ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ГОУВПО «Пермский государственный университет»

С.М. Шаврин, Л.Н. Лядова, С.И. Чуприна

Моделирование и проектирование информационных систем

Учебно-методическое пособие

Пермь 2007

УДК 004.41
ББК 32.97
Ш14

Шаврин С.М.

Ш14 Моделирование и проектирование информационных систем: учеб.-метод. пособие / С.М. Шаврин, Л.Н. Лядова, С.И. Чуприна; Перм. гос. ун-т.– Пермь, 2007. – 152 с.: ил.

ISBN 5-7944-1035-3

Рассматриваются основные понятия, используемые при моделировании и проектировании информационных систем, а также теоретические основы разработки информационных систем различных классов. Сравниваются существующие методологические подходы к моделированию и проектированию информационных систем, в частности, структурный и объектно-ориентированный. Приведены основные сведения об унифицированном языке моделирования UML.

Пособие может быть полезно как преподавателям, ведущим занятия по дисциплинам, связанным с изучением и использованием информационных технологий, так и студентам, обучающимся разработке, сопровождению и эксплуатации информационных систем в различных предметных областях.

Рецензент – доктор физико-математических наук, профессор, директор учебного центра «Информатика» *С.В. Русаков*

Данное пособие является победителем конкурса, проведенного Пермским государственным университетом в ходе реализации инновационной образовательной программы «Формирование информационно-коммуникационной компетентности выпускников классического университета в соответствии с потребностями информационного общества» в рамках приоритетного национального проекта «Образование».

УДК 004.41
ББК 32.97

ISBN 5-7944-1035-3

СОДЕРЖАНИЕ

I. ОРГАНИЗАЦИОННО-МЕТОДИЧЕСКИЙ РАЗДЕЛ

Цели и задачи курса

Цель изучения курса – освоение современных методов и средств моделирования и проектирования информационных систем на базе унифицированного языка моделирования UML, а также формирование навыков их самостоятельного практического применения.

Задачи курса:

- введение в теорию моделирования;
- обзор и сравнение основных подходов к разработке программного обеспечения;
- обзор языка UML, его средств и возможностей;
- изучение языка UML применительно к моделированию и проектированию информационных систем;
- приобретение опыта использования языка UML на различных этапах жизненного цикла информационных систем.

Требования к уровню освоения содержания курса

При изучении курса студенты должны:

- получить *базовые навыки использования языка UML* на различных этапах жизненного цикла информационных систем;
- научиться *«читать» чужие модели*;
- получить опыт *создания своих легкочитаемых моделей* средней сложности;
- должны *научиться выбирать диаграммные методы*, наиболее подходящие для решения тех или иных задач.

Кроме того, студенты должны *усвоить основные концепции объектно-ориентированного проектирования*.

Место курса в системе основной образовательной программы

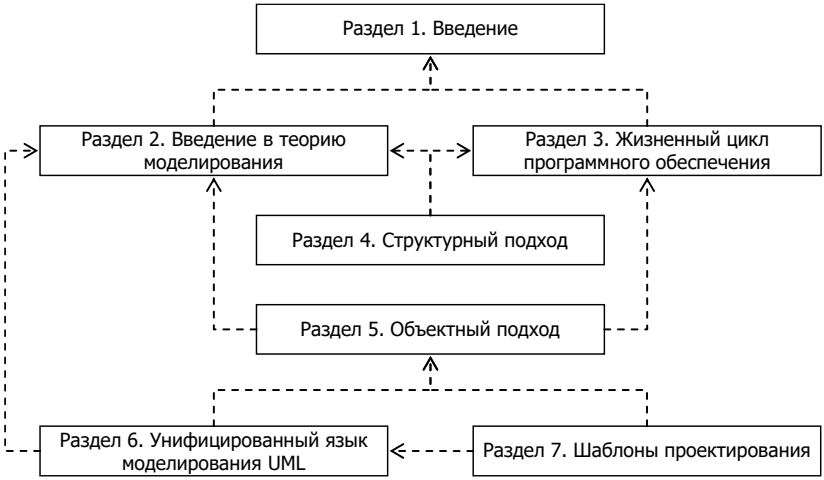
При изучении данного курса используются знания и навыки, полученные при изучении курсов «Языки программирования и методы трансляции», «Системное и прикладное программное обеспечение» и «Системы управления базами данных».

Полученные знания и навыки будут востребованы при выполнении курсовых и выпускных квалификационных работ.

II. СОДЕРЖАНИЕ КУРСА

Связь между разделами

На приведенной ниже диаграмме представлены взаимозависимости разделов данного курса. Пунктирные стрелки на этой и других диаграммах показывают рекомендуемый порядок изучения материала: стрелка означает зависимость материала раздела от знаний и навыков, полученных при изучении других разделов пособия, составляющих базу для освоения материала выбранного раздела. Качество освоения материала раздела зависит от полученных ранее (при изучении разделов, указанных стрелками) знаний и навыков.

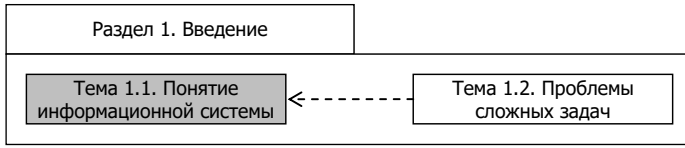


КОНСПЕКТ ЛЕКЦИЙ

Раздел 1. Введение

Тема 1.1. Понятие информационной системы

Тематический контекст



Краткое содержание

1. Понятие информационной системы.
2. Основные задачи, стоящие перед разработчиком информационной системы.
3. Основной критерий качества информационной системы.

В литературе и сети Internet можно найти множество различных определений информационной системы. Ниже дается одно из них, которое, по мнению авторов, является достаточно простым и в то же время емким.

Определение. *Информационная система* – это комплекс информационных ресурсов и технологий, предназначенный для сбора, хранения и обработки данных в рамках некоторой предметной области. Под информационными ресурсами понимаются как программные, так и аппаратные.

Достоинством данного определения является то, что явно выделяются три основные функции информационной системы, а следовательно, и три основные задачи, которые надо решить при ее разработке. Этими задачами являются организация сбора, хранения и обработки данных.

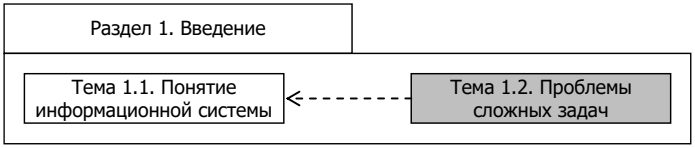
Здесь стоит обратить внимание на то, что информационная система работает с данными. Сразу возникает вопрос: а почему она информационная? Где информация? Ответ на этот вопрос прост. Информация извлекается из данных при их обработке. Собственно, основным назначением информационной системы является предоставление возможности извлечения информации из массива данных. Качество информационной системы в первую очередь определяется тем, насколько быстро и просто можно получить необходимую пользователю информацию. Все остальное вторично.

Вопросы для самоконтроля

1. Что такое информационная система?
2. Какие задачи стоят перед разработчиком информационной системы?
3. В чем разница между данными и информацией?
4. Каков основной критерий качества информационной системы?

Тема 1.2. Проблемы сложных задач

Тематический контекст



Краткое содержание

1. Проблема разбиения.
2. Проблема языка.
3. Проблема процесса.
4. Понятия методологии и технологии.

Разработка информационной системы – это сложная задача. Рассмотрим проблемы, которые возникают при решении таких задач. Прежде всего, заметим, что восприятие сложных сущностей человеком ограничено, т.е. человеку сложно охватить всю задачу сразу. Естественным решением является разделение сложной задачи на несколько более простых, каждую из которых можно легко понять и решить. Здесь возникает первая проблема – *проблема разбиения*. Как осуществлять разбиение задачи? По какому принципу выделять подзадачи?

Вторая проблема – *проблема языка* – связана с тем, что при решении сложных задач постоянно возникает необходимость четко выражать свои мысли на материальном носителе. Например, может понадобиться описать результаты анализа задачи, чтобы вернуться к ним позже. Кроме того, сложные задачи обычно решаются группой людей, которым необходимо общаться между собой. Для того чтобы это общение было эффективным, необходим язык, при помощи которого можно рассуждать,

описывать полученные результаты и принятые решения. Самым универсальным языком является естественный язык человека, однако его недостатки широко известны. Среди них можно перечислить неоднозначность, громоздкость, отсутствие наглядности и пр.

И, наконец, третья большая проблема – *это проблема процесса*. Решение сложной задачи обычно продолжительно во времени, состоит из множества маленьких шагов и, как уже говорилось выше, выполняется группой людей. Соответственно возникает проблема организации и планирования процесса решения задачи. На какие этапы разбить решение задачи? Что делать сначала, а что потом? Сколько времени займет тот или иной этап и все решение в целом? Необходимо ответить на все эти вопросы для того, чтобы иметь возможность управлять процессом решения задачи и в итоге решить ее.

После того как в области информационных технологий были осознаны проблемы сложных задач, а также был накоплен некоторый опыт их решения, начали появляться различные методологии и технологии, призванные повысить эффективность разработки программного обеспечения.

Прежде чем продолжить рассуждения, дадим определения методологии и технологии.

Определение. *Методология* – учение о структуре, логической организации, способах и средствах деятельности. Если коротко, то это *наука о методах*. В узком смысле методология является *синонимом подхода*.

Определение. *Технология* – определённая последовательность действий, средства и инструменты для достижения желаемого результата; способ преобразования данного в необходимое. Технология применяет научные достижения на практике. Целью технологии является производство продукта: материальная технология создает материальный продукт, информационная технология – информационный продукт.

Рассмотрим проблему процесса. На сегодняшний день основные, общие для большинства подходов этапы процесса сложились, это

– *анализ* – исследование проблемы (а не поиск ее решения);
– *проектирование* – поиск логического решения задачи,

обеспечивающего выполнение основных требований;
- *реализация* – конструирование системы в соответствии с логическим решением;
- *тестирование* – поиск и устранение ошибок.

На практике обычно выделяются еще некоторые этапы, такие как планирование, внедрение, поддержка и пр. Однако в рамках данного пособия интерес представляют только названные выше четыре основных этапа (в особенности первые два – анализ и проектирование). Более подробно процесс разработки рассматривается в третьем разделе данного пособия.

Решение проблемы разбиения играет ключевую роль в определении способа решения задачи в целом. Различные методологии разработки информационных систем предлагают различные способы разбиения, в чем и заключается их принципиальное различие. Например, структурный подход подразумевает использование функциональной декомпозиции задачи, а объектный подход предлагает разбивать систему на объекты. В четвертом и пятом разделах данного пособия подробно рассматриваются эти основные подходы.

Проблеме языка посвящена большая часть пособия. На сегодняшний день наибольшее распространение получили диаграммные методы, т.к. графическое представление информации наиболее наглядно. Многие методологии предлагают свои собственные диаграммные методы, с некоторыми из которых можно ознакомиться в четвертом и пятом разделах данного пособия. Кроме того, шестой раздел посвящен унифицированному языку моделирования UML, который на сегодня является промышленным стандартом в рамках объектно-ориентированного подхода.

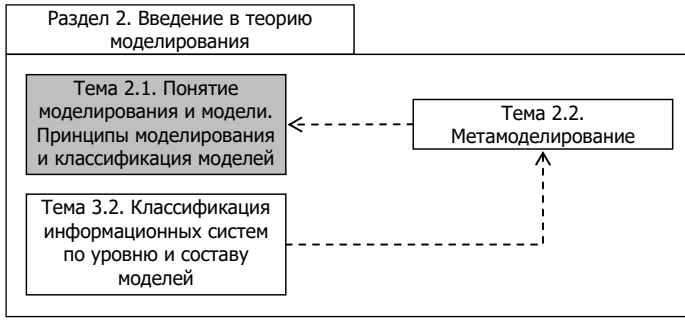
Вопросы для самоконтроля

1. В чем заключается суть проблемы разбиения?
2. В чем заключается суть проблемы языка?
3. В чем заключается суть проблемы процесса?
4. Чем методология отличается от технологии?

Раздел 2. Введение в теорию моделирования

Тема 2.1. Понятие моделирования и модели. Принципы моделирования и классификация моделей

Тематический контекст



Краткое содержание

1. Понятие моделирования как метода научного познания.
2. Понятие модели, свойства модели, разделение моделей на материальные и идеальные.
3. Четыре принципа моделирования. Классификация моделей: по точке зрения на систему, по Бучу, по степени абстракции.
4. Классификация моделей.

Определение. *Моделирование* – это метод научного познания, заключающийся в изучении некоторого объекта посредством его модели.

Определение. *Модель* – это объект-заменитель объекта–оригинала, который находится в определенном соответствии с оригиналом и обеспечивает изучение некоторых его свойств.

Определения дают представление о моделировании, однако, чтобы понять его суть, необходимо сравнить этот метод с другими.

Исторически первым методом научного познания является эксперимент, суть которого заключается в извлечении знаний об объекте путем непосредственного воздействия на него. Так, например, постепенно нагревая свинцовую болванку, можно вы-

яснить температуру ее плавления.

Эксперимент дает знания о свойствах конкретного объекта в определенных условиях, чего, конечно же, недостаточно для познания мира в целом. На помощь в этом случае приходят такие методы, как индукция и дедукция, которые позволяют на базе некоторых фактов делать общие или частные выводы. Например, проведя серию опытов со свинцовыми болванками разной формы, можно заключить, что температура плавления от нее не зависит.

Эксперимент вместе с индукцией/дедукцией является мощным инструментом познания, который исправно снабжает человечество знаниями об окружающем мире. Однако с некоторых пор начали появляться ситуации, в которых проведение эксперимента невозможно или нежелательно по каким-либо причинам. Среди всевозможных причин можно указать физические, экономические, политические, этические и пр. Например, строить самолет для того, чтобы определить его аэродинамические качества, экономически невыгодно. Лучше это сделать при помощи модели самолета в аэродинамической трубе. Другой пример: при проверке безопасности автомобиля при лобовом столкновении имеет смысл использовать манекен, а не живого водителя или пассажира. Это и есть моделирование.

Подводя итог, можно сказать, что моделирование имеет смысл использовать в тех случаях, когда возникает необходимость в изучении объектов или явлений, непосредственное наблюдение за которыми невозможно, затруднено, дорого, слишком продолжительно во времени и пр.

Перейдем теперь к моделированию информационных систем. В чем здесь особенность? В чем трудность изучения информационной системы? Зачем ее вообще изучать? Чтобы ответить на эти вопросы, начнем с конца.

Информационную систему *надо изучать*, т.к. разработчик должен точно знать, что требуется сделать. В противном случае он не сможет управлять разработкой и рискует сделать не то, что надо заказчику. *Сложность изучения* информационной системы заключается в том, что ее еще нет. Более того, обычно имеется две еще не существующих (виртуальных) системы. Звучит странно, но это так. Одна система – в голове заказчика, а вторая – это система, которую может сделать разработчик. Эти системы различаются в силу технических, технологических и про-

чих ограничений, с которыми должен считаться разработчик. Кроме того, бывают ситуации, когда заказчик смутно представляет, что ему нужно, или разработчик знает, как можно сделать лучше. Следовательно, для успешного взаимодействия между заказчиком и разработчиком нужны *две модели*: модель того, *что хочет заказчик,* и модель того, *что может разработчик*.

Перейдем к моделям. Как было сказано выше, *модель является заменителем оригинала* и *обеспечивает изучение некоторых его свойств*. Понятие модели имеет принципиальное значение для человека, т.к. все, что он знает о мире, является моделью этого мира. Выделим два основных свойства моделей:

1. Модель позволяет акцентировать наиболее важные (с точки зрения решаемой задачи) аспекты изучаемого объекта, абстрагируясь от менее важных.
2. Модель всегда «хуже» объекта-оригинала, она всегда является некоторым его упрощением.

Приведенное выше определение модели является слишком общим. Уточним понятие модели для задачи моделирования информационных систем.

Определение. *Модель* – это абстрактное описание на некотором формальном языке некоторых аспектов системы, важных с точки зрения цели моделирования.

Цели моделирования могут быть различными, например: понять предметную область, проанализировать поведение системы во времени, максимально точно записать принятое проектное решение и т.д. Таким образом, для различных целей имеем различные модели.

Моделирование имеет богатую историю, и длительный опыт его применения позволил сформулировать четыре основных принципа.

Первый принцип. Выбор модели оказывает определяющее влияние на подход к решению проблемы и на то, как будет выглядеть это решение. Иначе говоря, необходимо подходить к выбору модели вдумчиво. Правильно выбранная модель высветит самые серьезные проблемы разработки и позволит проникнуть в самую суть задачи, что при ином подходе было бы невозможно. Неправильная модель заведет в тупик, поскольку внимание будет сосредоточено на несущественных деталях.

Данный принцип удобно разобрать на примере парадигм

программирования. Каждая парадигма предлагает некоторый язык, в терминах которого решается задача. Любое решение, описанное на этом языке, можно считать моделью двоичного кода, который существует во время выполнения. Рассмотрим логическую парадигму. В этом случае предлагается описывать решение в терминах логических утверждений, предоставляются механизмы прямого/обратного вывода и механизм отката. Такой подход удобен для решения задач искусственного интеллекта и совершенно не подходит, например, для финансовых вычислений. Функциональная парадигма, напротив, очень удобна для вычислений, но в то же время ее нельзя применить для обработки сложных структур данных. Объектно-ориентированная парадигма позволяет легко справиться с множеством взаимодействующих объектов, обладающих сложной структурой и поведением, однако работа с логическими утверждениями вызовет определенные трудности.

Второй принцип. Каждая модель может быть воплощена с разной степенью абстракции. Иногда достаточно поверхностного описания системы, например, для общения с заказчиком, однако бывают случаи, когда требуется работа на уровне битов, например, при спецификации межсетевого интерфейса.

Третий принцип. Лучшие модели – те, что ближе к реальности. Под близостью к реальности в данном случае следует понимать фиксацию существенных особенностей объекта, а не излишне подробное его описание. Модель, загроможденная деталями, может только запутать, хотя ее первичное назначение – вносить ясность. Детальность модели должна соответствовать цели моделирования. В основе этого принципа лежит следующая идея. Поскольку модель всегда упрощает реальность, задача состоит в том, чтобы это упрощение не повлекло за собой каких-либо существенных потерь.

Четвертый принцип. Нельзя ограничиваться созданием только одной модели. Наилучший подход при разработке любой нетривиальной системы – использовать совокупность нескольких моделей, почти независимых друг от друга. Если система сложная, то учет всех ее аспектов в одной модели приведет к ее чрезвычайной сложности. Лучше иметь несколько моделей, делающих акценты на разных сторонах системы.

На самом высоком уровне все модели можно разделить на *материальные* и *идеальные*. Материальные модели воспроизво-

дят объект моделирования так сказать «во плоти». Примером материальной модели может служить модель самолета в аэродинамической трубе. К идеальным моделям относятся все описательные модели, т.е. они описывают объект моделирования на некотором языке. Например, фраза «паспорт – это документ, удостоверяющий личность» является моделью паспорта. В рамках данного пособия будут рассматриваться именно идеальные модели.

Классификация по точке зрения на систему. Следующим способом классификации моделей является классификация по точке зрения на систему (в соответствии с четвертым принципом моделирования). Модели можно разделить на следующие виды:

- *статические* – описывают структурные свойства;
- *динамические* – описывают поведенческие свойства;
- *функциональные* – описывают функциональные свойства.

Эти три вида моделей представляют собой три ортогональных взгляда на систему, но в то же время они связаны между собой (рис. 1).

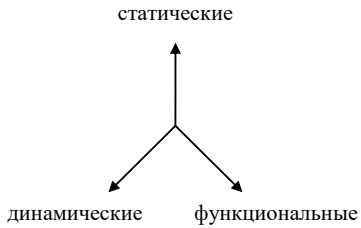


Рис. 1. Классификация моделей по точке зрения на систему

Статическая модель описывает составные части системы, их структуру, связи между ними и операции, которые они могут выполнять. Операции статической модели являются событиями динамической и функциями функциональной.

Динамическая модель описывает последовательность выполнения шагов в процессе функционирования системы. Она дает объяснение, в результате чего была вызвана та или иная операция статической модели и вычислена функция функциональной модели.

Функциональная модель описывает преобразования, осу-

ществляемые системой. Она раскрывает содержание операций статической модели и событий динамической.

Связи между такими моделями явные и обозримые, поэтому можно понять каждую модель в отдельности.

Относительная важность этих моделей зависит от предмета моделирования. В неинтерактивных вычислительных задачах важнее функциональная модель, в интерактивных системах – динамическая, а статическая модель важна для любых систем с нетривиальными структурами данных.

В объектном подходе функциональную модель обычно не выделяют, т.к. все видимые преобразования сводятся к изменению состояния объектов, которое отражается в поведенческих моделях.

Классификация Буча (в соответствии с четвертым принципом моделирования). Приведенная выше классификация является общей и применима к моделям любых систем. Применительно к программным системам вообще и к информационным системам в частности существует более подробная классификация, которую предложил Грэди Буч.

Программная система наиболее оптимально может быть описана при помощи следующих пяти взаимосвязанных моделей: *модель прецедентов*, *модель проектирования*, *модель процессов*, *модель реализации* и *модель развертывания* (рис. 2).

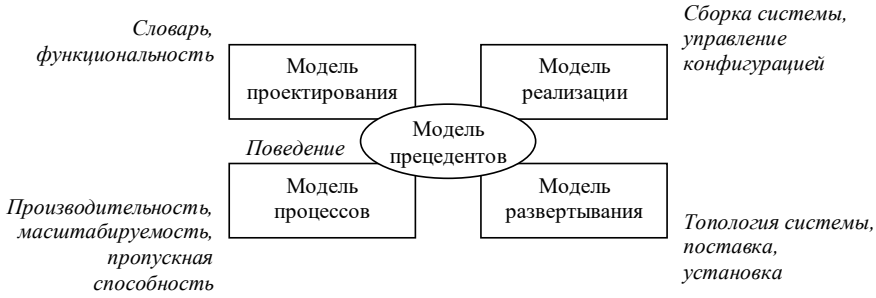


Рис. 2. Классификация Буча

Модель прецедентов – охватывает прецеденты, которые описывают поведение системы, наблюдаемое конечными пользователями, аналитиками и тестировщиками.

Модель проектирования – охватывает классы, интерфейсы и кооперации, формирующие словарь задачи и ее решения. Эта

модель поддерживает прежде всего функциональные требования, предъявляемые к системе, т.е. те услуги, которые она должна предоставлять конечным пользователям.

Модель процессов – охватывает потоки и процессы, формирующие механизмы параллелизма и синхронизации в системе. Данная модель описывает главным образом производительность, масштабируемость и пропускную способность системы.

Модель реализации – охватывает компоненты и файлы, используемые для сборки и выпуска конечного программного продукта. Данная модель предназначена в первую очередь для управления конфигурацией версий системы, составляемых из независимых (до некоторой степени) компонентов и файлов, которые могут по-разному объединяться между собой.

Модель развертывания – охватывает узлы, формирующие топологию аппаратных средств системы, на которых она выполняется.

Классификация по степени абстракции (в соответствии со вторым принципом моделирования). По степени абстракции модели можно разделить на следующие виды:

- *концептуальные модели* – высокоуровневый взгляд на задачу в терминах предметной области;
- *модели спецификации* – определяют внешний вид и внешнее поведение системы;
- *модели реализации* – отражают внутреннее устройство системы, конкретный способ реализации внешнего облика и наблюдаемого поведения.

Необходимо четко представлять разницу между этими моделями. Рассмотрим пример. На рис. 3 представлены три модели человека. Атрибут «Имя» на концептуальной модели (рис. 3, *а*) означает, что у человека есть имя. Причем имеется в виду человек из предметной области. На модели спецификации (рис. 3, *б*) тот же самый атрибут «Имя» означает, что класс «Человек» может сообщить имя и имеется механизм его обновления, т.е. в данном случае атрибут соответствует понятию свойства из языков программирования, однако нет никакой информации о том, как это свойство реализуется. И только на модели реализации (рис. 3, *в*) мы видим, что свойство «Имя» реализуется при помощи скрытого поля и двух методов доступа.

Если говорить о диаграммах классов языка UML, которые использовались в примере выше (рис. 3), то по их внешнему ви-

ду, как правило, удается определить, какая модель имеется в виду. На концептуальных моделях обычно отсутствуют операции, а также информация о видимости атрибутов и их типах. В модели спецификации добавляются операции и информация о типах, однако видимость по-прежнему не указывается, т.к. предполагается, что везде она «public». В модели реализации отображаются все члены класса – как скрытые (private, protected), так и открытые (public). Подробнее о диаграммах классов смотрите в соответствующих разделах пособия.

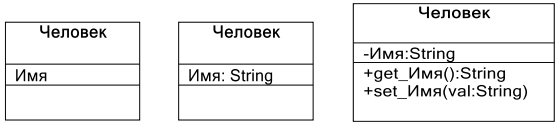


Рис. 3. Пример моделей человека
а – концептуальная модель, б – модель спецификации, в – модель реализации

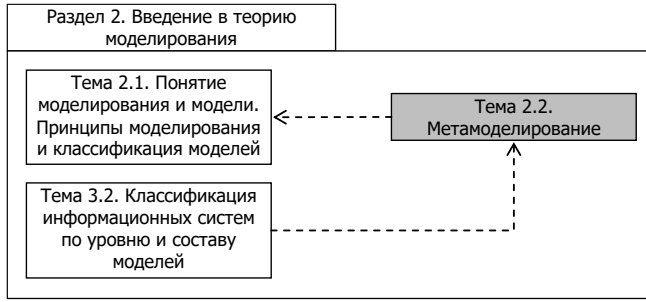
В заключение несколько слов об относительной важности этих моделей. Концептуальную модель и модель спецификации необходимо строить всегда (или почти всегда). Построение модели реализации является довольно трудоемким и достаточно механическим занятием, поэтому ее имеет смысл описывать в особо сложных случаях для иллюстрации некоторого нестандартного решения, использованного программистом или рекомендуемого ему.

Вопросы для самоконтроля

1. Что такое модель?
2. Каково основное свойство модели?
3. Когда нужно применять модель?
4. В чем особенность моделирования информационных систем?
5. Каковы основные принципы моделирования и в чем их суть?
6. Как модели можно классифицировать?

Тема 2.2. Метамоделирование

Тематический контекст



Краткое содержание

1. Понятие метамодели.
2. Разделение метамоделей на лингвистические и онтологические, связь между ними.

Как было сказано выше, модели абстрагируются от несущественных в заданном контексте деталей, что позволяет сосредоточить внимание на наиболее важных аспектах системы. Безусловно, это очень удобный инструмент, однако в последнее время все чаще возникает необходимость работать на более высоком уровне абстракции. Это связано с тем, что модели, как и все в этом мире, подвержены изменениям: может измениться постановка задачи или ее понимание, может понадобиться решить похожую задачу. В этом случае удобно было бы иметь некие «супермодели», которые описывали бы только общие свойства целого множества объектов, были бы в меньшей степени подвержены изменениям и применимы для множества схожих задач, – метамодели.

Понятие *метамодели* относительно новое, и не все еще понимают его суть. В литературе или Internet можно, например, встретить такое определение метамодели: метамодель – это модель модели. Действительно, если вспомнить определения других метасущностей (метаданные – это данные о данных, метазнания – это знания о знаниях), то по аналогии можно предложить такое определение. Однако оно приводит к определенным трудностям. Рассмотрим пример. На рис. 4 представлена организация и ее модель. Прямоугольники в данном случае обозначают

экземпляры классов, чьи имена написаны после двоеточия; сплошные линии – это связи между объектами, а пунктирная стрелка отражает зависимость и проведена от модели к моделируемому объекту.

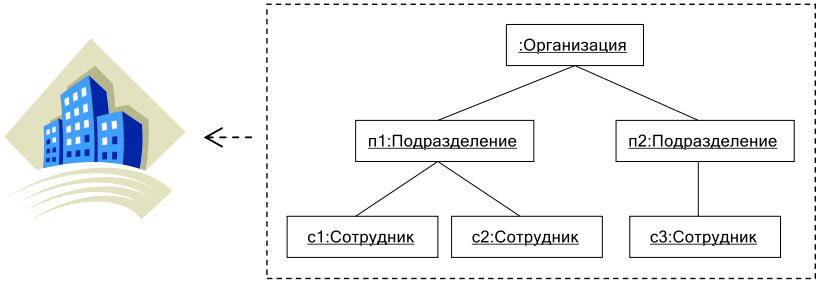


Рис. 4. Организация и ее модель

Модель на рис. 4 отражает интересующие нас аспекты некоторой организации, а именно состав подразделений и сотрудников. Упростим теперь эту модель. Можно, например, убрать информацию о подразделениях или вообще оставить только количество сотрудников (рис. 5). Модели на рис. 5 моделируют модель на рис. 4 и, следовательно, в соответствии с предложенным выше определением являются метамоделями. Однако они также применимы и к исходной организации и потому являются моделями. Таким образом, следуя приведенному определению, одну и ту же сущность можно назвать и моделью, и метамоделью, что недопустимо. Рассмотрим следующее определение.

