реализации многих современных численных методов можно сравнить с проблемами, возникающими при отображении области сложной геометрической формы на прямоугольную сетку перед решением уравнений в частных производных (какой бы простой ни была исходная задача, при таком отображении она обычно усложняется). Точно так же при переходе от реальных (например, физических) объектов к математическим абстракциям обычно теряются многие их свойства, которые могли бы облегчить моделирование. В данной же работе предлагается решать задачи в их исходной объектной постановке, без отображения на математические шаблоны типа матриц систем уравнений.

Целью данной главы служит анализ эффективности объектно-ориентированного подхода с точки зрения вычислительной математики как таковой (без учёта «имитационных» свойств). При этом решаются проблемы оптимального объектного представления различных понятий данной науки, а также выбираются конкретные численные методы решения задач различных типов, которые лучше подходят для такого представления.

Безусловно, за рубежом существует большое количество работ по анализу применимости объектно-ориентированных языков программирования к решению вычислительных задач [2,4,5]. Основной упор в этих работах делается на преодоление проблемы понижения скорости арифметических операций над большими массивами данных (векторами, матрицами) при использовании таких языков. Другими словами, речь идёт об универсальных рекомендациях вычислителям по эффективному программированию стандартных для процедурных языков конструкций на объектных языках. В частности, в [4] рассматриваются проблемы объектного представления матрицы (вектора), сплошной адресации элементов матриц, разделения данных между матрицей и её подматрицами, уменьшения числа вызовов конструкторов этих объектов. Во многих работах, например, в [2] приводятся результаты сравнения производительности объектных и процедурных языков, и обосновывается выбор наиболее производительных компиляторов. От объектно-ориентированного подхода в таких работах остаётся очень мало; наоборот, целью авторов является приблизить свойства объектно-ориентированных языков к процедурным (процедурные сложно использовать в задачах, связанных, например, с пользовательским интерфейсом). В частности, не делается попыток отказаться от матричного представления данных многих численных методов. Поэтому сами алгоритмы численных методов остаются неизменными, и научных работ по их объектной трактовке не возникает.

Данная работа не претендует на анализ и оптимизацию производительности низкоуровневых вычислений на объектных языках программирования; в ней исследуются способы использования преимуществ объектов на значительно более высоком уровне. Прежде всего, такие преимущества появляются за счёт отказа от матричного представления многих математических задач. При этом, конечно, возникают проблемы реализации многих численных методов (своих для каждой задачи), основанных на таком представлении. Именно на поиск конкретных вычислительных алгоритмов и направлена данная работа, – в отличие от большинства работ по объектно-ориентированным вычислениям, которые рассматривают лишь языковые конструкции, общие для многих алгоритмов.

Естественным приложением научных работ данной тематики (в том числе и настоящей работы) является создание «объектно-ориентированных вычислительных библиотек». Несмотря на то, что их создано очень много, большинство предназначено для решения задач линейной алгебры, вычисления сложных функций и решения недифференциальных уравнений. Более сложные задачи, как обосновывается в [2], должны решаться самостоятельно пользователями этих базовых алгебраических библиотек. В данной работе предлагается иная концепция объектно-ориентированной библиотеки (см. 3.4).

## Объектная интерпретация понятий вычислительной математики

### Системы уравнений

В разделе 2.2.1 было обосновано, что элемент вычислительной модели должен представлять собой набор семантически связанных параметров, к которым привязаны расчётные алгоритмы. Другими словами, элемент трактовался как объект (в смысле ООП) и соответствовал либо типичным математическим объектам, либо типичным объектам предметной области. Ниже как раз обсуждается соответствие элемента универсальным математическим понятиям, без привязки к конкретной предметной области. Примером чисто математического элемента является система уравнений, а «семантически связанными параметрами», объединяемыми элементом, в этом случае являются искомые переменные и коэффициенты системы. Если коэффициенты системы – параметры элемента – не являются постоянными, а зависят от решения системы, то эту зависимость также можно представлять как объект или даже совокупность объектов (функций); однако такое представление описывалось выше и не имеет прямого отношения к рассматриваемому здесь математическому объекту.

На базе одной и той же системы уравнений могут, как правило, решаться задачи нескольких типов. Например, для системы обыкновенных дифференциальных уравнений может ставиться задача Коши или краевая задача. При решении обеих задач может использоваться один и тот же элемент, что является одним из основных преимуществ объектно-ориентированного представления численных методов. Более того, если для решения краевой задачи применять удобные алгоритмы (например, метод «стрельбы»), то не требуется создавать специальные методы элемента – достаточно в цикле использовать метод, отвечающий за решение задачи Коши.

Другое преимущество объектно-ориентированного подхода связано с простотой совместного решения уравнений различных типов. Например, в одной части моделируемой системы может происходить чисто конвективный перенос и решаться гиперболическое уравнение, а в другой – диффузионный перенос и, соответственно, параболическое уравнение. Данный подход инвариантен также относительно размерности задачи и позволяет рассматривать дискретно-непрерывные системы, то есть решать совместно алгебраические и дифференциальные уравнения.

Ещё одно математическое достоинство объектов заключается в их инвариантности относительно типа решаемой задачи. Является ли задача статической, динамической или вообще обратной задачей идентификации параметров модели – во всех этих случаях используются одни и те же расчётные классы элементов. Различие между этими задачами заключается только в алгоритме обновления состояния элементов, и этот алгоритм занимает всего несколько строчек кода. В случае динамической задачи обновление происходит вплоть до заданного времени, в случае статической – до получения заданной точности. В случае обратной задачи получение заданной точности происходит много раз в цикле, итерации которого соответствуют различным значениям искомых параметров модели, а изменение этих параметров происходит согласно какому-либо методу решения нелинейных уравнений.

Таким образом, основной элемент объектно-ориентированной модели – система уравнений – рассчитывает значения неизвестных системы (возможно, динамику их изменения во времени) по значениям своих коэффициентов (и правой части), взаимодействуя при этом с другими объектами, если коэффициенты или правая часть зависят от решения. Ниже принципиальные алгоритмы таких расчётов рассматриваются более подробно.

### Схемы решения уравнений

В соответствии с методологией вычислительной математики, для решения системы уравнений (даже если она является простейшей алгебраической) необходимо сначала перейти к её дискретному аналогу – схеме. В схему входят значения параметров элемента (неизвестные и коэффициенты системы уравнений) на нескольких последовательных временных слоях или на нескольких итерациях. При стандартной (процедурной) реализации схемы разным слоям (или итерациям) соответствуют различные переменные (точнее, массивы переменных), что приводит к высокой сложности алгоритмов, к трудностям в их изменении, в совместном использовании нескольких алгоритмов и т.п.

Этого можно избежать, если использовать объектно-ориентированный подход не только на уровне элементов, но и на более низком уровне – на уровне их параметров. Параметр, рассматриваемый как объект, объединяет несколько значений, которые соответствуют нескольким моментам времени (или нескольким итерациям). У этого объекта имеются методы, вычисляющие разностные аппроксимации производных параметра различных порядков, методы, эффективным (по быстродействию) способом смещающие историю изменения параметра на один шаг по времени (на одну итерацию), и т.д. Объектно-ориентированная трактовка параметра имеет большое значение при создании компактных и развиваемых библиотек численных методов, однако с точки зрения предмета данной главы она важна и для описания расчётных схем.

Система уравнений, к какому бы типу она ни относилась, может быть аппроксимирована несколькими схемами (семействами схем), и с каждой схемой, в свою очередь, может быть ассоциировано несколько вычислительных алгоритмов. Как следствие, у соответствующего системе объекта (элемента) может быть несколько методов, которые по-разному решают одну и ту же задачу. Однако такой подход неудобен тем, что разные схемы могут иметь различный набор вспомогательных параметров, которые к тому же не должны иметь ничего общего с параметрами исходной системы уравнений. В связи с этим предлагается выделить вспомогательные параметры из элемента в отдельный объект – схему. Одновременно решается проблема слишком большого количества методов у элемента: различные варианты реализации каждой схемы программируются в соответствующих им объектах, а элементу остаётся лишь вызывать методы схемы, передавая им свои параметры. Таким образом, элемент, содержащий данные об исходной задаче, не хранит и не использует информацию о решающем эту задачу численном методе.

Разделение элемента и схемы имеет и другое преимущество: данные схемы могут быть полностью формализованы (т. е. приведены к форме, удобной для расчётов и математического анализа), в то время как параметры элемента могут сильнее отражать структуру предметной области (например, потребности имитационного моделирования запрещают делать их безразмерными). Численные методы, реализуемые схемой, совершенствуются гораздо реже, чем изменяются элементы, которые могут различаться для каждой предметной области, поэтому их разделение на разные объекты весьма целесообразно.

Выше обсуждалась внутренняя структура системы уравнений – элемента вычислительной модели – и его взаимодействие с такими объектами как зависимости, параметры, схемы. Однако немалое значение имеет также взаимодействие нескольких систем уравнений между собой. Конечно, было бы хорошо, если вся совокупность параметров модели могла бы быть формализована в виде одной системы уравнений какого-либо типа. Однако сложные модели обычно приводят ко многим системам уравнений разных типов; кроме того, даже если тип систем одинаков, не всегда целесообразно объединение их в одном расчётном элементе. К примеру, слабо связанные системы обыкновенных дифференциальных уравнений, матрицы Якоби которых имеют сильно отличающиеся спектры, гораздо эффективнее решаются по отдельности, поскольку для них целесообразно выбирать различный шаг по времени и даже различные схемы.

Проблема оптимизации решения больших систем уравнений путем их расщепления на несколько лучше обусловленных систем меньшей размерности пока не рассматривалась в вычислительной математике. Это связано с тем, что реализовать совместное решение этих систем с помощью процедурного подхода крайне затруднительно. Дело в том, что системы связаны между собой через общие данные, в то время как процедурным образом представленные алгоритмы их решения должны работать независимо. Использование объектного подхода позволяет избежать сложных приёмов разделения одних и тех же данных между процедурами решения разных систем: ничто не мешает одному и тому же параметру входить в несколько различных элементов – систем уравнений (а также в элементы других типов). Самый удобный среди процедурных приём разделения данных существует в языке FORTRAN (оператор EQUIVALENCE), однако он бессилен, если разные системы уравнений решает одна и та же процедура (проблема в том, что процедура, в отличие от класса, не может иметь нескольких экземпляров).

Помимо проблемы разделения данных между системами, объектный подход решает задачу выбора последовательности их переходов к следующему моменту времени. Поскольку шаг по времени может различаться для разных систем (более, того, он может меняться со временем), системы (и любые другие элементы) необходимо упорядочивать по времени в виде очереди, что возможно только за счёт их объектного представления. Головной элемент очереди осуществляет переход к следующему моменту времени, после чего он переносится в то место очереди, которое отстоит от её головы на величину текущего шага по времени для этого элемента.

Таким образом, на языке объектов не только эффективно реализуются не только общепринятые схемы решения систем уравнений (понимаемые как связи между переменными на нескольких временных слоях или итерациях), но и макро-схемы совместного решения нескольких систем.

### Пространственные сетки и сеточные шаблоны

Выше был описан процесс взаимодействия между элементами, совместно решающими систему уравнений через разделение её на несколько слабо связанных систем. В вычислительной математике существуют также пространственно-распределённые задачи, для которых разделение на подсистемы (каждая из которых соответствует одному сеточному узлу) является стандартным, менее искусственным и единственно возможным способом решения. В таких задачах есть условие локальности связей между сеточными узлами, благодаря которому элементу, рассчитывающему этот узел, достаточно взаимодействовать только с небольшим числом соседних узлов (узлов шаблона).

Ниже для краткости сам узел пространственной сетки (а не система уравнений, соответствующая ему) фигурирует в качестве расчётного элемента. Связям между такими элементами соответствует отношение упорядоченности сеточных узлов по координатам (в случае использования неструктурированных сеток это есть отношение соседства сеточных узлов) – см. рис 3.1.

Упомянутая выше локальность связей означает, что значение какого-либо параметра в рассчитываемом сеточном узле определяется только значениями в узлах шаблона (например, ). Поэтому любой численный метод оказывается привязанным к узлу, то есть к элементу, и для его реализации данный узел не обязан иметь информацию о пространственно-удалённых узлах, с которыми у него нет связей. Это полностью соответствует объектной модели. Если же формулировать аппроксимацию уравнений в частных производных на процедурно-ориентированном языке системы линейных уравнений, то получается следующее: сначала искусственно создаётся проблема в виде огромной разреженной матрицы этой системы, а затем эта проблема преодолевается с помощью каких-нибудь специальных способов хранения и обработки этой матрицы в памяти компьютера.

i.4

i.4.k

i

i.2

i.1

i.3

i-1

i, j

j-1

j+1

i+1

i+2

**Рис. 3.1.** Соответствие между традиционным и объектным представлением пространственной сетки

Взаимодействие между элементами, решающими систему уравнений в частных производных, – сеточными узлами – носит принципиально иной характер, нежели взаимодействие между элементами-системами. Системы связаны между собой только через общие параметры, которые для одной системы являются искомыми, а для других входят в коэффициенты или правые части уравнений. Сеточные узлы отнюдь не всегда должны использовать напрямую параметры, принадлежащие к соседним узлам шаблона. Для получения необходимой информации о соседнем узле (которая может представлять собой сложную комбинацию его параметров) часто удобнее вызывать его метод. Вообще, вычислительные алгоритмы могут быть рассредоточены по методам нескольких разнотипных элементов, которые вызывают друг друга в процессе расчётов.

Взаимодействие сеточных узлов во времени также более интенсивно, чем описанные выше для элементов-систем последовательные переходы на следующий слой. Исследование устойчивости алгоритмов возможно лишь, если все сеточные узлы (по крайней мере, все узлы одной области интегрирования) обновляют своё состояние с одним и тем же шагом по времени. Поэтому элементы-узлы, в отличие от слабо связанных систем, не могут быть упорядочены по времени в виде очереди, и возникает проблема последовательности их обновления. Для решения этой проблемы естественно создавать специальный объект – область интегрирования, – который ответственен за порядок перехода сеточных узлов на следующий временной слой (или на следующую итерацию). Однако важен не только сам порядок перехода, но и то, какие значения параметров соседних элементов (до перехода или после) использует сеточный узел при расчёте своих параметров. Поэтому каждый узел должен также хранить информацию о том, обновлялось ли его состояние на данном временном шаге (итерации). Приведённое рассуждение касается не только систем с распределёнными параметрами, но и любых подсистем, взаимодействие элементов которых слишком интенсивно, чтобы позволять им в хаотическом порядке обновлять своё состояние.

Таким образом, при решении уравнений в частных производных ещё более эффективно, чем в других разделах вычислительной математики, используется объектно-ориентированный подход. Это проявляется как в более активном взаимодействии основных расчётных элементов (сеточных узлов), так и в появлении в дополнение к элементам и их параметрам ещё одного, более высокого, уровня объектов – уровня подсистем (областей интегрирования).

### Ресурсоёмкость

Важным свойством, характеризующим любой численный метод, является его требовательность к машинным ресурсам. Эти ресурсы делятся на два основных типа – ресурсы процессорного времени и ресурсы оперативной памяти. Считается, что использование объектно-ориентированных языков программирования ведёт к большим затратам и того, и другого (по сравнению с процедурными языками) [4]. Следовательно, приемлемость объектов в качестве вычислительных единиц подлежит обоснованию.

Действительно, с каждым объектом необходимо связывать некоторую метаинформацию, для размещения которой расходуется дополнительная память. Относительное количество такой памяти можно было бы существенно уменьшить, если минимальными объектами в расчётной программе являлись бы элементы (даже при решении распределённых задач элементы – сеточные узлы – содержат намного больше полезной информации, чем той, которая необходима лишь для их существования в виде объектов). Однако, как показано в разделе 3.2.2, минимальным объектом должен являться параметр, содержащий всего несколько вещественных чисел; поэтому сокращать расходование памяти путём увеличения размера объектов вряд ли целесообразно.

Объектно-ориентированный подход может экономить память вычислительных программ за счёт совершенно других механизмов. Дело в том, что в процедурно-ориентированных программах часто встречаются массивы большой длины, заведомо достаточной для хранения всех параметров задачи, хотя часто выделенная под них память используется лишь на несколько процентов. Даже если язык поддерживает задание длины массива во время выполнения программы (в FORTRAN такая возможность появилась только в стандарте 1990-го г.), не всегда этим можно пользоваться, поскольку длина массива во время выполнения может неоднократно меняться. И уж совсем непреодолимые затраты памяти возникают при использовании многомерных массивов, когда для разных значений их медленно меняющегося индекса быстро меняющийся индекс должен изменяться в различных пределах. ООП решает эти проблемы автоматически, поскольку массив является объектом. В частности, элементы многомерного массива – тоже массивы – могут иметь различную длину. Помимо самих массивов, в объектно-ориентированные языки обычно включаются классы – оболочки массивов, которые могут динамически (оптимальным образом) менять их длину в зависимости от реального количества элементов в них. (Правда, динамическое изменение длины плохо отражается на быстродействии.)

Если же посмотреть на проблему выделения излишнего количества памяти с точки зрения численных методов, то можно заметить малое количество (всего несколько процентов) ненулевых элементов матриц Якоби, которые встречаются при решении больших систем обыкновенных дифференциальных уравнений. Поскольку объектно-ориентированные численные методы не используют матричное представление систем уравнений, они позволяют (в разы и даже в десятки раз) сократить количество необходимой памяти для их хранения. Одновременно уменьшается число лишних арифметических операций (умножений на нуль, приращений переменной цикла и т.д.).

Использование объектно-ориентированных языков увеличивает расходование ресурсов процессора, но в них существуют механизмы, максимально снижающие этот эффект. Действительно, наследование и полиморфизм уменьшают скорость вычислений, поскольку при вызове какого-либо метода у элемента происходит выбор среди одноимённых методов его класса и родительских классов (этот выбор называется динамическим, или поздним, связыванием). Однако быстродействие можно повысить, например, путём ещё одного наследования от данного класса и объявления полученного класса конечным (не способным иметь подклассов). В языке Java это позволяет компилятору не использовать динамическое связывание методов полученного класса, а также осуществлять автоматическую замену вызова методов их кодом, если он не слишком велик. В некоторых языках (С++) такая замена (inline-подстановка) может быть явно затребована программистом.