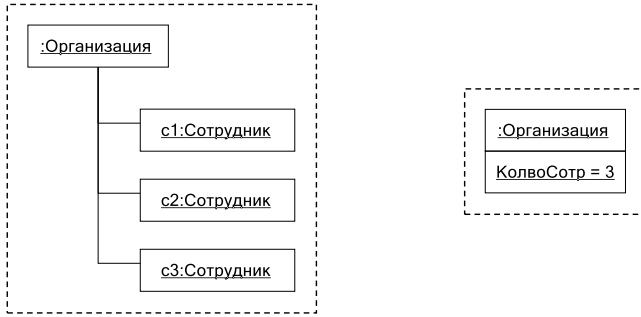


Рис. 5. Другие модели организации

Определение. *Метамодель* – это модель языка моделирования.

Исходя из данного определения все модели на рис. 4 и 5 являются *простыми моделями*. Все они моделируют исходную организацию, а также друг друга (более абстрактные модели можно считать моделями более подробных). Тогда возникает вопрос: а где же метамодель?

Заметим, что все модели на рис. 4 и 5 описаны при помощи одного и того же языка. Если их проанализировать, то можно выделить всего три элемента: *объект, связь* и *слот*. Объекты и связи были рассмотрены выше, а примером слота является количество сотрудников в модели на рис. 5 на правой модели. Если смоделировать этот язык, то получится метамодель. Пример метамодели представлен на рис. 6.

Рис. 6. Пример метамодели

На рис. 6 используется нотация диаграмм классов, которые подробно рассматриваются далее. Здесь ограничимся лишь кратким комментарием. Прямоугольник соответствует классу и состоит из трех разделов: имя, раздел атрибутов и раздел операций. Сплошные линии обозначают ассоциации, а цифры определяют множественность («*» означает «много»).

Рис. 7. Пример метамодели

Вернемся к моделям на рис. 4 и 5. Если продолжить их анализ, то можно выделить еще одну группу понятий: *организация*, *подразделение* и *сотрудник*. Это тоже язык, который можно смоделировать. На рис. 7 представлена метамодель, в терминах

которой можно описать все три модели на рис. 4 и 5. Ромбы на концах ассоциаций обозначают отношения типа «часть–целое» (закрашенный ромб – это более сильная связь).

Подведем итог. Для некоторой организации были построены три модели и две метамодели. Все три модели похожи друг на друга в том смысле, что все они описывают исходную организацию, только с различной степенью детализации. Метамодели же принципиально различны. Первая метамодель (рис. 6) описывает предметно-независимый язык, а вторая метамодель (рис. 7) – предметно-зависимый язык. Такие метамодели называют *лингвистическими* и *онтологическими* соответственно.

Определение. *Лингвистическая метамодель* – это метамодель, которая описывает предметно-независимый язык моделирования.

Определение. *Онтологическая метамодель* – это метамодель, которая описывает предметно-зависимый язык моделирования.

Внимательный читатель может заметить, что на рис. 7 изображена концептуальная модель или, другими словами, модель предметной области. Здесь нет никакого противоречия. Модель предметной области – это модель языка, на котором описывается ее состояние и, следовательно, метамодель для некоторого множества его (состояния) моделей. Таким образом, в рассуждениях о моделях и метамоделях важную роль играет контекст.

Обратимся вновь к рис. 6 и 7. Очевидно, что представленные на них модели описаны на одном и том же языке (*классы*, *ассоциации*, *атрибуты*). Следовательно, модель этого языка будет их онтологической метамоделью (или *мета*-метамоделью для моделей на рис. 4 и 5). Если же мы рассмотрим графическую нотацию, использованную на рис. 6 и 7, то модель этого языка (*прямоугольники*, *дуги*, *ромбы*, *надписи*) будет лингвистической метамоделью.

Вернемся еще раз к рис. 6. Метамодель на этом рисунке моделирует язык, на котором можно описывать только «плоские модели». Действительно, предоставляется, по сути, единственная абстракция «объект», при помощи которой предлагается описывать задачу. Все объекты равноправны и, следовательно, находятся на одном уровне. На практике же удобно иметь возможность описывать сразу несколько онтологических уровней модели.

На рис. 8 приведен пример многоуровневого онтологического моделирования. Пунктирная стрелка означает отношение «Класс–Экземпляр» и указывает на класс. Полый треугольник на конце сплошной линии означает обобщение и указывает на более общее понятие.

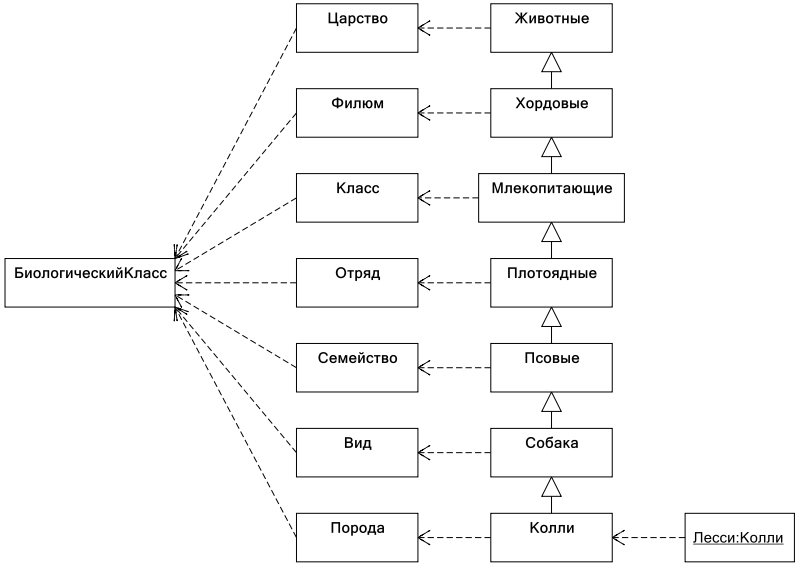


Рис. 8. Пример многоуровневого онтологического моделирования

В данном случае модель состоит из четырех онтологических уровней. На самом нижнем уровне находится объект, представляющий конкретную собаку «Лесси». На втором уровне находится часть биологической классификации. Третий уровень содержит основания классификации. И, наконец, на последнем онтологическом уровне находится самый общий класс, все экземпляры которого имеют отношение к биологии. Заметим, что каждый онтологический уровень определяет мини-язык. Каждый нижестоящий уровень описывается в терминах вышестоящего уровня. Например, колли – это конкретный экземпляр породы, а псовые – это одно из семейств. Если проследить всю цепочку от «Лесси», то получится, что «Колли» – это класс, «Порода» – метакласс, а «БиологическийКласс» – это мета-метакласс.

Определение. *Метакласс* – это класс, экземплярами которого являются другие классы.

Определение. *Класс* – это совокупность объектов, обладающих схожей структурой, поведением и семантикой.

Определение. *Семантика* – это суть, значение, смысл некоторого предмета или явления.

На рис. 9 представлен пример того, как реализуется многоуровневое онтологическое моделирование. Здесь используется единая лингвистическая метамодель, в терминах которой описываются онтологические уровни.

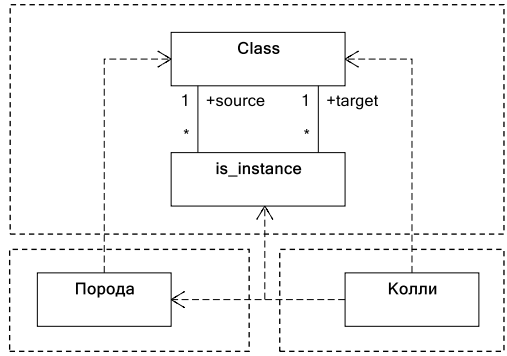


Рис. 9. Реализация многоуровневого онтологического моделирования

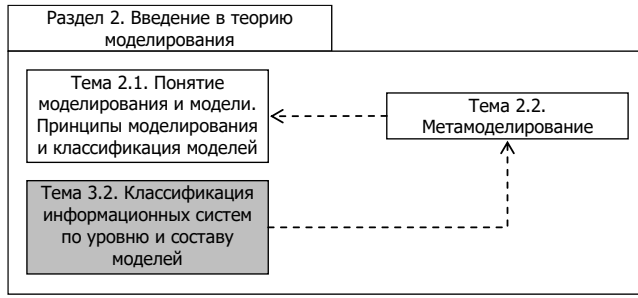
Обратите внимание, что отношение «Класс-Экземпляр» между классами «Порода» и «Колли» является экземпляром класса «is_instance», определенного в лингвистической метамодели.

Вопросы для самоконтроля

1. Что такое метамодель? Дайте определение метаметамодели.
2. В чем состоит разница между лингвистическими и онтологическими метамоделями?
3. Можно ли определить метамодель как модель модели? Обоснуйте свой ответ.
4. Придумайте свои примеры лингвистических и онтологических метамоделей.

Тема 2.3. Классификация информационных систем по уровню и составу моделей

Тематический контекст



Краткое содержание

1. Классическая четырехуровневая иерархия моделей.
2. Классификация информационных систем по количеству, уровню и способу использования моделей.

Рассмотрим классическую четырехуровневую иерархию моделей (рис. 10). В данной иерархии каждый вышестоящий уровень определяет язык для описания нижестоящего. В конкретных случаях уровней может быть меньше, в большем же количестве уровней до сих пор не было необходимости.

Рис. 10. Классическая четырехуровневая иерархия моделей

Разберем подробнее уровни классической иерархии применительно к задаче моделирования информационных систем. На уровне `M0` находятся данные, описывающие состояние предметной области, т.е. модель состояния. Уровень `M1` является онтологической метамоделью для уровня `M0` и содержит модель предметной области. Уровень `M2` определяет лингвистическую метамодель для уровней `M1` и `M0`. Другими словами, на уровне `M2` находится модель языка моделирования, с которым работают аналитики, разработчики, CASE-средства и пр. Самый верхний

уровень (M3) определяет язык, на котором описываются метамодели уровня M2 и который обычно описывается на самом себе.

В зависимости от количества уровней и способа их использования информационные системы и технологии их разработки можно разделить на несколько классов.

Традиционные информационные системы (рис. 11). В традиционной информационной системе внутри системы находятся данные, описывающие состояние предметной области. Эти данные соответствуют некоторой модели предметной области, которая может быть описана на любом языке (в том числе и на естественном языке человека). Модель разрабатывается аналитиками, после чего разработчики реализуют ее с помощью средств программирования. В случае изменения модели приходится переписывать и перекомпилировать исходные коды системы.

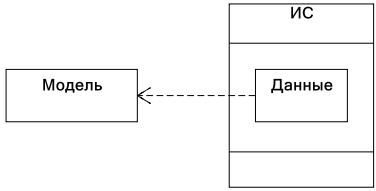


Рис. 11. Традиционные информационные системы

Традиционные CASE-технологии (рис. 12). В традиционных CASE-технологиях модель предметной области определяется формально и находится внутри CASE-средства. Модель описывается в терминах метамодели, которая может быть определена на любом языке. Как и в случае с традиционной информационной системой, метамодель разрабатывается аналитиками, после чего реализуется разработчиками. Изменение метамодели влечет за собой переписывание и перекомпиляцию CASE-средства, однако такие изменения происходят крайне редко.

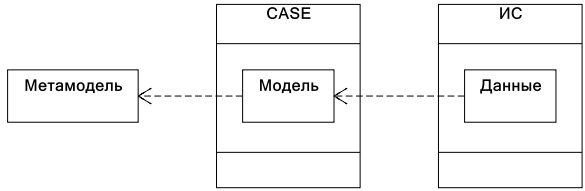


Рис. 12. Традиционные CASE-технологии

CASE-средство предоставляет инструменты для создания и редактирования моделей, а также позволяет частично сгенерировать код информационной системы. Полученная на выходе система обычно реализует все необходимые структуры данных, обеспечивает доступ к БД и предоставляет стандартный интерфейс пользователя. Поведенческие и функциональные аспекты дописываются вручную. В случае изменения модели CASE-средство позволяет заново сгенерировать код системы, при этом код, добавленный программистами, сохраняется (при соблюдении определенных правил его написания). После повторной генерации обычно требуется ручная доработка кода. Достоинством подхода является то, что существенно экономится время на начальных этапах разработки. Кроме того, поддерживается соответствие между системой и моделью. В предыдущем случае зачастую оказывается так, что реализация далеко уходит от начальной модели (которая не обновляется), что, в свою очередь, приводит к неуправляемости разработки и некоторым другим проблемам.

Информационные системы, управляемые метаданными (см. рис. 13). В этом случае также используются три уровня моделей, однако модель предметной области находится внутри информационной системы. Таким образом, система выступает в роли интерпретатора, а модель – в роли «управляющей программы». Недостатком такого подхода является то, что снижается универсальность, т.к. нет возможности что-то дописать вручную. Соответственно, метамодель должна быть максимально мощной. К достоинствам следует отнести тот факт, что при изменении модели не требуется повторять кодирование и компиляцию – информационная система просто начинает работать в соответствии с новой моделью.

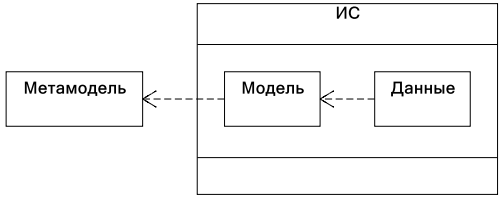


Рис. 13. Информационные системы, управляемые метаданными

В качестве примера реальной системы, использующей данный подход, можно привести систему «1С: Предприятие». Этот подход используется также в CASE-технологии «METAS», разрабатываемой в АНО науки и образования «Институт компьютинга» при участии сотрудников кафедры математического обеспечения вычислительных систем Пермского государственного университета.

Технология DSM с генерацией кода (рис. 14). Аббревиатура DSM расшифровывается как *Domain Specific Modeling*, что можно перевести как *моделирование в терминах предметной области*. В данном случае для решения каждой задачи применяется свой язык моделирования, в котором используются исключительно понятия и отношения из предметной области. Вариант реализации данной технологии представлен на рис. 14. Здесь используется мета-метамодель, которая реализуется meta-CASE-средством. При помощи этого средства описывается метамодель, которая определяет предметно-зависимый язык моделирования. На основе этой модели генерируется CASE-средство – с его помощью описывается модель предметной области и генерируется информационная система. Meta-CASE и CASE-средство могут быть объединены в одну систему, что проиллюстрировано пунктирным контуром (рис. 14).

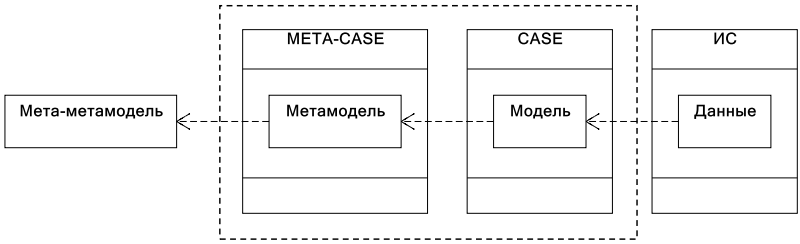


Рис. 14. Технология DSM с генерацией кода

Использование *предметно-зависимого языка* (DSL – Domain Specific Language) позволяет существенно упростить процесс создания моделей предметной области, в котором могут принимать активное участие эксперты заказчика – специалисты в рассматриваемой предметной области. Прочие плюсы и минусы, связанные с генерацией кода, совпадают с соответствующими плюсами и минусами традиционной CASE-технологии.

Технология DSM с интерпретацией метаданных (рис. 15). Данный вариант является комбинацией двух предыдущих. Метамодель, модель и данные находятся внутри информационной системы.

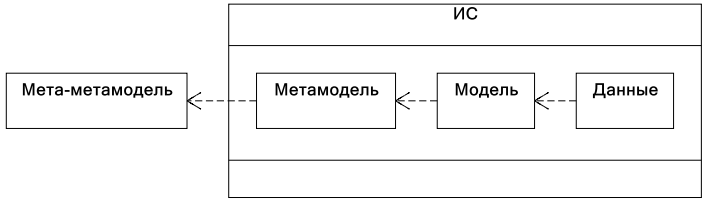


Рис. 15. Технология DSM с интерпретацией метаданных

Для того чтобы данный подход оказался применимым на практике, необходимо, чтобы мета-метамодель была максимально выразительной. Интерпретация сразу двух уровней метамоделей приводит к ощутимой потере производительности, однако при достаточной выразительности мета-метамодели получается чрезвычайно гибкая система.

Вопросы для самоконтроля

1. Назовите и охарактеризуйте уровни классической че-

тырехуровневой иерархии моделей.
2. На какие классы можно разделить информационные системы по количеству, уровню и способу использования моделей?
3. Подумайте, почему классическая иерархия включает именно четыре уровня?
4. Проклассифицируйте известные Вам информационные системы по количеству, уровню и способу использования моделей.

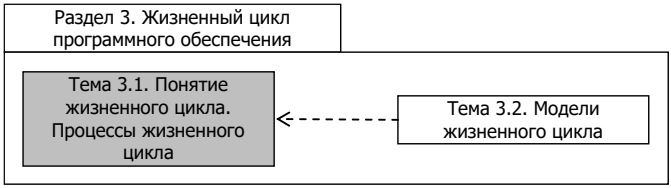
Задания для самостоятельной работы

1. Измените классическую четырехуровневую иерархию моделей с учетом разделения метамоделей на лингвистические и онтологические.
2. Уточните классификацию информационных систем с учетом разделения метамоделей на лингвистические и онтологические.

Раздел 3. Жизненный цикл программного обеспечения

Тема 3.1. Понятие жизненного цикла. Процессы жизненного цикла

Тематический контекст



Краткое содержание раздела

1. Понятие жизненного цикла.
2. Основные нормативные документы, регламентирующие этапы и состав процессов жизненного цикла.
3. Основные, вспомогательные и организационные процессы жизненного цикла.
4. Процесс разработки.
5. Процесс документирования.

Определение/ *Жизненный цикл информационных систем* – это период времени, начинающийся с момента принятия решения о необходимости создания информационной системы и заканчивающийся в момент завершения ее эксплуатации.

Российские и международные стандарты. Жизненный цикл любой информационной системы состоит из отдельных процессов, причем эти процессы могут быть как параллельными (например, разработка и документирование), так и последовательными (например, разработка и эксплуатация). Основным нормативным документом, регламентирующим состав процессов жизненного цикла информационной системы, является международный стандарт ISO/IEC 12207: 1995 «Information Technology – Software Life Cycle Processes» (ISO – International Standard Organization, IEC – International Electrotechnical Commission). Стандарты определяют структуру жизненного цикла, содержащую *процессы*, *действия и задачи*, которые должны быть выполнены при создании информационной системы.

В России создание программного обеспечения регламентировалось стандартами ГОСТ ЕСПД (единой системы программной документации – серия ГОСТ 19.XXX). Однако стандарты ориентированы на разработку небольших программ, создаваемых отдельными программистами. Создание автоматизированных систем, в состав которых входит программное обеспечение, регламентируется стандартами ГОСТ 34.602-89 («Информационная технология. Комплекс стандартов на автоматизированные системы.»), ГОСТ 34.601-90, ГОСТ 34.603-92. Но и в этих стандартах также недостаточно отражены процессы создания современных распределенных информационных систем, функционирующих в неоднородной среде, включающих средства интеграции с внешними системами и т.п., а некоторые положения этих стандартов просто устарели. Поэтому лучше ориентироваться на международные стандарты.

В соответствии со стандартом ISO/IEC 12207 все процессы жизненного цикла разделены на три группы:

- *основные процессы* (приобретение, поставка, разработка, эксплуатация и сопровождение);
- *вспомогательные процессы*, обеспечивающие выполнение основных (документирование, управление конфигурацией, обеспечение качества, верификация, аттестация, совместная оценка, аудит, разрешение проблем);
- *организационные процессы* (управление, создание инфраструктуры, усовершенствование, обучение).

Процесс разработки. В рамках данного пособия особый интерес представляет процесс *разработки*, который охватывает работы по созданию программного обеспечения и его компонентов в соответствии с заданными требованиями, включая оформление документации, подготовку материалов, необходимых для проверки работоспособности системы, обучения персонала и т.д.

Традиционно выделяют следующие этапы создания системы и ее эксплуатации:

- анализ требований;
- проектирование;
- кодирование (программирование);
- тестирование и отладка программ;
- эксплуатация и сопровождение.

Для современных корпоративных систем *процесс разра-*

ботки можно разбить на большее число этапов. Рассмотрим эти тапы.

Подготовительная работа начинается с выбора модели жизненного цикла информационной системы (см. след. тему), соответствующей масштабу, значимости и сложности проекта. Дальнейшие действия и задачи зависят от выбранной модели. На данном этапе выбираются и согласовываются стандарты, методы и средства разработки.

Анализ требований к системе – это определение ее функциональных возможностей, пользовательских требований, требований к надежности и безопасности, к внешним интерфейсам и т.д. Требования определяются исходя из информационных потребностей, критериев реализуемости и возможностей разработчика.

Проектирование архитектуры системы на высоком уровне – определение компонентов ее оборудования, программного обеспечения и операций, выполняемых эксплуатирующим систему персоналом. Архитектура должна соответствовать требованиям, предъявленным к системе, а также принятым проектным стандартам и методам.

Анализ требований к программному обеспечению предполагает определение следующих характеристик для каждого компонента программного обеспечения:

- функциональные возможности (включая среду функционирования и характеристики производительности);
- внешние интерфейсы;
- спецификации надежности и безопасности;
- эргономические требования;
- требования к используемым данным;
- требования к установке и приемке;
- требования к пользовательской документации;
- требования к эксплуатации и сопровождению.

Проектирование архитектуры включает следующие задачи (для каждого компонента системы):

- перевод требований к системе в архитектуру, определяющую на высоком уровне структуру системы, состав его компонентов;
- разработку и документирование программных интерфейсов системы и баз данных;

– разработку предварительной версии пользовательской документации;
– разработку и документирование предварительных требований к тестам и плана интеграции системы.

Детальное проектирование включает следующие задачи:
– описание компонентов системы и интерфейсов между ними на более низком (детальном) уровне, достаточном для последующего самостоятельного кодирования и тестирования;
– разработку и документирование детального проекта базы данных;
– обновление (при необходимости пользовательской документации);
– разработку и документирование требований к тестам и плана тестирования;
– обновление плана интеграции компонентов системы.

Кодирование и тестирование охватывает следующие задачи:
– *программирование* (*кодирование*) и *документирование* каждого компонента системы и базы данных, а также совокупности тестовых процедур и данных для них;
– *тестирование* и *отладка* каждого компонента на соответствие установленным требованиям и документирование результатов каждого теста;
– *обновление* (при необходимости) *пользовательской документации*;
– *обновление* (при необходимости) *плана интеграции*.

Интеграция программного обеспечения предусматривает сборку разработанных компонентов системы в соответствии с планом. Для агрегированных компонентов разрабатываются квалификационные требования (наборы критериев или условий, которые должны быть выполнены, чтобы квалифицировать программный продукт как готовый к использованию).

Квалификационное тестирование программного обеспечения выполняет разработчик в присутствии заказчика по всем разделам требований. При этом проверяется соответствие документации требованиям и ее адекватность разработанным компонентам.

Интеграция системы заключается в сборке всех компонентов информационной системы (включая программное обеспечение и оборудование).

Квалификационное тестирование системы устанавливает соответствие созданной информационной системы совокупности всех требований к ней. При этом производится проверка и оформление полного комплекта документации.

Установка программного обеспечения информационной системы осуществляется ее разработчиком в среде и на оборудовании, предусмотренных договором. В процессе установки проверяется работоспособность.

Приемка системы предусматривает оценку результатов квалификационного тестирования и установки. Выполняется заказчиком с помощью разработчика (поставщика). Разработчик передает систему заказчику в соответствии с договором и обеспечивает обучение персонала и поддержку.

Эксплуатация системы – это процесс, который охватывает действия и задачи оператора (т.е. организации, эксплуатирующей систему). Этот процесс включает подготовительную работу (планирование работ, выполняемых в процессе эксплуатации, установку эксплуатационных стандартов, определение процедур разрешения проблем возникающих в процессе эксплуатации). Кроме того, при вводе в эксплуатацию очередной редакции системы осуществляется эксплуатационное тестирование, после чего она передается в эксплуатацию.

Процесс сопровождения предусматривает действия и задачи, выполняемые сопровождающей организацией (возможно, разработчиком). Этот процесс активизируется при необходимости изменения (модификации программного обеспечения, его модернизации или адаптации, повышения производительности, которые вызваны проблемами или необходимостью настройки на новые условия, изменившиеся требования, необходимостью переноса в другую среду). Изменения в программном обеспечении могут вноситься и в целях исправления ошибок. Процесс сопровождения должен гарантировать целостность системы, миграцию имеющихся компонентов информационной системы в изменившуюся среду.

При модификации программного обеспечения и реструкту-

ризации базы данных должны выполняться те же действия, что и при разработке системы. Условия сопровождения должны оговариваться договором.

Процесс документирования. Одним из основных *вспомогательных процессов* является процесс *документирования*, предусматривающий формализованное описание информации, создаваемой в течение всего жизненного цикла информационной системы. Каждый этап жизненного цикла сопровождается разработкой документации (технического задания, эскизного и технического проектов, описания программы, руководств пользователей, программистов, администраторов и т.п.).

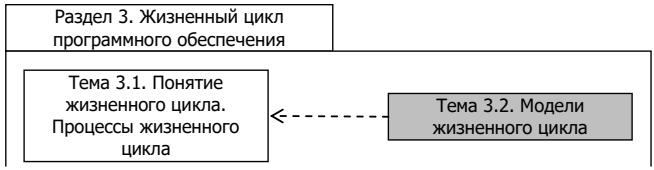
С точки зрения разработчика чрезвычайно важно документирование результатов этапов анализа и проектирования. Для этого обычно используют графические языки моделирования, такие как UML.

Вопросы для самоконтроля

1. Что такое жизненный цикл?
2. Какие нормативные документы регламентируют этапы и состав процессов жизненного цикла?
3. Опишите основные, вспомогательные и организационные процессы жизненного цикла.
4. Какие основные этапы можно выделить в процессе разработки информационной системы?

Тема 3.2. Модели жизненного цикла

Тематический контекст



Краткое содержание

1. Каскадная модель жизненного цикла.
2. Спиральная модель жизненного цикла.

Существуют различные модели жизненного цикла, определяющие порядок исполнения этапов и критерии переходов от одного этапа к другому. Каждая из моделей имеет определенные преимущества при реализации в конкретных условиях. Самое широкое распространение получили *каскадная* и *спиральная* модели жизненного цикла. Рассмотрим их подробнее.

Каскадная модель. Каскадная модель возникла в 70-е гг. XX в. и предполагает, что переход к следующему этапу происходит после полного завершения предыдущего этапа (рис. 16).

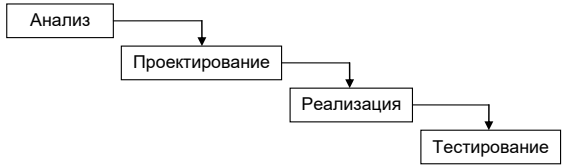


Рис. 16. Каскадный жизненный цикл

Достоинством каскадного жизненного цикла является его простота. Каждый этап завершается созданием полного комплекта документации, и есть возможность для выполнения очередного этапа привлечь стороннюю организацию или просто другой отдел. Однако существуют и минусы. Во-первых, каскадный жизненный цикл дает поздние результаты. Заказчику можно что-то показать не раньше, чем начнется этап тестирований. Такая ситуация существенно повышает риски. Во-вторых, каскадный жизненный цикл не предусматривает возврата на пройденные этапы, необходимость в котором на практике возникает гораздо чаще, чем хотелось бы.

Спиральная модель. Спиральная модель жизненного цикла появилась в конце 80-х гг. Ее возникновение тесно связано с развитием объектно-ориентированного подхода. В данном случае процесс разработки разбивается на итерации, для каждой из которых последовательно выполняются этапы анализа, проектирования, реализации и тестирования (рис. 17). Итерации могут быть как короткими (экстремальное программирование), так и длинными (более традиционные подходы). На каждой итерации берется небольшая часть системы и реализуется в необходимом объеме. С каждой новой итерацией функциональность системы растет.

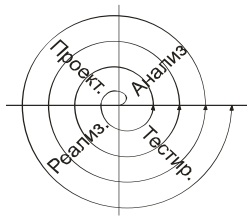


Рис. 17. Спиральный жизненный цикл

Достоинством такого подхода является то, что он лучше отражает действительность. В данном случае возврат, например, к этапу анализа не является чрезвычайной ситуацией. Кроме того, спиральный жизненный цикл дает быстрые результаты. Уже после первой итерации можно что-то показывать заказчику и, при необходимости, уточнять задачу.

При использовании спиральной модели в процессе проектирования есть возможность провести анализ возможных рисков и издержек. Еще одно преимущество модели – ориентация на развитие и модификацию программного обеспечения в процессе его проектирования, что особенно важно при создании сложных информационных систем.

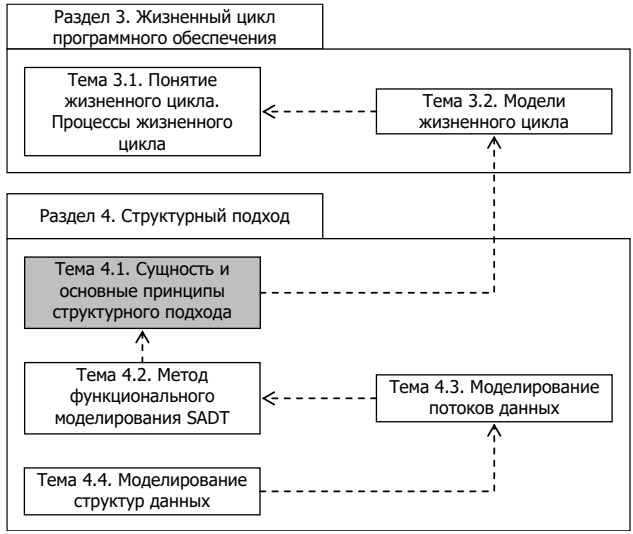
Вопросы для самоконтроля

1. Какие существуют преимущества и недостатки у каскадного жизненного цикла?
2. Какие существуют преимущества и недостатки у спирального жизненного цикла?

Раздел 4. Структурный подход

Тема 4.1. Сущность и основные принципы структурного подхода

Тематический контекст



Краткое содержание

1. Сущность структурного подхода. Базовые принципы структурного подхода: а) «разделяй и властвуй», б) принцип иерархического упорядочения, в) абстрагирования, г) принцип непротиворечивости, д) принцип структурирования данных.
2. Плюсы и минусы структурного подхода.

Исторически первым был структурный подход, который возник в конце 60-х вместе с появлением методологии SADT (Structured Analysis and Design Technique), которая позже вошла в набор стандартов IDEF (Icam DEFinition). Набор этих стандартов был разработан в рамках программы ICAM (Integrated Computer Automated Manufacturing), проводимой по инициативе ВВС США.

Суть структурного подхода заключается в *функциональной декомпозиции*, т.е. задача представляется как одна большая функция, которая преобразует входные данные в выходные. Данная функция разбивается на подфункции, которые, в свою очередь, тоже разбиваются на подфункции, и так до тех пор, пока в очередном разбиении не отпадает необходимость вследствие тривиальности соответствующей функции.

В качестве языка используется графическая нотация диаграмм SADT. Кроме диаграмм SADT в структурном подходе используются также диаграммы потоков данных (*DFD – Data Flow Diagram*), на которых изображаются потоки данных между функциональными узлами системы, а также диаграммы «сущность–связь» (*ERD – Entity-Relationship Diagram*), которые описывают структуры данных.

Принципы структурного подхода:

1. *Разделяй и властвуй*. Для решения сложной задачи используется ее разбиение на несколько более простых подзадач.
2. *Иерархическое упорядочивание*. Решение задачи представляется в виде иерархии описаний (функций, потоков данных), где описания вышестоящих уровней детализируются при помощи описаний нижестоящих уровней.
3. *Абстрагирование*. При решении задачи рассматриваются только важные ее аспекты; все незначительное отбрасывается.
4. *Формализация*. Все принятые решения описываются формально.
5. *Непротиворечивость*. Никакая часть системы не может противоречить никакой другой. Все должно быть согласованно и находиться в соответствии.

Достоинства структурного подхода. Пожалуй, самым большим достоинством структурного подхода является возможность эффективной реализации. Действительно, единственное, на что приходится тратить процессорное время кроме решения собственно задачи, – это на вызовы функций и возвраты из них. Таким образом, структурный подход оказывается полезным в

тех случаях, когда важна максимальная производительность, например, для задач реального времени.

Недостатки структурного подхода. Основным недостатком структурного подхода является его чувствительность к изменениям – малейшее изменение в постановке задачи может привести к необходимости серьезной переделки всей иерархии, т.к. все функции жестко связаны в иерархию и определяются друг через друга. В случае с информационными системами, когда постановка часто меняется (например, вследствие естественной эволюции предметной области), такое свойство подхода является крайне нежелательным.

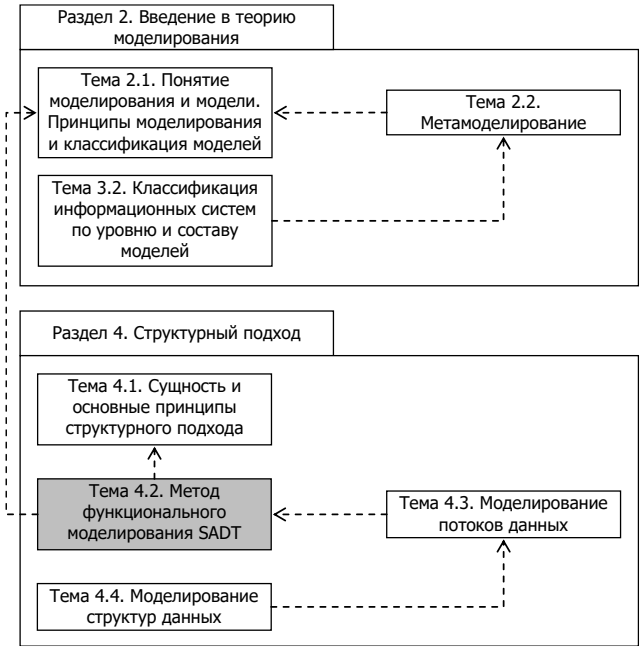
Модель процесса. Для структурного подхода характерен *каскадный жизненный цикл* системы, когда каждый последующий этап решения задачи выполняется после полного завершения предыдущего этапа. Это является следствием принципа иерархической упорядоченности. Система (как и ее описание) представляет собой иерархическую структуру, в которой вышестоящие уровни определяются через нижестоящие. Любую вершину в иерархии можно считать определенной только в том случае, если определены все ее потомки. Другими словами, иерархия не может быть определена наполовину. Вследствие этого в рамках структурного подхода разработка информационной системы ведется строго поэтапно – сначала полностью выполняется этап анализа (строится полная иерархия моделей анализа), затем полностью выполняется этап проектирования (строится полная иерархия моделей проектирования) и т.д.

Вопросы для самоконтроля

1. В чем сущность структурного подхода?
2. Определите базовые принципы структурного подхода.
3. Почему структурному подходу свойственен каскадный жизненный цикл?
4. Почему структурный подход позволяет добиться наиболее эффективной реализации?

Тема 4.2. Метод функционального моделирования SADT

Тематический контекст



Краткое содержание

1. Основные понятия.
2. Состав функциональной модели.
3. Построение иерархии диаграмм.
4. Типы связей между функциями.

SADT. Методология SADT представляет собой совокупность методов, правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области. Функциональная модель SADT отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями.

Функции. Основным элементом функциональной модели SADT является *функция*, которая на диаграмме изображается в виде прямоугольника (рис. 18). Слева у функции находятся входы, а справа – выходы. Кроме того, у функции могут быть

управляющие параметры – данные, которые непосредственно функцией не обрабатываются, но от которых зависит результат, например, процентная ставка. Дополнительно может указываться исполнитель (или его еще называют механизмом) – лицо или техническое средство, которое выполняет данную функцию.

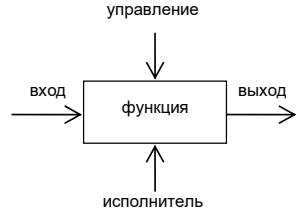
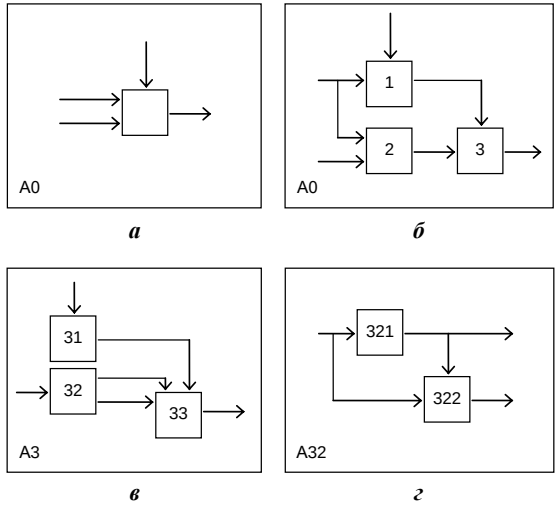


Рис. 18. Графическая нотация диаграмм SADT

Иерархия диаграмм. Принцип построения диаграмм заключается в следующем. Сначала строится корневая диаграмма, на которой моделируемый объект (вся система в целом или какая-то ее подсистема) представляется в виде одной функции, для которой определяются все входные параметры, выходные величины, управляющая информация, а также исполнители (рис. 19, *а*). После этого полученная диаграмма детализируется путем разбиения основной функции на несколько подфункций (рис. 19, *б*). Далее процесс детализации рекурсивно повторяется для каждой нетривиальной подфункции (рис. 19, *в*, *г*). Таким образом строится иерархия диаграмм.

Каждая диаграмма в модели имеет свой номер, который отражает ее положение в иерархии диаграмм. Например, диаграмма с номером `A32` является детализацией функции № 2 на диаграмме `A3`, которая, в свою очередь, является детализацией функции № 3 на диаграмме `A0`.

При построении иерархии диаграмм необходимо придерживаться двух основных правил. Во-первых, все дуги, у которых нет начала или конца на данной диаграмме, должны начинаться или соответственно продолжаться на родительской диаграмме. Например, если у функции № 2 на диаграмме `A0` есть два входа и один выход, то на диаграмме `A2` может быть только две внешних входящих дуги, одна внешняя исходящая и ни одной внешней дуги управления. Выполнение данного требования является необходимым условием полноты и непротиворечивости полученной функциональной модели.



а *б* *в* *г*

Рис. 19. Детализация функций на дочерних диаграммах

Второе правило касается детальности разбиения. Не следует сразу разбивать функцию на множество маленьких подфункций, лучше выделить несколько крупных подфункций, которые впоследствии можно детализировать на отдельных диаграммах. Критерием может служить правило «7 ± 2», т.е. на диаграмме не должно быть более пяти-семи (в крайнем случае, девяти) функций.

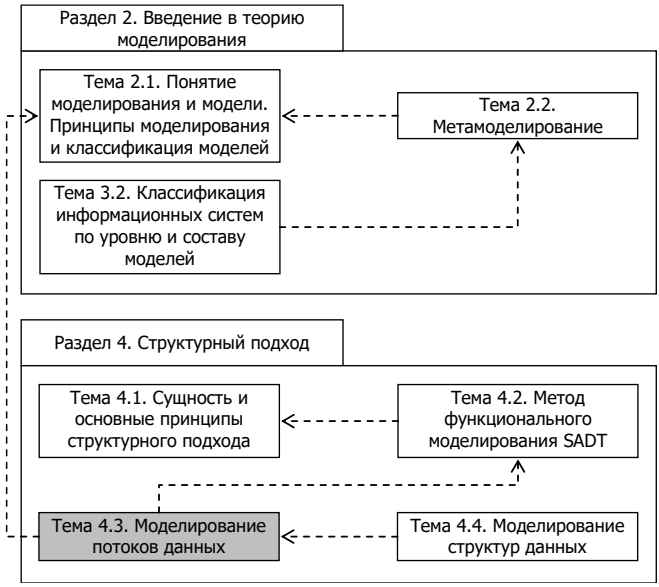
Именование функций и дуг. На рис. 19 используются неименованные функции и дуги, поскольку это чисто иллюстративный пример. На практике же каждой функции и дуге должно быть присвоено осмысленное имя, отражающее ее суть.

Вопросы для самоконтроля

1. Что изображается на диаграммах SADT?
2. Каким образом функции упорядочиваются иерархически?
3. Что такое вход, выход, исполнитель и управление?

Тема 4.3. Моделирование потоков данных

Тематический контекст



Краткое содержание

1. Основные понятия.
2. Состав диаграмм потоков данных.
3. Построение иерархии диаграмм.

Моделирование потоков данных во многом похоже на построение функциональных моделей SADT, различаются лишь акценты. В соответствии с данной методологией модель системы определяется как иерархия диаграмм потоков данных (DFD), описывающих асинхронный процесс преобразования информации от ее ввода в систему до выдачи пользователю. Диаграммы верхних уровней иерархии (контекстные диаграммы) определяют основные процессы или подсистемы ИС с внешними входами и выходами. Они детализируются при помощи диаграмм нижнего уровня. Такая декомпозиция продолжается до тех пор, пока дальнейшая детализация не потеряет смысл ввиду простоты полученных процессов.

Моделируемая система с точки зрения потоков данных выглядит следующим образом. Источники информации, называемые внешними сущностями, порождают информационные потоки, переносящие информацию к подсистемам или процессам. Те в свою очередь преобразуют информацию и порождают новые потоки, которые переносят информацию к другим процессам или подсистемам, накопителям данных или внешним сущностям – потребителям информации. Соответственно на диаграммах потоков данных используются следующие элементы: *внешние сущности*, *потоки данных*, *подсистемы*, *процессы* и *накопители данных*. Рассмотрим их подробнее.

Внешние сущности. Внешняя сущность – это объект, находящийся за пределами границы системы и являющийся источником или потребителем информационных потоков. В качестве внешних сущностей могут выступать как физические лица (например, заказчик), так и другие системы (например, платежная система).

На диаграмме внешние сущности изображаются в виде прямоугольника, который находится над плоскостью листа и отбрасывает на него тень (рис. 20). Такое обозначение призвано подчеркнуть «внешность» соответствующего объекта.

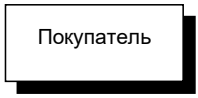


Рис. 20. Пример внешней сущности

Подсистемы. Подсистема является группирующим элементом модели и позволяет объединять родственные процессы или другие подсистемы. Данные непосредственно подсистема не обрабатывает и, следовательно, она должна обязательно содержать хотя бы один процесс или накопитель данных.

На диаграммах потоков данных подсистемы изображаются в виде прямоугольника со скругленными краями (рис. 21). В верхней части фигуры указывается номер подсистемы, а в средней – ее название, которое должно представлять собой предложение с подлежащим и соответствующими определениями и дополнениями. Кроме этого в нижней части фигуры дополнительно может указываться имя проектировщика.

Рис. 21. Пример подсистемы

Процессы. Процессы являются преобразующими сущностями, которые в соответствии с некоторым алгоритмом преобразуют входные потоки данных в выходные. Физически процесс может быть реализован различными способами. Например, преобразование может осуществлять некоторый отдел организации, программа, устройство и т.д.

Графическое представление процесса на диаграмме совпадает с таковым для подсистемы (рис. 21). Разница заключается в способе их именования. В качестве имени процесса должно указываться предложение с активным недвусмысленным глаголом в неопределенной форме, за которым следуют существительные в винительном падеже. Кроме того, в нижней части фигуры указывается физическая реализация процесса.

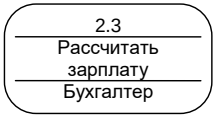


Рис. 22. Пример процесса

Накопители данных. Накопитель данных представляет собой абстрактное устройство, предоставляющее услуги сохранения и извлечения данных. Физически накопителем данных может являться картотека, таблица базы данных, файл на диске и т.д. Способ изображения накопителей на диаграмме потоков данных проиллюстрирован на рис. 23.

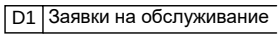


Рис. 23. Пример накопителя данных

Накопители данных идентифицируются с помощью буквы «D» и номера. Имя накопителя должно отражать содержание

хранимых данных. Набор накопителей является прообразом базы данных и должен быть увязан с информационной моделью при дальнейшем анализе.

Потоки данных. Поток данных определяет информацию, передаваемую через некоторое соединение от источника к приемнику. Реальный поток данных может быть информацией, передаваемой по кабелю между двумя устройствами, пересылаемыми по почте письмами, магнитными лентами или дискетами, переносимыми с одного компьютера на другой и т.д.

Поток данных на диаграмме изображается линией, оканчивающейся стрелкой, которая показывает направление потока (рис. 24). Каждый поток данных имеет имя, отражающее его содержание.

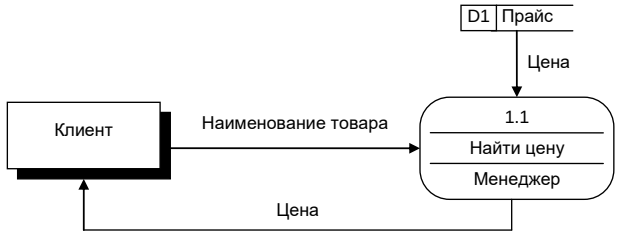


Рис. 24. Пример диаграммы потоков данных

Иерархия диаграмм. При построении диаграмм потоков данных используется тот же принцип, что и при построении SADT-диаграмм. Сначала строится корневая диаграмма, на которой определяются внешние сущности и генерируемые или потребляемые ими потоки данных. Система на корневой диаграмме представляется в виде одной подсистемы (процесса) или в виде набора основных подсистем (процессов). Далее полученная диаграмма уточняется на дочерних диаграммах до тех пор, пока на очередном шаге полученные процессы не окажутся примитивными. Таким образом, как и в случае с методологией SADT, получаем иерархию диаграмм.

При построении иерархии диаграмм потоков данных необходимо придерживаться правил, аналогичных правилам, изложенным в предыдущем разделе, т.е. все потоки данных, у которых нет начала или конца на данной диаграмме, должны начи-

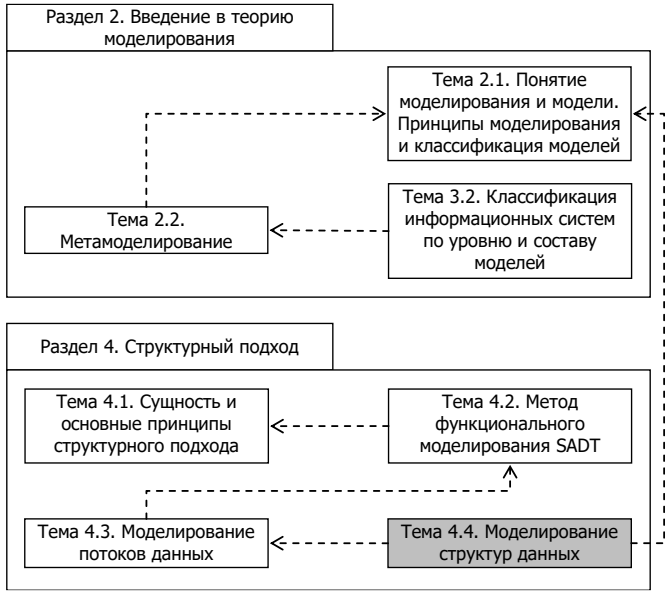
наться или, соответственно, продолжаться на родительской диаграмме. Кроме того, количество сущностей на каждой диаграмме должно удовлетворять правилу «7 ± 2».

Вопросы для самоконтроля

1. Что изображается на диаграммах потоков данных?
2. Чем подсистема отличается от процесса?
3. Можно ли остановить процесс детализации на диаграмме, содержащей подсистемы? Почему?

Тема 4.4. Моделирование структур данных

Тематический контекст



Краткое содержание

1. Основные понятия.
2. Состав диаграмм «сущность–связь».
3. Сущности, связи, атрибуты.

Метод Баркера. Цель моделирования данных состоит в обеспечении разработчика информационной системы концептуальной схемой базы данных в форме одной модели или нескольких локальных моделей, которые относительно легко могут быть отображены в любую систему баз данных.

Наиболее распространенным средством моделирования данных являются *диаграммы «сущность–связь»* (*ER-диаграммы, Entity Relationship Diagrams, ERD*). С их помощью определяются важные для предметной области объекты (сущности), их свойства (атрибуты) и отношения друг с другом (связи). Диаграммы «сущность-связь» непосредственно используются для проектирования реляционных баз данных.

Нотация ER-диаграммы была впервые введена П. Ченом (Chen) и получила дальнейшее развитие в работах Баркера. Метод Баркера отличается большей наглядностью и выразительностью. Именно он и будет рассматриваться далее.

Определение. *Сущность* (*Entity*) – это реальный либо воображаемый объект, имеющий большое значение для рассматриваемой предметной области, информация о котором подлежит хранению. Графическое изображение сущности представлено на рис. 25.

Рис. 25. Пример сущности

Каждая сущность должна обладать некоторыми свойствами:

- каждая сущность должна иметь уникальное имя, и к одному и тому же имени должна всегда применяться одна и та же интерпретация. Одна и та же интерпретация не может применяться к различным именам, если только они не являются псевдонимами;
- сущность обладает одним или несколькими атрибутами,

которые либо принадлежат сущности, либо наследуются через связь;
- сущность обладает одним или несколькими атрибутами, которые однозначно идентифицируют каждый экземпляр сущности;
- каждая сущность может обладать любым количеством связей с другими сущностями модели.

Определение. *Связь* (*Relationship*) – поименованная ассоциация между двумя сущностями, значимая для рассматриваемой предметной области. Связь – это ассоциация между сущностями, при которой, как правило, каждый экземпляр одной сущности, называемой родительской сущностью, ассоциирован с произвольным (в том числе нулевым) количеством экземпляров второй сущности, называемой сущностью-потомком, а каждый экземпляр сущности-потомка ассоциирован в точности с одним экземпляром сущности-родителя. Таким образом, экземпляр сущности-потомка может существовать только при существовании сущности родителя.

Связи может даваться имя, выражаемое грамматическим оборотом глагола и помещаемое возле линии связи. Имя каждой связи между двумя данными сущностями должно быть уникальным, но имена связей в модели не обязаны быть уникальными. Имя связи всегда формируется с точки зрения родителя, так что предложение может быть образовано соединением имени сущности-родителя, имени связи, выражения степени и имени сущности-потомка.

Графические обозначения для степени связности и обязательности приведены на рис. 26.

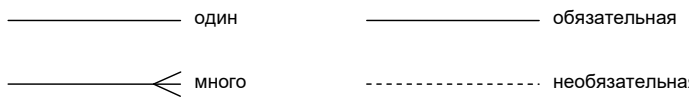


Рис. 26. Графическая нотация для связей

На рис. 27 приведен пример использования связей на ER-диаграмме. В данном случае личность может иметь несколько документов (паспорт, загранпаспорт, военный билет и т.д.) и ученую степень; одну и ту же ученую степень могут иметь несколько личностей, а с каждым документом связана одна и только одна личность.

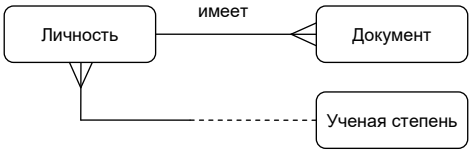


Рис. 27. Пример использования связей

Определение: *Атрибут* – любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности.

Каждый атрибут идентифицируется уникальным именем, выражаемым грамматическим оборотом существительного, описывающим представляемую атрибутом характеристику. Атрибуты изображаются в виде списка имен внутри блока ассоциированной сущности, причем каждый атрибут занимает отдельную строку.

Атрибут может быть либо обязательным (помечается символом «*»), либо необязательным (помечается символом «○»). Обязательность означает, что атрибут не может принимать неопределенных значений. Кроме того, атрибут либо может быть описательным (т.е. обычным дескриптором сущности), либо входить в состав уникального идентификатора (помечается символом «#»). Примеры различных видов атрибутов приведены на рис. 28.

Рис. 28. Пример ER-диаграммы

Определение. *Уникальный идентификатор* – это атрибут или совокупность атрибутов и/или связей, предназначенная для уникальной идентификации каждого экземпляра данного типа сущности. В случае полной идентификации каждый экземпляр данного типа сущности полностью идентифицируется своими собственными ключевыми атрибутами, в противном случае в его идентификации участвуют также атрибуты другой сущности-родителя. Например, на рис. 28 для идентификации документа используются его серия, номер, а также идентификатор вида, что изображается при помощи перечеркнутого конца связи.

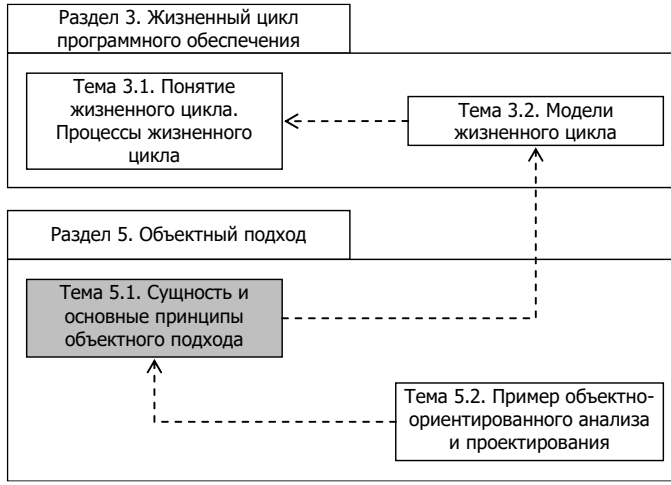
Вопросы для самоконтроля

1. Что изображается на диаграммах «сущность–связь»?
2. Что такое сущность, связь и атрибут?
3. Как диаграммы «сущность–связь» отображаются на реляционную модель данных?

Раздел 5. Объектный подход

Тема 5.1. Сущность и основные принципы объектного подхода

Тематический контекст



Краткое содержание

1. Сущность объектно-ориентированного подхода.
2. Базовые принципы объектно-ориентированного подхода: уникальность, классификация, инкапсуляция, наследование, полиморфизм.
3. Плюсы и минусы объектно-ориентированного подхода.

Первые объектно-ориентированные методы появились в начале 80-х гг. XX в., однако основные результаты были получены спустя 10 лет, когда Айвар Джекобсон, Джим Рембо и Грейди Буч представили свои подходы – *Objectory, OMT* (*Object Modeling Technique* – техника объектного моделирования) и *Booch* соответственно. Эти три метода были впоследствии объединены в один, из которого вырос *унифицированный язык моделирования – UML* (*Unified Modeling Language*).

Суть объектного подхода заключается в *объектной декомпозиции*, т.е. система представляется в виде совокупности объектов, которые в процессе взаимодействия обмениваются сообщениями. Дадим определение объекта.

Определение. *Объект* – это самостоятельная, самодостаточная сущность, обладающая состоянием, поведением и семантикой.

Принципы объектно-ориентированного подхода. Объектно-ориентированному подходу присуще большинство принципов структурного подхода, а именно: «разделяй и властвуй», абстрагирование, формализация и непротиворечивость. Кроме того, выделяются следующие специфичные для него принципы:

1. *Уникальность*. Каждый объект уникален. Это самый базовый принцип, о котором многие даже не задумываются. Он означает, что *никакие два даже абсолютно идентичных объекта нельзя считать одним и тем же объектом*. Например, математика не различает два числа «5», однако в рамках объектно-ориентированного подхода два объекта, каждый из которых представляет число «5», будут различными. В терминах языков программирования это означает, что *каждый объект имеет свой адрес в памяти*, и две объектные переменные считаются равными только в том случае, если они указывают на один и тот же адрес.

2. *Классификация*. Все *объекты объединяются в классы по принципу сходства структуры, поведения и семантики*. Определение класса будет дано ниже. Здесь лишь заметим, что объект первичен по отношению к классу. Именно объекты объединяются в классы, формируют их. Ошибочно считать, что если в программе сначала описываются классы, а потом создаются их экземпляры, то класс первичен. Класс появляется только после анализа множества необходимых экземпляров.

3. *Инкапсуляция*. Есть два понятия инкапсуляции. В узком смысле инкапсуляция – это *совместное «хранение» данных и методов их обработки*. Это лишь частное свойство, которое нередко ошибочно считается единственным. В широком смысле инкапсуляция – это *скрытие реализации за интерфейсом*, т.е. объект обладает внутренней, известной лишь ему структурой и интерфейсом, через который другие объекты с ним взаимодействуют. Одним из важнейших следствий инкапсуляции является требование того, чтобы доступ к полям класса осуществлялся

только через его методы. Разделение реализации и интерфейса позволяет менять реализацию, не влияя на способ взаимодействия с объектом.