Следует также заметить, что с точки зрения быстродействия объектно-ориентированные численные методы имеют и свои преимущества. Дело в том, что в стандартной процедуре решения задачи математической физики значения параметров представляются в виде массивов большой длины (по числу сеточных узлов), доступ к которым осуществляется в одном или нескольких циклах по узлам. Поэтому время расходуется не только на расчёт, но и на сложение инкрементируемых переменных циклов с числами 1, -1, 2, -2 и т. д. (см. рис. 3.1) и на доступ к значениям параметров в массиве (вычисление адреса памяти по индексу массива). В случае объектной реализации численных методов в элементах хранятся явные ссылки на соседние элементы, у которых можно явно запросить значения их параметров, безо всякой арифметики адресов. При этом требуется больше памяти, но зато исключаются значительные затраты на целочисленную арифметику.

### Некоторые специфические понятия

Наконец, необходимо рассмотреть то, как согласуются с объектно-ориентированным подходом менее универсальные понятия вычислительной математики, чем описанные выше понятия системы уравнений, схемы и сетки. Прежде всего, существуют многообразные свойства расчётных схем: явность, устойчивость, монотонность, консервативность, гибридность и т.д. Свойства устойчивости и монотонности являются чисто математическими и не имеют отношения к реализации схем; поэтому объектно-ориентированный подход не привносит в эти понятия ничего нового. Почти то же можно сказать о консервативности (свойство, имеющее смысл для гиперболических уравнений), однако практика показывает, что схемы, являющиеся удобными для своей объектной реализации, обычно одновременно являются консервативными. Это связано с тем, что объектные схемы всегда ближе к физике задачи, и реализующие их элементы соответствуют реальным объектам.

Гибридные схемы решения уравнений в частных производных (прежде всего гиперболических, поскольку именно они имеют требующие гибридизации разрывные решения) несколько проще реализуются с помощью объектно-ориентированного подхода, чем с помощью процедурного. Благодаря тому, что схема как объект в разделе 3.2.2 была отделена от самого расчётного элемента (в данном случае – от сеточного узла), для совместного использования в каждом узле двух схем, совершенно не нужно изменять ни код элемента, ни код схем. Достаточно создать класс гибридной схемы (один класс для всех возможных пар элементарных схем!), которая будет в соответствии с градиентом решения интерполировать результаты, получаемые по каждой из двух схем.

Несколько менее впечатляющим выглядит применение объектно-ориентированного подхода к реализации неявных схем. С одной стороны, самый простейший объектно-ориентированный численный метод для уравнений практически любого типа (который не следит за тем, на каком временном слое или на какой итерации берутся значения параметров) фактически является полуявным (треугольным), в то время как простейший процедурный численный метод – явный. С другой стороны, решение какой-либо вспомогательной системы уравнений на верхнем слое объектно-ориентированным способом достаточно сложно. В случае обыкновенных дифференциальных уравнений вспомогательная система уравнений нелинейная, и единственный разумный способ её решения состоит в привязке к основному расчётному элементу – к дифференциальной системе – ещё одного элемента (алгебраической системы). На каждом шаге своего решения дифференциальная система должна изменять некоторые коэффициенты алгебраической, после чего итерировать её схему до получения заданной точности.

Реализация неявных схем в случае пространственно-распределённых задач возможна без использования вспомогательных элементов, но лишь ценой существенного усложнения структуры и функций основных элементов – сеточных узлов. Большинство алгоритмов решения уравнений на верхнем временном слое не являются локальными, в отличие от самих схем для уравнений в частных производных, поэтому теряются некоторые преимущества объектного подхода по отношению к таким уравнениям. К примеру, алгоритм прогонки (даже монотонной прогонки) требует двукратного обращения к методам одного и того же сеточного узла: первый раз для вычисления прогоночных коэффициентов, а второй – для расчёта искомых параметров в этих узлах. Код метода прогонки получается более компактным и развиваемым, если выделить прогоночные коэффициенты в отдельный элемент, связанный с каждым сеточным узлом. Прямой ход прогонки состоит в цепочке вызовов методов расчёта именно этих вспомогательных элементов, и только на обратном ходу прогонки сами сеточные узлы последовательно вызывают друг у друга методы обновления своего состояния. Следует заметить, что распространение простейшего алгоритма прогонки на более сложные задачи (многомерные, с циклическими граничными условиями и т.д.) приводит к серьёзным трудностям, а объектно-ориентированная немонотонная прогонка вообще, по-видимому, невозможна. Если же решать уравнения на верхнем слое итерационными методами, то возникает необходимость в описанных выше вспомогательных элементах алгебраической системы уравнений, привязанных, в данном случае, к каждому сеточному узлу. Тем не менее в разделе 3.3.2.2 показывается, что и в случае неявных схем ООП может стимулировать развитие численных методов.

Выше речь шла о применимости объектов к некоторым типам схем; осталось рассмотреть их эффективность с точки зрения некоторых видов пространственных сеток. Отсутствие регулярной структуры сетки вполне соответствует объектному подходу, поскольку он (в отличие от процедурного) также не предполагает упорядочение узлов по координатам в массивах. Более того, так как для нерегулярных сеток характерно использование в разных узлах шаблонов разного размера и схем различного типа, объектный подход выглядит намного эффективнее процедурного в плане хранения коэффициентов этих схем. Он также облегчает алгоритмы поиска наилучших узлов шаблона.

Методология адаптивных сеток, применяющаяся для получения решений с большими градиентами, плохо соотносится с принятым в данной работе подходом. Если сеточные узлы просто перемещаются в пространстве (скапливаясь в областях «разрывов»), а их число остаётся неизменным, то никаких неудобств представление узлов в виде объектов не вызывает. Правда, и преимуществ никаких это представление не несёт: на каждом шаге по времени состояние узлов меняется полностью (это не согласуется с их объектной природой), причём за это отвечают не сами узлы (практически не взаимодействующие друг с другом), а внешний алгоритм. Если же число узлов со временем может изменяться, то объектный подход приведёт к недопустимым затратам ресурсов: нельзя конструировать (и уничтожать) тысячи объектов на каждом шаге по времени.

Описанные выше отношения между объектными трактовками различных понятий вычислительной математики резюмируются на рис. 3.2, а изложенные соображения о применимости объектно-ориентированного подхода к этим понятиям – в табл. 3.1.

|  |  |
| --- | --- |
|  | Уровень последовательности вычислений |
| Уровень расчётов по формулам |
| Уровень взятия и присвоения значений |
| **Рис. 3.2.** Отношения агрегации между классами объектов, соответствующими основным понятиям вычислительной математики | |
| *Примечание* 1. Рисунок соответствует объектной нотации UML, цифры около конца связи между классами означают её кратность (число объектов, которое может содержать агрегирующий объект) | |
| *Примечание* 2. Кратность связи класса элемента с самим собой (0..\*) указана для случая однотипных элементов (например, узлов сетки); при использовании сложных методов элемент может иметь ещё одну принципиально иную связь с элементом другого типа | |

**Таблица 3.1**

Применимость ООП к некоторым понятиям вычислительной математики

|  |  |  |
| --- | --- | --- |
| **Понятие** | **Степень соответствия ООП** | |
| Система уравнений | Выше среднего | ООП наиболее полезен, если совместно решаются несколько связанных систем, особенно разных типов |
| Схема | Средняя | объектное представление схемы нужно лишь для создания легко развиваемых библиотек численных методов; объектное представление параметра схемы значительно сокращает код, но увеличивает ресурсоёмкость программ; объектное представление элементов (систем) позволяет упорядочивать их переходы к следующим моментам времени (или итерациям) |
| Пространственная сетка и шаблон | Высокая | объектное представление узлов сетки позволяет гораздо эффективнее использовать характерное для распределённых задач условие локальности связей между узлами |
| Гибридная схема | Выше среднего | ООП позволяет при гибридизации использовать ранее реализованные негибридные схемы, не изменяя их |
| Неявная схема | Ниже среднего | объектная реализация неявных схем столь же сложна, как и процедурная, а в случае пространственно-распределённых задач теряется локальность алгоритмов и многие связанные с ней преимущества ООП |
| Нерегулярная сетка | Высокая | нерегулярные сетки с помощью ООП реализовать даже проще, чем регулярные |
| Адаптивная сетка | Низкая | объектная реализация алгоритмов перемещения сеточных узлов слабо отличается от процедурной, а динамическое создание (удаление) узлов в случае использования ООП недопустимо |

## Объектное представление конкретных численных методов

### Обыкновенные дифференциальные уравнения

Все методы решения обыкновенных дифференциальных уравнений (ОДУ) делятся на одношаговые и многошаговые, и их объектная реализация существенно различается.

В принципе, при объектном представлении параметров очень удобно хранить в них любое требуемое число их значений на последовательных шагах по времени, что создаёт предпосылки для реализации многошаговых схем. Однако правая часть уравнений обычно состоит не только из параметров, но и из объектов-функций и объектов-операций, для которых хранение истории своего изменения во времени является избыточным. Поэтому целесообразно создавать специальные объекты (класса «параметр»), в каждом из которых хранить последовательность значений одного компонента вектора правой части. Кроме того, сложные задачи, для которых имеет смысл применять объектно-ориентированный подход, обычно приводят к жёстким системам уравнений, в связи с чем необходимо изменять шаг по времени в ходе расчётов, а к этому многошаговые схемы можно приспособить только за счёт существенного усложнения алгоритма. Такие задачи также часто содержат не только сосредоточенные, но и распределённые подсистемы, для расчёта которых многослойные схемы применяются редко. Поэтому использование многошаговых схем для ОДУ нецелесообразно и с точки зрения единообразия расчётов всех частей рассматриваемой системы.

С другой стороны, в пользу объектного представления таких схем можно привести тот факт, что по сравнению с процедурным представлением достаточно просто можно реализовать переход от одношаговых схем к многошаговым (это необходимо для «разгона» многошаговых схем на первых шагах) и обратный переход в процессе расчётов. Для этих переходов достаточно только поменять в объекте-системе ссылку на объект-схему. Следует также обратить внимание, что значения правой части системы ОДУ в случае использования многошаговых схем вычисляются при тех значениях аргументов, которые соответствуют решению системы: .

В случае одношаговых схем порядок их аппроксимации можно сделать выше первого только за счёт вычисления правой части при значениях аргументов, рассчитываемых по сложным формулам: , . При условии объектного представления аргументов (параметров) это приводит к необходимости присваивать им рассчитанные значения непосредственно перед вычислением правой части, после чего нужно «стирать» эти значения из истории изменения аргументов. К счастью, это можно делать чисто техническими способами (не пересматривая саму объектную модель): так как аргументы правой части являются одновременно параметрами элемента, представляющего систему уравнений, он может их изменять без особых проблем. В целом, использование объектно-ориентированных одношаговых методов решения ОДУ (схем типа Рунге-Кутты) предпочтительнее многошаговых (схем Адамса или Гира): не нужно «разгонять» метод на первых шагах и создавать специальные объекты для хранения динамики изменения правой части уравнения, легче изменять шаг по времени и связывать систему с другими расчётными элементами.

Ещё один эффективный метод решения ОДУ, связанный с переходом к продолженным системам (например, схемы Обрешкова), вряд ли имеет смысл рассматривать с точки зрения ООП, который целесообразно применять, прежде всего, для сочетания математических методов с имитационными моделями. Дело в том, что схемы типа Обрешкова предполагают аналитическое дифференцирование правой части системы, что невозможно в имитационных моделях, содержащих большое количество экспертных зависимостей.

Наконец, необходимо описать алгоритм реализации неявных схем решения ОДУ. Независимо от того, относятся ли они к одношаговым или многошаговым схемам, возникает необходимость решения на каждом шаге системы нелинейных алгебраических уравнений. Как было указано в разделе 3.2.5, для этого целесообразно привязать к каждому элементу, решающему ОДУ, элемент алгебраической системы. Если для каждой дифференциальной системы имеется своя алгебраическая, то начальное приближение для её решения никуда не исчезает в промежутке между шагами по времени дифференциальной системы, что повышает скорость сходимости (хотя и увеличивает расходование памяти). Кроме того, множественность алгебраических систем имеет смысл, когда не требуется на каждом шаге пересчитывать матрицу Якоби правой части исходной системы (с этой матрицей однозначно связана итерационная матрица алгебраической системы).

Таким образом, объектная реализация численных методов решения систем обыкновенных дифференциальных уравнений не приводит к непреодолимым трудностям, однако и не имеет принципиальных преимуществ. Предпочтение следует отдавать одношаговым схемам (типа Рунге-Кутты), хотя комбинирование их с многошаговыми схемами с помощью объектно-ориентированного программирования реализуется относительно просто.

Более оптимистичный вывод можно сделать при анализе дифференциально-разностных уравнений (точнее, дифференциально-разностных уравнений запаздывающего типа). Как было отмечено в разделе 3.2.2, объектная трактовка параметров расчётного элемента позволяет хранить в них совершенно разное количество их значений, относящихся к различным моментам времени. Соответственно, те параметры, входящие в дифференциально-разностное уравнение с запаздыванием, могут хранить историю своего изменения ровно такой длины, которая нужна для нахождения их значения в момент времени, отстоящий на величину максимального запаздывания от текущего момента. Для вычисления (интерполяции) значения в произвольный момент времени такие параметры должны также иметь ссылку на объект, хранящий моменты времени, которые соответствуют их истории (такой объект часто бывает один и тот же для многих параметров). Отсюда видно, что относительно простые дифференциально-разностные уравнения при условии применения ООП и при достаточном количестве памяти компьютера можно решать, не прибегая к сложным численным методам.

### Уравнения в частных производных

Анализ объектно-ориентированного представления методов решения уравнений в частных производных можно проводить согласно классификации этих уравнений на гиперболические, параболические и эллиптические. Однако не меньший интерес представляет сравнительный анализ объектных методов для этих типов уравнений, а также возможностей для использования для них одних и тех же объектов.

Многослойные схемы для уравнений в частных производных с точки зрения ООП имеют те же преимущества и недостатки, что и многошаговые схемы для обыкновенных дифференциальных уравнений, а также дополнительное достоинство, связанное с возможностью использования в шаблоне меньшего количества сеточных узлов, различающихся пространственными координатами. Объектная реализация многослойных схем немногим отличается от реализации двухслойных схем, которые описываются ниже.

#### Методы для гиперболических уравнений

Особенностью наиболее распространённых схем решения гиперболических задач является их явность, поскольку неявные схемы хуже описывают возникающие в таких задачах волновые процессы. Как было отмечено в разделе 3.2.5, явные схемы гораздо проще неявных реализуются с помощью объектов. Перед описанием алгоритма их реализации следует обратить внимание на тот факт, что при решении динамических уравнений (в отличие от эллиптических) схемы высоких порядков аппроксимации практически не используются. Более того, для гиперболических уравнений не существует линейных монотонных схем выше первого порядка аппроксимации (теорема Годунова), а гибридные схемы, позволяющие преодолеть это ограничение, включают схемы первого порядка. Поэтому необходимо рассмотреть, прежде всего, объектное представление двуслойных схем первого порядка аппроксимации для простейшего уравнения переноса в *n-*мерном пространстве (схемы для систем уравнений сводятся к ним преобразованием Римана).

Шаблон таких схем имеет, помимо рассчитываемого узла, *(n+1)* сеточных узлов, которые для монотонности должны служить вершинами фигуры, включающую точку (т. н. базовую точку) пересечения характеристики, которая проходит через рассчитываемый узел, с нижним слоем или с границей области интегрирования. Максимальную точность имеет схема на шаблоне, узлы которого находятся на минимальном расстоянии от базовой точки (доказательство данного утверждения, справедливого для шаблонов любого размера, в данной работе не приводится). При использовании процедурного похода для поиска ближайших к базовой точке сеточных узлов обычно каждый раз при построении схемы (в случае уравнения с переменными коэффициентами – на каждом шаге!) перебираются все узлы области интегрирования.

Объектно-ориентированный подход позволяет существенно увеличить быстродействие за счет того, что поиск для каждого узла можно вести направленно, начиная с ближайших к нему соседей. Список соседних узлов, упорядоченный по расстоянию, строится только один раз, а в процессе расчётов у каждого элемента списка вызывается метод, вычисляющий расстояние от него до передаваемой ему в качестве аргумента базовой точки, а также вызывающий такой же метод у своих соседей. Если в этом методе проверять условия приближения узлов к базовой точке, то *(n+1)* требуемых узлов находятся по кратчайшему пути. Конечно, и в процедурной программе для каждого узла можно хранить номера некоторого числа ближайших соседей (это число, кстати, фиксировано, в отличие от узлов-объектов). Однако подобный алгоритм в этом случае можно реализовать только через вызов сложной рекурсивной процедуры, требующей доступа к большому количеству не нужных ей данных и, главное, зависящей от размерности задачи. В данном же случае алгоритм поиска не зависит не только от размерности пространства, но и от численного метода (поскольку этот алгоритм содержится в элементе, а не в его схеме).

Схемы чётного (в частности, второго) порядка аппроксимации гиперболических уравнений обладают большой дисперсионной (и малой диффузионной) ошибкой, в связи с чем их решения сильно осциллируют на разрывах. Поэтому среди немонотонных схем имеет смысл рассматривать только схемы третьего порядка на шаблонах, содержащих на нижнем слое *(n+3)* узлов. Ближе всего к монотонным находятся схемы, шаблон которых также лежит на минимальном расстоянии от базовой точки. Следовательно, описанный выше для схем первого порядка алгоритм быстрого поиска оптимального шаблона остаётся практически без изменений, а значит, элемент, который реализует наименее осциллирующую схему третьего порядка, можно унаследовать от элемента, реализующего наиболее точную монотонную схему. Как было показано в разделе 3.2.5, алгоритм гибридизации этих двух схем на основе объектного подхода является ещё более простым.