4. *Наследование*. Наследование означает, что классы могут объединяться в иерархии наследования. Структура, поведение и семантика объектов наследуются вниз по иерархии. Наверху находятся более общие классы, внизу – более специфичные. Для наследования действует «правило подстановки». Суть его сводится к тому, что с объектами классов-потомков (специфичных) можно работать через интерфейс класса-родителя (общего), т.е. вместо объектов родителя можно подставлять объекты любого потомка.

5. *Полиморфизм*. Данный принцип означает возможность доступа к нескольким реализациям через один интерфейс. Способом реализации полиморфизма в объектно-ориентированных языках программирования является механизм переопределения унаследованных методов.

Достоинства объектно-ориентированного подхода. Основным достоинством является устойчивость к изменениям и возможность повторного использования решений. Эти свойства подхода вытекают их того, что вместо жестко связанных между собой функций используются относительно независимые объекты, чья реализация никому, кроме них самих, неизвестна. Соответственно изменения получаются локальными и есть возможность использовать готовые объекты в другом контексте.

Недостатки объектно-ориентированного подхода. По сравнению со структурным подходом, объектно-ориентированный подход проигрывает в производительности. Основными причинами потери производительности являются инкапсуляция и полиморфизм. Действительно, для доступа к полям через методы и для поиска нужного метода по таблице виртуальных методов требуется процессорное время.

Модель процесса. Для объектно-ориентированного подхода характерен *спиральный жизненный цикл* системы. В отличие от структурного подхода, объектный подход не использует иерархических структур, в которых определение родителя не имеет смысла без определения всех его потомков. Вместо этого применяется система объектов, каждый из которых можно анализировать, проектировать, реализовывать и тестировать практиче-

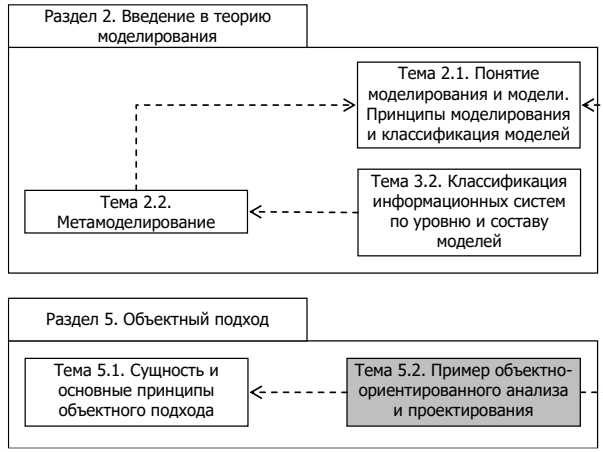
ски независимо от других. Таким образом, появляется возможность формирования итераций, на каждой из которых выполняются все основные этапы разработки для относительно небольшой группы объектов.

Вопросы для самоконтроля

1. В чем заключается сущность объектно-ориентированного подхода?
2. Чем объектно-ориентированный подход принципиально отличается от структурного подхода?
3. Что такое уникальность, классификация, инкапсуляция, наследование и полиморфизм?
4. Почему в рамках объектно-ориентированного подхода возможно использование спирального жизненного цикла?
5. На что тратится дополнительное процессорное время при использовании объектно-ориентированного подхода?
6. В чем преимущества объектно-ориентированного подхода перед структурным подходом?

Тема 5.2. Пример объектно-ориентированного анализа и проектирования

Тематический контекст



Краткое содержание

1. Пример объектно-ориентированного анализа и проектирования для задачи «Игра в кости».
2. Построение модели прецедентов, построение концептуальной модели, проектирование поведения и статической структуры системы.

Основная идея объектно-ориентированного анализа и проектирования состоит в рассмотрении предметной области и логического решения задачи с точки зрения совокупности объектов (понятий или сущностей).

Объектно-ориентированный анализ. В процессе объектно-ориентированного анализа основное внимание уделяется выделению и описанию понятий предметной области, их структуры и связей между ними. Например, в задаче автоматизации библиотеки среди понятий могут присутствовать «библиотека», «издание», «читатель» и др.

Объектно-ориентированное проектирование. В процессе объектно-ориентированного проектирования определяются логические программные объекты, а также их взаимосвязи и взаимодействие между ними. Эти объекты характеризуются атрибутами и операциями. Например, класс «издание» может содержать атрибут «название» и операцию «выдать».

Объектно-ориентированное конструирование. В процессе объектно-ориентированного конструирования определенные логические объекты реализуются средствами объектно-ориентированных языков программирования, таких как C++, Java, C#, VB.NET и т.д.

Рассмотрим пример объектно-ориентированного анализа и проектирования. В данном примере будет использоваться язык UML, который подробно будет рассматриваться дальше. Задача сейчас состоит не в том, чтобы досконально разобраться во всех диаграммах, а в том, чтобы получить представление о процессе объектно-ориентированного анализа и проектирования и месте языка UML в нем.

Задача. Рассмотрим игру в кости. Игрок бросает два кубика. Если сумма очков равна 7, то он считается победителем, в противном случае – проигравшим.

Формулировка требований. Можно выделить несколько видов требований, например, технические, физические, экономические и пр. В данном случае особый интерес представляют прежде всего функциональные требования, т.е. требования к функциональности системы. Для описания и анализа функциональных требований Айвар Джэкобсон предложил использовать *прецеденты* (*UseCase*) – структурированные описания процесса взаимодействия пользователя с системой. На рис. 29 представлена диаграмма прецедентов для данной задачи, на которой изображен один прецедент и один *актор* (*пользователь системы*).

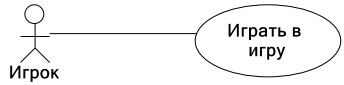


Рис. 29. Диаграмма прецедентов

Ниже приводится текстовое описание прецедента «Играть в игру», в котором дается высокоуровневая характеристика процесса взаимодействия игрока с системой. Диаграмма прецедентов является единственной диаграммой языка UML, в которой большое внимание уделяется текстовому описанию.

Название: Играть в игру

Акторы: Игрок

Описание: Этот прецедент начинается в тот момент, когда игрок берет и бросает кости. Если сумма очков равна 7, то игрок считается победителем, в противном случае – проигравшим.

Определение концептуальной модели. Целью данного этапа является построение модели предметной области. Для этого необходимо произвести ее декомпозицию, которая заключается в идентификации понятий, атрибутов и ассоциаций, имеющих важное значение для решения задачи. Источником понятий, атрибутов и ассоциаций являются описания прецедентов. Результат анализа отображается в концептуальной модели, которая иллюстрируется с помощью диаграмм понятий (рис. 30).

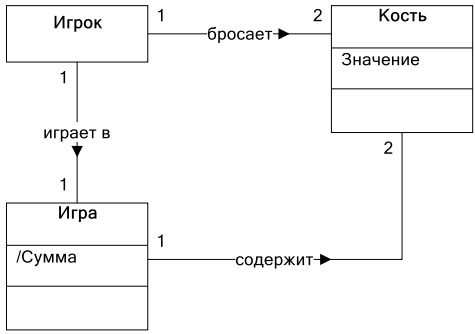


Рис. 30. Концептуальная модель

Важно заметить, что концептуальная модель – это не описание программных компонентов. Она отражает понятия реального мира в терминах предметной области. Как реализовывать эти понятия при помощи классов – это задача, которую предстоит решить на этапе проектирования.

Проектирование взаимодействия. В объектно-ориентированной среде существует некоторый набор объектов, которые взаимодействуют друг с другом посредством передачи сообщений. Целью данного этапа является выделение объектов и определение взаимодействия между ними. Результаты этапа отображаются на диаграммах взаимодействия (рис. 31).

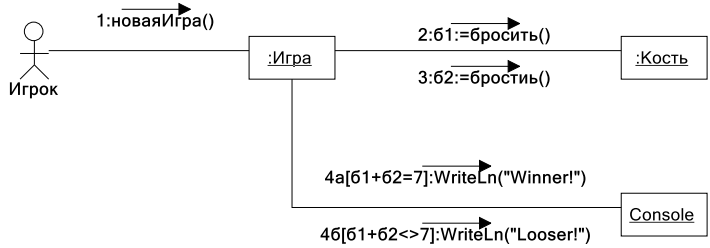


Рис. 31. Диаграмма взаимодействия

Проектирование классов. Целью данного этапа является определение набора и структуры классов, а также отношений между ними. Основным источником информации являются диаграммы взаимодействия. Результат этапа отображается на диаграмме классов (рис. 32).

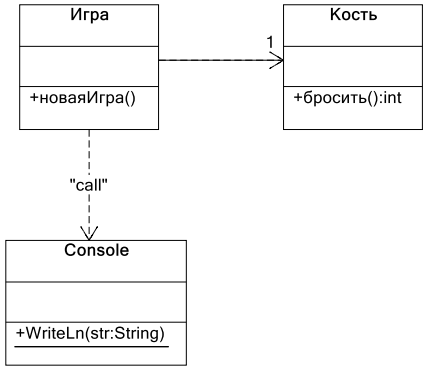


Рис. 32. Диаграмма классов

В отличие от концептуальной модели, эта диаграмма не иллюстрирует понятия реального мира, а описывает программные сущности.

Задания для самостоятельной работы

1. Попробуйте расширить пример. Действуя по аналогии, добавьте в систему таблицу рекордов.
2. Попробуйте провести объектно-ориентированный анализ и проектирование для задачи «Игра в 21». Задача заключается в следующем. Есть два игрока и колода карт. Игрокам раздается по две карты. Каждый игрок может взять себе еще несколько карт (сначала один, потом второй). После этого игроки открывают свои карты и сравнивают набранные очки (стоимость картинок – 10, туз – 1 или 11 по выбору игрока). Задача набрать 21 очко (два туза считаются самой старшей комбинацией). Побеждает тот игрок, у которого больше очков, но в пределах 21. Игрок, набравший больше 21, считается проигравшим. Если игроки набрали равное количество очков или оба превысили 21, то считается, что победителя нет.

Раздел 6. Унифицированный язык моделирования UML

Тема 6.1. Обзор языка UML

Тематический контекст

Краткое содержание

1. История UML.
2. Определение языка UML.
3. Архитектура, управляемая моделью (MDA).
4. Стандарт MOF.

На ранних этапах развития объектно-ориентированного подхода сложилась такая ситуация, которую назвали «война методов». Было предложено свыше 50 методов, причем каждый объявлялся самым лучшим и удобным. Это привело к тому, что руководители проектов выбирали метод практически случайно.

Людям, которые переходили из организации в организацию, приходилось проходить повторное обучение. Более того, после неудачного опыта с одним методом руководители пытались перейти на другой, что приводило к еще более плачевным последствиям. Это был кризис объектно-ориентированного подхода.

Первые шаги в направлении стандартизации были предприняты в 1994 г., когда Джим Рэмбо (James Rumbaugh) перешел из General Electrics в Rational, где работал Грэйди Буч (Grady Booch), и они решили объединить свои методы (OMT и BOOCH соответственно). В 1995 г. Rational покупает Objectory AB вместе с Айваром Джэкобсоном (Ivar Jacobson) – автором методологии с одноименным названием Objectory. Совместными усилиями они пытаются создать унифицированный язык моделирования, который положил бы конец войне методов, – UML (Unified Modeling Language). В 1997 г. OMG (Object Management Group) утверждает версию UML 1.1 и включает ее в набор своих стандартов. OMG является международной организацией, в которую входят такие гиганты, как Microsoft, IBM, Oracle, HP и др.

На сегодня UML является, пожалуй, самым известным языком моделирования, который поддерживается огромным количеством инструментальных средств и широко используется на практике. Последней версией языка на момент написания данного пособия является версия UML 2.0.

Определение. *UML* – это *графический язык моделирования*, представляющий собой систему обозначений, базирующуюся на диаграммах, и предназначенный для визуализации, спецификации, конструирования и документирования систем, в которых большая роль принадлежит программному обеспечению.

Разберем это определение подробно.

UML – это язык. Любой язык состоит из словаря и правил, позволяющих комбинировать входящие в него слова и получать осмысленные конструкции. Словарь и правила языка UML объясняют, как создавать и читать хорошо определенные модели, но ничего не сообщают о том, какие модели и в каких случаях нужно создавать. Это задача всего процесса разработки программного обеспечения. В данном пособии рассматривается некоторый обобщенный процесс, который позволяет продемонстрировать большинство диаграмм языка UML.

UML – это язык визуализации. Безусловно, наиболее точным описанием системы является ее исходный код, однако с усложнением системы уменьшается количество информации о системе, которую можно извлечь из кода за фиксированное время. Более того, ухудшается и качество получаемой информации, т.к. читая код, человек акцентирует внимание на частностях, упуская при этом общие вещи. Известно, что графическое представление информации является наиболее наглядным. UML позволяет визуализировать как саму систему, так и артефакты, прямо не относящиеся к программированию, например модель предметной области.

UML – это язык спецификации. Под специфицированием понимается построение точных, недвусмысленных и полных моделей. UML позволяет специфицировать все существенные решения, касающиеся анализа, проектирования и реализации, которые должны приниматься в процессе разработки и развертывания системы.

UML – это язык конструирования. UML не является языком визуального программирования, но модели, созданные с его помощью, могут быть непосредственно переведены на различные языки программирования. Это позволяют сделать, например, такие инструментальные средства, как Visio, Telelogic Tau, Rational Rose, CASEBERRY и пр. Таким образом, UML *позволяет решить задачу прямого проектирования*, т.е. обеспечивает *возможность генерации кода по модели*. Возможна также реализация и *обратного проектирования*, т.е. *построения модели по исходному коду*. Наиболее продвинутые CASE-средства используют комбинированный подход, т.е. поддерживается соответствие кода и модели.

UML – это язык документирования. Компания, выпускающая ПО, помимо исполняемого кода производит и другие артефакты, например, требования к системе, описание архитектуры, проект, исходный код, тесты и т.д. UML позволяет решить проблему документирования системной архитектуры и всех ее деталей, предлагает язык для формулирования требований к системе и даже позволяет организовать качественное тестирование. Кроме того, сама технология разработки, используемая компанией, может быть описана на UML.

Следует заметить, что сфера применения UML не ограничивается моделированием программного обеспечения, хотя, конечно, он наиболее приспособлен именно для этого. Объектная парадигма является достаточно универсальной, и везде, где она применима, применим и UML.

UML входит в целое семейство *стандартов OMG*, которые формируют *MDA – Model Driven Architecture* (*архитектура, управляемая моделью*). Основная идея MDA – использование моделей (в отличие от исходного кода) в качестве основного артефакта при разработке программных систем. Один из стандартов MDA, о котором здесь имеет смысл поговорить, – это *Meta-Object Facility* (*MOF*).

Определение. *MOF* – это стандарт OMG, определяющий язык и платформу для спецификации, построения и управления технологически нейтральными метамоделями.

На рис. 33 представлена иерархия метамоделей, которую использует OMG. На самом верхнем уровне находится модель MOF – модель языка, на котором определяются такие языки моделирования, как UML или IDL (*Interface Definition Language – язык определения интерфейсов*). Большинство спецификаций OMG (в том числе и MOF) определены в терминах MOF.

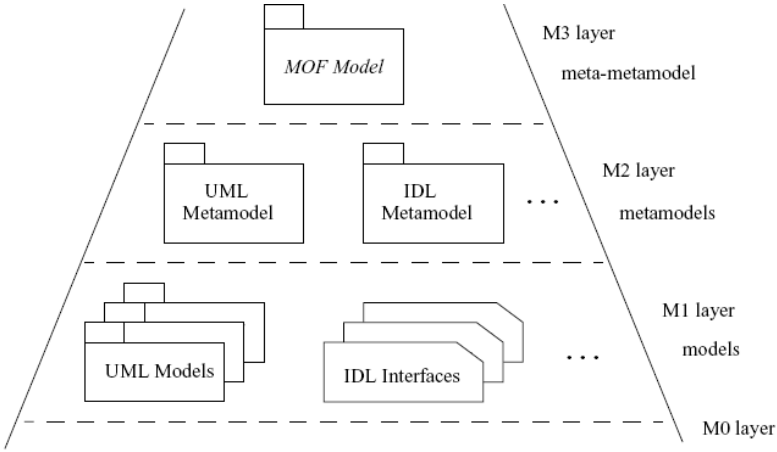


Рис. 33. Иерархия метамоделей

Среди стандартов OMG есть *стандарт XMI* (*XML Metadata Interchange – обмен метаданными в формате XML*), который позволяет любое MOF-описание преобразовать в XML. Соответственно любое CASE-средство, которое реализует MOF и поддерживает XMI, сможет «понять» любой язык моделирования, описанный в терминах MOF. Достаточно лишь загрузить соответствующий XMI-документ.

На втором уровне иерархии находятся языки моделирования, входящие в состав стандартов OMG, такие как UML и IDL. Первый уровень содержит создаваемые пользователями модели, которые описывают конкретные интерфейсы, предметные области и пр. И, наконец, на уровне `M0` находятся экземпляры пользовательских моделей, например данные, описывающие состояние предметной области.

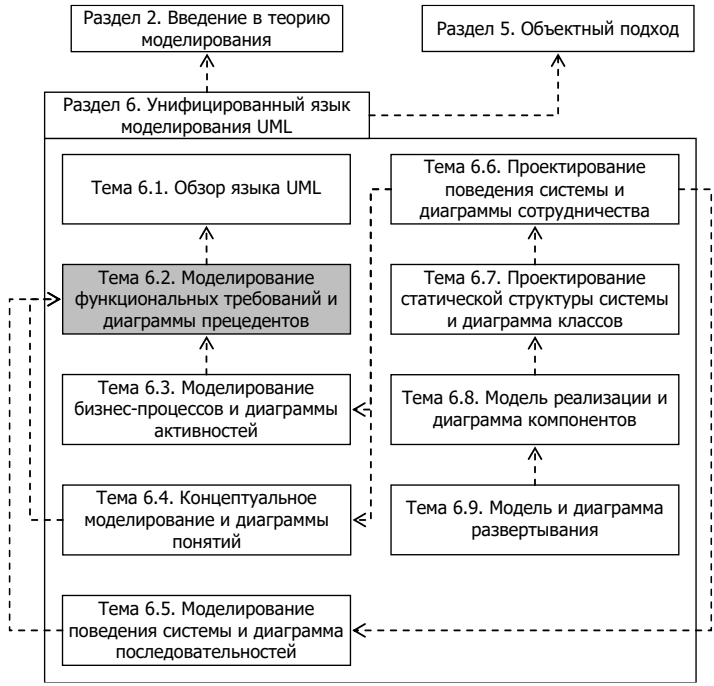
Синтаксически MOF представляет собой упрощенный вариант диаграмм классов языка UML, которые рассматриваются ниже. Если уметь читать диаграммы классов, то разобраться в любой MOF-модели не составит труда.

Вопросы для самоконтроля

1. Что такое UML? Прокомментируйте определение.
2. В чем заключается основная идея MDA?
3. Какова связь между MOF и UML?

Тема 6.2. Моделирование функциональных требований и диаграммы прецедентов

Тематический контекст



Краткое содержание

1. Понятие прецедента, свойства прецедентов.
2. Задачи пользователя и системные взаимодействия.
3. Основные элементы диаграммы прецедентов: акторы, прецеденты, отношения.
4. Порядок построения модели.

Диаграмма прецедентов (UseCase Diagram, UCD) пришла в UML из методологии Objectory, которую разработал Айвар Джэкобсон (Ivar Jacobson). Это единственная диаграмма UML, в которой большое внимание уделяется текстовым описаниям, а графическая нотация носит иллюстративный характер. В некоторых случаях допускается опускать текстовые описания, однако это не рекомендуется.

Основная идея прецедентов заключается в выявлении и описании ситуаций, когда пользователи взаимодействуют с системой. Основное внимание здесь обращается на то, что надо пользователю, и на то, как он это получает при помощи системы. Таким образом, выделив и описав все варианты использования (use case) системы, мы получаем описание функциональных требований. Построение модели прецедентов является первым шагом работы над проектом.

Определение. *Прецедент* – описание множества содержательно-близких сценариев взаимодействия акторов (внешних агентов) с системой, которое осуществляется с целью получения акторами некоторого полезного результата при помощи системы.

Здесь важно обратить внимание на то, что прецедент может охватывать сразу несколько сценариев, следуя которым актор может достичь своей цели. Например, рассмотрим прецедент «Отправка e-mail» из некоторой почтовой системы. Основные шаги отправки письма хорошо известны: нужно написать текст письма, указать тему и адресата, и, наконец, отправить его. Однако в каждом конкретном случае возможны варианты. Например, адрес получателя можно написать самостоятельно или выбрать из адресной книги, отправку можно отложить, сохранив письмо как черновик, и т.д. Это и есть сценарии. Как видно из этого примера, сценарии обычно похожи друг на друга и различаются только в деталях. Отношение между прецедентом и сценариями такое же, как и отношение между классом и его экземплярами. Т.е. сценарий является экземпляром прецедента.

Рассмотрим *основные свойства прецедентов*:

1. Прецедент охватывает некоторую очевидную для пользователя функцию, т.е. пользователь должен представлять себе суть процесса и, самое главное, знать, что он может получить в конце.
2. Прецедент может быть как небольшим, так и достаточно крупным (например, сделать список, сделать предметный указатель).

Прецедент решает некоторую дискретную и важную для пользователя задачу. За решение таких задач обычно платят деньги. Таким образом, «Нажать на кнопку <Печать>» – это

плохой прецедент, а «Сформировать и напечатать отчет» – вполне достойный.

Источником прецедентов являются требования, которые пользователь предъявляет к системе, т.е. это описание тех функций, которые он хотел бы видеть в системе.

В процессе взаимодействия пользователя и системы можно выделить два аспекта: *задачи пользователя* и *системные взаимодействия*.

Определение. *Задачи пользователя* – это задачи из предметной области. Они важны для пользователя, от их выполнения зависит его душевное и материальное благосостояние.

Определение. *Системные взаимодействия* – запросы к функциональности системы. С помощью системных взаимодействий пользователь решает свои задачи.

Прецеденты соответствуют задачам пользователя и описываются в терминах системных взаимодействий.

Рассмотрим основные элементы диаграммы прецедентов. К ним относятся следующие элементы: *акторы*, *прецеденты* и *отношения*. Рассмотрим их подробно.

Определение. *Актор* – это внешний по отношению к системе объект. Актором может быть человек или другая система.

Важно понимать, что актор – это не конкретный человек, а роль, которую он играет по отношению к системе. Одну и ту же роль могут исполнять несколько человек, и, наоборот, один человек может исполнять несколько ролей. Например, в организации, занимающейся поставкой компьютерного оборудования, может быть несколько менеджеров, но роль «Менеджер» будет одна, т.к. все они решают одни и те же задачи. С другой стороны, один из менеджеров может быть по совместительству еще и кассиром. В данном случае один человек выполняет две роли. Способ изображения акторов на диаграммах представлен на рис. 34.

Рис. 34. Графическая нотация для изображения актора

Процесс выделения акторов тесно связан с определением границ системы. Определив границы системы, можно идентифицировать ее внешние и внутренние свойства. Внешняя среда представляется только акторами. Например, рассмотрим магазин и две роли – покупатель и кассир. Если в качестве границ системы выбрать весь магазин в целом, то актором будет только покупатель, т.к. кассир в данном случае является внутренним ресурсом системы, выполняющим определенные функции. Если же в качестве границ системы выбрать программно-аппаратную часть торговой системы, то актором будет кассир и, возможно, покупатель (если покупатель взаимодействует с системой напрямую, например, для ввода pin-кода кредитной карты).

Выбор границ системы определяется целями моделирования. Если планируется разработка приложения или устройства, то имеет смысл определить границы системы в соответствии с программно-аппаратным обеспечением. Если же речь идет об экономической реорганизации предприятия, то в качестве границ системы следует выбрать все предприятие в целом.

Прецеденты. В простейшем случае прецедент представляет собой один или несколько абзацев текста, описывающих взаимодействие акторов и системы. Однако гораздо удобнее и эффективнее использовать шаблоны, состоящие из набора полей. UML не определяет стандартного способа описания прецедентов, и в каждом случае выбирается наиболее удобный формат. Как минимум обычно выделяют следующие поля: *Название*, *Акторы* и *Описание*.

Прецеденты можно разделить на высокоуровневые и развернутые.

Определение. *Высокоуровневый прецедент* – это прецедент, который кратко описывает основные моменты взаимодействия акторов с системой.

Высокоуровневые прецеденты удобно использовать на начальном этапе формулирования требований к системе. Они позволяют быстро осознать степень сложности и функциональность системы. Прецеденты высокого уровня имеют слабое отношение к конкретным проектным решениям, и, следовательно, их можно использовать в различных проектах. Пример высокоуровневого прецедента представлен ниже.

Название: Продажа товара

Акторы: Кассир

Описание: Покупатель подходит к кассе с товарами, которые он желает приобрести. Кассир регистрирует их и сообщает общую сумму покупателю. Покупатель оплачивает и покидает магазин с приобретенными товарами.

Определение. *Развернутый прецедент* – это прецедент, который детально описывает процесс взаимодействия акторов с системой.

При описании развернутых прецедентов обычно выделяют *основной поток*, *подпотоки* и *альтернативные потоки*.

Определение. *Основной поток* – это описание типичного, наиболее вероятного хода событий.

Определение. *Подпоток* – это логически законченная последовательность основного потока, которую имеет смысл описать отдельно (например, используется несколько раз).

Определение. *Альтернативный поток* – это отклонения от основного потока, связанные с возникновением различного рода исключений и нестандартных ситуаций. В альтернативном потоке могут пропускаться или изменяться шаги основного потока, а также добавляться новые.

При описании развернутых прецедентов обычно используют таблицы с двумя столбцами: *действия акторов* и *отклик системы*. Если поток простой, то бывает достаточно текстового описания. Пример развернутого прецедента приведен ниже.

Название: Продажа товара

Акторы: Кассир, Покупатель

Краткое описание: Покупатель подходит к кассе с товарами, которые он желает приобрести. Кассир регистрирует их и сообщает общую сумму покупателю. Покупатель оплачивает и покидает магазин с приобретенными товарами.

Триггер: Покупатель подходит к кассе с товарами, которые он желает приобрести.

Основной поток:

Действия акторов	Отклик системы
1. Кассир сканирует штрих-код каждого товара (E_1) и при необходимости указывает количество.	2. Подтверждает регистрацию новой товарной позиции и сообщает название и цену товара.
3. После завершения ввода информации кассир информирует об этом систему.	4. Сообщает кассиру общую сумму продажи.
5. Кассир сообщает сумму покупателю (E_2). При оплате наличными покупатель передает деньги кассиру. Кассир информирует об оплате систему и при необходимости выдает сдачу. При оплате карточкой выполняется подпоток S_1.	6. Генерирует подробный чек.

Подпотоки:

S_1: Оплата кредитной карточкой. Покупатель вставляет карточку в терминал, вводит pin-код и подтверждает снятие со счета требуемой суммы. Система сообщает о проведении оплаты. Прецедент продолжается.

Альтернативные потоки:

E_1: Ручной ввод кода. При невозможности успешного сканирования штрих-кода кассир вводит его вручную. Прецедент продолжается.
E_2: Отмена продажи. Если покупатель не может оплатить товар, то кассир отменяет продажу. Прецедент завершается.

В данном прецеденте были использованы некоторые дополнительные поля. Поле «Краткое описание» содержит общую характеристику прецедента и, по сути, повторяет текст соответствующего высокоуровневого прецедента. Поле «Триггер» определяет событие, которое инициирует начало прецедента. При описании основного потока места возможного возникновения исключительных ситуаций помечались именем соответствующего альтернативного в скобочках. Диаграмма, иллюстрирующая данный прецедент, представлена на рис. 35.

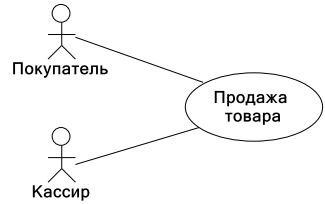


Рис. 35. Продажа товара

Прецеденты полезно разделять по степени важности на следующие категории: *главные*, *второстепенные* и *дополнительные*.

Определение. *Главные прецеденты* – это прецеденты, без реализации которых невозможно функционирование системы.

Главные прецеденты обычно охватывают наиболее часто используемые функции системы и должны реализовываться в первую очередь. Примерами главных прецедентов в системе розничной торговли могут являться «Продажа товара» и «Приход товара».

Определение. *Второстепенные прецеденты* – это прецеденты, без реализации которых возможно частичное функционирование системы.

Второстепенные прецеденты обычно охватывают важные, но относительно редко используемые функции системы. К ним обычно относятся всевозможные отчеты, инвентаризация и пр. Такие прецеденты реализуются во вторую очередь, а иногда даже после начала опытной эксплуатации системы.

Определение. *Дополнительные прецеденты* – прецеденты, которые не влияют на основное функционирование системы.

Дополнительные прецеденты обычно охватывают желательные, но не обязательные функции системы. Назначением подобных прецедентов является упрощение, ускорение или какое-либо другое улучшение некоторых аспектов работы с системой. В определенных ситуациях подобные прецеденты можно вообще не реализовывать.

Еще одним способом классификации прецедентов является их разделение на *идеальные* и *реальные*.

Определение. *Идеальные прецеденты* – это развернутые прецеденты, отражающие процесс взаимодействия акторов с системой без указания конкретных технологий ввода/вывода информации.

При описании идеальных прецедентов проектные решения, особенно связанные с интерфейсом пользователя, опускаются. Соответственно идеальные прецеденты не зависят от реализации и могут быть использованы в различных проектах. Выдержка из идеального прецедента для системы «Банкомат» приведена ниже.

```
… Клиент идентифицирует себя. Система предлагает выбрать
требуемую функцию. Клиент выбирает «Запрос остатка». …
```

Определение. *Реальные прецеденты* – это развернутые прецеденты, отражающие процесс взаимодействия акторов с системой в терминах конкретных проектных решений и на основе конкретных технологий ввода-вывода.

Крайним случаем реальных прецедентов является подробное описание пользовательского интерфейса с указанием названий окон, кнопок, полей и пр. элементов управления, с которыми работает пользователь. Такие прецеденты особенно полезны при написании документации пользователя. Выдержка из реального прецедента для системы «Банкомат» приведена ниже.

```
… Клиент вставляет свою карту в приемник. Система отобра-
жает на экране приглашение к вводу pin-кода. Клиент
вводит pin-код при помощи цифровой клавиатуры. Для каж-
дой введенной цифры система отображает на экране символ
«X». Пользователь нажимает кнопку «Ввод». …
```

Рассмотрим подробнее шаблон описания прецедентов. В вышеприведенных примерах использовались такие поля, как *Название*, *Акторы*, *Краткое описание*, *Триггер*, *Описание*, *Основной поток*, *Подпотоки* и *Дополнительные потоки*. Кроме этих полей на практике бывает удобно использовать и некоторые другие, такие как, например: *Автор*, *Дата создания*, *Дата последнего изменения*, *Назначение*, *Предусловие*, *Результат успешного завершения*, *Результат неудачного завершения*. Назначение всех этих полей достаточно очевидно. Дополни-

тельных комментариев требует разве что предусловие. Предусловие – это условие, которое должно быть выполнено перед началом выполнения прецедента; прецедент не может начаться, пока не выполнятся все его предусловия. Например, предусловием в прецеденте «Оформление расходной накладной» для некоторого торгового предприятия может являться следующее условие: клиент должен иметь при себе оплаченный счет.

Отношения. На диаграммах прецедентов используются четыре вида отношений: *ассоциация*, *обобщение* и *зависимость*.

Определение. *Ассоциация* – это устойчивое структурное отношение между сущностями.

Отношения ассоциации используются между акторами и прецедентами. Ассоциация между актором и прецедентом означает, что данный актор участвует в данном прецеденте. Пример ассоциации см. на рис. 36.

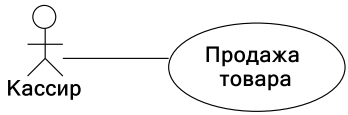


Рис. 36. Пример ассоциации

Определение. *Обобщение* – это отношение между элементом-родителем и элементом-потомком, при котором потомок является частным случаем родителя и наследует его структуру, поведение и семантику.

Отношение обобщения используется либо между акторами, либо между прецедентами. Отношение обобщения между акторами означает, что актор-потомок может участвовать во всех прецедентах, в которых участвует актор-родитель (допускается несколько родителей), плюс в некоторых еще. Например, администратору системы могут быть доступны все функции обычных пользователей плюс некоторые администраторские функции, которые доступны только ему. Пример использования отношения обобщения между акторами приведен на рис. 37.

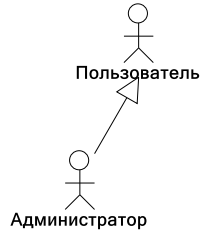


Рис. 37. Пример отношения обобщения между акторами

Отношение обобщения между прецедентами означает, что дочерние прецеденты используют некоторую общую схему взаимодействия акторов с системой, и различаются только в конкретных шагах. Все акторы, связанные при помощи ассоциаций с родительским прецедентом, участвуют во всех дочерних прецедентах. Родительский прецедент иногда делают абстрактным (т.е. не содержащим каких-либо конкретных шагов), что иллюстрируется при помощи курсива. Пример отношения обобщения между прецедентами приведен на рис. 38.

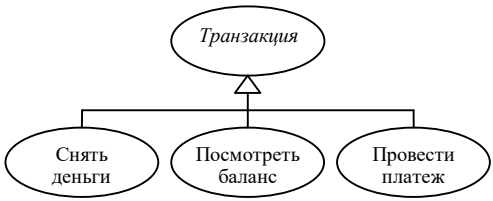


Рис. 38. Пример отношения обобщения между прецедентами

Определение. *Зависимость* – это семантическое отношение между двумя сущностями, которое означает, что одна сущность (зависимая) некоторым образом зависит от другой сущности (независимой) и изменения структуры, поведения или семантики независимой сущности могут повлечь за собой изменения в структуре, поведении или семантике зависимой сущности.

На диаграммах прецедентов используются два вида отношений зависимости – это *зависимость включения* и *зависимость расширения*.

Определение. *Зависимость включения* – эта такая зависимость между прецедентами, в которой один прецедент (базовый) явно включается в другой прецедент.

Зависимость включения применяется, когда одна и та же последовательность действий используется в нескольких прецедентах. Кроме того, если один из подпотоков некоторого прецедента получается достаточно большим, то его имеет смысл вынести в отдельный прецедент и использовать зависимость включения. Включаемый прецедент никогда не используется самостоятельно. Для обозначения места включения в тексте описания базового прецедента можно использовать другой шрифт или оператор `include`, например `include(«Оплата КК»)`. Пример использования зависимости включения приведен на рис. 39.

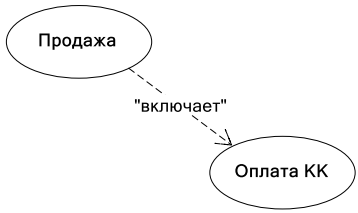


Рис. 39. Пример использования зависимости включения

Определение. *Зависимость включения* – эта такая зависимость между прецедентами, в которой один прецедент (расширяющий) неявно включается в другой прецедент (расширяемый).

Зависимость расширения применяют для моделирования таких частей прецедента, которые пользователь воспринимает как необязательное поведение системы. Данная зависимость также используется для моделирования подпотоков, которые выполняются только при определенных условиях или для больших альтернативных потоков.

Для использования зависимости расширения в расширяемом прецеденте необходимо определить точки расширения – те места, куда при определенных условиях включается расширяющий прецедент. В тексте расширяемого прецедента точки расширения можно выделять при помощи круглых скобок. Для

расширяющего прецедента необходимо указать соответствующую точку расширения расширяемого прецедента.

Рассмотрим пример. На рис. 40. изображены два прецедента – «Разместить заказ» и «Разместить срочный заказ». При оформлении заказа устанавливается его приоритет. В случае если устанавливается строчный приоритет, то необходимо выполнить некоторые дополнительные действия, которые описываются в прецеденте «Разместить срочный заказ».

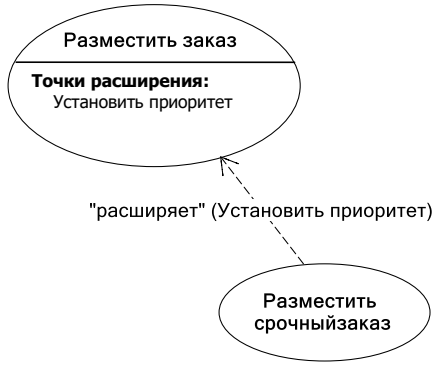


Рис. 40. Пример использования зависимости расширения

Порядок построения модели. Для построения модели прецедентов рекомендуется придерживаться следующей последовательности шагов:

1. Определить границы системы, идентифицировать акторов и прецеденты.
2. Записать все прецеденты в высокоуровневой форме и разбить их по категориям: главные, второстепенные, дополнительные.
3. Выделить наиболее важные и рискованные прецеденты и записать их в развернутой идеальной форме.
4. Определить отношения между прецедентами и проиллюстрировать их на диаграмме.

Реальные прецеденты строятся по мере необходимости на этапе проектирования, либо по требованию заказчика.

Бонусы модели прецедентов. Как говорилось выше, основным назначением модели прецедентов является фиксация функциональных требований к системе. Однако это не единственный полезный эффект, который можно из нее извлечь. Рассмотрим некоторые дополнительные бонусы модели прецедентов.

Планирование итераций. Каждый прецедент описывает решение некоторой законченной задачи пользователя и является естественным кандидатом на единицу планирования разработки.

Определение профилей пользователей. Каждый прецедент указывает акторов, которые в нем участвуют. Соответственно для каждого пользователя можно определить набор функций системы, которые ему необходимы. На основе этой информации можно формировать профили пользователей и организовывать систему безопасности.

Оптимизация системы. Каждый прецедент выделяет основную, наиболее типичную и соответственно наиболее часто используемую цепочку действий пользователей. Соответственно эту цепочку действий можно оптимизировать как внутри системы, так и на интерфейсе (сократить количество кликов, переходов между окнами и пунктами меню).

Организация тестирования. Каждый прецедент описывает ожидаемый отклик системы на действия пользователя. Эту информацию можно использовать для организации тестирования системы.

Формирование документации пользователя. Каждый прецедент описывает действия, которые должен совершить пользователь для того, чтобы решить свою задачу. Эта информация является источником для формирования документации пользователя. При определенной инструментальной поддержке ее можно сформировать в автоматическом или полуавтоматическом режиме.

Типичные ошибки

Типичной ошибкой является выделение бессодержательных прецедентов (в крайнем случае – одного единственного, например «Работа системы»). В любой нетривиальной системе должно быть несколько прецедентов, каждому из которых должна соот-

ветствовать вполне конкретная задача пользователя.

Другой типичной ошибкой является включить в обязанности системы выполнение фантастических действий. В качестве примера можно взять следующую формулировку: «читатель допускается в читальный зал без верхней одежды и посторонних книг». Понятно, что если речь идет об автоматизации библиотеки, то на текущем уровне развития техники данная формулировка смысла не имеет. Однако ее можно использовать, если в качестве границ системы была выбрана вся библиотека в целом и, например, стоит задача ее реорганизации.

Вопросы и задания для самоконтроля

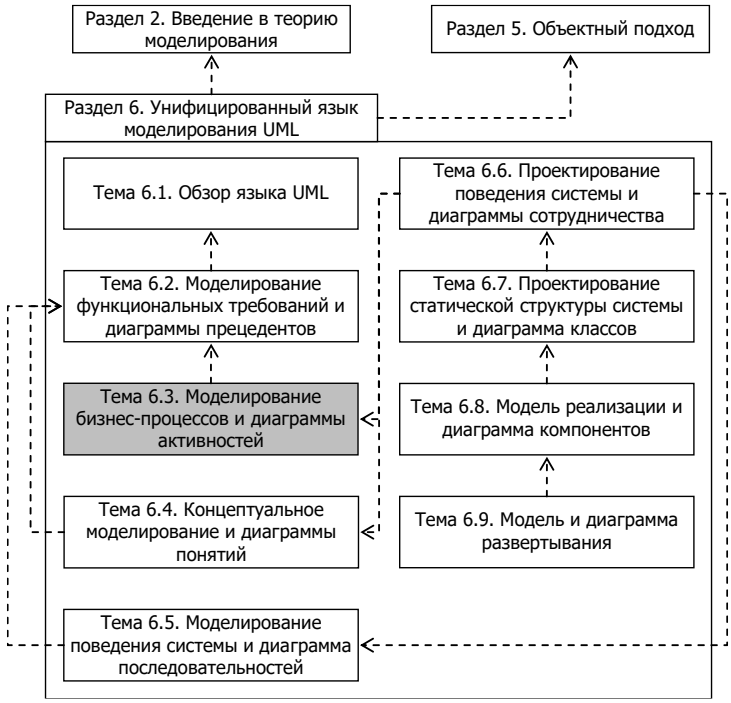
1. Что такое прецедент?
2. Какова связь между задачами пользователя и системными взаимодействиями?
3. Что такое актор? Как он связан с пользователями системы?
4. Как выбор границ системы влияет на построение модели прецедентов?
5. Назовите три способа классификации прецедентов.
6. Какие отношения можно использовать в модели прецедентов?
7. В чем разница между зависимостью включения и зависимостью расширения?
8. Как прецеденты могут помочь при оптимизации и тестировании системы?

Задания для самостоятельной работы

1. После изучения темы 5.4 постройте концептуальную модель диаграммы прецедентов.
2. Постройте модель прецедентов для учебного задания.

Тема 6.3. Моделирование бизнес-процессов и диаграммы активностей

Тематический контекст



Краткое содержание

1. Понятие бизнес-процесса.
2. Элементы диаграммы активностей: «плавательные дорожки», активности, синхронизаторы/разветвители.

Определение. *Бизнес-процесс* – это устойчивый процесс (последовательность работ), соотнесенный с отдельным видом производственно-хозяйственной деятельности компании и обычно ориентированный на создание новой стоимости; иерархия взаимосвязанных функциональных действий, реализующих одну (или несколько) из целей организации.

Определение бизнес-процесса в чем-то похоже на определение прецедента, однако это разные понятия и необходимо чет-

ко понимать разницу между ними. Прецедент описывает решение некоторой частной задачи пользователя, в то время как бизнес-процесс ориентирован на достижение глобальных целей компании. Если речь не идет о комплексной автоматизации всего предприятия, то зачастую бизнес-процессы выходят за рамки создаваемой системы, в то время как прецедент жестко связан с ней. Обычно бизнес процесс включает несколько прецедентов.

Из вышесказанного следует, что описание бизнес-процесса позволяет понять контекст возникновения прецедентов. Для описания бизнес-процессов удобно использовать диаграмму активностей (Activity Diagram, AD) языка UML. Пример диаграммы активностей для процесса продажи товара «под заказ» представлен рис. 41.

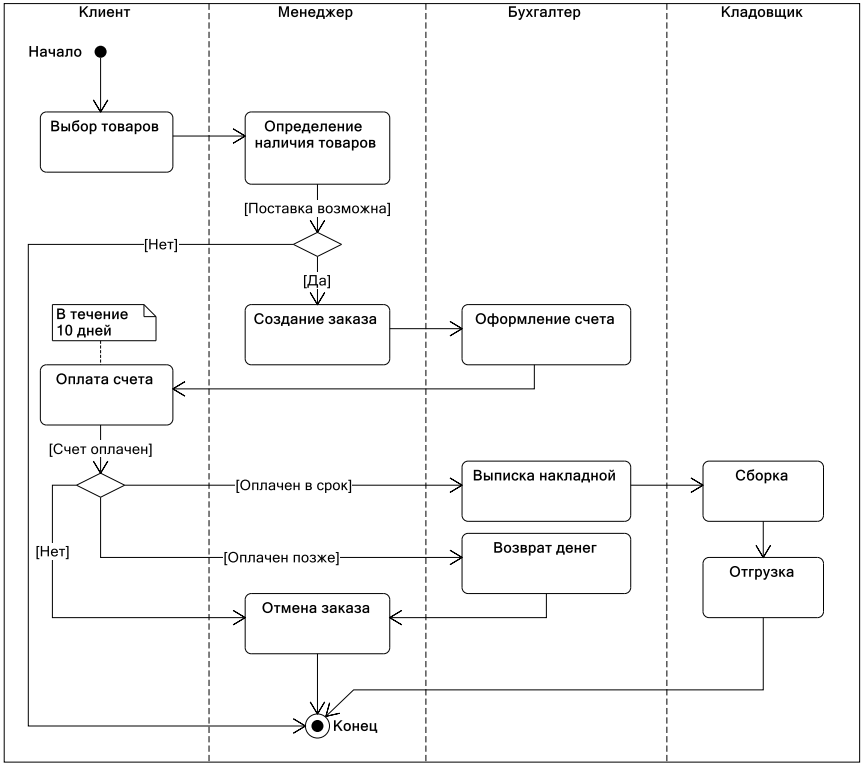


Рис. 41. Пример описания бизнес-процесса

Элементы диаграммы активностей

Разберем основные элементы диаграммы активностей, используемые при описании бизнес-процессов. Прежде всего, любая диаграмма активностей начинается с начального состояния, обозначаемого жирной точкой, и заканчивается конечным состоянием, обозначаемым жирной точкой в окружности (рис. 41). Начальное состояние определяет точку входа в процесс, а конечное – точку выхода. На диаграмме допускается только одно начальное состояние; конечных состояний может быть несколько, но лучше, чтобы оно также было единственным. Если не получается построить диаграмму активностей с единственным конечным состоянием без пересечения линий, то лучше разбить ее на несколько диаграмм (например, одну крупную и несколько более детальных).

У любого бизнес-процесса всегда есть участники. Для каждого участника на диаграмме активностей создается так называемая плавательная дорожка – прямоугольная область диаграммы, в которой располагаются активности данного участника. Между собой плавательные дорожки разделяются пунктирными линиями (рис. 41).

Основным элементом диаграммы активностей является *активность*, которую иногда еще называют деятельностью. Дадим определение.

Определение. *Активность* – это состояние объекта (группы объектов), в котором он (они) выполняет некоторые действия.

Активность на диаграмме изображается в виде прямоугольника со скругленными углами и именем активности внутри. Переходы между активностями показываются при помощи стрелок (рис. 41).

При описании бизнес-процессов нередко встречаются ситуации, когда выбор активности, в которую следует перейти на следующем шаге, зависит от некоторого условия. Для отражения подобных особенностей процесса используется *условный переход*, который изображается в виде ромба (рис. 42). Альтернатив может быть 2 или больше, и они должны полностью покрывать все возможные варианты разрешения условия.

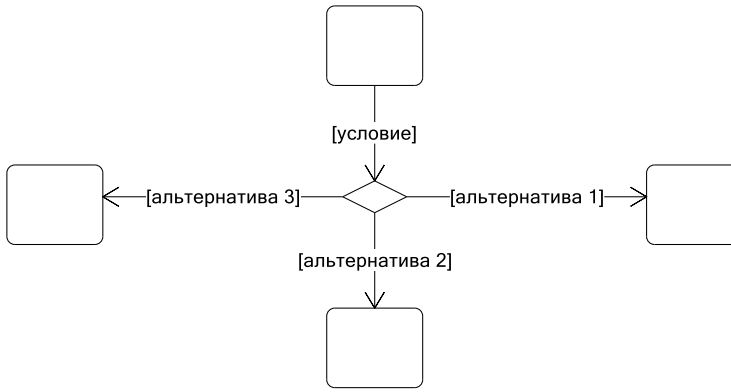


Рис. 42. Условный переход

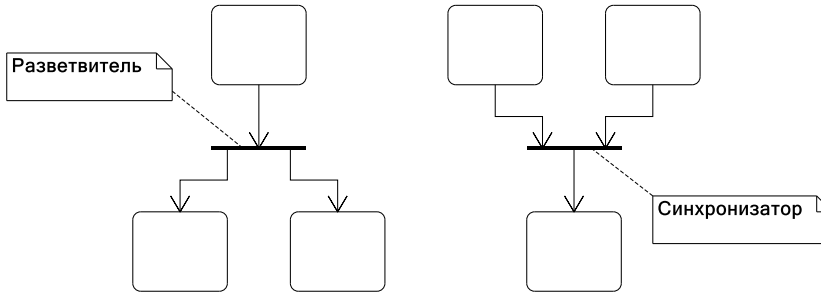


Рис. 43. Сложные переходы

Кроме условных переходов существуют еще *сложные переходы*, которые разделяются на *разветвители* и *синхронизаторы* (рис. 43). Разветвители используются для разделения (распараллеливания) потока управления (на рисунке слева), а синхронизаторы – наоборот, для слияния нескольких потоков (на рисунке справа).

Наличие на диаграмме *разветвителя* означает, что при его срабатывании сразу два объекта переходят в активное состояние. Рассмотрим, например, предметную область «Гонки Формула-1» и процесс «Обслуживание машины на pit-стопе». В данном случае при заезде машины на pit-стоп в активное состояние переходят сразу две команды техников, одна из которых меняет резину, а вторая – заливает бензин. Для моделирования такой ситуации необходимо использовать разветвитель (рис. 44).

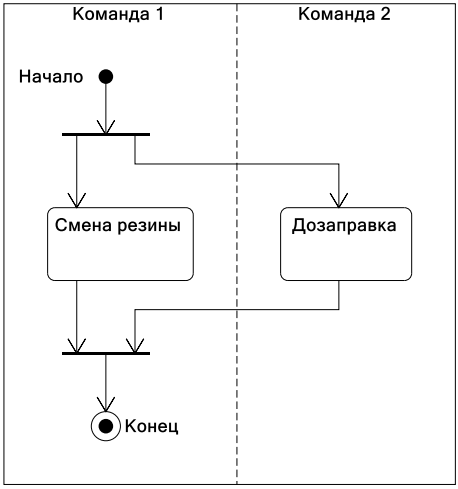


Рис. 44. Пример использования сложных переходов

Наличие на диаграмме *синхронизатора* означает, что данный переход срабатывает только в том случае, если завершились все его входящие активности. Например, очевидно, что машина может уехать с pit-стопа только в том случае, если обе команды техников закончили свою работу. Для моделирования такой ситуации необходимо использовать синхронизатор (рис. 44).

Необходимо четко понимать разницу между синхронизатором и состоянием с несколькими входами. Например, бизнес-процесс, изображенный на рис. 41, может прийти в конечное состояние либо в случае невозможности поставки товара, либо после отмены заказа, либо после отгрузки товара. Напротив, процесс, изображенный на рис. 44, может прийти в конечное состояние только после того, как завершится и смена резины, и дозаправка.

Диаграммы активностей обычно строят одновременно с диаграммами прецедентов. Эти два вида диаграмм дополняют друг друга. Диаграммы активностей позволяют понять бизнес-процесс заказчика, а прецеденты позволяют определить место разрабатываемой системы в этом бизнес-процессе.

Вопросы для самоконтроля

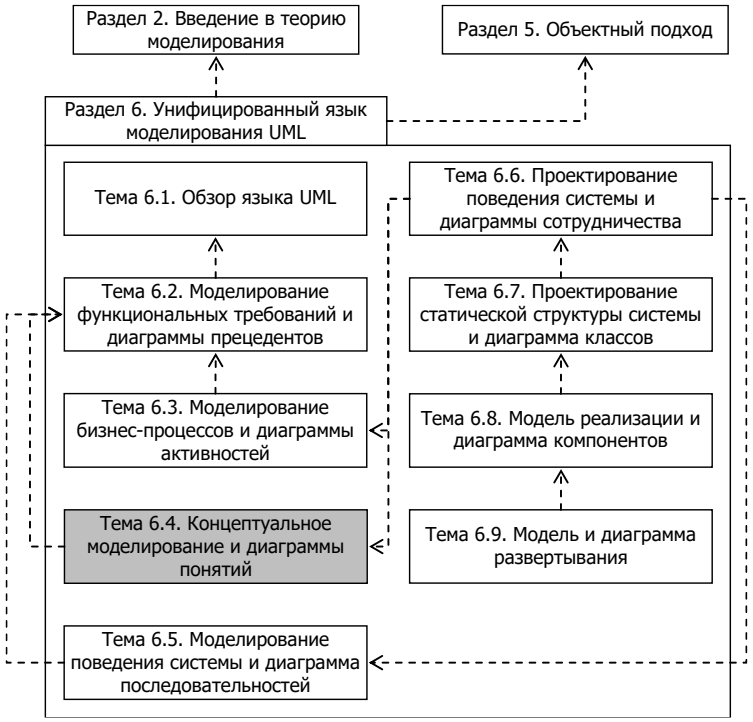
1. Что такое активность?
2. Для чего нужны «плавательные дорожки»?
3. В чем разница между моделью прецедентов и моделью бизнес-процессов?

Задания для самостоятельной работы

1. После изучения темы 5.4 постройте концептуальную модель диаграммы активностей.
2. Постройте диаграммы активностей для учебного задания, приведенного в конце пособия.

Тема 6.4. Концептуальное моделирование и диаграммы понятий

Тематический контекст



Краткое содержание

1. Понятия и их атрибуты.
2. Идентификация понятий.
3. Ассоциации, идентификация ассоциаций, роли.
4. Ограничения.
5. Отношение обобщения, правило «100%», правило «*is_a*», абстрактные понятия, многомерная множественная классификация.
6. Отношение агрегации, композитная и коллективная агрегация, идентификация отношений агрегации.
7. Порядок построения концептуальной модели.

Определение. *Концептуальная модель предметной области* – это модель, отражающая основные понятия предметной области, их структуру и взаимосвязи.

Результат концептуального моделирования отображается на диаграмме понятий, которая является частным случаем диаграммы классов языка UML. На рис. 45 приведен пример диаграммы понятий, иллюстрирующей часть концептуальной модели предметной области «Система розничной торговли».

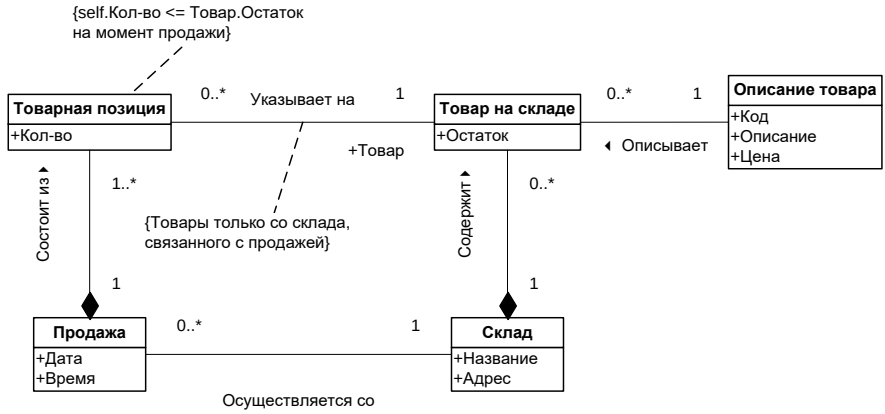


Рис. 45. Пример диаграммы понятий

На диаграммах понятий обычно отображаются следующие элементы: *понятия* и их *атрибуты*, отношения *ассоциации* и *обобщения*, а также *ограничения*. Разберем эти элементы подробно.

Понятие

Прежде всего, необходимо разобраться с понятием *понятия*. Говоря обще, *понятие* – это представление некоторой идеи или объекта в сознании человека. Формальное определение понятия дается ниже.

Определение. *Понятие* – это тройка: *символ*, *содержание* и *расширение*, где символ – это слово или образ, представляющий понятие; содержание – это определение понятия, а расширение – это множество примеров, к которым понятие применимо.

Рассмотрим в качестве примера понятие окружности. Сим-

волом в данном случае будет слово «окружность», определением – «геометрическое место точек, равноудаленных от данной», а расширением – множество всевозможных окружностей с разными радиусами.

Идентификация понятий. Для построения концептуальной модели необходимо прежде всего выделить понятия предметной области. Основным источником понятий являются текстовые описания предметной области (прецеденты). Точного метода выделения понятий нет. На практике можно использовать *непосредственное выделение понятий* из описания предметной области или *поиск по списку стандартных категорий*. Рассмотрим эти методы.

Суть непосредственного выделения понятий из текстовых описаний сводится к поиску имен существительных и их рассмотрению в качестве кандидатов на понятия. В данном случае надо понимать, что между именами существительными и понятиями нет однозначного соответствия, т.к. для естественного языка человека характерны такие явления, как *синонимы* и *омонимы*. Кроме того, имя существительное может соответствовать атрибуту (например, дата) или операции (например, завершение).

Богатый опыт моделирования позволил человечеству сформулировать список стандартных категорий понятий, которые обычно имеют значение (табл. 1). Данным списком можно воспользоваться для уточнения состава понятий концептуальной модели предметной области.