#### Методы для параболических уравнений

В отличие от гиперболических уравнений, построение схем высокого порядка аппроксимации параболических уравнений необходимо только для узкого класса сильно нелинейных задач. Кроме того, монотонность параболических схем не является столь существенным свойством, поскольку эти уравнения сами по себе содержат диффузионный член, сглаживающий нефизические осцилляции решения. Поэтому поиск оптимального шаблона не так сильно улучшает качества численного метода, и эффективность реализации этого поиска на основе объектов не так важна. С другой стороны, желательно избегать применения явных параболических схем, особенно для нелинейных уравнений, так как условие Куранта для них () гораздо более ограничительное, чем для явных гиперболических схем ().

Рассмотрим сначала случай регулярной сетки. Если на верхнем слое аппроксимировать только значения пространственных производных искомой функции, то рассчитывать неявную схему можно методом прогонки, а в многомерном случае – также методом переменных направлений или методом дробных шагов с независимыми прогонками по каждой координате. С точки зрения ООП метод переменных направлений предпочтительнее метода многоточечной прогонки, поскольку позволяет использовать те же объекты схемы трёхточечной прогонки, что и в одномерных задачах. Для этого даже необязательно создавать специальный класс метода переменных направлений, поскольку ***n*** объектов метода трёхточечной прогонки имеют все необходимые данные (одну связь со своим сеточным узлом, две связи с такими же объектами-схемами для соседних узлов и набор коэффициентов схемы) и не должны быть связаны между собой. Таким образом, на этапе подготовки к расчётам «рядом» со структурой связанных внутренних сеточных узлов строится ***n***аналогичных структур из объектов трёхточечной прогонки. Перед каждым шагом узел изменяет в соответствующем ему объекте метода лишь коэффициенты схемы (если коэффициенты уравнения переменные), а структура связей на протяжении всех расчётов остаётся неизменной. Граничные узлы (в которых заданы краевые условия первого рода) привязаны к объектам метода, соответствующим ближайшему внутреннему узлу; кроме того, на разных границах области интегрирования граничные узлы должны быть двух разных типов. Узлы первого типа инициируют прямой ход прогонки в цепочке объектов метода, а узлы второго типа инициируют обратный ход в цепочке внутренних узлов (после того, как до них дойдет очередь прямого хода прогонки – см. рис. 3.3).

В случае явных схем объект области интегрирования содержит неструктурированное множество сеточных узлов, которые он может в произвольном порядке переводить на следующий шаг по времени. Однако области интегрирования, предназначенная для узлов с неявными методами, должна отличать от остальных граничные узлы первого типа, самостоятельно переводя на следующий шаг только эти узлы. Для реализации метода переменных направлений такая область должна также иметь счётчик кратности (в двумерном случае – чётности) шага, а также несколько массивов граничных узлов первого типа (число массивов равно размерности пространства). Если коэффициенты и/или правая часть уравнения зависят от решения, то метод прогонки и метод переменных направлений можно дополнить дополнительными простыми итерациями на каждом (полном) шаге по времени (значения искомой функции на предыдущем временном слое являются хорошим начальным приближением, поэтому более сложные итерационные методы здесь избыточны). За эти итерации (происходящие вплоть до получения заданной точности) также отвечает объект области интегрирования.

Внутренний узел

Граничный узел типа 2

Граничный узел типа 1

Схема трёхточечной прогонки по ***x***

Схема  
прогонки по ***y***

***x***

***y***

**Рис. 3.3.** Связи объектов, реализующих метод переменных направлений в случае 2D. Стрелки обозначают направление вычислений, пунктирные линии – связи по данным.

Следует заметить, что изложенный алгоритм расчёта неявной схемы в многомерном случае существенно опирается на регулярность сетки, в то время как большинство реальных задач имеют сложную геометрию. Поэтому, какой бы удобной ни была объектно-ориентированная реализация метода переменных направлений, необходимо рассмотреть также алгоритм метода многоточечной прогонки, который возможен и на нерегулярной сетке. Этот алгоритм включает, во-первых, построение связей между элементами и схемами сеточных узлов и, во-вторых, несколько этапов вычислительного процесса на каждом шаге по времени, соответствующих расчёту приграничных коэффициентов, прямому ходу и обратному ходу прогонки.

Для аппроксимации параболического уравнения с первым порядком в *n*-мерном случае требуется шаблон, содержащий *(2n+1)* сеточных узлов (включая рассчитываемый). Если фиксировать этот шаблон (как это было сделано в случае регулярной сетки), то построение связей происходит только один раз перед расчётами. Если же коэффициенты уравнения сильно изменяются в ходе расчётов, то построение оптимального шаблона можно проводить на каждом шаге или через несколько шагов (как для нелинейных гиперболических уравнений), однако в силу резкого понижения быстродействия это делать нецелесообразно (см. также начало данного раздела). Поэтому каждый внутренний узел необходимо связать с *2n* ближайшими узлами, как и для гиперболических схем (см. 3.3.2.1), что позволяет использовать для «параболических» и «гиперболических» элементов один и тот же суперкласс. Последующий алгоритм связывания объектов схемы многоточечной прогонки, присоединенных ко внутренним узлам, более сложен. Для того, чтобы реализация метода не слишком зависела от размерности, предлагается создавать специальные объекты связей между объектами схемы, которые на этапе расчёта будут хранить соответствующие прогоночные коэффициенты. Эти связи в объекте схемы делятся на ***n***входных и ***n***выходных связей. Входные связи необходимы при вычислении прогоночных коэффициентов для выходных связей на прямом ходу прогонки, а выходные связи используются самими сеточными узлами для расчёта их значений на обратном ходу. Разделение на входные и выходные связи происходит автоматически в ходе процесса связывания, который инициируется граничным сеточным узлом первого типа (см. выше), беспорядочным образом распространяясь по сетке вплоть до граничных узлов второго типа. Следует заметить, что перед беспорядочным связыванием объектов схем внутри области интегрирования необходимо, чтобы все граничные узлы образовали связи с ***n*** приграничными объектами схемы; в противном случае граничный узел (как узел первого типа) может связаться с объектом схемы, который уже связал себя с данным узлом (как с узлом второго типа). Запрещение двунаправленных связей между граничным узлом и объектом схемы позволяет самостоятельно (без затухания) распространяться по сетке как процессу построения связей, так и процессам расчёта, основанным на этих связях. Тем не менее целесообразно проводить связывание объектов схемы более упорядоченно; а именно, нужно строить сначала связи с узлом, направление на который составляет наименьший угол с направлением на предыдущий узел цепочки (см. рис. 3.4). Чем более прямыми являются цепочки сеточных узлов, тем более устойчивым является метод прогонки.

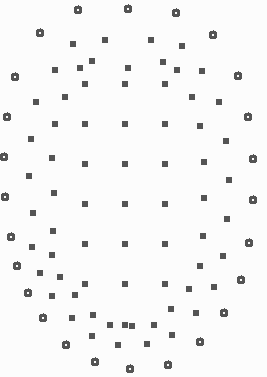
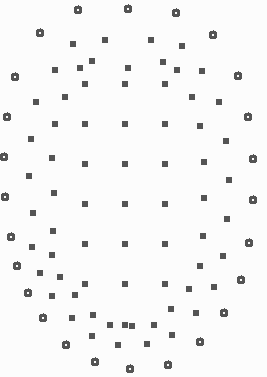
Алгоритм перехода некоторой области интегрирования на один шага по времени, как было отмечено выше, состоит из нескольких этапов. В случае регулярной сетки для последовательного выполнения всех этих этапов область интегрирования на каждом шаге инициирует процесс расчёта только один раз (точнее, делает это в цикле, каждая итерация которого полностью рассчитывает одномерную цепочку объектов). В данном случае сначала граничными узлами первого типа рассчитываются все приграничные прогоночные коэффициенты, затем инициируется процесс расчёта объектами схемы всех остальных прогоночных коэффициентов в их связях, и только после его окончания через какой-либо граничный узел второго типа инициируется расчёт внутренних узлов (последние два процесса самостоятельно распространяются по сетке). Разделение на узлы первого и второго типа происходит автоматически при построении связей (если какой-либо объект схемы связался с некоторым граничным узлом, то он помечается как узел второго типа; в противном случае – остаётся узлом первого типа). Поэтому на первом этапе шага по времени можно не вычислять прогоночные коэффициенты в схемах, связанных с граничными узлами второго типа.

**Б**

**А**

Граничные узлы типа 1

Граничные узлы типа 2



**Рис. 3.4.** Алгоритм построения связей между узлами двумерной нерегулярной сетки для реализации метода пятиточечной прогонки: А – с оптимизацией направления прогонки, Б – беспорядочное связывание узлов (без оптимизации направления)

Описанный объектно-ориентированный алгоритм реализации неявных схем для уравнений в частных производных методом многоточечной прогонки на нерегулярной сетке не имеет известных автору процедурных аналогов. Основным его преимуществом является то, что многомерная область перед прогонкой не преобразуется в квазиодномерную структуру (в квазидиагональную матрицу). Распространение всех объектных алгоритмов для неявных численных методов на случай более сложных краевых условий (выше предполагались условия первого рода) идейно ничем не отличается от случая чисто процедурных алгоритмов.

#### Методы для эллиптических уравнений

Возможность использования для решения эллиптических уравнений расчётные классы, реализованные для параболических уравнений, делает нецелесообразным создание классов, основанных на принципиально отличных методах. Поэтому нужно лишь рассмотреть вопрос об использовании описанных в разделе 3.3.2.2 объектов для эллиптических задач. Поскольку основное отличие между параболическими и эллиптическими схемами заключается в алгоритме выбора оптимального шага по времени (итерационного параметра), то поставленный вопрос касается только одного из расчётных объектов – области интегрирования. Так как классов области интегрирования, соответствующих различным численным методам, может быть несколько, то нецелесообразно расширять возможности каждого из них алгоритмом изменения итерационного параметра. Лучше реализовать этот алгоритм в отдельном классе (который и является, собственно, схемой эллиптического уравнения). Поскольку и для параболических уравнений должен быть предусмотрен объект, выбирающий шаг по времени (многие нелинейные параболические уравнения – это жесткие системы в бесконечномерном пространстве), то не нужно создавать многочисленные подклассы «параболических» областей интегрирования, умеющие взаимодействовать с эллиптической схемой. Достаточно привязывать к обычным «параболическим» областям какую-либо эллиптическую схему в качестве объекта выбора шага. При таком подходе методы решения эллиптических уравнений и вспомогательные по отношению к ним «параболические» методы можно развивать независимо друг от друга.

Наконец, следует рассмотреть ещё один вычислительный алгоритм, который пригоден для расчёта эллиптических уравнений, но не описывался при анализе параболических задач (в разделе 3.3.2.2) ввиду низкого порядка аппроксимации по времени. Речь идёт о методе Зейделя, относящегося к классу треугольных итерационных методов. Реализация этого метода, формально являющегося неявным, в случае применения процедурного подхода практически ничем не отличается от явных методов, а в случае объектного подхода – даже проще явных. Алгоритм состоит в неупорядоченном переводе сеточных узлов на следующую итерацию таким образом, что позже рассчитывающиеся узлы используют те значения в ранее рассчитанных узлах, которые относятся к верхнему слою. Очевидно, при этом не требуется хранить в узле информацию о том, был ли этот узел рассчитан на текущей итерации. Замечательной особенностью метода Зейделя является то, что он не требует вычисления границ спектра итерационной матрицы, что в случае методов с выбором итерационного параметра приводит к громоздким вычислениям, плохо согласующимся с объектным подходом.

## Пример использования: объектно-ориентированная библиотека численных методов для задач гидромеханики и массопереноса (теплопереноса)

Описанный в данной главе подход реализует объектно-ориентированная библиотека численных методов, которая содержит более сорока элементов и в настоящее время участвует в решении задач комплексного моделирования организма человека (см. раздел 2.3). Об эффективности подхода свидетельствует то, что эта библиотека, способная рассчитывать основные типы алгебраических, дифференциальных, дифференциально-разностных уравнений и уравнений математической физики, была разработана всего за один (человеко-)месяц. Это говорит о том, что не только описанный в разделе 2.2.1 процесс создания модели, но и процесс разработки логических блоков этой модели в случае использования объектно-ориентированного подхода намного быстрее процедурно-ориентированного. Эти два подхода отличаются друг от друга тем же, чем сборка компьютера из микросхем отличается от его сборки из отдельных транзисторов.

Предлагаемая библиотека предназначена для простейших гидродинамических расчётов течений жидкости (газа) и расчётов переноса (веществ или тепла) этими течениями. Среда, несущая жидкость, может быть относительно произвольной: от дискретной системы трубок до пористой среды с таким же законом пропорциональности между потоком и градиентом давления. Очевидно, уравнения, аналогичные гидродинамическим, (уравнения Кирхгоффа) описывают токи в электрических цепях или проводящих средах; поэтому часть элементов данной библиотеки подходит также к электротехническим задачам.

Следует заметить, что рассматриваемая библиотека не столь универсальна, как описанные в разделе 3.3 численные методы. Кроме того, она использует отнюдь не все преимущества объектно-ориентированного подхода, упомянутые в разделе 3.2. В частности, в силу простоты применяемых численных методов они не разделяются на два объекта – элемент и схему, что сильно уменьшает возможности их дальнейшего развития.

При описании библиотеки используется понятие среды, не упомянутое в основной части главы по той причине, что оно необходимо отнюдь не во всех моделях. Среда – это набор глобальных данных, которые заведомо используются большим числом расчётных элементов, имеющих для хранения ссылки на среду специальное поле. Название данного объекта определилось физическим смыслом элементов данной библиотеки, которые хранят в нём свойства протекающей по ним субстанции. Применение среды позволяет экономить память, поскольку в противном случае многие элементы содержали бы одинаковые данные; кроме того, одновременное изменение этих данных было бы затруднено.

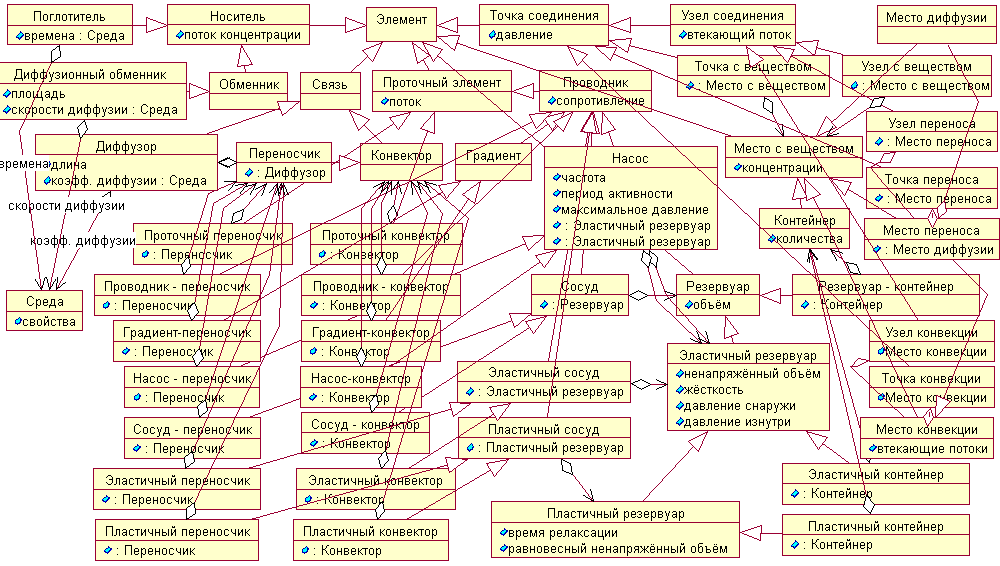
### Использование библиотеки

Рассматриваемая библиотека может использоваться двумя способами. Во-первых, её классы легко встраиваются в любую объектно-ориентированную программу (естественно, написанную на том же языке, что и сама библиотека, то есть на языке Java). При этом разработчик программы, целью которого является реализация конкретной модели или небольшого набора однотипных моделей, может при необходимости путём наследования от элементов библиотеки создавать свои расчётные классы, более приспособленные для решения его задач. Во-вторых, данная библиотека классов может быть использована как составная часть инструментальных средств визуальной разработки моделей. При этом создание новых классов не обязательно (хотя возможно!), поскольку даже из нескольких десятков элементов библиотеки, визуально связывая их в разных комбинациях, можно построить очень большое число моделей соответствующей предметной области. Пример модели, созданный с помощью инструментального средства на базе данной библиотеки, описан в разделе 2.3.2.

В обоих рассмотренных случаях библиотека входит в состав исполняемого кода программы. Однако максимальной гибкостью обладает разрабатываемое автором инструментальное средство, которое рассматривает классы библиотеки как данные, которые можно в наглядном виде представлять, изменять и расширять новыми классами. Перед запуском расчётов эти данные компилируются в исполняемый код, причём особенности языка Java (в частности, его интерпретируемость) позволяют использовать этот код в той же программе, которая редактирует классы библиотеки, не перезапуская её. Такой подход даёт возможность не разделять систему моделирования на расчётные и интерфейсные программы (что затруднило бы управление моделированием и визуализацию получаемых результатов).

Описание библиотеки с точки зрения её использования приведено в приложении к данной работе – ввиду большого объёма этого описания. Для использования библиотеки при конструировании разнообразных моделей достаточно приведённой в приложении информации, в то время как для расширения возможностей библиотеки и для приспособления её к конкретным предметным областям необходима также объектно-ориентированная структура библиотеки, описанная ниже в разделе 3.4.2.