Для изображения понятий на диаграмме понятий используется нотация для класса без раздела операций (см. диаграмму классов). Если понятие не имеет атрибутов или они не важны в данном контексте, то раздел атрибутов может опускаться (рис. 46).

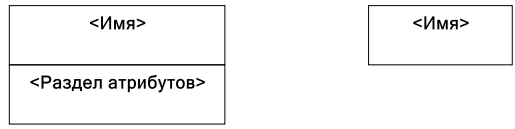


Рис. 46. Варианты графической нотации для изображения понятий

Таблица 1. Примеры категория понятий

Категория понятия	Пример
Физические или материальные объекты	Самолет
Спецификации, элементы дизайна или описания объектов	Описание полета, описание товара
Места	Магазин, Аэропорт
Транзакции	Продажа, Платеж, Бронирование
Элемент транзакции	Товарная позиция
Роли людей	Кассир, Пилот
Контейнеры других объектов	Склад, Магазин, Самолет
Содержимое контейнеров	Элемент, Пассажир, Товар
Другие компьютерные или электромеханические системы, внешние по отношению к данной системе	Банк (для банкомата)
Абстрактные понятия	Голод, Клаустрофобия
Организации	Отдел продаж, Библиотека
События	Продажа, Полет
Процессы (обычно не представляются в виде понятий)	Продажа продукта, Бронирование места
Правила и политики	Правила возврата товара, политика выдачи кредитов
Каталоги	Каталог товаров, книг
Записи финансовой, трудовой, юридической и др. деятельности	Чек, Трудовой контракт
Руководства, книги	Должностная инструкция

Ассоциации

Ассоциации используются для связывания понятий между собой. В подавляющем большинстве случаев диаграмма понятий представляет собой граф, вершинами которого являются понятия, а дугами – ассоциации (рис. 47). Повторим определение ассоциации.

Рис. 47. Пример ассоциации

Определение. *Ассоциация* – это устойчивое структурное отношение между сущностями (в данном случае – понятиями).

Основным способом выделения ассоциаций является выделение по определению. В данном случае можно предложить следующее правило. Если экземпляры некоторых двух понятий могут участвовать в устойчивых структурных связях друг с другом и есть необходимость в хранении информации об этих связях, то в концептуальную модель необходимо добавить ассоциацию между этими понятиями.

Другим способом является поиск по списку стандартных категорий ассоциаций (табл. 2). В этом случае берутся два понятия, между которыми подозревается наличие ассоциации, и проверяются по списку.

Таблица 2. Стандартные категории ассоциаций

Категория ассоциации	Пример
A является физической частью B	Крыло – Самолет
A является логической частью B	Товарная позиция – Продажа
A физически содержится в/на B	Товар – Полка, Пассажир – Самолет
A логически содержится в B	Описание товара – Каталог
A является описанием B	Описание товара – Товар
A известен / зарегистрирован / записан / включен в отчет / содержится в B	Продажа – Система розничной торговли
A является членом B	Ученик – Класс, Игрок – Команда
A является организационной единицей B	Отдел – Магазин
A использует или управляет B	Кассир – Система розничной торговли, Пилот – Самолет
A взаимодействует с B	Покупатель – Кассир
A связан с транзакцией B	Покупатель – Платеж
A является транзакцией, которая связана с другой транзакцией B	Платеж – Продажа
A следует за B	Остановка – Остановка
A является собственностью B	Самолет – Авиакомпания, Система розничной торговли – Магазин

Из приведенного выше списка особую важность имеют следующие категории ассоциаций:

– A является физической или логической частью B;

– A физически или логически содержится в B;
– A записан в B.

Как и у понятий, у ассоциаций есть имена. В языке UML есть определенный стандарт именования ассоциаций. Имя ассоциации должно содержать некоторую глагольную форму, причем при прочтении подряд имени одного понятия, имени ассоциации и имени другого понятия, должна получаться осмысленная фраза. По умолчанию, диаграмма понятий читается слева направо и сверху вниз. Если какую-то ассоциацию необходимо читать в противоположном направлении, то это отмечается при помощи закрашенной стрелки (рис. 48).

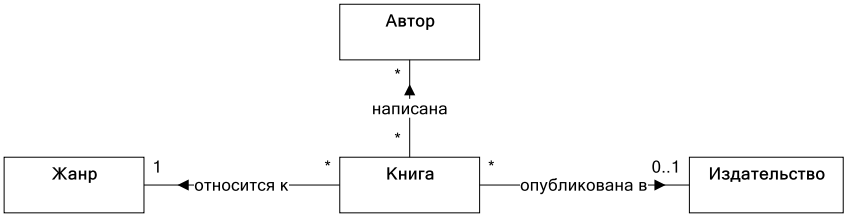


Рис. 48. Именование ассоциаций

Роли

Каждый конец ассоциации называется ролью. Роль имеет две базовые характеристики: *имя* и *кратность*. Имя определяет название роли, в которой выступает понятие в данной ассоциации. Если имя роли не указано, то считается, что оно совпадает с именем понятия. В примере на рис. 49 человек, выступая в роли классного руководителя, руководит классом. Данную ассоциацию следует читать следующим образом: «Классный руководитель руководит классом». Аналогичным образом читается и вторая ассоциация.

Рис. 49. Роли ассоциации

Кратность роли определяет, сколько экземпляров понятия, соответствующего данной роли, может быть одновременно связано с одним экземпляром понятия, соответствующего противоположной роли. Из примера на рис. 45 следует, что классом одновременно может руководить только один руководитель, но учиться в нем могут сразу несколько учеников (символ «*» означает «много»). Кратность роли может задаваться числом, промежутком (пример: 3..5), перечислением (пример: 2,5,7) или произвольной комбинацией указанных способов.

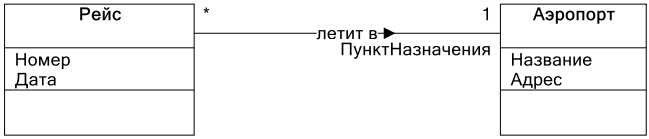
При поиске ассоциаций следует придерживаться следующих рекомендаций:

1. Гораздо важнее идентифицировать понятия, чем ассоциации.
2. Изучение ассоциаций не должно отнимать слишком много времени, но должно приносить максимальный эффект.
3. Избегайте использования избыточных ассоциаций.

Атрибуты

Перейдем теперь к рассмотрению атрибутов. При построении концептуальной модели иногда возникает вопрос: что считать атрибутом, а что считать понятием. Дадим определение атрибута.

а



б

Рис. 50. Варианты моделирования рейса

Определение. *Атрибут* – это именованная характеристика сущности.

У атрибута есть имя и тип. В большинстве случаев тип атрибута должен быть примитивным (строка, число, логический, дата, перечисление). Если оказывается, что тип атрибута не элементарен или есть сомнения, то лучше сделать этот атрибут отдельным понятием и использовать ассоциацию. Например, на рис. 50, *а* атрибут «`Пункт назначения`» в большинстве случаев будет иметь сложную структуру. В соответствии с требованием о примитивности типа атрибута более предпочтительным вариантом является второй (рис. 50, *б*).

Есть другой способ отличить понятие от атрибута: необходимо решить, важно ли выполнение требования уникальной тождественности (т.е. важно ли различать два одинаковых значения этого атрибута)? Если да, то это понятие, в противном случае – скорее всего атрибут. Например, в крупном мегаполисе названия некоторых улиц в различных районах могут совпадать. Следовательно, использование улицы в качестве атрибута понятия «`Адрес`» нежелательно, т.к. приведет к путанице адресов (рис. 51, *а*). В данном случае имеет смысл использовать отдельное понятие «`Улица`» (рис. 51, *б*).

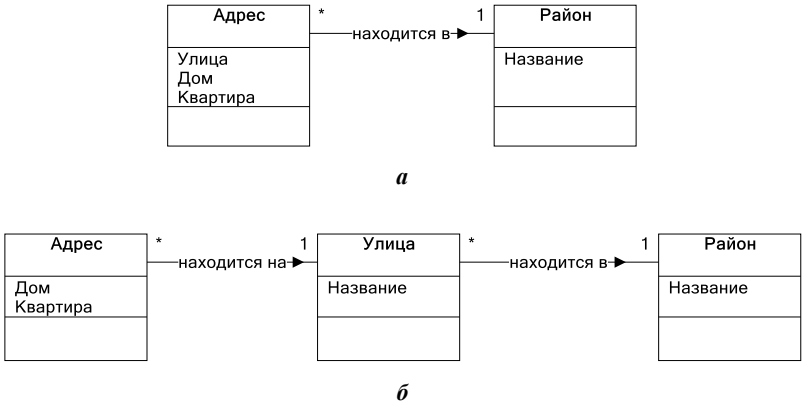


Рис. 51. Варианты моделирования адреса

Тип, изначально считающийся примитивным, может быть представлен в виде не примитивного в следующих случаях:

- если он состоит из отдельных частей (например, адрес или ФИО);
- если с этим типом обычно ассоциируются операции, такие как синтаксический анализ и проверка (например, ИНН или номер кредитной карты);
- если тип используется для задания количества с единицами измерения (например, цена за шт./кг/литр).

Ограничения

С любым элементом диаграммы понятий (или любой другой диаграммы языка UML) может быть связано одно или несколько *ограничений*. Ограничения *дополняют семантику стандартных элементов* и могут задаваться как в произвольной форме, так и формально. В качестве формального языка чаще всего используется целевой язык программирования или специальный язык OCL (Object Constraint Language – язык объектных ограничений), который входит в семейство стандартов OMG. На диаграмме ограничения заключаются в фигурные скобки и связываются с другими элементами при помощи пунктирных линий. Примеры ограничений можно найти на рис. 45.

Обобщение

Отношение *обобщения* уже рассматривалось для модели прецедентов, где оно использовалось между акторами и между прецедентами. В концептуальной модели обобщение используется между понятиями. Далее вместо термина «понятие» будет использоваться термин «тип», т.к. он более удобен. Повторим определение обобщения.

Определение. *Обобщение* – это отношение между элементом-родителем и элементом-потомком, при котором потомок является частным случаем родителя и наследует его структуру, поведение и семантику.

Элемент-родитель обычно называется *супертипом*, а элемент-потомок – *подтипом*. Движение от подтипа к супертипу называется обобщением, а от супертипа к подтипу – специализацией (рис. 52).

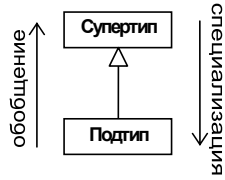


Рис. 52. Обобщение и специализация

Для проверки правильности использования отношения обобщения существует два правила: правило «100%» и правило «*is_a*». Отношение обобщения считается правильным, если выполняются оба эти правила. Рассмотрим их подробно.

Правило «100%». 100% определения супертипа должны быть применимы к подтипу. Подтип должен удовлетворять 100% определения своего супертипа. Это касается как атрибутов, так и ассоциаций.

Модель, приведенная на рис. 53, удовлетворяет сформулированному выше правилу, т.к. каждый конкретный платеж также характеризуется датой, суммой и ассоциацией с продажей.

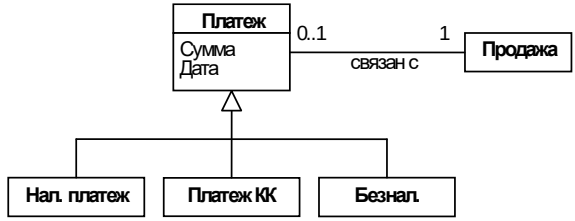


Рис. 53. Модель платежа

Правило «is_a». Все элементы множества (расширения) подтипа должны быть элементами множества (расширения) его супертипа, т.е. подтип – это (*is-a*) супертип.

На рис. 54 проиллюстрированы множества типов для модели платежа. Действительно, каждый конкретный платеж является платежом. Следовательно, любой экземпляр (элемент множества), например безналичного платежа, будет также и экземпляром (элементом множества) платежа. Таким образом, модель платежа на рис. 53 удовлетворяет обоим правилам и, следовательно, является корректной.

Рис. 54. Множества типов для модели платежа

Покажем, что выполнения только одного правила недостаточно. Рассмотрим, например, два понятия: окружность и эллипс. Для простоты будем считать, что оси эллипса расположены параллельно осям координат. Использование только правила «100%» может привести к варианту, показанному на рис. 55, *а*. Данная модель является ошибочной, т.к. эллипс не является частным случаем окружности. Использование только правила «*is_a*» может привести к варианту, показанному на рис. 55, *б*, который также является неправильным, т.к. подтип не наследует структуру супертипа. Использование обоих правил исключает первые два варианта и приводит к правильной модели, представленной на рис. 55, *в*.

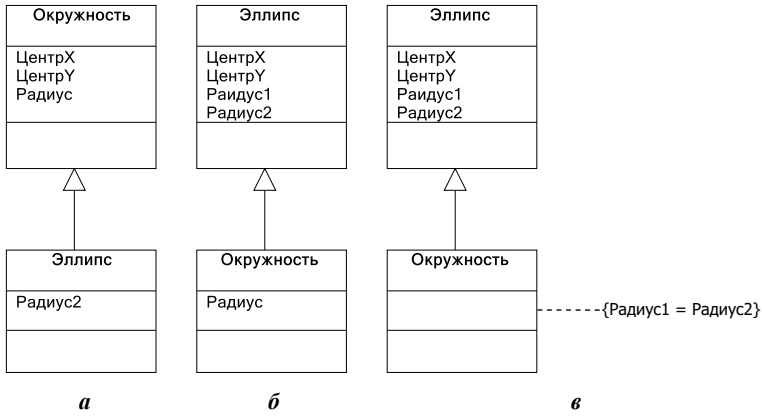


Рис. 55. Способы моделирования окружности и эллипса

Правила «100%» и «*is_a*» определяют, когда можно использовать обобщение, но ничего не говорят о том, когда его нужно использовать. Например, на рис. 56 представлена фор-

мально правильная, но в большинстве случаев бесполезная иерархия понятий. Ниже приводятся рекомендации, когда следует выделять подтипы (специализация) и супертипы (обобщение).

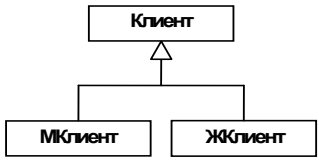


Рис. 56. Модель клиента

Тип *следует разбивать на подтипы*, если:

– подтип имеет дополнительные атрибуты, интересные с точки зрения задачи (рис. 57, *а*);
– подтип имеет дополнительные ассоциации, интересные с точки зрения задачи (рис. 57, *а*);
– подтипу соответствует понятие, управляемое, обрабатываемое, реагирующее или используемое способом, отличным от способа, определяемого супертипом или другим подтипом (рис. 57, *б*).

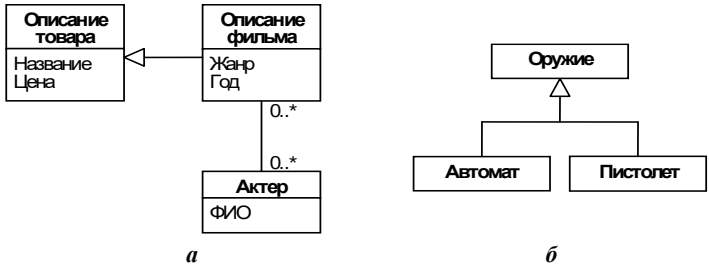


Рис. 57. Примеры использования обобщения

Для группы типов *следует вводить супертип*, если:

– потенциальные подтипы будут удовлетворять правилам «100%» и «*is_a*»;
– все подтипы имеют один и тот же атрибут, который можно вынести в супертип;
– все подтипы имеют одну и ту же ассоциацию, которую

можно выделить и связать с супертипом.

Прямые и косвенные экземпляры

Прежде чем двигаться дальше, введем понятия прямых и косвенных экземпляров.

Определение. Экземпляр некоторого понятия называется *прямым*, если он больше не является экземпляром ни одного другого понятия, находящегося ниже по иерархии наследования.

Определение. Экземпляр некоторого понятия называется *косвенным*, если он также является экземпляром, по крайней мере, одного понятия, находящегося ниже по иерархии наследования.

Абстрактные понятия

Выделение супертипов позволяет упрощать модели, повышает их выразительность и зачастую приводит к появлению абстрактных понятий.

Определение. *Абстрактное понятие* – это понятие, которое не имеет прямых экземпляров.

Отсутствие прямых экземпляров означает, что множества подтипов полностью покрывают множество супертипа. Сравните рис. 54 с рис. 58. В последнем случае любой элемент множества «Платеж» попадает в одно из подмножеств.

Рис. 58. Множество экземпляров абстрактного понятия «Платеж»

Для изображения абстрактных понятий на диаграмме понятий используется курсивное написание имени (рис. 59).

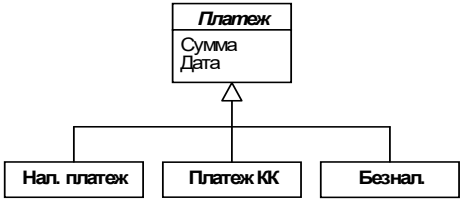


Рис. 59. Пример абстрактного понятия

Многомерная множественная классификация

В языке UML имеется возможность осуществления многомерной специализации родительского понятия. В этом случае его экземпляры будут классифицироваться сразу по нескольким признакам (в нескольких измерениях) и соответственно принадлежать сразу нескольким классам. Более того, предусматривается возможность того, что даже в рамках одного измерения специализации экземпляры могут принадлежать нескольким классам. Ключевым понятием многомерной множественной классификации является понятие *дискриминатора*.

Определение. *Дискриминатор* – представление разбиения, которому принадлежит отношение обобщения. Каждое разбиение представляет собой ортогональное измерение специализации родительского понятия.

На рис. 60 представлен пример с двумя дискриминаторами – «Пол» и «Роль». Соответственно для человека определяются два измерения специализации (два разбиения). Важно понимать, что каждый косвенный экземпляр родительского понятия должен быть экземпляром, по крайней мере, одного понятия из каждого разбиения.

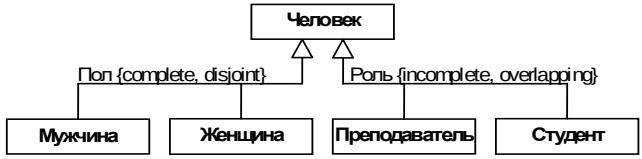


Рис. 60. Двумерная классификация людей по полу и роли в системе

Каждый дискриминатор кроме имени разбиения может также определять его свойства, которые указываются в фигур-

ных скобках (рис. 60). Для разбиений определены две пары стандартных свойств.

- `complete` означает, что разбиение полное, т.е. каждый экземпляр родительского понятия обязательно является также и экземпляром по крайней мере одного из дочерних понятий в данном разбиении, т.е. базовое понятие является абстрактным в данном измерении;
- `incomplete` означает, что разбиение неполное, т.е. допускаются прямые экземпляры базового понятия; неполнота разбиения подразумевает, что могут появиться дополнительные дочерние понятия;
- `disjoint` означает, что разбиение непересекающееся, т.е. никакой экземпляр одного дочернего понятия не может быть также экземпляром другого дочернего понятия;
- `overlapping` – означает, что разбиение пересекающееся, т.е. допускается, чтобы экземпляр одного дочернего понятия был одновременно экземпляром другого дочернего понятия.

Если свойства разбиения не указаны или указаны не полностью, то по умолчанию считается, что разбиение неполное непересекающееся (`incomplete`, `disjoint`). Если не указано имя разбиения, то оно считается безымянным.

Агрегация

При моделировании структурных отношений между понятиями нередко приходится сталкиваться с такой ситуацией, когда отношение является неравноправным, т.е. одно понятие выступает в роли главного, а второе – в роли починенного. Для отражения таких особенностей предметной области используется агрегация.

Определение: *Агрегация* – это вид ассоциации, моделирующий неравноправные отношения типа «часть–целое».

В качестве примера агрегации можно привести отношения между рукой и пальцами, папкой и файлами, таблицей и строками, и т.д. При построении диаграмм, имя ассоциации в отношениях агрегации зачастую не указывается, т.к. подразумевается имя «имеет часть» или «является частью».

Агрегация бывает двух видов: *композитная* и *коллектив-*

ная.

Композитная агрегация (обычно ее называют *композицией*) – это сильная связь; она подразумевает, что со стороны составного объекта кратность не может превышать единицу. Считается, что составной объект является владельцем своих частей, а части не могут существовать без целого. Композитная агрегация изображается в виде закрашенного ромба со стороны составного объекта (рис. 61).

Рис. 61. Пример композитной агрегации

Коллективная агрегация (обычно ее называют просто *агрегацией*) – это более слабая связь, которая подразумевает, что со стороны составного объекта кратность может быть больше единицы. Кроме того, допускается участие в нескольких агрегациях. В отличие от композиции, при уничтожении составного объекта не предполагается уничтожение частей. Коллективная агрегация изображается в виде пустого ромба со стороны составного объекта (рис. 62).

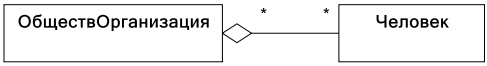


Рис. 62. Пример коллективной агрегации

Правила идентификации отношения агрегации

Вводить агрегацию следует в следующих случаях:

- время жизни компонента ограничено временем жизни составного объекта, т.е. есть зависимость создания/удаления (например, файл–каталог);
- в физическом или логическом агрегате очевидно наличие отношения «часть–целое» (например, машина–двигатель);
- некоторые свойства составного объекта распространяются и на его компоненты, например, место их расположения (к примеру, папка–документ);
- операции, применяемые к составному объекту, осуществляются и над его частями, например, разрушение,

перемещение, запись, копирование (например, файл–каталог).

Порядок построения концептуальной модели

При построении концептуальной модели можно руководствоваться следующей последовательностью шагов:

1) составить список понятий;
2) идентифицировать их атрибуты;
3) идентифицировать ассоциации;
4) произвести разбиение на подтипы и выделение супертипов;
5) выделить отношения агрегации.

Эти шаги можно выполнять как для всей модели в целом, так и для ее частей.

Рекомендации по построению диаграмм понятий

Нет ничего хуже, чем неаккуратная, непонятная диаграмма. При построении любых диаграмм, и диаграмм понятий в частности, старайтесь делать их простыми, наглядными, легко читаемыми. Для этого следует придерживаться приведенных далее правил.

Не стоит пытаться все вместить в одну диаграмму. Лучше разбить модель на несколько относительно независимых частей и построить несколько диаграмм. При этом некоторые части модели придется повторить на нескольких диаграммах – в этом нет ничего страшного. При разбиении имеет смысл придерживаться правила «7 ± 2». Именно такое количество понятий на диаграмме является оптимальным.

Каждая диаграмма должна акцентировать внимание на определенном аспекте системы. Для этого имеет смысл скрывать несущественные в данном контексте элементы модели (атрибуты, ассоциации, ограничения).

Располагайте понятия равномерно друг относительно друга. Линии отношений рисуйте параллельно осям координат. Не допускайте пересечения линий и излишних изломов (максимум 2). Некоторые аналитики совершенно напрасно пренебрегают эстетической стороной построения диаграмм. С моделями этих людей работать потом неприятно.

Вопросы для самоконтроля

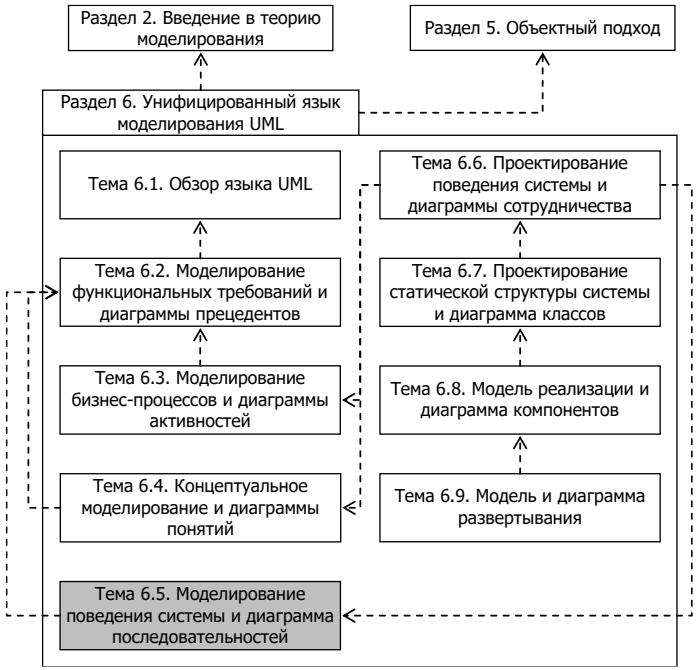
1. Что такое концептуальная модель?
2. Как выделять понятия и ассоциации?
3. Как правильно выделять атрибуты?
4. Для чего нужны роли?
5. В чем суть правил «100%» и «is_a»?
6. В чем особенность абстрактных понятий?
7. В чем разница между коллективной и композитной агрегацией?

Задания для самостоятельной работы

1. Подумайте, почему требуются два правила: «100%» и «*is_a*»? Почему не достаточно одного из них?
2. Подумайте, почему многомерная множественная классификация не используется в языках программирования?
3. Постройте концептуальную модель диаграммы понятий.
4. Постройте диаграммы понятий для учебного задания.

Тема 6.5. Моделирование поведения системы и диаграмма последовательностей

Тематический контекст



Краткое содержание

1. Модель поведения системы, системные сообщения и системные операции.
2. Принцип черного ящика.
3. Диаграмма последовательностей.
4. Основные элементы диаграммы: объекты, линии жизни, сообщения, периоды активации.
5. Порядок построения диаграммы.
6. Описание системных операций.

Модель поведения системы

Модель поведения системы завершает этап анализа. Данная модель строится в основном на базе модели прецедентов. Система в данном случае рассматривается как черный ящик. Ос-

новной целью построения модели поведения системы является определение системных операций и системных сообщений.

Определение. *Системное сообщение* – это входящее сообщение, сгенерированное актором для системы и переданное через ее границу.

Определение. *Системная операция* – это операция, выполняемая системой при получении системного сообщения.

Как видно из приведенных выше определений, системные операции и системные сообщения тесно связаны. Имена сообщений и соответствующих им операций совпадают. Разница между ними состоит в том, что сообщение является мгновенным стимулирующим воздействием, а операция представляет собой «продолжительную» реакцию на это воздействие.

Для отображения состава, порядка и параметров системных сообщений используется диаграмма последовательностей.

Напомним, что каждый прецедент описывает несколько связанных сценариев взаимодействия акторов с системой. Каждый сценарий – это некоторая цепочка системных взаимодействий, которые осуществляются при помощи передачи системных сообщений. Соответственно для каждого сценария каждого прецедента можно построить диаграмму последовательностей, однако на практике этого обычно не требуется. В большинстве случаев бывает достаточно построить диаграммы только для основных потоков и наиболее интересных альтернативных потоков и подпотоков.

Для построения диаграммы необходимо выделить из текста описания прецедента все генерируемые акторами сообщения и их параметры и отобразить их на диаграмме в порядке генерации. Таким образом, диаграмма последовательностей представляет собой формальное представление некоторого сценария прецедента. Пример диаграммы последовательностей для основного потока прецедента «Продажа товара» представлен на рис. 63.

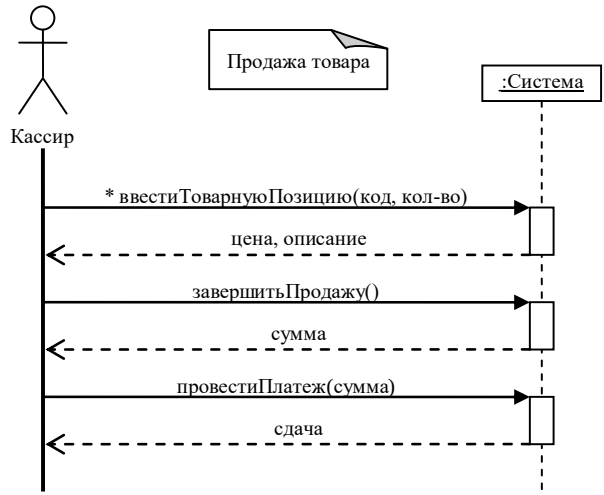


Рис. 63. Диаграмма последовательностей для прецедента «Продажа товара»

Основными элементами диаграммы последовательностей являются *объекты* (в т.ч. акторы) и *сообщения*.

Объекты

Все системные объекты на диаграммах последовательностей изображаются в виде прямоугольников, внутри которых указывается имя объекта и его тип, которые разделяются двоеточием. Если у объекта нет имени или оно не важно, то указывается только двоеточие и класс. Имя и класс объекта подчеркиваются. Примеры изображения объектов приведены на рис. 64.