### Объектно-ориентированная структура библиотеки



**Рис. 3.5.** Диаграмма классов библиотеки численных методов для задач гидромеханики и массопереноса (теплопереноса) в нотации UML

На рис. 3.5 представлена диаграмма классов элементов библиотеки, соответствующая сокращённой объектной нотации UML (Unified Modeling Language). Каждый класс на диаграмме характеризуется (помимо названия) физически значимыми данными (параметрами); а вспомогательные данные, необходимые для реализации численных методов, опущены. Например, опущены элементы, с которыми взаимодействуют экземпляры данного класса, поскольку эта информация представлена в табл. 4 приложения. В связи с тем, что язык программирования Java, на котором написана библиотека, не поддерживает множественного наследования классов, аналогичный множественному наследованию результат получен за счёт механизма агрегации (делегирования некоторых функций объекта агрегируемому объекту), а также за счёт использования интерфейсов (на диаграмме не показаны). По соображениям наглядности диаграмма русифицирована, и в ней опущены названия всех методов элементов. Более подробная информация о физическом смысле элементов библиотеки и их назначении приведена в приложении.

## Резюме

В данной главе с точки зрения вычислительной математики проанализирован предложенный в главе 2 подход к построению и реализации моделей сложных систем. Для многих понятий, встречающихся в области численных методов, (система уравнений, схема, сетка и т.п.) разработаны объектные эквиваленты и указаны алгоритмы их взаимодействия. Обоснована эффективность использования объектно-ориентированных методов, связанных с некоторыми более специфическими понятиями (например, с гибридными схемами или нерегулярными сетками), в то время как для других понятий (например, для неявных схем и адаптивных сеток) отмечены (и, отчасти, решены) проблемы их объектного представления. Исследован также вопрос о повышенной ресурсоёмкости объектных вычислений по сравнению с процедурными, на который, однако, дан неоднозначный ответ. Многие свойства объектно-ориентированного подхода проиллюстрированы на широко известных численных методах, из которых выбраны наиболее ему соответствующие, а также указаны методы, которые следует реализовывать только в процедурных программах. Показано, что основанные на объектах алгоритмы реализации численных методов, в особенности на нерегулярных сетках, могут стимулировать развитие этих методов как таковых. Эффективность подхода также показана при описании структуры библиотеки численных методов, связанной с конкретной предметной областью (гидромеханика и массоперенос).

# Многокомпонентные базы данных как средство поддержки методологии вычислительного эксперимента

## Постановка задачи и предпосылки для её решения

До сих пор были описаны проблемы оптимального представления структуры моделей (глава 2) и численных методов (глава 3) в оперативной памяти компьютера, прежде всего в процессе создания моделей и выполнения расчётов с ними. В данной главе процесс моделирования рассматривается на большем временном интервале, затрагивается хранение данных модели в постоянной памяти и работа со многими версиями моделей. В этой области также возникает несколько проблем, которые эффективно решаются на основе объектно-ориентированного подхода.

### Проблемы хранения данных компьютерных моделей

#### Неоднородность данных и необходимость СУБД для моделирования

Прежде всего, данные математических и имитационных моделей (а особенно комплексных моделей, соединяющие в себе оба подхода) имеют чрезвычайно различную природу. Их можно условно классифицировать по следующим критериям:

1. входные и выходные (важно для всех моделей);
2. данные о предметной области и данные о расчётных алгоритмах (для моделей с редактируемыми численными методами);
3. качественные и количественные (для моделей с редактируемой структурой);
4. структурные и функциональные (для моделей с экспертными зависимостями);
5. статические и динамические (для нестационарных моделей);
6. точные (константные) и неточные (часто изменяемые);
7. распределённые и сосредоточенные данные (для математических моделей, основанных на уравнениях в частных производных).

Для всех этих типов данных способ их хранения должен, так или иначе, отличаться. Если для хранения в оперативной памяти компьютера структуры данных объектно-ориентированных языков программирования в какой-то мере решают эту проблему, то для хранения данных в постоянной памяти объектно-ориентированного подхода (по крайней мере, в его традиционном понимании) недостаточно. Объектно-ориентированные базы данных – интенсивно развивающееся в настоящее время направление теории и реализации БД – могут облегчить унифицированное хранение входных и выходных, качественных и количественных, распределённых и сосредоточенных данных, но отнюдь не статических и динамических; они также не в состоянии оптимизировать хранение точных (часто изменяемых) и неточных (редко изменяемых) данных.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **неоднородные данные** | | | | |  | | вычислительные | |
|  | | | | | | | | |
| качественные | |  | количественные | | |  | данные ПО | |
|  | | | | | | | | |
| структурные |  | | | распределённые | |  | | входные |
|  | | | | | | | | |
| функциональные | |  | сосредоточенные | | |  | выходные | |
|  | | | | | | | | |
| динамические | |  | неточные (часто изменяемые) | | | |  | |
|  | | | | | | | | |
| статические | |  | точные (редко изменяемые) | | | |  | |

**Рис. 4.1.** Неоднородность данных моделей

Таким образом, без преувеличения можно сказать, что при моделировании сложных систем возникает самые большое число типов данных, чем в любой другой области применения компьютеров. И тем не менее, в настоящее время не существует не только системы управления базой данных (СУБД), которая бы поддерживала эту особенность задач моделирования, не существует даже теоретической основы для такой системы. С одной стороны, это обусловлено тем, что на проблему хранения данных вычислители не обращают внимания, а разработчики СУБД не задумываются над потребностями относительно узкой предметной области, реализация которых требует серьёзных исследований. Если же говорить об объективных причинах, то разнообразие данных моделей действительно кажется слишком большим, чтобы применять к ним какой-либо универсальный подход. Как следствие, разработчики каждой модели придумывают свою нестандартизованную файловую структуру для хранения её «уникальных» данных, и эта структура не отвечает даже тем требованиям, которые предъявляются к обычным базам данных.

В данной части работы (в главе 4) исследуются общие черты данных, возникающих в разнообразных моделях, показывается, что они не являются уникальными, и на основе этого разрабатывается подход к стандартной СУБД, которой удобно пользоваться как в универсальных системах моделирования, так и в частных моделях.

#### Методология вычислительного эксперимента и её отражение в данных

Построение модели любой системы начинается с качественного описания входящих в неё подсистем, их параметров и связей, то есть с создания схемы модели (см. рис. 4.2 и табл. 4.1). Например, если создаётся распределённая модель, данные схемы включают геометрию системы и построенную на ней сетку. При этом используются метаданные о тех вычислительных алгоритмах, которые предполагается использовать, и эти метаданные тоже должны быть явно представлены и где-то храниться. После создания схемы задаются числовые значения всех её параметров, а также функциональные зависимости между ними. После того, как и эта информация заложена в базу данных, модель формально готова к вычислениям. Однако на этапе использования модели возникает ещё множество динамических результатов, которые соответствуют конкретному сценарию моделирования, то есть набору некоторых исходных данных.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | | |  | |  | База данных | | | |  |  | |
| Ожидаемые  результаты | | | |  | | |  |  | |  | |  |  | |
|  | | | Метаданные 1 | | |  | | Метаданные 2 | | |
|  | |  | | |  | |  |  | |  | |  |  | |
|  | |  | | | Схема 1.1 | | |  | | Схема 1.2 | | | …… |
|  | |  | | |  | |  |  | |  | |  |  | |
|  | | Модель 1.1.1 | | | | |  | Модель 1.1.2 | | | | …… |  | |
|  | |  | | |  | |  |  | |  | |  |  | |
| Сценарий 1.1.1.1 | | | | |  | | Сценарий 1.1.1.2 | | | …… | |  |  | |

**Рис. 4.2.** Влияние методологии моделирования на структуру данных

**Таблица 4.1**

Классификация данных модели по их роли в процессе моделирования

|  |  |
| --- | --- |
| **Компонент БД** | **Типы данных модели** |
| Метаданные | данные о расчётных алгоритмах |
| Схема | качественные, структурные, точные (редко изменяемые) |
| Модель | количественные, функциональные, неточные (часто изменяемые) |
| Сценарий | входные, выходные, динамические |

Конечно, результаты моделирования далеко не сразу начинают соответствовать ожиданиям. Поэтому в процессе расчётов часто приходится возвращаться на этап идентификации констант модели, реже – на этап определения качественной схемы этой модели, а иногда даже на этап выбора вычислительных алгоритмов (см. рис. 4.2). Чтобы произвести изменение всего одного параметра (например, входного параметра сценария) и при этом не испортить уже имеющиеся наработки, обычно создают резервную копию всей базы данных. На самом деле достаточно создать версию только одного компонента базы данных – в данном случае, сценария. Если после этого потребуется подняться на уровень выше и изменить какую-либо константу модели, достаточно будет сделать это один раз, а не дважды, ведь это изменение автоматически отразится на обеих версиях сценария.

### Анализ пригодности существующих способов создания динамических баз данных к задачам моделирования

База данных (БД) в современном её понимании – это статическая совокупность данных, характеризующая состояние некоторой системы в определённый момент времени. Конечно, иногда требуется хранить и динамику развития системы. Обычно хранится лишь динамика изменения некоторых (заранее определённых) параметров системы, и для этого создаются специальные части БД (например, таблицы), в которых для каждого объекта (параметра) вводится специальный атрибут – время появления в БД. В принципе, этот стандартный подход можно применить и к системам с переменной структурой, характеризуя временем абсолютно все данные БД и создавая в ней каждый раз новые объекты вместо того, чтобы изменять старые. Однако такой подход весьма неэкономичен как в смысле объёма хранимой информации, так и в смысле скорости выполнения запросов на выборку данных (выборку всех данных в некоторый момент времени или, наоборот, выборку некоторых данных во все моменты времени). Отсутствие реально работающих динамических баз данных – лишнее доказательство неприемлемости описанного стандартного подхода к хранению истории развития систем. Современные СУБД поддерживают журнализацию вносимых в БД изменений (запоминание истории) только в целях отката и восстановления БД после сбоев.

Тем не менее, потребность одновременно хранить в БД некоторое число состояний рассматриваемой системы существует. Часто она формулируется как хранение версий БД, без привязки к динамике, то есть ко временной последовательности состояний. Наиболее распространена поддержка версий в средствах проектирования и разработки программных продуктов – в CASE-средствах. Однако такие системы версификации либо слишком зависят от структуры данных, либо являются внешними по отношению к БД. В последнем случае под версиями понимаются не зависящие друг от друга экземпляры БД, которые снабжаются некоторыми дополнительными атрибутами (в том числе временем создания, по которому происходит упорядочение версий). Поддержка версий была бы весьма полезна при создании не только программных продуктов, но и разнообразных моделей сложных систем, – хотя бы для того, чтобы в качестве версий хранить различные наборы входных данных модели с соответствующими им результатами моделирования. Однако в этой области универсальных средств версификации вообще не существует.

Проблемы, возникающие при работе с несколькими версиями, хранящимися как независимые экземпляры БД, можно проиллюстрировать на следующем примере. Пусть последовательные состояния системы отличаются лишь небольшим числом параметров, но имеют самостоятельное значение, то есть их имеет смысл хранить в качестве версий. Тогда соответствующие версиям экземпляры БД практически не будут содержать полезной информации, многократно дублируя подавляющее большинство данных. Хорошо, если версии считаются законченными, и в них не придётся впоследствии ничего изменять. Если же возможна корректировка исходных данных о системе (например, оказался неучтённым какой-либо фактор, действующий изначально или начиная с некоторого момента времени), стандартный подход к версификации приведёт не только к неоправданному расходованию ресурсов (постоянной памяти), но и к необходимости многократного повторения одной и той же работы над каждым экземпляром БД.

Другой пример касается не линейного упорядочения состояний системы во времени, а иерархического упорядочения состояний её частей. Многие БД объединяют относительно статичные, редко изменяемые данные (качественные, более-менее надёжные) и динамичные, часто изменяемые (количественные, неточно известные). При физическом разделении версий БД, соответствующих различным значениям часто изменяемых данных, возникают описанные выше проблемы дублирования данных и повторения однообразных действий над версиями при изменении статичных данных. Однако в данном случае дело осложняется тем, что изменение статичных данных желательно сопровождать удвоением числа существующих версий: половина версий будет соответствовать их старым значениям, а половина – новым. Расположение версий в их временной последовательности в этом случае абсолютно не соответствует смыслу их отличий друг от друга, который требует представления версий в виде иерархической структуры. Сравнение же версий (проясняющее смысл их различия) обычно проводится полным сравнением всех данных, что также неприемлемо, так как версии заведомо отличаются не сильно. Иерархия версий встречается в существующих средствах разработки программных продуктов, однако эта иерархия жёстко привязана к структуре данных, в частности, имеет фиксированное число уровней (например, нижний уровень – версии файлов модулей (классов), средний уровень – версии директорий групп модулей (пакетов, компонентов), верхний уровень – версии проекта в целом).

Наконец, ещё одно неприятное следствие представления экземпляра БД как единого целого проявляется при попытке повторного использования накопленных данных в близких задачах. Очевидно, некоторая часть БД (статичные данные, в терминологии предыдущего примера) является достаточно универсальной, чтобы быть полезной в других сходных БД. Но чтобы её использовать, нужно создать копию существующей БД, и «вручную» удалить из неё всё, что не имеет отношения к новой задаче, оставив только универсальную часть. При этом не может быть и речи о том, чтобы параллельно производить корректировку этой части в «старой» и «новой» БД.

### Предлагаемый подход к СУБД для моделирования

Таким образом, в теории и реализации баз данных существуют следующие проблемы, особенно заметные в приложении к потребностям моделирования:

1. неэкономичность хранения (и сравнения) слабо различающихся состояний систем (версий БД), связанная с многократным дублированием данных;
2. многократное повторение одной и той же работы над версиями БД при их единообразной корректировке;
3. линейное (не иерархическое) упорядочение версий, не дающее представления о смысле различий между версиями;
4. невозможность повторного использования частей БД для решения близких задач.

Очевидно, для решения этих проблем нужно представлять БД в виде набора зависящих друг от друга частей (версий), имеющих похожую структуру (схему), но различные (не дублирующиеся) данные. Такие части ниже называются компонентами, а состоящая из них БД – многокомпонентной (МКБД). МКБД в одних случаях можно трактовать как БД со встроенной поддержкой версий, а в других – как БД для хранения истории изменения данных.

Задача состоит в том, чтобы разбиение БД на отдельные компоненты, решив перечисленные выше проблемы, не привело ни к понижению скорости выполнения запросов, ни к рассогласованию данных из разных компонентов, ни к потере целостного восприятия каждой версии БД пользователем. В качестве дополнительного преимущества многокомпонентности желательно также повысить эффективность разделения работы по заполнению БД между несколькими пользователями. Последнее особенно необходимо при рассмотрении сложных моделей, поскольку такие модели требуют привлечения нескольких специалистов различного профиля для разработки своих частей.

### Основные идеи и предпосылки создания МКБД

Для решения поставленной задачи необходимо, чтобы компонент базы данных (КБД) обладал следующим особенностями:

1. КБД содержит только те данные, которые уникальны для связанной с ним версии.
2. Остальные данные, общие для нескольких версий, предоставляются редактирующему данный КБД пользователю из других компонентов.
3. Для этого компоненты образуют иерархическую структуру, в которой каждый нижестоящий КБД может иметь доступ к данным вышестоящего, которые могут быть изменены, удалены и расширены новыми данными.
4. При удалении данных КБД происходит проверка (и восстановление) ссылочной целостности не только для него, но и для всех нижестоящих компонентов.

Перечисленные особенности (по крайней мере, первые три) обнаруживают сильное сходство МКБД с объектно-ориентированным подходом, применяющимся, в частности, при разработке современного программного обеспечения. Анализ применимости понятий объектно-ориентированного программирования (ООП) к потребностям баз данных (см. раздел 4.2.1) позволяет ещё более расширить возможности МКБД. В частности, из ООП в МКБД можно перенести понятие абстрактных данных, которое оказывается очень полезным при создании версий БД (хотя его не имеет смысла рассматривать в задачах о хранении временной последовательности состояний БД).

Следует заметить, что объектно-ориентированные базы данных (ООБД) не имеют никакого отношения к проблемам, решаемым с помощью МКБД. В ООБД основное внимание уделяется проблеме хранения данных сложных типов (классов, абстрактных типов) вместе с привязанными к данным методами их обработки; этим расширяются возможности реляционных баз данных (РБД), которые имеют дело с данными простых (предопределённых) типов. Многокомпонентный же подход к базам данных ничего не предполагает о типах данных: если данные имеют достаточно сложную структуру, имеет смысл хранить их в соответствии с объектно-ориентированной моделью, если относительно простую – можно пользоваться преимуществами реляционной модели. Другими словами, МКБД может являться РБД, ООБД или ОРБД, лишь бы соответствующие структуры данных существовали там в нескольких экземплярах (компонентах), иерархически связанных между собой. Поскольку упорядочение данных на уровне компонентов не имеет отношения к их организации на более низком уровне, систему управления многокомпонентными базами данных имеет смысл реализовывать как надстройку над существующими СУБД.

## Содержание многокомпонентного подхода к БД

### Изложение многокомпонентного подхода с точки зрения ООП

Суть многокомпонентных баз данных, их свойства и преимущества по сравнению с однокомпонентными приведены в таблице 4.2, вместе с аналогичными утверждениями объектно-ориентированного программирования. В таблице 4.3 перечислены основные термины МКБД и аналогичные им термины ООП.

**Таблица 4.2**

Сравнение МКБД с объектно-ориентированным подходом.

|  |  |
| --- | --- |
| **Содержание объектно-ориентированного подхода** | **Содержание многокомпонентного подхода к проектированию БД** |
| *Суть ООП состоит в следующем:*   1. Программа представляет собой совокупность объектов, объединяющих данные и методы их обработки.    1. Смысл объекта (поведение) отделён от его реализации (структуры) за счёт того, что методы могут не содержать кода, а данные имеют определённую область видимости.    2. Данные объектов (поля) также могут быть объектами. 2. Каждый объект относится к определённому типу (классу), т.е. является его экземпляром. 3. Тип (класс) может наследовать часть данных и методов от других типов (классов), то есть быть их подтипом.   *Примечание* к пункту 1.2: в одном и том же поле могут храниться объекты определённого типа и всех его подтипов. | *Суть МКБД состоит в следующем:*   1. База данных представляет собой совокупность компонентов (КБД), каждый из которых объединяет набор таблиц, а точнее – набор групп таблиц (документов).    1. Смысл КБД (структура, поведение здесь ни при чём) отделён от его реализации (от количественного наполнения, структура здесь ни при чём) за счёт того, что данные могут быть абстрактными (неполными и/или несамодостаточными) и могут иметь определённую область видимости. 2. Каждый КБД имеет определённую схему, состоящую из схем документов, а те, в свою очередь – из схем отдельных таблиц. 3. КБД может наследовать часть данных от других КБД, то есть быть их подкомпонентом |
| *Примечание* к пункту 1. Об объединении данных и методов в МКБД (в отличие от ООП) говорить не имеет смысла, поскольку методы являются данными. Активные БД, в которых методы привязаны к данным, здесь можно упоминать только в связи с частным типом данных – с данными об алгоритмах, без которых обычно не обходится создание систем, требующих МКБД, но это не относится к теме данной работы.  *Примечание* к пункту 1.2. Данные компонентов (документы, таблицы) не могут быть компонентами (в отличие от ООП, где данные объектов могут быть объектами), поскольку у всех компонентов примерно одинаковая схема, её можно только сужать.  *Примечание* к пункту 2. Схема не есть полный аналог класса (типа): в отличие от ООП, у разных КБД одна и та же схема может оказаться лишь чисто случайно, хотя созданию однотипных КБД существенно способствует фиксированный набор схем документов, из которых может состоять схема КБД.  *Примечание* к пункту 3. Наследование в МКБД принципиально отличается от ООП, согласно которому наследоваться должны схемы, а не сами компоненты, причём наследование всегда должно приводить к расширению. | |
| *Выделяются следующие свойства ООП:*   1. Абстракция – выделение общих черт сходных объектов в абстрактную сущность – класс, а также отделение реализации методов класса от внешнего интерфейса (протокола, контрактных обязательств) этого класса. 2. Инкапсуляция – отделение внутреннего устройства объектов (структур данных и вспомогательных операций с ними) от поведения этих объектов (от методов, входящих в интерфейс). 3. Иерархичность – упорядочение объектов с помощью некоторых отношений.    1. Агрегация – связывание объектов на основе отношения включения («быть частью / состоять из»).    2. Наследование – расположение классов объектов по уровням с помощью отношения специализации («быть разновидностью / быть обобщением»).       1. Одиночное наследование – наследование, при котором класс имеет только один родительский класс (суперкласс).       2. Множественное наследование – наследование, при котором класс может иметь несколько суперклассов (строго говоря, схема отношений классов при множественном наследовании имеет уже не иерархическую, а сетевую структуру). 4. Типизация – способ защититься от использования объектов одного типа вместо другого (например, путём вызова у объекта тех методов, которых его класс не имеет). С типизацией связан полиморфизм – переопределение методов в разных классах одного типа. 5. Параллелизм – поддержка одновременного выполнения нескольких процессов, возможная благодаря чёткому разделению программы на отдельные объекты. 6. Сохраняемость – способность объекта существовать во времени (независимо от породившего его процесса) и/или в пространстве (перемещаясь с одного адреса на другой).   *Примечание:* свойства 1-3 являются обязательными для любой объектно-ориентированной системе, в то время как свойства 4-6 поддерживаются не всеми объектно-ориентированными языками программирования. | *Выделяются следующие свойства МКБД:*   1. Абстракция – проявляется не в выделении общих черт сходных экземпляров КБД в абстрактную сущность – схему КБД: более или менее абстрактными являются сами экземпляры, каждый из которых в отдельности обычно имеет лишь часть данных, необходимых для работы системы. 2. Инкапсуляция (необязательное свойство) – проявляется в наличии различных областей видимости данных (документов), однако это служит не для отделения внутреннего устройства КБД от их внешнего поведения, а для удобства разделения работы над БД. 3. Иерархичность – упорядочение КБД с помощью некоторых отношений.    1. Агрегация – отсутствует.    2. Наследование – расположение КБД по уровням с помощью отношения наполнения (доработки) («быть реализацией / быть структурой»).       1. Одиночное наследование – наследование, при котором КБД имеет один родительский КБД (суперкомпонент).       2. Множественное наследование – имеет сомнительный смысл, а реализация затруднительна в связи с противоречащими данными суперкомпонентов. 4. Типизация – очень сильная, поскольку в рамках каждой схемы БД свобода выбора типа компонента ограничена числом комбинаций из документов схемы. 5. Полиморфизм – переопределение данных компонентов (а не методов, как в ООП!) в их подкомпонентах. 6. Параллелизм – говорить об этом не имеет смысла, поскольку БД сама по себе статична, (хотя использовать параллельно несколько КБД удобнее, чем несколько стандартных экземпляров БД). Параллелизм в БД может означать параллельную работу нескольких пользователей над разными КБД одной БД. 7. Сохраняемость – свойство, очевидное для любых БД.   *Примечание:* таким образом, основными «объектно-ориентированными» свойствами МКБД являются абстракция (как неполнота и как несамодостаточность), наследование и полиморфизм компонентов. |
| *ООП имеет следующие преимущества*  *перед другими подходами:*   1. компактность программ, уменьшающая их стоимость и время их разработки; 2. возможность повторного использования классов, программ и целых проектов; 3. эффективное разделение работы между дизайнерами и программистами; 4. быстрая адаптация программ к изменяющимся задачам и требованиям; 5. уменьшение риска разработки сложных программ за счёт интеграции и тестирования их компонентов на протяжении всего времени разработки, а не на последнем её этапе. 6. ориентация на человеческое восприятие мира, позволяющая программировать даже без понимания деталей работы компьютера. | *Многокомпонентные БД имеют*  *следующие преимущества*  *перед однокомпонентными:*   1. компактность БД (размер которой зависит от числа её версий менее чем линейно), это уменьшает стоимость БД и повышает скорость наполнения её данными; 2. возможность повторного использования даже полных иерархий КБД; 3. эффективное разделение работы по заполнению различных по смыслу КБД; 4. быстрая адаптация версий БД (одновременно большого их числа) к изменению данных, и даже их схемы; 5. уменьшение риска разработки сложных БД за счёт интеграции и тестирования их частей на протяжении всего времени разработки, а не на последнем её этапе. |

**Таблица 4.3**

Соответствие между терминологией ООП и МКБД

|  |  |  |
| --- | --- | --- |
| *Термин ООП* | *Термин МКБД* | |
| программа | база данных, БД | |
| объект | компонент, КБД | |
| класс объекта | схема БД (которая, однако, не произвольна, а состоит из схем документов заданной схемы БД) | |
| подкласс /суперкласс | подкомпонент/суперкомпонент (не подсхема/суперсхема) | |
| поле класса | схема документа (состоящая из схем таблиц) | |
| значение поля | документ (с таблицами и их данными) | |
| доступ к полям (открытый (public), защищённый (protected) или закрытый (private); абстрактный (abstract) или конечный (final)) | | доступ к документам (открытый (protected), закрытый для записи (final) или закрытый для чтения (private)) |
| абстрактность класса | неполнота КБД и/или несамодостаточность схемы КБД | |

Таким образом, МКБД больше всего напоминает иерархию наследования классов, каждый из которых имеет только один экземпляр. Экземпляр (КБД) состоит преимущественно из значений составных полей (из связанных по смыслу таблиц разных документов), некоторые из которых являются в той или иной мере абстрактными (неполными, если нет части строк в некоторых связанных таблицах компонента, или несамодостаточными, если нет соответствующих документов в компоненте). Все компоненты некоторой БД имеют почти одинаковые схемы, которые могут отличаться отсутствием некоторых документов. Абстракция (в смысле неполноты), наследование и полиморфизм относятся к экземпляру КБД, а не к его схеме. Компонент может быть самодостаточной (самостоятельной), только неполной (незаконченной) версией (иметь схему, совпадающую со схемой БД, лишь не полностью заполненную данными), а может быть заведомо несамодостаточной частью (хотя необязательно неполной!), содержащей не все документы общей схемы БД. Полная версия обычно состоит из нескольких наследуемых друг от друга КБД, неполных и несамодостаточных. Однако неполнота КБД и несамодостаточность его схемы – это разные трактовки понятия абстрактности в БД.

Данные в МКБД могут иметь следующие значения атрибута «доступ»:

1. открытый (соответствует модификатору protected в ООП, разрешая подкомпонентам чтение и запись данных);
2. закрытый для записи (соответствует модификатору final для методов в ООП, разрешая подкомпонентам только чтение данных);
3. закрытый для чтения (соответствует модификатору private в ООП, разрешая доступ к данным только из текущего компонента).

*Примечание* 1: значение доступа имеет смысл присваивать не каждому объекту, хранимому в БД (не каждой строке таблицы), а каждому документу каждого КБД.

*Примечание* 2: модификатору abstract из ООП соответствует атрибут полноты КБД, но его нельзя задать, он определяется совокупным состоянием КБД и его суперкомпонентов.

### Изложение многокомпонентного подхода с точки зрения теории БД

#### Объекты базы данных и связи между ними

Существующую науку о БД многокомпонентный подход почти не затрагивает, допуская проектировать таблицы (отношения) в соответствии с любой существующей (реляционной, объектно-ориентированной или объектно-реляционной) моделью. Другим словами, таблица может рассматриваться как в реляционном виде (т.е. находиться, по крайней мере, в первой нормальной форме), так и в объектно-ориентированном виде (т.е. иметь в качестве доменов списковые данные, сложные типы, подтипы и т.д.).

Отличия МКБД проявляются не на уровне столбцов таблицы (атрибутов отношений), а на более высоком уровне, который в стандартных БД отсутствует. Самые простые СУБД представляют БД в виде неструктурированного множества таблиц, более сложные вводят понятие группы таблиц (каталога, схемы и т.п.), которое позволяет строить из таблиц более сложную логическую структуру. При рассмотрении МКБД группа таблиц также активно применяется. Ниже группа таблиц называется документом ввиду особенностей своего представления пользователю и в силу необходимости отличать её от каталога или схемы. Однако, кроме этого относительно стандартного понятия, МКБД использует понятие компонента базы данных (КБД) – некоторого промежуточного объекта между группой таблиц и всей базой данных.

В однокомпонентной БД: , , ∅,   
в структурированной однокомпонентной БД: , , ∅, , ∅ ],   
в многокомпонентной БД , , ∅, , ∅, , ∅ ] },   
где:



*Т* – схема таблицы (отношения);

*Д* – схема документа (группы таблиц);

*КБД* – схема компонента БД;

*БД* – схема БД.

Отсюда видно, что даже с точки зрения схем компонент базы данных (КБД) не есть просто ещё один уровень иерархического упорядочения таблиц, аналогичный документу (группе таблиц). Схемы КБД являются нестрогими подмножествами схемы БД, имея друг с другом непустое пересечение, – в отличие от схем документов, которые представляют собой разбиение схемы КБД (то есть не пересекаются). Другими словами, КБД могут иметь все таблицы схемы БД, являясь не её частями, а версиями (вариантами).

Однако основное отличие компонентов от документов проявляется не в схемах, в самих данных. В то время как данные документов разных версий однокомпонентной БД независимы, данные компонентов – версий многокомпонентной БД – являются зависимыми; они организованы в некоторую иерархию, принципиально отличную от рассмотренной выше иерархии схем.

Два компонента в этой иерархии (наследования) могут находиться в отношении наполнения (в отношении доработки, в отношении «структура/реализация»). Компонент, находящийся выше некоторого КБД в иерархии, называется суперкомпонентом (родительским компонентом, предком), а находящийся ниже – подкомпонентом (дочерним компонентом, потомком). Часть данных суперкомпонента используется в его подкомпонентах, и часть этих данных там изменяется (удаляется); кроме того, в них появляются новые данные.

Дерево иерархии компонентов обычно строится таким образом, что самые точные, редко изменяемые данные помещаются ближе к корню дерева, а самые ненадёжные, часто изменяемые – на его листьях. Это позволяет в большинстве случаев перед созданием новой версии копировать лишь один лист дерева, а не целую ветвь (и тем более не всё дерево). В БД могут существовать абсолютно надёжные данные, имеющие к тому же весьма общий характер, так что на них могут опираться все остальные данные. Такие данные относятся к корневому компоненту (ККБД), который является суперкомпонентом для всех существующих в БД компонентов. Если ККБД является единственным компонентом МКБД, то она неотличима от однокомпонентной БД.

#### Абстракция данных

Ещё одна идея МКБД заключается в том, что не все компоненты могут иметь полный набор данных, необходимый для их использования в приложениях. Связанные между собой данные (например, качественные и количественные данные, характеризующие один и тот же объект) могут храниться в разных таблицах, относящихся, как правило, к разным документам. Это позволяет одни данные (качественные) относить к суперкомпоненту, а разные варианты других данных (количественных) – к его подкомпонентам-версиям. В этом случае суперкомпонент является неполным (абстрактным), он не может быть использован как таковой, но может служить общей основой (заготовкой) для своих подкомпонентов, избавляя пользователя от многократного повторения одной и той же работы (в рассмотренном примере – от работы по изменению качественных данных).

Возможность работы с абстрактными данными проявляется в том, что таблицы МКБД делятся на первичные и вторичные. Первичная таблица, хранящая обычно общую структуру каких-либо реальных объектов, может содержать ссылки на другие таблицы (внешние ключи) лишь среди своих неключевых атрибутов. Ключ вторичной таблицы совпадает с ключом соответствующей ей первичной, то есть одновременно является внешним ключом. Как следствие, во вторичной таблице (в нескольких вторичных таблицах) могут храниться только данные, конкретизирующие описание объектов первичной таблицы.

Данное рассуждение проведено для какого-либо конкретного КБД, однако различение первичных и вторичных таблиц имеет смысл, только если в БД имеется, по крайней мере, два КБД: абстрактный суперкомпонент, который хранит прежде всего данные в первичной таблице (а вторичной таблицы может не иметь вообще), и подкомпонент, который хранит недостающие данные во вторичной таблице. Иерархия компонентов считается полной (неабстрактной), если она в сумме имеет равное число объектов (строк) в первичной таблице и в любой соответствующей ей вторичной таблице.

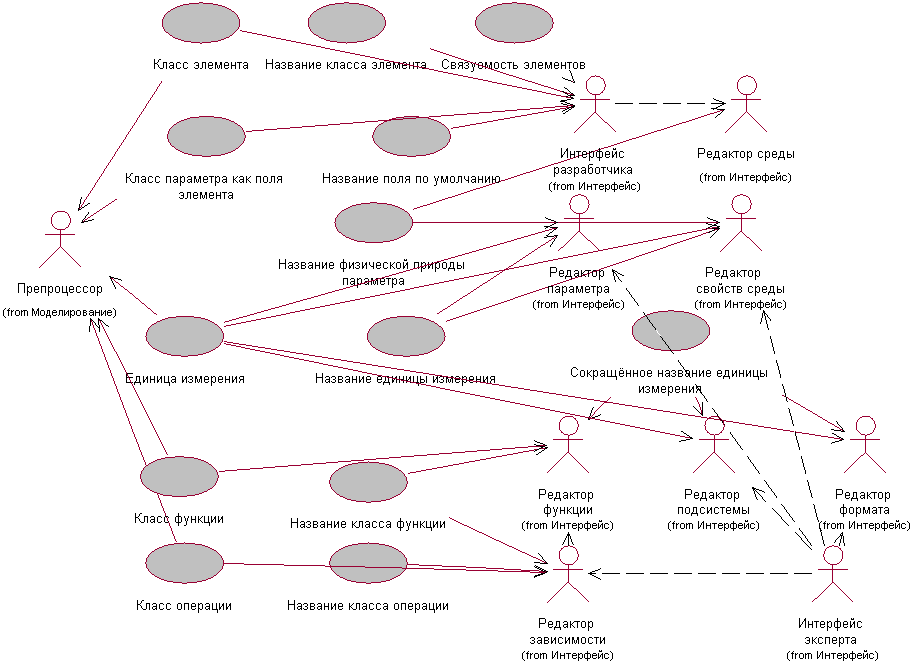
## Пример использования: архитектура базы данных обобщённой модели

Изложенный выше подход к многокомпонентным базам данных иллюстрируется в данном разделе на примере структуры конкретной БД, хранящей для специального инструментального средства (см. разделы 1.2 и 3.4.1) информацию о компьютерных моделях достаточно общего вида. Предлагаемая структура БД предназначена для относительно простых имитационных моделей, она не является многокомпонентной в полном смысле слова, имея фиксированное (равное четырем) число уровней компонентов. Кроме того, описываемая ниже БД не содержит ни информации о численных методах расчёта моделей, ни распределённых по пространству данных (обе этих части БД развиваются автором в настоящее время).

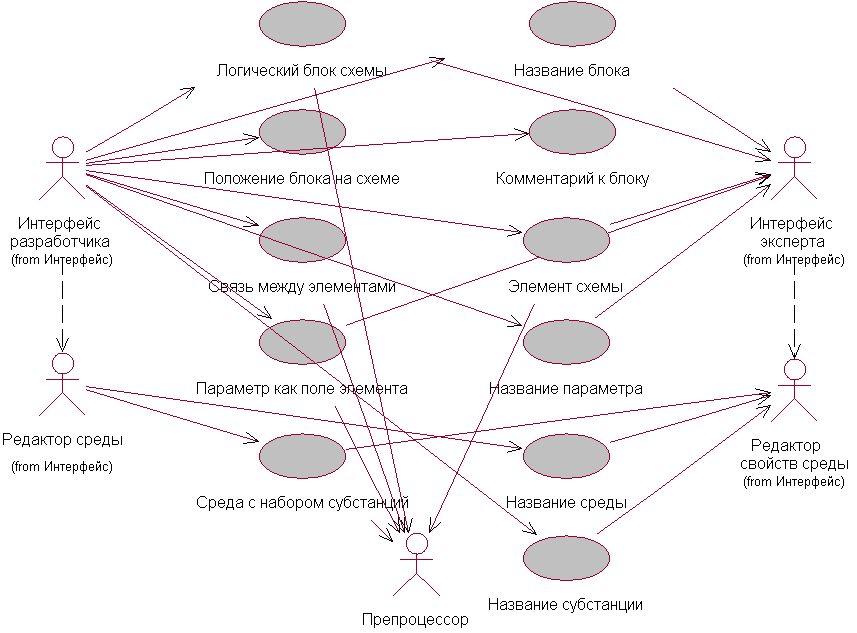
### Описание схемы БД

Особенности методологии моделирования, приведшие к разделению базы данных обобщённой модели на четыре функционально различные части (данные сценария, модели, схемы и метаданные), изложены в разделе 4.1.1.2 и проиллюстрированы на рис. 4.2. Ниже приводится более конкретное описание схемы базы данных – названия таблиц и способы их использования. На рисунках таблицы изображены в виде овалов, использующие их блоки программы – в виде "человечков", а способы их использования (направления передачи данных) – в виде стрелок. Данная нотация широко применяется средствах проектирования программных продуктов (CASE-средствах) и носит название Use Case View. Схемы отдельных таблиц, которые обычно представляются на диаграммах Logical View, здесь не рассматриваются, поскольку особенности МКБД проявляются лишь на более высоком, чем атрибуты таблиц, уровне.