Рис. 64. Примеры объектов

При построении модели поведения системы используется только один системный объект «:Система», который представляет всю систему в целом. Для изображения акторов используется тот же элемент, что и на диаграммах прецедентов.

У каждого объекта есть линия жизни, которая направлена

сверху вниз. Линии жизни обычных объектов изображается при помощи тонкой или пунктирной линии. Для того чтобы подчеркнуть активную природу акторов, их линии жизни изображаются жирной линией. В UML также есть понятие *активного объекта* – объекта, владеющего потоком управления. Линии жизни активных объектов также изображаются при помощи жирных линий. Если объект уничтожается в ходе взаимодействия, то его линия жизни оканчивается крестом (рис. 65). В модели поведения системы данный элемент не используется.

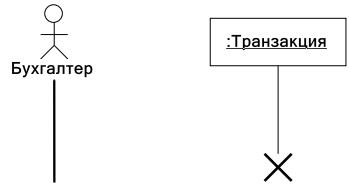


Рис. 65. Линии жизни объектов

Сообщения

В языке UML определено следующих три типа сообщений: *синхронные*, *асинхронные* и *ответные*.

Определение. *Синхронное сообщение* – это сообщение, после отправки которого отправитель дожидается завершения его обработки.

Определение. *Асинхронное сообщение* – это сообщение, после отправки которого отправитель может выполнять другие действия, не дожидаясь завершения его обработки.

Определение. *Ответное сообщение* – это сообщение, уведомляющее отправителя о завершении обработки его сообщения. Может использоваться для передачи результата.

Синхронные сообщения обозначаются при помощи сплошной линии с закрашенным треугольником на конце, асинхронные – сплошной линией со стрелкой на конце, а ответные – пунктирной линией со стрелкой на конце (рис. 66).

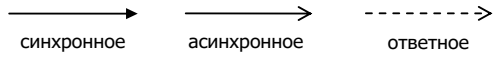


Рис. 66. Типы сообщений

Каждое сообщение кроме ответных сообщений имеет имя и, при необходимости, параметры. Имена сообщений должны содержать глаголы в повелительном наклонении и отражать те действия, которые должны быть выполнены в ответ на сообщение (рис. 67).

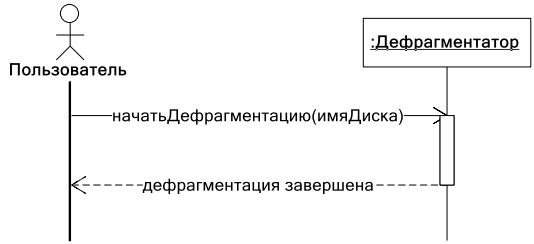


Рис. 67. Имена и параметры сообщений

На приведенном выше рисунке используется необязательный элемент, который называется *периодом активации* (белый прямоугольник на линии жизни объекта). Период активации иллюстрирует действия объекта по обработке сообщения (время владения потоком управления). Данный элемент позволяет повысить наглядность диаграммы, особенно в тех случаях, когда используются асинхронные сообщения.

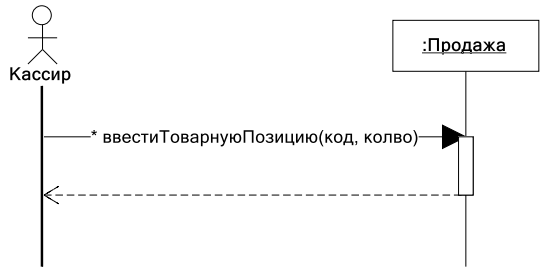


Рис. 68. Повторяющиеся сообщения

При анализе взаимодействия между объектами нередко встречается такая ситуация, когда некоторое сообщение отправляется несколько раз, причем обычно заранее неизвестно сколько. Для отражения такой ситуации на диаграммах последовательностей используется символ «*» перед именем сообщения,

который означает, что сообщение может быть отправлено ноль, один или более раз (рис. 68).

Если требуется изобразить на диаграмме последовательностей сразу два или более сценариев, которые отличаются только некоторыми шагами, то для этого можно использовать условные сообщения – сообщения, которые отправляются только при выполнении определенного условия. Условие отправки на диаграмме записывается перед именем сообщения в квадратных скобках и может быть сформулировано в произвольной форме (рис. 69).

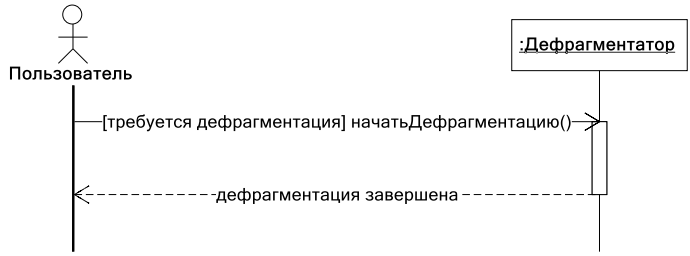


Рис. 69. Условные сообщения

Чаще всего условные сообщения встречаются в тех случаях, когда в зависимости от некоторого условия отправляется либо одно, либо другое (или одно из нескольких) сообщение. Такие сообщения называются альтернативными и изображаются на диаграмме при помощи условных сообщений, начала которых соединены (см. рис. 70).

Строго говоря, на рис. 70 изображены два параллельных сообщения с взаимоисключающими условиями отправки. Параллельность сообщений означает, что порядок их отправки неважен, в том числе допускается их одновременная отправка, что и иллюстрируется совмещенным началом соответствующих стрелок на диаграмме.

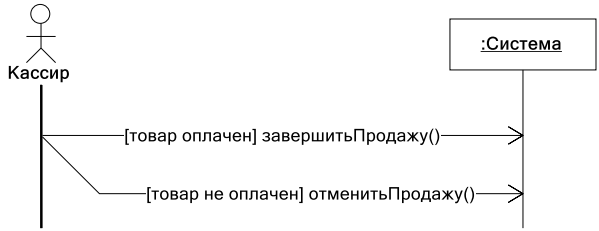


Рис. 70. Альтернативные сообщения

Описание системных операций

Каждому системному сообщению соответствует *системная операция*. Функциональность системы определяется набором системных операций. Прежде чем приступать к реализации системы (или ее части), необходимо определить все системные операции. Для этого можно использовать *описания системных операций.*

Определение. *Описание системной операции* – это документ, описывающий предполагаемые результаты ее выполнения; он составляется в декларативной форме и акцентирует внимание на том, что должно произойти, а не на том, как этого достичь.

В UML не определяется стандарта для описания системных операций. На практике рекомендуется использовать шаблон со следующими полями:

- *имя* – имя системной операции;
- *обязанности* – краткое описание того, за что отвечает данная системная операция;
- *ссылки* – прецеденты, в которых используется данная системная операция;
- *примечание* – комментарии, замечания о реализации и т.д.;
- *исключения* – возможные исключительные ситуации и реакции на них;
- *предусловия* – условия, которые должны быть выполнены перед началом операции;
- *постусловия* – описание изменений в системе, которые должны произойти после выполнения системной операции.

Самым главным разделом описания является раздел постусловий. Принцип его формирования следующий: необходимо сравнить состояние до выполнения операции с требуемым состоянием после ее выполнения и выделить все изменения. Все изменения на уровне данных можно свести к созданию/удалению экземпляров, формированию/разрыву связей и изменению значений атрибутов. При описании постусловий рекомендуется каждое изменение явно относить к одному из вышеописанных классов.

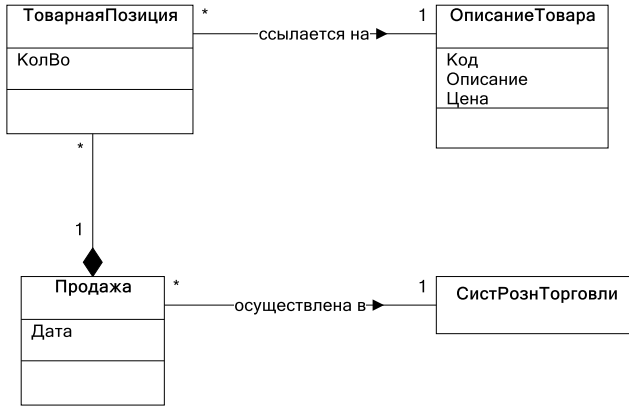


Рис. 71. Концептуальная модель продажи

Рассмотрим описание системных операций на примере операции «ввестиТоварнуюПозицию». Пусть данная операция используется в прецеденте «Продажа товара» для некоторой системы розничной торговли. На рис. 71 приведена часть концептуальной модели, которая понадобится для описания.

Имя: ввестиТоварнуюПозицию(код, количество)

Обязанности: Добавить товарную позицию к текущей продаже.

Ссылки: Прецедент «ПокупкаТовара»

Примечание: Использовать самый быстрый доступ к БД.

Исключения: Если код товара не известен, то запросить повторный ввод.

Предусловия: нет

Постусловия:

- Для новой продажи создан объект :Продажа (создание экземпляра).
- Для новой продажи объект :Продажа связан с объектом :СистРознТорговли (формирование связи).
- Создан объект :ТоварнаяПозиция (создание экземпляра).
- Объект :ТоварнаяПозиция связан с объектом :Продажа (формирование связи).
- Атрибут :ТоварнаяПозиция.Кол-во принял значение «количество» (модификация атрибута).
- Объект :ТоврнаяПозиция связан с объектом :ОписаниеТовара на основе соответствия кода товара (формирование связи).

На этапе анализа нет необходимости генерировать полный и точный набор постусловий. Некоторые интересные детали прояснятся на этапе проектирования. Новые открытия инициируют новые итерации. Это одно из преимуществ итеративной разработки.

Наиболее типичной ошибкой при описании системных операций является невключение формирования связей в число постусловий операции. Установка связей играет особо важную роль при создании новых экземпляров.

Описание системных операций является достаточно трудоемкой задачей и в относительно простых задачах обычно не используется. Однако стоит заметить, что кроме своего непосредственного назначения описания системных операций можно использовать для тестирования системы.

Вопросы для самоконтроля

1. В чем разница между системным сообщением и системной операцией? Что общего между ними?
2. В чем заключается принцип «черного ящика»?
3. Какие виды сообщений бывают и чем они различаются?

Задания для самостоятельной работы

1. Постройте концептуальную модель диаграммы последовательностей.
2. Постройте диаграммы последовательностей для учебного задания. При необходимости опишите системные операции.

Типичные ошибки

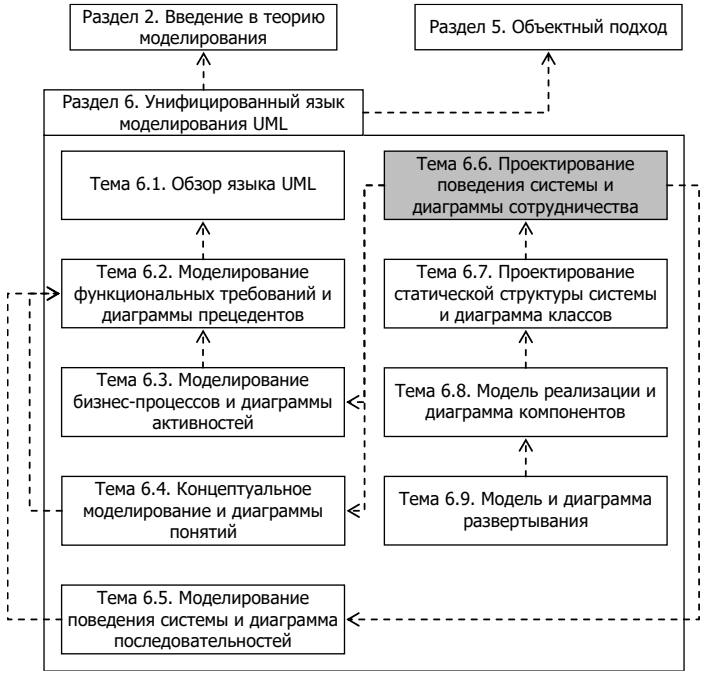
Типичной ошибкой является использование бессодержательных сообщений. При добавлении сообщения на диаграмму подумайте о соответствующей ему системной операции. Что она должна сделать? В чем состоит ее обязанность? Если Вы не можете коротко и четко ответить на эти вопросы, то имеет смысл пересмотреть принятое решение.

Рекомендации по построению диаграмм последовательностей

При выборе имен системных сообщений используйте глаголы в повелительном наклонении, например: «`создатьТоварнуюПозицию`», «`сохранитьДокумент`», «`получитьОстаток`» и т.д.

Тема 6.6. Проектирование поведения системы и диаграммы сотрудничества

Тематический контекст



Краткое содержание

1. Этап проектирования, основная цель этапа.
2. Два вида диаграмм взаимодействия: диаграммы последовательностей и диаграммы сотрудничества.
3. Диаграммы сотрудничества.
4. Понятия сотрудничества и взаимодействия.
5. Элементы диаграммы сотрудничества: объекты, связи, сообщения.
6. Работа со стандартными коллекциями.
7. Сообщения классам.
8. Видимость объектов.

По завершении этапа анализа разработчик информационной системы должен получить ответ на вопрос «что надо сделать?» и иметь на руках следующий набор артефактов:

– *модель прецедентов* (описания прецедентов, диаграммы прецедентов);
– *концептуальная модель* (диаграммы понятий);
– *модель поведения системы* (диаграммы последовательностей, описания системных операций).

Следующим этапом является этап *проектирования*, на котором разрабатывается логическое решение задачи. Цель этапа проектирования – ответить на вопрос «как добиться требуемой функциональности системы?». *Основными задачами* данного этапа являются

– построение диаграмм взаимодействия,
– построение диаграмм классов.

Диаграммы взаимодействия

Описания системных операций, которые были созданы на этапе анализа, содержат спецификацию постусловий выполнения операции. Однако в них нет информации о том, какие программные объекты отвечают за выполнение системной операции и каким образом они достигают требуемого состояния системы. Для раскрытия содержания системных операций используются диаграммы взаимодействия.

Диаграмма взаимодействия иллюстрирует процесс обмена сообщениями между объектами системы. В основе этого взаимодействия лежит необходимость выполнения постусловий из

описания соответствующей системной операции. Для каждой системной операции строится своя диаграмма взаимодействия. При проектировании поведения системы она рассматривается как белый ящик, и основное внимание уделяется описанию взаимодействия составных частей системы.

В UML определено два типа диаграмм взаимодействия: *диаграмма последовательностей* (SD – Sequence Diagram) и *диаграмма сотрудничества* (COD – Collaboration Diagram). Обе эти диаграммы одинаково выразительны, но делают акцент на различных аспектах взаимодействия.

Диаграммы последовательностей имеют выделенную ось времени и позволяют сосредоточить внимание на последовательности сообщений (рис. 72).

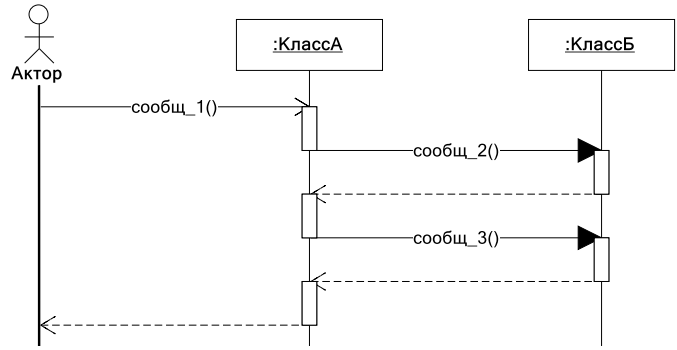


Рис. 72. Диаграмма последовательностей

Диаграммы сотрудничества акцентируют внимание на контексте взаимодействия. На них нет выделенной оси времени, зато явно указываются связи между объектами. Для задания порядка отправки сообщений используется их нумерация. Пример на рис. 73 содержательно идентичен примеру, приведенному на рис. 72.

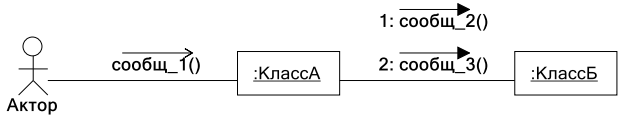


Рис. 73. Диаграмма сотрудничества

На практике обычно отдается предпочтение диаграммам сотрудничества, т.к. они более компактны и явно определяют контекст взаимодействия, что на этапе проектирования немаловажно. Далее рассматриваются именно диаграммы сотрудничества.

Диаграмма сотрудничества

На диаграммах сотрудничества выделяют два аспекта – это *сотрудничество* и *взаимодействие*. Определим эти понятия.

Определение. *Сотрудничество* – это состав объектов взаимодействия и связи между ними.

Определение. *Взаимодействие* – это последовательность сообщений, которыми обмениваются объекты взаимодействия.

Данные аспекты диаграммы сотрудничества связаны следующим образом. На диаграмме может присутствовать как сотрудничество с взаимодействием, так и одно сотрудничество. В рамках одного сотрудничества может осуществляться несколько взаимодействий (отображается на разных диаграммах).

Сотрудничество формирует контекст, в котором происходит взаимодействие объектов. Для описания сотрудничества используются практически те же самые элементы, что и для концептуальной модели (рис. 74).

Рис. 74. Пример неименованного и именованного объекта

В ходе взаимодействия объекты могут создаваться и уничтожаться. Созданные объекты помечаются как `{new}`, уничтоженные – `{deleted}`, а созданные и затем уничтоженные (временные) – `{transient}` (рис. 75).

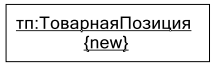


Рис. 75. Пометка созданного объекта

Связи между объектами показываются при помощи дуг (рис. 76). Причем, в отличие от ассоциаций, в данном случае множественность не указывается. Каждая связь связывает ровно два объекта, а соответствующие ассоциации определяют, какие объекты в каких связях могут участвовать.

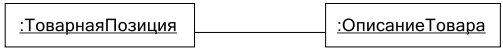


Рис. 76. Пример связи

Допускается использование агрегации и ролей (рис. 77).

Для моделирования внешней среды используются акторы (рис. 78).

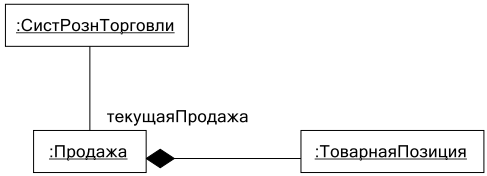


Рис. 77. Использование ролей и агрегации

Рис. 78. Моделирование внешней среды

Перейдем к рассмотрению взаимодействия. Взаимодействие объектов осуществляется путем обмена сообщениями. Сообщение может принимать различные формы, например: вызов метода, посылка сигнала или инициирование события. На диаграммах сотрудничества используются синхронные и асинхронные сообщения, для изображения которых используются те же графические элементы, что и для диаграмм последовательностей. Сообщения могут передаваться только вдоль линий связи между объектами (рис. 79).

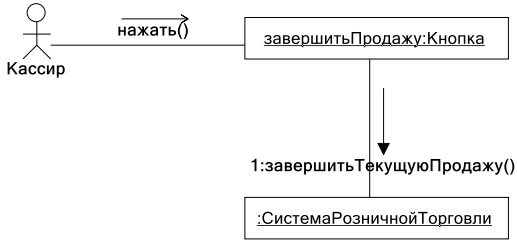


Рис. 79. Отправка сообщений

Каждое сообщение имеет метку, которая включает как минимум имя сообщения и круглые скобки. Полный формат метки сообщения имеет следующий вид:

```
<Предшественники> / <Номер> [<Условие>]:
                  <Переменная> := <Имя> ( <Аргументы> )
```

Предшественники – это список номеров сообщений, обработка которых должна завершиться прежде, чем данное сообщение может быть послано. Если у сообщения есть предшественники, то после них ставится *слэш*. В примере на рис. 80 четыре класса обмениваются асинхронными сообщениями. Суть данного взаимодействия может сводиться, например, к следующему. По запросу пользователя объект класса `A` опрашивает состояние удаленных объектов `б1` и `б2`, после чего результаты опроса сообщает объекту класса `B`. Соответственно, сообщение 3 может быть отправлено только после того, как объект класса `A` получит сообщения `1.1` и `1.2`.

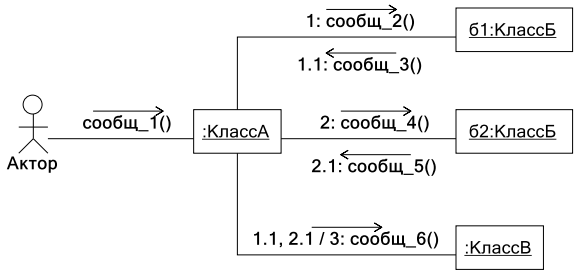


Рис. 80. Пример обмена асинхронными сообщениями

Номер сообщения определяет его положение в цепочке сообщений. Нумерация бывает *плоской* и *иерархической*. При плоской нумерации с каждым сообщением связывается натуральное число, определяющее абсолютный порядковый номер сообщения. Однако такую нумерацию неудобно использовать в случае с асинхронными сообщениями, т.к. в этом случае определить абсолютный порядковый номер бывает невозможно. В связи с этим на практике применяется иерархическая нумера-

ция, при которой сообщению присваивается относительный номер «внутри» родительского сообщения – сообщения, в ходе обработки которого было сгенерировано данное. Например, сообщение `сообщ_5` на рис. 80 было сгенерировано в ответ на сообщение `сообщ_4` с номером `2` и, следовательно, имеет номер `2.1`.

Для отображения параллельных сообщений к номеру сообщения дописывается буква, например, `3.1а` и `3.1б`. Это означает, что порядок отправки сообщений неважен (т.е. при наличии возможности их можно отсылать одновременно). Например, для сообщений `сообщ_2` и `сообщ_4` на рис. 80 порядок их отправки не важен, и диаграмму можно нарисовать так, как показано на рис. 81.

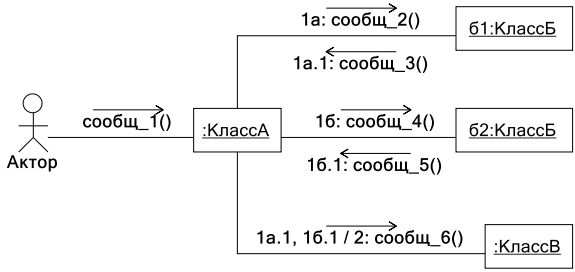


Рис. 81. Параллельные сообщения

Условие, как и на диаграммах последовательностей, определяет условие посылки сообщения, или итерацию, например, `[продажа не создана]` или `*[для всех товарных позиций]`. При помощи условий и параллельных сообщений можно организовать взаимоисключающие сообщения, например, `1а [условие]` и `1б [not условие]`.

Переменная – это имя переменной, куда помещается результат обработки сообщения, например:

```
тп := найтиТоварнуюПозицию(код)
```

Этот идентификатор может быть использован в качестве аргументов других сообщений или в условиях.

Имя – это имя сообщения. Требования к именованию сообщений такие же, как и для диаграммы последовательностей.

В UML выделяется два стандартных сообщения `Create` и `Destroy`, которые отвечают за создание и уничтожение объектов соответственно.

Аргументы – это список параметров сообщения, заключенный в круглые скобки. Если сообщение не имеет параметров, то пишутся пустые скобочки.

Работа с коллекциями объектов

Для стандартных коллекций объектов, таких как список, массив и пр., в UML введено специальное понятие *мультиобъекта*. Пример мультиобъекта приведен на рис. 82. В имени мультиобъекта после двоеточия указывается тип элементов коллекции.

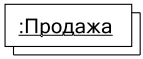


Рис. 82. Пример мультиобъекта

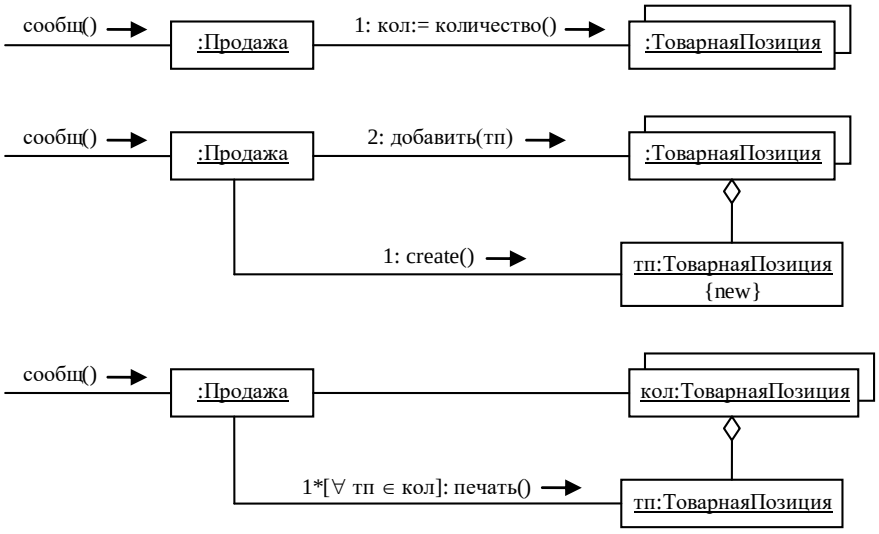


Рис. 83. Примеры работы с коллекциями

Важно понимать разницу между мультиобъектом и специализированной коллекцией. Мультиобъект – это стандартная кол-

лекция объектов, которая реализуется такими классами, как, например, ArrayList в .NET. Соответственно мультиобъекту можно отправлять только простые сообщения, типа добавить/удалить элемент, получить количество элементов и т.д. Если нужна сложная логика (например, получить все продажи за некоторый период), то необходимо создавать отдельный класс. На рис. 83 представлены некоторые примеры работы с коллекцией товарных позиций.

Сообщения классу

Сообщения могут передаваться не только объектам, но и самим классам. Сообщение классу соответствует вызову статического метода. На рис. 84 представлен пример получения текущей даты при помощи сообщения классу. Класс на диаграмме отличается от объекта тем, что его имя не подчеркнуто и пишется без двоеточия.

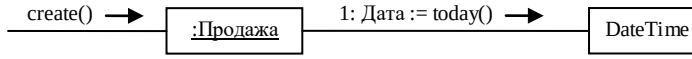


Рис. 84. Пример сообщения классу

Видимость объектов

Видимость между объектами является *необходимым условием для обмена сообщениями*. Для отправки сообщения объектом А объекту Б необходимо, чтобы объект Б был виден объекту А. Существуют следующие четыре способа обеспечения видимости объекта Б объектом А:

1. *Посредством атрибутов (ассоциаций)*. Ссылка на объект Б является значением некоторого атрибута объекта А.
2. *Посредством параметров*. Ссылка на Б передается в качестве параметра в некоторый метод объекта А.
3. *Локальная видимость*. Ссылка на объект Б хранится в локальной переменной некоторого метода объекта А.
4. *Глобальная видимость*. Объект Б доступен глобально объекту А.

Кроме того, каждый объект видит сам себя. Способ обеспе-

чения видимости между объектами пишется в кавычках со стороны видимого объекта (рис. 85). Если способ обеспечения видимости не указан, то подразумевается видимость посредством ассоциаций.

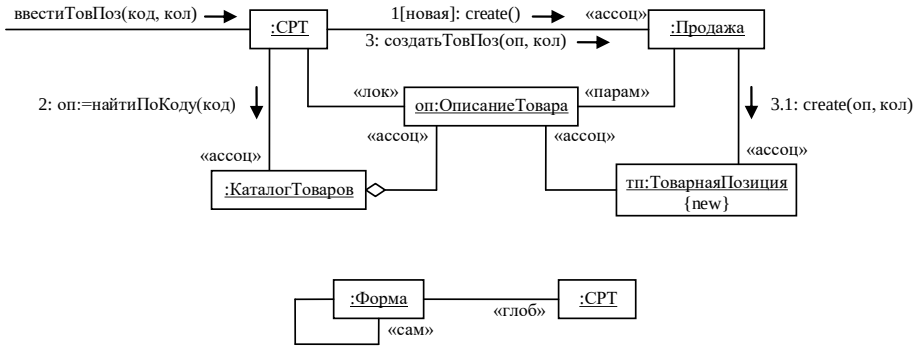


Рис. 85. Видимость между объектами

В представленном выше списке способы обеспечения видимости перечислены в порядке убывания их приоритета. Если некоторый объект виден несколькими способами, то следует на диаграмме указывать тот способ, который имеет больший приоритет. Например, на рис. 85 объект `оп: ОписаниеТовара` передается в качестве параметра конструктора при создании объекта `тп: ТоварнаяПозиция`, после чего между ними формируется связь, соответствующая ассоциации «ссылается на» (рис. 71). Поскольку видимость посредством ассоциации имеет больший приоритет, чем видимость посредством параметра, то на диаграмме (рис. 85) указывается именно она.