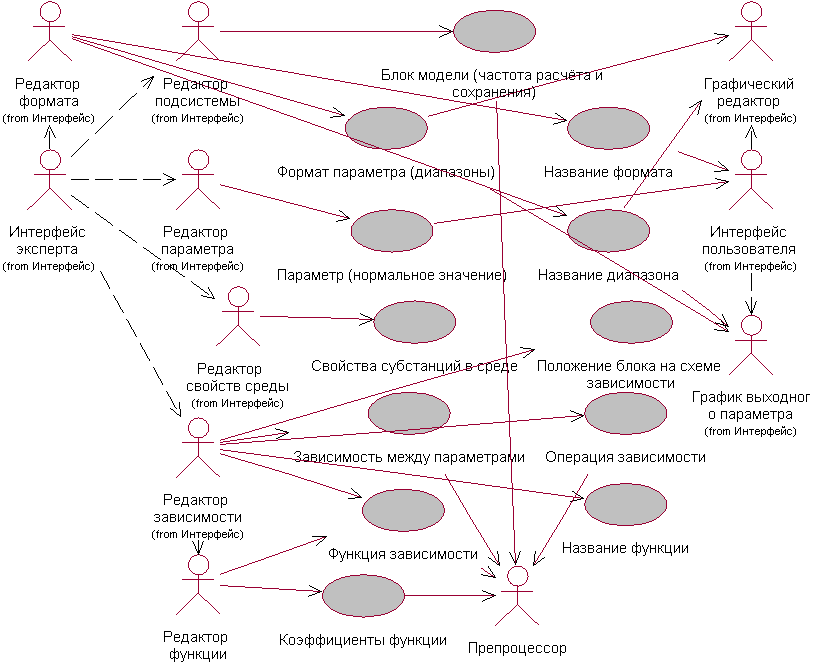
При описании схемы базы данных модели таблицы, хранящие информацию о способе визуального представления этой модели, рассматриваются наряду с основными таблицами (используемыми препроцессором для загрузки данных в солвер). Это сделано в связи с тем, что «визуальные» таблицы ещё больше увеличивают общую неоднородность данных модели, усиливая, тем самым, потребность в применении МКБД.



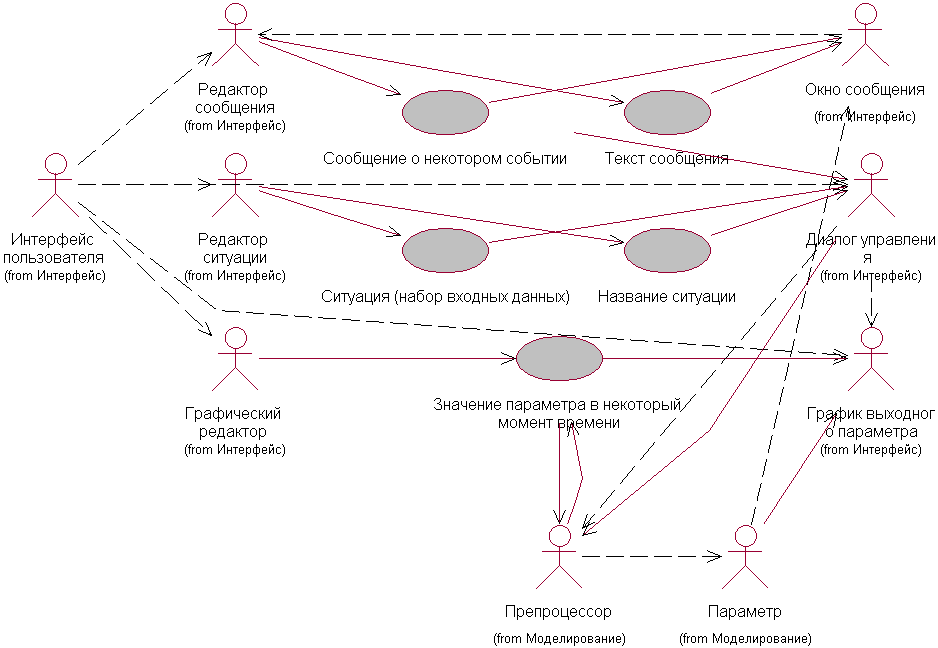
**Рис. 4.3.** Таблицы метаданных



**Рис. 4.4.** Таблицы данных схемы модели



**Рис. 4.5.** Таблицы данных модели



**Рис. 4.6.** Таблицы данных сценария

### Преимущества и недостатки четырёхуровневой БД

Таким образом, чтобы база данных соответствовала общепринятой методологии моделирования, она должна представлять собой иерархию различных по смыслу компонентов – данные сценария, данные модели, данные схемы и метаданные. С помощью одной и той же модели можно рассчитать много сценариев, одна и та же схема может быть реализована многими моделями, а на основе одного и того же численного алгоритма может быть построено много схем. Это избавляет не только от проблем с нехваткой места на диске компьютера при создании версий больших моделей, но и от многократного повторения рутинной работы при одновременном изменении этих версий.

Ещё одним аргументом в пользу описанной четырёхуровневой структуры базы данных является то, что она легко позволяет разделить работу по моделированию между несколькими разработчиками. Для того, чтобы сложная модель оказалась эффективной, адекватной и при этом ещё кому-то нужной, в процесс её создания должно вовлекаться несколько участников. Как минимум, среди них должен быть вычислительный математик, который бы работал на уровне метаданных и схемы, эксперт соответствующей предметной области, который бы давал свои экспертные оценки для параметров модели, и конечный пользователь, которые бы определял, какие исходные данные он хочет вводить на уровне сценария и какие результаты получать.

С помощью разделения базы данных на несколько компонентов в какой-то мере решается проблема чрезвычайной неоднородности данных моделирования. Данные каждого компонента могут физически храниться в той форме, которая им адекватна. Например, для количественных данных модели более-менее подходит реляционная форма хранения, для качественных данных схемы – сетевая, для метаданных – объектная, а для динамических данных сценария – просто неформатированный файл. Однако следует заметить, что таким же преимуществом обладает любая структуризация данных, например, разделение их не на компоненты, а просто на группы таблиц.

Основной недостаток рассмотренной структуры базы данных в том, что она является слишком жёсткой. Использование этой структуры в реальных моделях показало, что не всегда можно чётко отделить, к примеру, качественные данные от количественных. Особенно это касается распределённых моделей, в которых геометрия модели определяет не только связи между сеточными узлами, но и коэффициенты расчётной схемы. Ещё менее чёткой является грань между точными, редко изменяемыми данными модели и неточными, часто изменяемыми. А ведь вся идеология изложенного подхода основана на том, что чем чаще приходится изменять данные в процессе моделирования, тем дальше от корня иерархического дерева компонентов базы данных они должны храниться.

Способ устранения отмеченного недостатка четырёхуровневой БД очевиден – это ещё большая иерархичность базы данных, ещё большая абстракция понятия её компонента и чёткое отделение этого понятия от понятия документа (группы таблиц). Другими словами, число уровней дерева базы данных не должно быть фиксированным, и она должна выглядеть примерно так, как показано на рис. 4.7. При этом один компонент может содержать и схему, и модель, и сценарий, а другой (находящийся ниже в иерархии) – всего одно значение одного параметра.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | | |  |  | База данных | |  |  |
|  | | | |  | |  |  |  |  |  |
| … | | Компонент 1 | |  | Компонент 2 | |
|  | |  | | |  |  |  |  |  |  |
| … | |  | | Компонент 1.1 | |  | Сценарий 1.2 | | …… |
|  | |  | | |  |  |  |  |  |  |
| … | | Компонент 1.1.1 | | | |  | Сценарий 1.1.2 | | …… |  |

**Рис. 4.7.** Архитектура базы данных обобщённой модели

## Резюме

В данной главе исследованы проблемы, возникающие при использовании стандартных систем управления базами данных для задач моделирования и связанные с неэкономичностью хранения слабо различающихся версий моделей. Показано, что решение этих проблем возможно путём создания нового (многокомпонентного) подхода к проектированию баз данных, который основан на некоторых идеях объектно-ориентированного проектирования и соответствует методологии вычислительного эксперимента. Данный подход, имеющий потенциальные приложения отнюдь не только в области создания систем моделирования, изложен как с точки зрения ООП, так и с точки зрения существующей теории баз данных. Его эффективность показана на примере базы данных обобщённой модели, для которой описана её логическая структура и методология использования.

# Заключение

Таким образом, в данной работе проведён анализ возможностей применения объектно-ориентированного подхода к задачам моделирования сложных систем. Показано, что ООП позволяет сблизить методы, принятые вычислительной математике и имитационном моделировании, и тем самым совместно использовать их достоинства.

Путём анализа подходов, принятых в существующих объектных средствах моделирования, выбрана оптимальная трактовка понятия объекта – элемента вычислительных моделей. Эта трактовка позволяет быстро создавать, легко развивать и наглядно представлять не только сами модели, но и численные методы, которые рассчитывают эти модели. Для большинства понятий вычислительной математики разработаны объектные эквиваленты и указаны алгоритмы их взаимодействия. Исследована эффективность объектно-ориентированных численных методов по сравнению с процедурно-ориентированными – как с точки зрения их алгоритмической сложности, так и с точки зрения требовательности к вычислительным ресурсам.

Проведён анализ путей использования объектно-ориентированного подхода к решению проблемы оптимального хранения данных, возникающих при создании моделей и при вычислительных экспериментах с ними. Разработанный в результате многокомпонентный подход к системам управления базами данных имеет значение не только для моделирования, но и для других задач, которые используют последовательности или иерархии версий состояния некоторой системы.

Эффективность всех перечисленных теоретических подходов показана на примерах их применения к конкретным практическим задачам. Для иллюстрации использования объектно-ориентированного подхода при построении моделей кратко описана модель организма человека, для демонстрации эффективности ООП при реализации вычислительных алгоритмов рассмотрена библиотека численных методов для задач гидромеханики и массопереноса, а в качестве примера многокомпонентной базы данных приведена структура базы данных обобщённой модели.

На основе данной работы создано инструментальное средство, которое позволяет автоматизировать процесс создания и использования сложных моделей, и сочетает достоинства имитационного и математического моделирования на основе объектно-ориентированного подхода. С помощью этого инструментального средства созданы описанные в качестве примеров модель организма человека, рассчитывающая эту модель библиотека численных методов и хранящая её многокомпонентная база данных.

Список литературы

1. R. F. Boisvert, S. Browne, J. Dongarra, E. Grosse. Digital software and data repositories for support of scientific computing. In N. Adam et al., editors, Advances in Digital Libraries, number 1082 in Lecture Notes in Computer Science, pages 61-72. Springer-Verlag, New York, 1996.
2. R. F. Boisvert, J. J. Dongarra, R. Pozo, K. A. Remington, and G. W. Stewart. Developing numerical libraries in Java. Concurrency: Practice and Experience, 10(11): 1117–1129, Sept. 1998. (http://www.cs.ucsb.edu/conferences/java98/papers/jnt.pdf)
3. E.Anderson, Z.Bai et al. LAPACK User’s Guide. SIAM, Philadelphia, second edition, 1995.
4. B. Blount, S. Chatterjee, An Evaluation of Java for Numerical Computing. The University of North Carolina. Jan. 1999. (ftp://ftp.cs.unc.edu/pub/users/cs/papers/sp-java.pdf)
5. R. Pozo. Template Numerical Toolkit for linear algebra: High performance programming with C++ and the Standard Template Library. International Journal of High Performance Computing Applications, 11(3), 1997. (http://math.nist.gov/tnt/).
6. Тормасов А.Г., Пашутин Р.А., Иванов В.Д., Петров И.Б. Объектно-ориентированный подход в создании сред поддержки сложных вычислений. Тезисы докладов XL научной конференции МФТИ, Долгопрудный, 1997.
7. Roberts, C.A., Dessouky, Y. M. An Overview of Object-Oriented Simulation. Simulation, vol. 70, no. 6, pp. 359-368. (http://www.scs.org/pubs/s98indaut.html)
8. Cubert, R.M., Fishwick, P.A. OOPM: An Object-Oriented Multimodeling and Simulation. Application Framework. Simulation, vol. 70, no. 6, pp. 379-395.
9. A. M. Uhrmacher. Concepts of Object-Oriented Simulation. Transactions of the Society for Computer Simulation, vol. 14, no. 2, pp. 59-68, 1997. (http://www.scs.org/pubs/t97toc.html)
10. Simulink Concepts. In: MATLAB User`s guide. MathWorks, inc. (www.mathworks.com).
11. Евдокимов А.В. Численное моделирование осреднённого по времени кровообращения человека. Выпускная квалификационная работа бакалавра. МФТИ, 1998.
12. Бурыкин А.А. Разработка методов компьютерного моделирования функциональных систем организма человека (на примере сердечно-сосудистой системы). Магистерская диссертация. Долгопрудный, МФТИ, 2000.

**Публикации автора по теме диссертации**

1. Евдокимов А.В., Объектно-ориентированный подход в математическом и имитационном моделировании. Тезисы докладов XLII научной конференции МФТИ, 1999.
2. Евдокимов А.В., Бурыкин А.А. О хранении и представлении данных в системах моделирования. Тезисы докладов XLII научной конференции МФТИ, Долгопрудный, 1999.
3. Бурыкин А.А., Евдокимов А.В. О применении объектно-ориентированного анализа при создании сложных компьютерных моделей в физиологии. Тезисы докладов XLII научной конференции МФТИ, Долгопрудный, 1999.

Приложение

**Описание объектно-ориентированной библиотеки численных методов для задач гидромеханики и массопереноса с точки зрения её использования**

Ниже кратко описывается физический смысл всех элементов библиотеки, их назначение и отношения друг с другом. Названия элементов подчеркнуты, а названия их параметров выделены курсивом. Для использования библиотеки при конструировании разнообразных моделей приведённой в данном разделе информации вполне достаточно, в то время как для расширения возможностей библиотеки и для приспособления её к конкретным предметным областям необходима также объектно-ориентированная структура библиотеки, описанная в разделе 3.4.2.

**Течение жидкости (газа)**

1. Резервуар предназначен для хранения некоторого *объёма* жидкости (газа). Его связи (которых может быть сколько угодно) с Проточными элементами и с Узлами соединения позволяют ему обмениваться жидкостью (газом) с другими резервуарами и системами проводников соответственно.
2. Эластичный резервуар, как и обычный Резервуар, имеет *объём* и методы его расчёта по втекающим и вытекающим *потокам*. Однако в данном элементе, помимо этого, заложена модель стенки резервуара (упругая модель). Для этого Эластичный резервуар характеризуется двумя *давлениями* (внутри и снаружи) и двумя свойствами стенки – *ненапряжённым объёмом* и *жёсткостью*, то есть отношением приращения давления к приращению объёма. После расчёта объёма вычисляется соответствующее упругой модели давление внутри резервуара, которое может быть использовано для регуляции потоков через связанные с данным элементом Проточные элементы и Узлы соединения.
3. Пластичный резервуар содержит те же параметры и использует те же модели расчёта *объема* и *давления*, что и Эластичный резервуар. Отличие между ними состоит в том, что упругая модель расчёта давления дополнена пластичной, что позволяет моделировать часто встречающееся явление релаксации напряжения в стенке резервуара. С этой целью в Пластичный резервуар добавлен ещё два параметра – характерное *время релаксации* *и равновесный ненапряжённый объём*. Если мгновенно увеличить (уменьшить) *объём* резервуара, то *давление* в нём сначала резко повысится (понизится), но за время релаксации за счёт постепенного роста (уменьшения) *ненапряжённого объёма* оно приблизится к своему первоначальному значению. Следовательно, ненапряжённый объём в случае пластичного резервуара рассчитывается им самим, и регулировать его извне не имеет смысла; зато равновесный ненапряжённый объём подлежит обязательной регуляции в зависимости от разности внешнего и внутреннего давлений (именно к нему со временем стремится значение ненапряжённого объёма, поэтому без регуляции не будет релаксации).
4. Точка соединения необходима для формализованного описания системы ветвящихся проводников (см. ниже). Данный элемент содержит всего один параметр – *давление* в точке, и сам никаких специфических действий над ним не производит. Точку соединения можно связывать с любым количеством элементов типа Проводник и одного из его подтипов (причём направление связи не имеет значения), однако самый обычный способ её использования – постановка на выходе элемента типа Насос, который в таком случае будет рассчитывать давление в точке.
5. Узел соединения расширяет возможности Точки соединения и отличается от неё тем, что самостоятельно рассчитывает своё давление на основе информации о сопротивлениях проводников, с которыми он связан, и о давлениях на их противоположных концах. При этом имеется возможность притока или оттока из узла некоторого количества жидкости (газа), для чего узел соединения содержит второй параметр – *втекающий поток*.
6. Проточный элемент является простейшим типом, характеризуясь только одним параметром – *потоком*, то есть объёмом жидкости (газа), протекающей через него в единицу времени. Связывается данный элемент каждым концом с не более чем одним элементом типа Резервуар или одного из его подтипов. Если связь направлена к резервуару, то его объём будет увеличиваться, если от резервуара – уменьшаться (естественно, при условии положительности потока).
7. Проводник является Проточным элементом, обладающим некоторым *сопротивлением*, которое определяется как коэффициент пропорциональности между разностью *давлений* на концах проводника и протекающим через него *потоком*. Соответственно, поток является уже не входным, а выходным параметром по отношению к данному элементу. Этот поток рассчитывается через разность давлений в Точках соединения или Узлах соединения, с которыми он должен быть связан (каждый конец проводника должен иметь ровно одну связь).
8. Градиент обеспечивает заданную разность давлений между Узлом соединения (который должен быть связан со входом данного элемента) и Точкой соединения (с выходом). Он имеет все свойства Проводника (*поток* и *сопротивление*), однако расчёт ведёт по той же формуле в обратном направлении, определяя из неё не поток, а давление на выходе (в Точке соединения). Градиент полезен не только в качестве источника (насоса), снабжённого внутренним сопротивлением, но и для постановки краевых условий с заданным потоком на границе.
9. Насос формально расширяет возможности Проводника, но представляет собой гораздо более сложный элемент, предназначенный для описания таких циклически меняющих свои параметры объектов, как желудочек сердца. Основной функцией насоса является расчёт потока и давления на выходе через известное давление на входе и выходное сопротивление. Насос имеет два однотипных набора параметров, относящиеся к двум фазам цикла его работы – активной (сокращение) и пассивной (расслабление). Эти наборы в точности соответствуют двум Эластичным резервуарам (см. ниже), однако некоторых параметров несколько отличается. В частности, ненапряжённый объём в активной фазе соответствует минимальному объёму, объём в пассивной фазе – ударному объёму, а ненапряжённый объём в пассивной фазе – просто объёму. Помимо этих параметров и рассмотренных выше *потока* и *сопротивления*, Насос характеризуется *частотой* (обратным периодом цикла), *длительностью активной фазы* и *максимальным давлением*, которое он может развивать при заданном объёме. В связи с тем, что некоторые закономерности работы такого насоса не могут быть описаны точно, но известны по экспериментальным данным, способ расчёта трёх параметров (максимального давления, минимального объёма и объёма в пассивной фазе) не фиксируется, и подлежит обязательному определению на уровне модели. Как и Проводник, Насос может быть связан с Точкой соединения или Узлом соединения, однако связь от Насоса к Узлу лишена смысла, поскольку узел рассчитывает своё давление, а в данном случае это должен делать Насос.
10. Сосуд является сочетанием параметров и функциональных возможностей, присущих двум типам элементов – Проводнику и Резервуару. Новым свойством является то, что *сопротивление* сосуда как проводника зависит от его диаметра, а значит, и от *объёма* сосуда как резервуара. С другой стороны, *поток* через сосуд определяется не только стационарным законом пропорциональности между ним и разностью давлений; при расчёте учитывается и нестационарная поправка к потоку, равная скорости изменения объёма и формально относящаяся не к сосуду, а к связанному с его выходным концом Узлу соединения (параметр *втекающий поток*). В принципе, систему сосудов можно ввести в схему как систему проводников и резервуаров, связанных узлами, но тогда соотношение между их объёмом и сопротивлением нужно искусственно вводить на уровне модели.
11. Эластичный сосуд сочетает свойства Проводника со свойствами Эластичного резервуара. Зависимость *сопротивления* и *потока* от *объёма* и скорости его изменения точно такая же, как в обычном Сосуде, а зависимость объёма от разности давлений и свойств стенки – такая же, как в Эластичном резервуаре. Отличие в том, что давления на концах сосуда как проводника напрямую определяют давление внутри сосуда как резервуара (оно принимается равным их полусумме).
12. Пластичный сосуд отличается от Эластичного сосуда только тем, что его *ненапряжённый объём* зависит от времени, если во времени изменяется *давление* в сосуде (см. Пластичный резервуар).   
     **Обмен веществ**
13. Контейнер является основой для всех элементов, способных содержать в себе набор некоторых субстанций (которые далее для определённости называются веществами). Каждое вещество в контейнере характеризуется как своей *концентрацией*, так и общим *количеством* (массой) в контейнере. При этом в данном элементе рассчитывается только количество, а концентрация вычисляется лишь в его подтипах. Любой контейнер может быть связан с несколькими элементами типов Носитель и Место конвекции (а также с элементами их подтипов), которые обеспечивают приток или отток веществ от контейнера.
14. Резервуар-контейнер благодаря наличию у него *объёма* (как у любого резервуара) имеет возможность рассчитать не только *количество*, но и *концентрации* веществ в контейнере.
15. Эластичный контейнер является Эластичным резервуаром, способным определять *концентрации* и *массы* веществ.
16. Пластичный контейнер сочетает свойства Пластичного резервуара и способность содержать в себе вещества.
17. Носитель обеспечивает отток веществ от Контейнера или его подтипов, поэтому он может иметь не более чем одну связь с контейнером на входе и ни одной связи на выходе. Скорость оттока формализуется с помощью параметра *поток вещества* каждого из «текущих» по данному элементу веществ, однако никаких расчётов сам Носитель не производит. Если поток вещества отрицателен, Носитель представляет собой источник вещества.
18. Поглотитель реализует простейшую модель расчёта оттока веществ из контейнера за счёт их поглощения, распада или какого-либо иного взаимодействия со средой. В этой модели поток концентрации пропорционален самой концентрации с коэффициентом, равным обратному характерному времени поглощения. *Время поглощения* для каждого вещества является свойством содержащейся в Поглотителе среды, а не его собственным свойством, поэтому в число параметров оно не входит.
19. Обменник, в отличие от Носителя, может не только служить в качестве источника или стока вещества, но и обеспечивать обмен веществ между двумя Контейнерами. Для этого данный тип элемента и все его подтипы может иметь не только входную, но и выходную связь. Расчёт *потока вещества* в Обменнике не проводится.
20. Диффузионный обменник содержит в себе модель, согласно которой *поток вещества* между двумя Контейнерами пропорционален разности концентраций в них и эффективной *площади* обменника. В таком случае коэффициент этой пропорциональности (*скорость диффузии*, то есть коэффициент диффузии, отнесённый к эффективной толщине «стенки» обменника) можно считать свойством содержащейся в данном элементе среды, а не его собственным параметром.   
     **Перенос веществ**
21. Место с веществом является основой для всех элементов, способных к переносу веществ. Каждое вещество в контейнере характеризуется только своей *концентрацией*, в то время как подобный данному элемент Контейнер снабжён также количествами веществ, которые зачастую не могут быть определены. Само по себе Место с веществом не может быть связано ни с одним элементом, что позволяет вводить произвольные законы изменения концентрации веществ в нём на уровне модели.
22. Точка с веществом описывается *давлением* и *концентрациями* содержащихся в ней веществ, однако в данный элемент не заложены модели их расчёта. Данный элемент может иметь связи с теми же элементами, что и обычная Точка.
23. Узел с веществом обладает всеми свойствами Узла соединения по отношению к расчёту своего *давления* и к связыванию с другими элементами, но также характеризуется и *концентрациями* веществ.
24. Место диффузии содержит в себе методы расчёта изменений *концентраций* содержащихся в нём веществ с помощью модели диффузионного переноса. Для реализации этой модели необходимо знать коэффициенты диффузии и эффективное расстояние между разными Местами диффузии, поэтому данный элемент может быть связан только с Диффузорами. Направление связей не имеет значения, количество их может быть любым. При расчётах предполагается, что соседние (связанные Диффузором) Места диффузии имеют приблизительно одинаковый объём, поэтому в случае невыполнения этого условия необходимо использовать Диффузионные обменники вместо Диффузоров и Контейнеры вместо Мест диффузии.
25. Диффузор, в отличие от сходного с ним Диффузионного обменника, потока вещества в себе не несёт и никаких расчётов производить не умеет. Он предназначен для связывания с Местами диффузии, которые самостоятельно рассчитывают концентрации своих веществ через хранящиеся в диффузорах *коэффициенты диффузии* (свойства среды, заполняющей Диффузоры) и эффективные *длины* самих Диффузоров. Под эффективнойдлиной Диффузора понимается среднее геометрическое между его реальной диффузионной длиной и отношением объёма к площади сечения.   
     **Конвективный перенос веществ**
26. Место конвекции, как и Место с веществом, содержит в себе набор *концентраций* веществ. Помимо них, данный элемент характеризуется *втекающими потоками концентрации*, с помощью которых он (и, главное, его подтипы) может связываться с Контейнерами и вместе с ними рассчитывать конвективный перенос веществ. Возможны также связи с элементами типа Конвектор, но не с элементами его подтипов, поскольку Место конвекции не имеет *давление* в числе своих параметров.
27. Точка конвекции отличается от Места конвекции наличием *давления*, благодаря которому все связанные с ней элементы подтипов Конвектора (обладающие сопротивлением) получают возможность рассчитывать свой *поток*. Этот поток, наряду с *размерами* конвекторов, Точка конвекции использует при расчёте своих *концентраций*.
28. Узел конвекции не только содержит *давление* среди своих параметров, но и рассчитывает его методами Узла соединения. Это накладывает запрет на соединение его входа с Градиентом-конвектором и Насосом-конвектором, которые сами рассчитывают своё выходное давление. Как и Место конвекции или Точка конвекции, данный элемент может быть связан с Контейнерами, однако благодаря наличию у него *втекающего потока* эти связи получают второе назначение, аналогичное назначению связей между Узлом соединения и Резервуарами.
29. Конвектор представляет собой Проточный элемент, обладающий помимо *потока* ещё и *размером*. Используя эти параметры, он способен участвовать в расчёте конвективного переноса веществ, проводимом Местами конвекции, которые должны быть связаны с каждым из его концов.
30. Проводник-конвектор, являясь одновременно Проводником и Конвектором, может участвовать в расчёте не только переноса веществ, но и протекающего через него *потока*. Поэтому он должен иметь связи не с Местом конвекции, а с Точкой конвекции или с Узлом конвекции.
31. Градиент-конвектор участвует в связях с Точкой конвекции и Узлом конвекции аналогично тому, как обычный Градиент связывается с Точкой соединения и Узлом соединения. По отношению к переносу веществ его роль совпадает с ролью любого конвектора.
32. Насос-конвектор соединяет в себе функциональность Насоса со способностью Конвектора по переносу веществ. Он может иметь точно такие же связи, как и Градиент.
33. Сосуд-конвектор, обладая *объёмом* (как и любой Сосуд), имеет возможность изменять в зависимости от него свои свойства по отношению к переносу веществ.
34. Эластичный конвектор, по сути, является Сосудом-конвектором с упругой стенкой, и подобно ему участвует в связях с Точками конвекции и (или) Узлами конвекции.
35. Пластичный конвектор имеет все достоинства Эластичного конвектора и плюс к ним обладает более реальными способом описания своих механических характеристик.  
     **Конвективно-диффузионный перенос веществ**
36. Место переноса, как и Место с веществом, содержит в себе набор *концентраций* веществ. Помимо них, данный элемент характеризуется *втекающими потоками концентрации*, с помощью которых он (и, главное, его подтипы) может связываться с Контейнерами и вместе с ними рассчитывать конвективный перенос веществ. Одновременно с конвективным в Месте переноса происходит диффузионный перенос веществ, если оно связано с элементами типа Переносчик (но не с элементами его подтипов, поскольку данный элемент не имеет *давление* в числе своих параметров). При этом используются имеющиеся у Переносчика *коэффициенты диффузии* по отношению к каждому веществу и его *длина*.
37. Точка переноса отличается от Места переноса наличием *давления*, благодаря которому все связанные с ней элементы подтипов Переносчика (обладающие сопротивлением) получают возможность рассчитывать свой *поток*. Этот поток, наряду с *размерами* переносчиков, Точка переноса использует при расчёте своих *концентраций*.
38. Узел переноса не только содержит *давление* среди своих параметров, но и рассчитывает его методами Узла соединения. Это накладывает запрет на соединение его входа с Градиентом-переносчиком и Насосом-переносчиком, которые сами рассчитывают своё выходное давление. Как и Место переноса или Точка переноса, данный элемент может быть связан с Контейнерами, однако благодаря наличию у него *втекающего потока* эти связи получают второе назначение, аналогичное назначению связей между Узлом соединения и Резервуарами.
39. Переносчик представляет собой Проточный элемент, обладающий помимо *потока* ещё и *размером*. Используя эти параметры вместе с имеющимися у него параметрами Диффузора, он способен участвовать в расчёте конвективно-диффузионного переноса веществ, проводимом Местами переноса, которые должны быть связаны с каждым из его концов.
40. Проводник-переносчик, являясь одновременно Проводником и Переносчиком, может участвовать в расчёте не только переноса веществ, но и протекающего через него *потока*. Поэтому он должен иметь связи не с Местом переноса, а с Точкой переноса или с Узлом переноса.
41. Градиент-переносчик является Градиентом, наделённым способностью к переносу веществ. Он может иметь точно такие же связи, как и любой переносчик, за исключением возможности соединить его выход с Узлом переноса.
42. Насос-переносчик соединяет в себе функциональность Насоса со способностью Переносчика по переносу веществ. Он может иметь точно такие же связи, как и любой переносчик, за исключением возможности соединить его выход с Узлом переноса.
43. Сосуд-переносчик, обладая *объёмом* (как и любой Сосуд), имеет возможность изменять в зависимости от него свои свойства по отношению к конвективной части переноса веществ.
44. Эластичный переносчик по сути, является Сосудом-переносчиком с упругой стенкой, и подобно ему участвует в связях с Точками переноса и (или) Узлами переноса.
45. Пластичный переносчик имеет все достоинства Эластичного переносчика и плюс к ним обладает более реальными способом описания своих механических характеристик.

**Таблица 1**

Физический смысл элементов библиотеки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *1. Элементы, переносящие что-либо* | | | | |
|  | Перенос жидкости (газа) | Конвективный перенос веществ | | Перенос веществ |
| Поток | Проточный элемент | Конвектор | | Переносчик |
| Сопротивление | Проводник | Проводник-конвектор | | Проводник-переносчик |
| Градиент | Градиент | Градиент-конвектор | | Градиент-переносчик |
| Период | Насос | Насос-конвектор | | Насос-переносчик |
| Объём | Сосуд | Сосуд-конвектор | | Сосуд-переносчик |
| Диффузор | |  | | |
|  | | | | |
| *2. Элементы, которые соединяют переносящие элементы* | | | | |
|  | Место | Точка | | Узел |
| Соединение | – | Точка соединения | | Узел соединения |
| Вещество | Место с веществом | Точка с веществом | | Узел с веществом |
| Конвекция | Место конвекции | Точка конвекции | | Узел конвекции |
| Перенос | Место переноса | Точка переноса | | Узел переноса |
| Место диффузии | |  | | |
|  | | | | |
| *3. Элементы, содержащие что-либо* | | | | |
|  | Объём | Эластичность | | Пластичность |
| Резервуар | Резервуар | Эластичный резервуар | | Пластичный резервуар |
| Проводник | Сосуд | Эластичный сосуд | | Пластичный сосуд |
| Конвектор | Сосуд-конвектор | Эластичный конвектор | | Пластичный конвектор |
| Переносчик | Сосуд-переносчик | Эластичный переносчик | | Пластичный переносчик |
| Контейнер | Резервуар-контейнер | Эластичный контейнер | | Пластичный контейнер |
| Контейнер | |  | | |
|  | | | | |
| *4. Элементы, переносящие вещества между контейнерами* | | | | |
| Носитель | Поглотитель | Обменник | Диффузионный обменник | |

*Примечание*. Сосуд-конвектор и сосуд-переносчик указаны в таблице дважды – в разделах 1 и 3.

**Таблица 2**

Параметры элементов библиотеки

|  |  |  |
| --- | --- | --- |
| *№* | *Элемент* | *Параметры* |
| 1 | Резервуар | объём |
| 2 | Эластичный резервуар | объём, *ненапряжённый объём*, *жёсткость*, давление изнутри, *давление снаружи* |
| 3 | Пластичный резервуар | объём, ненапряжённый объём, *жёсткость*, давление изнутри, *давление снаружи*, *время релаксации*, ***равновесный ненапряжённый объём*** |
| 4 | Точка соединения | *давление* |
| 5 | Узел соединения | давление, *втекающий поток* |
| 6 | Проточный элемент | *поток* |
| 7 | Проводник | поток, *сопротивление* |
| 8 | Градиент | *поток*, *сопротивление* |
| 9 | Насос | поток, *выходное сопротивление*, *частота*, *длительность активной фазы*, ***максимальное давление***, ударный объём, ***объём в пассивной фазе***, *выходная жёсткость*, давление в пассивной фазе, *давление снаружи в пассивной фазе*, объём в активной фазе, ***минимальный объём в активной фазе***, *жёсткость насоса*, давление в активной фазе, *давление снаружи в активной фазе* |
| 10 | Сосуд | поток, сопротивление, *объём* |
| 11 | Эластичный сосуд | поток, сопротивление, объём, *ненапряжённый объём*, *жёсткость*, давление изнутри, *давление снаружи* |
| 12 | Пластичный сосуд | поток, сопротивление, объём, ненапряжённый объём, *жёсткость*, давление изнутри, *давление снаружи*, *время релаксации*, ***равновесный ненапряжённый объём*** |
| 13 | Контейнер | *концентрации веществ*, количества веществ |
| 14 | Резервуар-контейнер | объём, концентрации веществ, количества веществ |
| 15 | Эластичный контейнер | объём, *ненапряжённый объём*, *жёсткость*, давление изнутри, *давление снаружи*, концентрации веществ, количества веществ |
| 16 | Пластичный контейнер | объём, ненапряжённый объём, *жёсткость*, давление изнутри, *давление снаружи*, *время релаксации*, ***равновесный ненапряжённый объём***, концентрации веществ, количества веществ |
| 17 | Носитель | *потоки веществ* |
| 18 | Поглотитель | потоки веществ, (времена поглощения) |
| 19 | Обменник | *потоки веществ* |
| 20 | Диффузионный обменник | потоки веществ, *эффективная площадь*, (скорости диффузии) |
| 21 | Место с веществом | *концентрации веществ* |
| 22 | Точка с веществом | *давление*, *концентрации веществ* |
| 23 | Узел с веществом | давление, *втекающий поток*, *концентрации веществ* |
| 24 | Место диффузии | концентрации веществ |
| 25 | Диффузор | *эффективная длина*, (коэффициенты диффузии) |
| 26 | Место конвекции | концентрации веществ, *втекающие потоки веществ* |
| 27 | Точка конвекции | *давление*, концентрации веществ, *втекающие потоки веществ* |
| 28 | Узел конвекции | давление, *втекающий поток*, концентрации веществ, *втекающие потоки веществ* |
| 29 | Конвектор | *поток*, *эффективный размер* |
| 30 | Проводник-конвектор | поток, *сопротивление*, *эффективный размер* |
| 31 | Градиент-конвектор | *поток*, *сопротивление*, *эффективный размер* |
| 32 | Насос-конвектор | поток, *выходное сопротивление*, *частота*, *длительность активной фазы*, ***максимальное давление***, ударный объём, ***объём в пассивной фазе***, *выходная жёсткость*, давление в пассивной фазе, *давление снаружи в пассивной фазе*, объём в активной фазе, ***минимальный объём в активной фазе***, *жёсткость насоса*, давление в активной фазе, *давление снаружи в активной фазе*, *эффективный размер* |
| 33 | Сосуд-конвектор | поток, сопротивление, *объём*, эффективный размер |
| 34 | Эластичный конвектор | поток, сопротивление, объём, *ненапряжённый объём*, *жёсткость*, давление изнутри, *давление снаружи*, эффективный размер |
| 35 | Пластичный конвектор | поток, сопротивление, объём, ненапряжённый объём, *жёсткость*, давление изнутри, *давление снаружи*, *время релаксации*, *равновесный ненапряжённый объём*, эффективный размер |
| 36 | Место переноса | концентрации веществ, *втекающие потоки веществ*, концентрации (копии) веществ |
| 37 | Точка переноса | *давление*, концентрации веществ, *втекающие потоки веществ*, концентрации (копии) веществ |
| 38 | Узел переноса | давление, *втекающий поток*, концентрации веществ, *втекающие потоки веществ*, концентрации (копии) веществ |
| 39 | Переносчик | *поток*, *эффективный размер*, *эффективная длина*, (коэффициенты диффузии) |
| 40 | Проводник-переносчик | поток, *сопротивление*, *эффективный размер*, *эффективная длина*, (коэффициенты диффузии) |
| 41 | Градиент-переносчик | *поток*, *сопротивление*, *эффективный размер*, *эффективная длина*, (коэффициенты диффузии) |
| 42 | Насос-переносчик | поток, *выходное сопротивление*, *частота*, *длительность активной фазы*, ***максимальное давление***, ударный объём, ***объём в пассивной фазе***, *выходная жёсткость*, давление в пассивной фазе, *давление снаружи в пассивной фазе*, объём в активной фазе, ***минимальный объём в активной фазе***, *жёсткость насоса*, давление в активной фазе, *давление снаружи в активной фазе, эффективный размер*, *эффективная длина*, (коэффициенты диффузии) |
| 43 | Сосуд-переносчик | поток, сопротивление, *объём*, эффективный размер, *эффективная длина*, (коэффициенты диффузии) |
| 44 | Эластичный переносчик | поток, сопротивление, объём, *ненапряжённый объём*, *жёсткость*, давление изнутри, *давление снаружи*, эффективный размер, *эффективная длина*, (коэффициенты диффузии) |
| 45 | Пластичный переносчик | поток, сопротивление, объём, ненапряжённый объём, *жёсткость*, давление изнутри, давление снаружи, время релаксации, ***равновесный ненапряжённый объём***, эффективный размер, *эффективная длина*, (коэффициенты диффузии) |

*Примечание*. Курсивом отмечены параметры, которые можно регулировать имитационными зависимостями или вводить в качестве входных параметров. Параметры насосов и пластичных элементов, выделенные жирным шрифтом, подлежат обязательной регуляции. В скобках указаны свойства сред, заполняющих элементы, а не их параметры.

**Таблица 3**

Алгоритмы расчёта элементов библиотеки

|  |  |  |
| --- | --- | --- |
| *№* | *Элемент* | *Алгоритм вычислений на одном шаге по времени* |
| 1 | Резервуар | (Расчёт связанных элементов.) Изменение объёма на величину , где  и  – потоки через входные и выходные элементы соответственно |
| 2 | Эластичный резервуар | (Расчёт объёма V как в резервуаре.) Регуляция *ненапряжённого объёма* V0, *жёсткости* E, *давления снаружи* P0. Расчёт давления изнутри: |
| 3 | Пластичный резервуар | (Расчёт объёма V как в резервуаре.) Регуляция *времени релаксации* T, ***равновесного ненапряжённого объёма*** V00. Расчёт ненапряжённого объёма V0 согласно дифференциальному уравнению . (Расчёт давления изнутри как в эластичном резервуаре.) |
| 4 | Точка соединения | Регуляция *давления* |
| 5 | Узел соединения | Регуляция *втекающего потока q.* Изменение объёма связанного резервуара (если таковой имеется) на величину . Расчёт давления , где Ri – сопротивления связанных проводников, Pi – давления на их противоположных концах, Qj – потоки через связанные активные проводники (насосы, градиенты),  – входное сопротивление |
| 6 | Проточный элемент | Регуляция *потока* |
| 7 | Проводник | Регуляция *сопротивления R.* Расчёт потока: , где P1, P2 – давления во входном и в выходном элементе соответственно |
| 8 | Градиент | Регуляция *потока Q, сопротивления R.* Расчёт давления P2 в выходном элементе через давление P1 во входном: |
| 9 | Насос | Регуляция *частоты f*, *длительности активной фазы Ts*, *выходной жёсткости E*, *жёсткости насоса K, давления снаружи в пассивной фазе P0d, давления снаружи в активной фазе P0S*. Регуляция *выходного сопротивления* или его расчёт по формуле , где P1 и P2 – давления во входном и в выходном элементе соответственно. Обязательная регуляция ***объёма в пассивной фазе*** *Vd*, затем ***минимального объёма*** *Vm* и ***максимального давления*** *Pm*. Расчёт ударного объёма  (, ), потока , давления в выходном элементе , давления в активной фазе , объёма в активной фазе . |
| 10 | Сосуд | Регуляция *объёма* V. Расчёт сопротивления , где R0 и V0 – нормальные значения. (Расчёт потока как в проводнике.) |
| 11 | Эластичный сосуд | Регуляция *ненапряжённого объёма* V0, *жёсткости* E, *давления снаружи* P0. Расчёт давления изнутри , где P1 и P2 – давления во входном и в выходном элементе соответственно. Расчёт объёма . (Расчёт сопротивления и потока как в сосуде.) |
| 12 | Пластичный сосуд | (Расчёт давления изнутри как в эластичном сосуде.) Регуляция *времени релаксации* T, ***равновесного ненапряжённого объёма*** V00. (Расчёт ненапряжённого объёма как в пластичном резервуаре.) (Расчёт объёма как в эластичном сосуде.) |
| 13 | Контейнер | (Расчёт связанных элементов.) Изменение количеств веществ *mk* на величину , где  и  – потоки веществ через входные и выходные элементы соответственно. Регуляция *концентраций веществ* |
| 14 | Резервуар-контейнер | (Расчёт связанных элементов.) (Расчёт объёма *V* как в резервуаре.) (Расчёт количеств веществ *mk* как в контейнере.) Расчёт концентраций веществ |
| 15 | Эластичный контейнер | (Расчёт связанных элементов.) (Расчёт объёма как в эластичном резервуаре.) (Расчёт количеств веществ как в контейнере.) (Расчёт концентраций веществ как в резервуаре-контейнере.) |
| 16 | Пластичный контейнер | (Расчёт связанных элементов.) (Расчёт объёма как в пластичном резервуаре.) (Расчёт количеств веществ как в контейнере.) (Расчёт концентраций веществ как в резервуаре-контейнере.) |
| 17 | Носитель | Регуляция *потоков веществ.* (Расчёт входного элемента.) |
| 18 | Поглотитель | Расчёт потоков веществ , где *mk* – количества веществ во входном элементе, *tk* – времена поглощения веществ средой. (Расчёт входного элемента.) |
| 19 | Обменник | Регуляция *потоков веществ.* (Расчёт входного и выходного элементов.) |
| 20 | Диффузионный обменник | Регуляция *эффективной площади диффузии*. Расчёт потоков веществ , где Сk и сk – концентрации веществ во входном и входном элементах соответственно, dk – скорости диффузии веществ в среде. (Расчёт входного и выходного элементов.) |
| 21 | Место с веществом | Регуляция *концентраций веществ* |
| 22 | Точка с веществом | Регуляция *давления*, *концентраций веществ* |
| 23 | Узел с веществом | Регуляция *втекающего потока*, *концентраций веществ. (*Расчёт давления [и объёма связанного резервуара] как в узле соединения.) |
| 24 | Место диффузии | Изменение концентраций веществ ck на величину , где Li – эффективная длина связанных диффузоров, ci,k – концентрации на их противоположных концах, Di,k – коэффициенты диффузии веществ в средах диффузоров. |
| 25 | Диффузор | Регуляция *эффективной длины* |
| 26 | Место конвекции | Регуляция *втекающих потоков веществ fk*. Изменение количеств веществ в связанном контейнере (если таковой имеется) на величину . Изменение концентраций веществ  на величину , где  и  – потоки через входные и выходные элементы соответственно,  и  – концентрации веществ на их противоположных концах,  и  – их эффективные размеры, . |
| 27 | Точка конвекции | Регуляция *давления, втекающих потоков веществ*. (Расчёт концентраций веществ [и их количеств в связанном контейнере] как в месте конвекции) |
| 28 | Узел конвекции | Регуляция *втекающего потока, втекающих потоков веществ.* (Расчёт давления [и объёма связанного резервуара] как в узле соединения.) (Расчёт концентраций веществ [и их количеств в связанном контейнере] как в месте конвекции) |
| 29 | Конвектор | Регуляция *потока*, *эффективного размера* |
| 30 | Проводник-конвектор | Регуляция *эффективного размера*. (Регуляция или расчёт остальных параметров как проводнике.) |
| 31 | Градиент-конвектор | Регуляция *эффективного размера*. (Регуляция или расчёт остальных параметров как в градиенте.) |
| 32 | Насос-конвектор | Регуляция *эффективного размера*. (Регуляция или расчёт остальных параметров как в насосе.) |
| 33 | Сосуд-конвектор | Регуляция *эффективного размера*. (Регуляция или расчёт остальных параметров как в сосуде.) |
| 34 | Эластичный конвектор | Регуляция *эффективного размера*. (Регуляция или расчёт остальных параметров как в эластичном сосуде.) |
| 35 | Пластичный конвектор | Регуляция *эффективного размера*. (Регуляция или расчёт остальных параметров как в пластичном сосуде.) |
| 36 | Место переноса | Регуляция *втекающих потоков веществ*. (Расчёт концентраций веществ [и их количеств в связанном контейнере] как в месте конвекции.) Копирование концентраций. (Расчёт концентраций веществ как в месте диффузии.) Копирование концентраций обратно. |
| 37 | Точка переноса | Регуляция *давления, втекающих потоков веществ.* (Расчёт концентраций веществ и их копий [и их количеств в связанном контейнере] как в месте переноса.) |
| 38 | Узел переноса | Регуляция *втекающего потока, втекающих потоков веществ.* (Расчёт давления [и объёма связанного резервуара] как в узле соединения.) (Расчёт концентраций веществ и их копий [и их количеств в связанном контейнере] как в месте переноса.) |
| 39 | Переносчик | Регуляция *потока*, *эффективного размера*, *эффективной длины*. |
| 40 | Проводник-переносчик | Регуляция *эффективного размера*, *эффективной длины*. (Регуляция или расчёт остальных параметров как проводнике.) |
| 41 | Градиент-переносчик | Регуляция *эффективного размера*, *эффективной длины*. (Регуляция или расчёт остальных параметров как в градиенте.) |
| 42 | Насос-переносчик | Регуляция *эффективного размера*, *эффективной длины*. (Регуляция или расчёт остальных параметров как насосе.) |
| 43 | Сосуд-переносчик | Регуляция *эффективной длины*. (Регуляция или расчёт параметров как сосуде.) Присвоение эффективному размеру значения объёма |
| 44 | Эластичный переносчик | Регуляция *эффективной длины*. (Регуляция или расчёт параметров как в эластичном сосуде.) Присвоение эффективному размеру значения объёма |
| 45 | Пластичный переносчик | Регуляция *эффективной длины*. (Регуляция или расчёт параметров как в пластичном сосуде.) Присвоение эффективному размеру значения объёма |

*Примечание 1.* При моделировании течений сначала рассчитываются элементы с давлениями (точки, узлы), затем – активные проводники (градиенты, насосы), а затем – пассивные проводники (обычные проводники, сосуды). Там, где не указано особо, соседние элементы рассчитываются на том шаге алгоритма, которое соответствует выбранному численному методу.

*Примечание 2*. Алгоритм расчёта элемента «насос» пояснён в [11]. Этот алгоритм не является таким универсальным, как большинство остальных, однако его реализация без элемента (чисто имитационным путём, через функциональные связи между параметрами) требует несколько десятков объектов с большим числом связей между ними.

**Таблица 4**

Допустимые связи между элементами библиотеки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *№* | *Элемент* | *Элементы для  связей со входом* | *Число связей* | |
| *На входе* | *На выходе* |
| 1 | Резервуар | 6;5,24 | 0-∞ | 0-∞ |
| 2 | Эластичный резервуар | 6;5,24 | 0-∞ | 0-∞ |
| 3 | Пластичный резервуар | 6;5,24 | 0-∞ | 0-∞ |
| 4 | Точка соединения | 7,10,11,12;8,9 | 0-∞;0-1 | [0-∞] |
| 5 | Узел соединения | 7, 10,11,12;1,2,3 | 0-∞;0-1 | [0-∞;0-1] |
| 6 | Проточный элемент | 1,2,3 | 0-1 | 0-1 |
| 7 | Проводник | 4,5 | 1 | 1 |
| 8 | Градиент | 4,5 | 1 | 1 |
| 9 | Насос | 4,5 | 1 | 1 |
| 10 | Сосуд | 4,5 | 1 | 1 |
| 11 | Эластичный сосуд | 4,5 | 1 | 1 |
| 12 | Пластичный сосуд | 4,5 | 1 | 1 |
| 13 | Контейнер | (6);17,18,19,20; 26,27,28;36,37,38 | 0-∞ | 0-∞ |
| 14 | Резервуар-контейнер | 6;17,18,19,20; 26,27,28;36,37,38 | 0-∞ | 0-∞ |
| 15 | Эластичный контейнер | 6;17,18,19,20; 26,27,28;36,37,38 | 0-∞ | 0-∞ |
| 16 | Пластичный контейнер | 6;17,18,19,20; 26,27,28;36,37,38 | 0-∞ | 0-∞ |
| 17 | Носитель | 13,14,15,16 | 0-1 | 0 |
| 18 | Поглотитель | 13,14,15,16 | 1 | 0 |
| 19 | Обменник | 13,14,15,16 | 0-1 | 0-1 |
| 20 | Диффузионный обменник | 13,14,15,16 | 1 | 1 |
| 21 | Место с веществом | 25,29,39 | 0-∞ | 0-∞ |
| 22 | Точка с веществом | 30,33,34,35;31,32 | 0-∞;0-1 | 0-∞ |
| 23 | Узел с веществом | 30, 33,34,35 | 0-∞ | 0-∞ |
| 24 | Место диффузии | 25 | 0-∞ | 0-∞ |
| 25 | Диффузор | 24,21 | 1 | 1 |
| 26 | Место конвекции | 29;13,14,15,16 | 0-∞;0-1 | 0-∞;0-1 |
| 27 | Точка конвекции | 30,33,34,35;31,32,13,14, 15,16 | 0-∞;0-1 | 0-∞;0-1 |
| 28 | Узел конвекции | 30,33,34,35; 13,14,15,16 | 0-∞;0-1 | 0-∞;0-1 |
| 29 | Конвектор | 26,21 | 1 | 1 |
| 30 | Проводник-конвектор | 27,28;23,24 | 1 | 1 |
| 31 | Градиент-конвектор | 27,28;23,24 | 1 | 1 |
| 32 | Насос-конвектор | 27,28;23,24 | 1 | 1 |
| 33 | Сосуд-конвектор | 27,28;23,24 | 1 | 1 |
| 34 | Эластичный конвектор | 27,28;23,24 | 1 | 1 |
| 35 | Пластичный конвектор | 27,28;23,24 | 1 | 1 |
| 36 | Место переноса | 39;13,14,15,16 | 0-∞;0-1 | 0-∞;0-1 |
| 37 | Точка переноса | 40,43,44,45;41,42,13,14, 15,16 | 0-∞;0-1 | 0-∞;0-1 |
| 38 | Узел переноса | 40,43,44,45; 13,14,15,16 | 0-∞;0-1 | 0-∞;0-1 |
| 39 | Переносчик | 36,21 | 1 | 1 |
| 40 | Проводник-переносчик | 37,38;23,24 | 1 | 1 |
| 41 | Градиент-переносчик | 37,38;23,24 | 1 | 1 |
| 42 | Насос-переносчик | 37,38;23,24 | 1 | 1 |
| 43 | Сосуд-переносчик | 37,38;23,24 | 1 | 1 |
| 44 | Эластичный переносчик | 37,38;23,24 | 1 | 1 |
| 45 | Пластичный переносчик | 37,38;23,24 | 1 | 1 |