Построение диаграмм сотрудничества является, пожалуй, самым сложным этапом, т.к. это творческий процесс, в рамках которого принимаются основные проектные решения. Человеческий опыт проектирования взаимодействия сформировал некоторые правила и вобрал удачные идеи, на которые имеет смысл ориентироваться. Одной из форм выражения этих идей являются шаблоны проектирования, некоторые из которых рассматриваются в следующей теме.

Вопросы для самоконтроля

1. В чем состоит основная цель этапа проектирования?
2. В чем разница между диаграммой последовательностей и диаграммой сотрудничества?
3. Что такое сотрудничество и взаимодействие? Как они связаны?
4. Какие сообщения можно посылать мультиобъектам?
5. Для чего нужна видимость между объектами?
6. Какие существуют способы обеспечения видимости?

Задания для самостоятельной работы

1. Постройте концептуальную модель диаграммы сотрудничества.
2. Постройте диаграммы сотрудничества для учебного задания.

Типичные ошибки

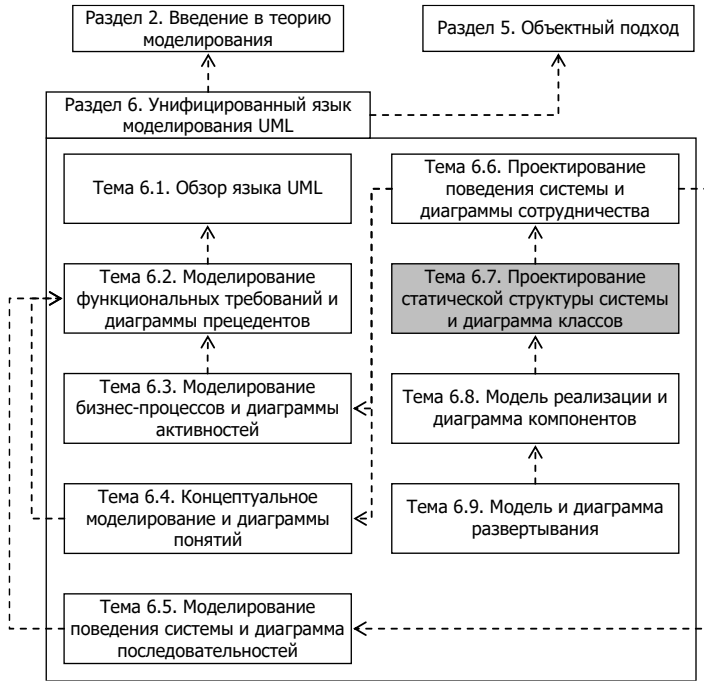
Типичной ошибкой является злоупотребление переменными. Если для реализации Вашего взаимодействия требуется много переменных, то, скорее всего, Ваши объекты обрабатывают не свои данные, что противоречит объектно-ориентированному подходу. Старайтесь свести количество сообщений к разумному минимуму.

Рекомендации по построению диаграмм сотрудничества

Диаграммы сотрудничества – это самые сложные для восприятия диаграммы. Старайтесь делать их максимально наглядными и простыми.

Тема 6.7. Проектирование статической структуры системы и диаграмма классов

Тематический контекст



Краткое содержание

1. Диаграмма классов. Видимость членов класса.
2. Описание атрибутов и операций.
3. Параметризованные классы.
4. Квалифицированные ассоциации.
5. Правила построения диаграммы классов.

Диаграмма классов

Построение диаграммы классов завершает этап проектирования. На данной диаграмме отражается статическая модель системы. Обычно диаграммы классов строятся параллельно с диаграммами сотрудничества, т.к. в различных ситуациях акцент делается либо на поведении, либо на структуре системы. Обе эти

диаграммы должны находиться в строгом соответствии, общие моменты которого можно определить следующими правилами:

- каждому объекту на диаграмме сотрудничества должен соответствовать класс на диаграмме классов;
- каждому входящему сообщению должны соответствовать операция или свойство класса принимающего его объекта;
- для каждой связи между объектами типа «ассоциация» на диаграмме классов должна присутствовать ассоциация;
- для прочих типов связей между объектами на диаграмме классов должны присутствовать зависимости.

На рис. 86 представлена диаграмма классов, которая была построена на основе рассмотренных выше диаграмм сотрудничества.

Многие элементы графической нотации диаграмм классов уже были введены при изучении диаграмм понятий. Рассмотрим новые элементы.

Операции

На диаграмме классов кроме раздела атрибутов используется дополнительно *раздел операций*. Как уже говорилось, *операции соответствуют входящим сообщениям*, которые должны уметь обрабатывать экземпляры класса. У операции есть имя, список параметров и, если операция что-то возвращает, то и тип результата.

Информация о типах

На диаграммах классов для всех членов (атрибуты, параметры и результат операций) должны быть указаны *типы данных*. В качестве типов можно использовать стандартные типы целевого языка программирования и имена других классов модели.

Информация об области видимости

Для каждого члена класса необходимо указать область его видимости. Существует три вида областей видимости:

- *public* (+) – член класса виден за его пределами;
- *protected* (#) – член класса виден только самому классу и его потомкам;
- *private* (–) – член класса виден только самому классу.

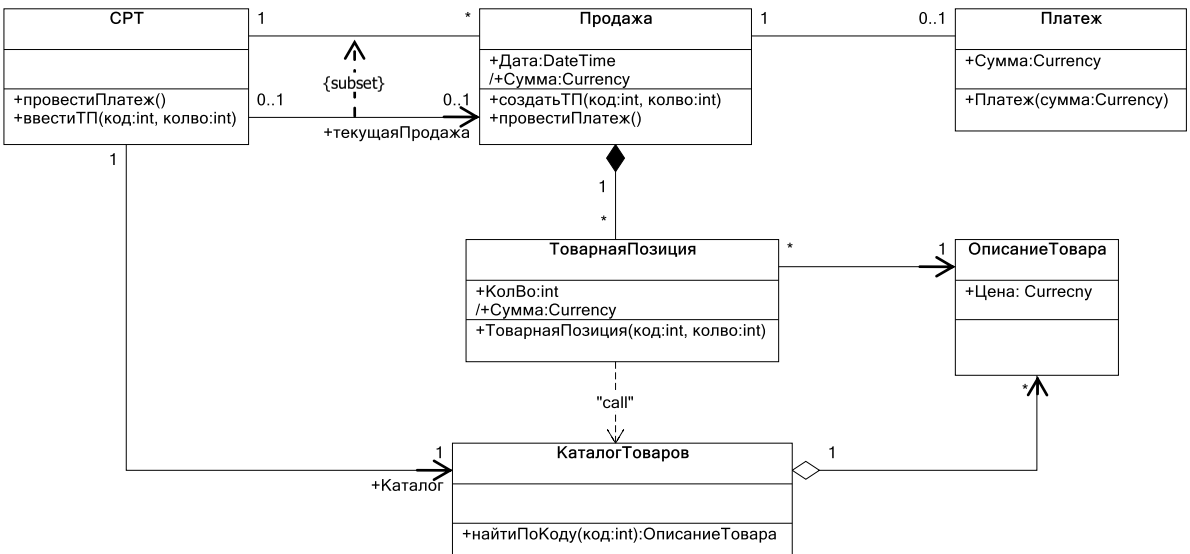


Рис. 86. Диаграмма классов для системы розничной торговли

На рис. 86 все члены классов имеют видимость *public*, т.к. на данной диаграмме изображена модель спецификации. В данном случае атрибут `Платеж.Сумма` следует воспринимать как свойство класса. На модели реализации данное свойство может быть реализовано как скрытое поле и пара методов доступа (рис. 87).

Вычислимые атрибуты

Вычислимые атрибуты – это такие атрибуты, чье *значение класс не хранит, а вычисляет при каждом обращении*. Примерами вычислимых атрибутов на рис. 86 являются `Продажа.Сумма` и `ТоварнаяПозиция.Сумма`. Перед именем вычислимого атрибута ставится *слэш*.

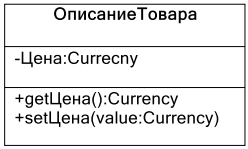


Рис. 87. Реализация свойства при помощи скрытого поля и двух методов доступа

Направление навигации

Стрелки на концах ассоциаций означают, что навигация осуществляется только в указанном направлении. Другими словами, видимость в направлении стрелки есть, а в противоположном направлении она отсутствует. Если направление навигации не указано, то подразумевается навигация в обе стороны.

Зависимости

Зависимости используются для отражения способов видимости, отличных от видимости посредством ассоциаций. Например, товарная позиция вызывает методы каталога товара (зависимость «call»), доступ к которому она получает через продажу и систему розничной торговли.

Вопросы для самоконтроля

1. Какие видимости членов класса бывают? В чем разница при использовании разных видов видимости?
2. Что такое параметризованный класс?
3. Как определить состав операций класса?

Задания для самостоятельной работы

1. Постройте концептуальную модель диаграммы классов.
2. Постройте диаграммы классов для учебного задания.

Рекомендации по построению диаграмм классов

При построении диаграмм классов придерживайтесь тех же правил, что и при построении диаграмм понятий. Дополнительно можно посоветовать воспользоваться CRC-карточками (Class Responsibility Card – карточка ответственности класса).

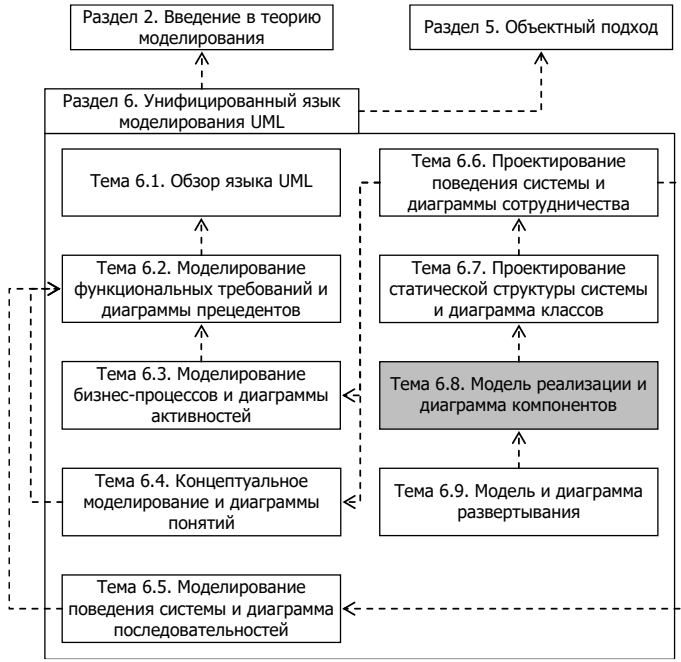
Подход заключается в следующем. Необходимо взять несколько небольших кусочков бумаги – по одному на каждый класс. Подходящий размер определите сами, но бумажки должны быть действительно небольшими, например, размером с один или два спичечных коробка. Далее для каждого класса на такой бумажке следует написать его обязанности, при этом нельзя мельчить и использовать излишние сокращения.

Если Вы затрудняетесь сформулировать обязанности некоторого класса, то стоит задуматься о его необходимости.

Если же Вам не хватает листочка, то следует рассмотреть вариант разбиения соответствующего класса на несколько более простых.

Тема 6.8. Модель реализации и диаграмма компонентов

Тематический контекст



Краткое содержание

1. Модель реализации.
2. Диаграмма компонентов. Основные элементы диаграммы компонентов: компоненты, интерфейсы, зависимости.
3. Использование стереотипов.

Модель реализации

В предыдущих разделах были рассмотрены модели анализа и проектирования, которые призваны ответить на вопросы «Что делать?» и «Как делать?» соответственно. Модель реализации завершает описание системы и акцентирует внимание на ее физической структуре. Под физической структурой в данном случае понимается как набор программных компонентов, так и аппаратные средства, на которых они размещаются и исполняются.

Для построения моделей реализации в UML используется два типа диаграмм – *диаграммы компонентов* и *диаграммы развертывания*. Рассмотрим диаграмму компонентов.

Диаграмма компонентов

Диаграмма компонентов представляет собой граф компонентов и зависимостей между ними. Она служит для отображения зависимостей между программными модулями, включая компоненты исходного или выполняемого кода. Пример диаграммы компонентов приведен на рис. 88.

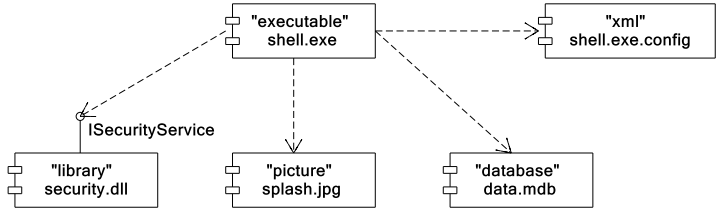


Рис. 88. Пример диаграммы компонентов

Компоненты

Основным элементом диаграммы компонентов является компонент, для изображения которого используется прямоугольник с двумя небольшими накладками с левой стороны (рис. 89).

Рис. 89. Пример компонента

Стереотипы

Как можно заметить из рис. 88, компоненты бывают различных типов. Среди наиболее часто используемых можно перечислить исполняемые файлы, изображения, библиотеки, базы данных, конфигурационные файлы, файлы помощи, исходные коды и пр. Для того чтобы различать компоненты, на диаграмме используют стереотипы, которые подписываются в кавычках над именем компонента (рис. 90).

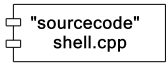


Рис. 90. Пример использования стереотипа

Пиктограммы

Для повышения наглядности диаграммы в стандарте UML предусмотрена возможность связать с каждым стереотипом некоторую пиктограмму, которая более точно, по сравнения со стандартной нотацией, отражает его суть (рис. 91). Однако не стоит забывать о том, что основное достоинство UML – это его стандартная нотация, понятная большинству разработчиков и аналитиков. Поэтому если принимается решение о введении нестандартного обозначения для некоторого стереотипа, то необходимо позаботиться о том, чтобы оно было максимально простым и понятным, находилось в рамках принятых в UML соглашений и не выбивалось и общего визуального ряда.

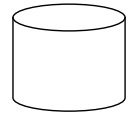


Рис. 91. Пример нестандартной нотации для компонента со стереотипом «database»

Интерфейсы

Если компонент представляет собой некоторый программный модуль, то в большинстве случаев он будет реализовывать один или несколько интерфейсов. Наличие этих интерфейсов является необходимым условием того, чтобы компонент можно было использовать в другом контексте или, при необходимости, заменить другим. Интерфейс на диаграмме компонентов обозначается так же, как это принято в COM-технологии – при помощи отрезка с окружностью на конце (рис. 92).

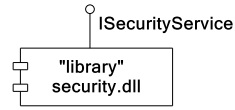


Рис. 92. Пример компонента, который реализует интерфейс

Зависимости

Одной из важнейших целей построения диаграмм компонентов является выделение и визуализация зависимостей между частями системы. Если один компонент потребляет услуги (например, библиотека классов) или ресурсы (например, изоб-

ражение) другого компонента, то он зависит от него. Зависимость между компонентами изображается при помощи пунктирной стрелки, направленной от зависимого компонента к независимому (рис. 93).

Рис. 93. Пример зависимости между компонентами

Компоненты, связанные отношением зависимости, должны находиться «близко» друг к другу. Степень «близости» (одна папка, один компьютер, локальная сеть) определяется требованиями конкретной ситуации. Если не отслеживать зависимости между компонентами, то высока вероятность того, что в процессе разработки система превратится в груду файлов, про многие из которых сложно будет сказать, нужны они или нет. Это усложняет разработку и приводит к засорению компьютеров конечных пользователей.

Вопросы для самоконтроля

1. Что такое компонент?
2. Как компонент связан с классами и интерфейсами?
3. Какие преимущества дает использование стереотипов на диаграмме компонентов?

Задания для самостоятельной работы

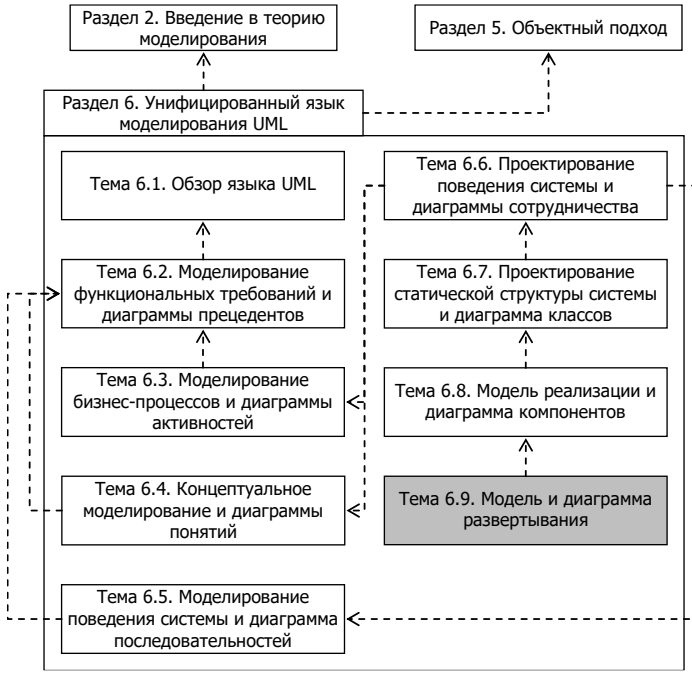
1. Постройте концептуальную модель диаграммы компонентов.
2. Постройте диаграмму компонентов для учебного задания.

Рекомендации по построению диаграммы компонентов

Одно из важнейших свойств компонента – его относительная независимость. Если на Вашей диаграмме присутствует много зависимостей, то это признак того, что компоненты спроектированы неправильно. Попробуйте перегруппировать классы так, чтобы большинство взаимодействий между ними происходило внутри компонентов. Однако не следует все классы объединять в один компонент – в этом случае Вы потеряете гибкость системы.

Тема 6.9. Модель и диаграмма развертывания

Тематический конспект



Краткое содержание

1. Модель развертывания.
2. Диаграмма развертывания.
3. Узлы. Разделение узлов на процессоры и устройства.
4. Использование стереотипов.

Модель развертывания

Для функционирования любой программной системы требуется аппаратное обеспечение. В случае информационных систем оно редко ограничивается одним компьютером и обычно представляет собой сложную систему аппаратных средств. Такая система может включать в себя клиентские компьютеры, компьютерные терминалы, серверы различного назначения, средства коммуникации, системы хранения данных, специализированные устройства (например, сканер штрих-кодов или датчик подачи топлива), сетевые принтеры и т.д. Каждое из этих

устройств обладает рядом характеристик (таких как тактовая частота, объем памяти, пропускная способность и т.д.), на значения которых разрабатываемая информационная система накладывает определенные ограничения. Все эти аспекты, безусловно, нуждаются в описании и являются составляющими модели развертывания системы.

Определение. *Модель развертывания* – это описание состава, характеристик и топологии аппаратных средств, а также распределения компонентов системы между ними.

Диаграмма развертывания

Для визуализации модели развертывания в UML используется диаграмма развертывания. Пример диаграммы развертывания приведен на рис. 94.

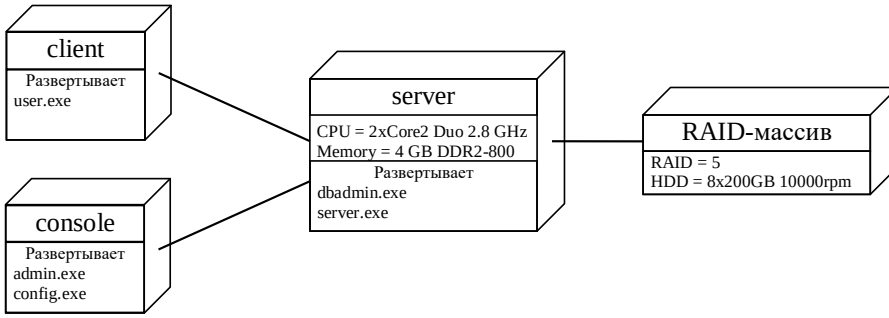


Рис. 94. Пример диаграммы развертывания

Узлы

Основным элементом диаграммы развертывания является *узел*, который изображается в виде куба. Дадим определение узла.

Определение. *Узел* – это элемент аппаратного комплекса, который представляет собой вычислительный ресурс, обычно обладающий как минимум некоторым объемом памяти. Совокупность компонентов может размещаться на узле, а также мигрировать с одного узла на другой.

Узлы обычно разделяют на *процессоры* и *устройства*. Разница между ними заключается в том, что процессоры обрабатывают информацию, например, сервер или аппаратная система шифрования, а устройства предоставляют услуги, например передача или хранение данных.

Стереотипы

Так же как и на диаграмме компонентов, на диаграмме развертывания часто используются стереотипы, для некоторых из которых вводится нестандартная нотация. При расширении диаграмм развертывания следует придерживаться тех же рекомендаций, что и для диаграмм компонентов. На рис. 95 приведен пример использования стереотипов, в котором для узлов «`Internet`» и «`Модемный пул`» были использованы нестандартные обозначения.

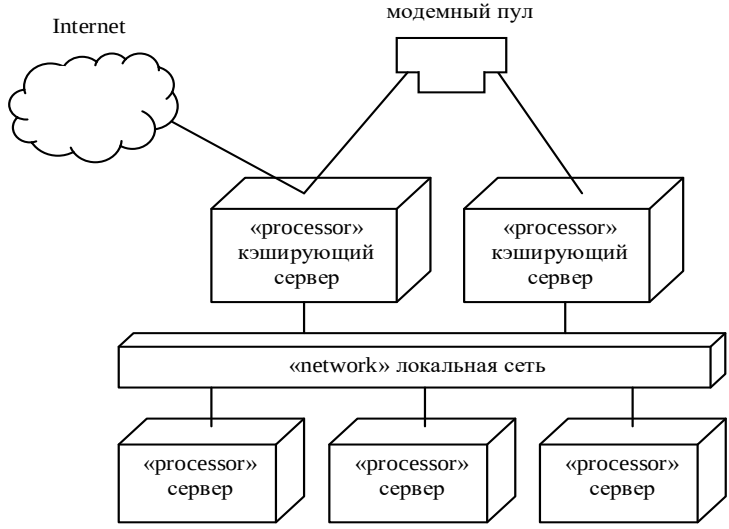


Рис. 95. Пример использования стереотипов на диаграмме развертывания

Вопросы для самоконтроля

1. Что такое узел?
2. В чем разница и какая связь между узлами и компонентами?
3. Чем отличается процессор от устройства? Приведите примеры процессоров и устройств.

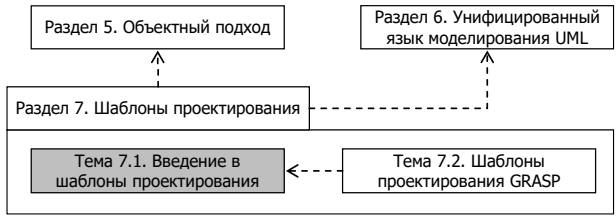
Задания для самостоятельной работы

1. Постройте концептуальную модель диаграммы развертывания.
2. Постройте диаграмму развертывания для учебного задания (задание приведено в конце пособия).

Раздел 7. Шаблоны проектирования

Тема 7.1. Введение в шаблоны проектирования

Тематический контекст



Краткое содержание

1. Обязанности, классификация обязанностей.
2. Понятие и назначение шаблонов проектирования.
3. Стандартный вид описания шаблона проектирования.

Обязанности

Занимаясь объектно-ориентированным проектированием, и в частности построением диаграмм взаимодействия, разработчик на самом деле занимаемся распределением обязанностей между объектами системы. Все обязанности можно разделить на *действия* и *знания*. К первой группе обязанностей относятся операции, *за выполнение которых отвечает объект*. Например, объектам класса «`Продажа`» может быть вменено в обязанность создавать товарные позиции. К обязанностям из группы действий относятся следующие виды обязанностей:

– выполнение некоторых действий самим объектом;
– инициация действий других объектов;
– управление и координирование действий других объектов.

Ко второй группе обязанностей относятся обязанности, *связанные с наличием некоторого рода информации*. Например, объектам класса «`Продажа`» может быть вменено в обязанность знание даты продажи. К обязанностям из группы знаний относятся следующие виды обязанностей:

– наличие информации о закрытых инкапсулированных данных;
– наличие информации о связанных объектах;

– наличие информации о вычислимых величинах.

Заметим, что один объект может выполнять несколько обязанностей. И наоборот, одну обязанность могу реализовывать несколько объектов. Например, за реализацию возможности отправки электронной почты, скорее всего, будет отвечать целая группа объектов.

Диаграммы взаимодействий отражают распределение обязанностей между объектами. Если на диаграмме объект (или класс) получает сообщение, то ему вменяется в обязанность его обработать. Кроме того, отправителю сообщения вменяется в обязанность видеть получателя, т.е. распределение обязанностей определяется взаимодействием (поток сообщений) и сотрудничеством (связи между объектами).

Человеческий опыт распределения обязанностей сконцентрирован в шаблонах проектирования (Design Patterns). Шаблоны не содержат новых идей, они систематизируют и формализуют существующие знания, принципы и удачные решения в области проектирования. Практически все шаблоны преследуют цель повышения гибкости, расширяемости системы.

По сути, шаблон проектирования – это обобщенное описание некоторой часто встречающейся задачи вместе с рекомендуемым способом ее решения, который был уже не единожды опробован на практике и доказал свою эффективность.

Шаблоны оформляются в виде структурированного описания, которое обычно включает следующие разделы:

– название шаблона;
– описание проблемы или задачи;
– описание решения;
– пример использования;
– рекомендации по использованию;
– описание достоинств и недостатков.

Стоит заметить, что у шаблонов проектирования нет недостатков в традиционном понимании этого слова. Скорее это цена, которую приходится платить за полезный эффект, получаемый от использования шаблона.

Вопросы для самоконтроля

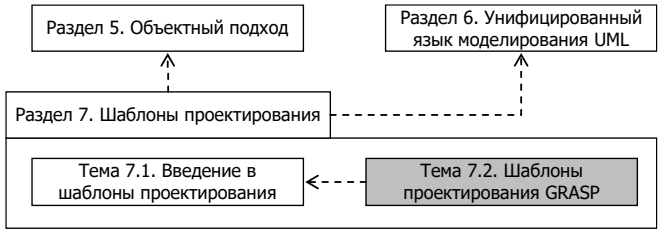
1. Что такое шаблон проектирования и зачем он нужен?
2. Какие виды обязанностей Вы знаете?
3. Как выглядит описание шаблона проектирования?

Дополнительная информация

В следующей теме рассматриваются общие шаблоны распределения обязанностей (GRASP). Для дальнейшего изучения шаблонов проектирования можно порекомендовать обратиться к книге Эриха Гаммы и др. «Примеры объектно-ориентированного проектирования. Паттерны проектирования».

Тема 7.2. Шаблоны проектирования GRASP

Тематический контекст



Краткое содержание

1. Обзор общих шаблонов распределения обязанностей (GRASP).
2. Шаблоны: «Эксперт», «Создатель», «Низкое связывание», «Высокое зацепление» и «Контроллер».

Шаблоны GRAPS

В данном разделе рассматриваются шаблоны GRASP (General Responsibility Assignment Patterns – Общие шаблоны распределения обязанностей). Это наиболее общие и часто (порой, неосознанно) используемые шаблоны проектирования. Некоторые из них напрямую следуют из принципов объектно-ориентированного подхода и могут показаться очень простыми и интуитивно понятными. Однако не стоит их недооценивать. Осознанное применение этих шаблонов поможет избежать многих проблем.

Шаблон Expert (Эксперт)

Проблема. Каков наиболее общий принцип распределения обязанностей между объектами при объектно-ориентированном проектировании?

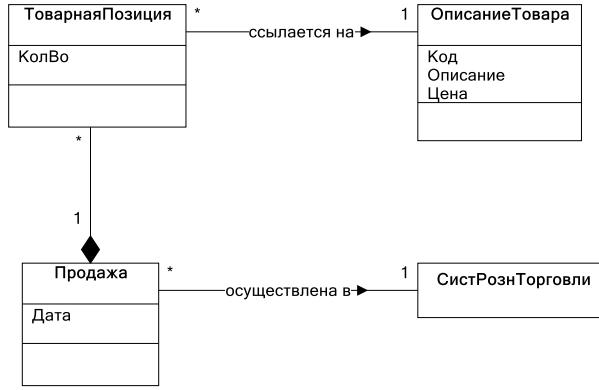


Рис. 96. Концептуальная модель продажи

Решение. Назначить обязанность информационному эксперту – объекту, который имеет информацию, необходимую для выполнения обязанности.

Например, в системе розничной торговли некоторому объекту необходимо знать общую сумму продажи. На рис. 96 представлена часть соответствующей концептуальной модели.

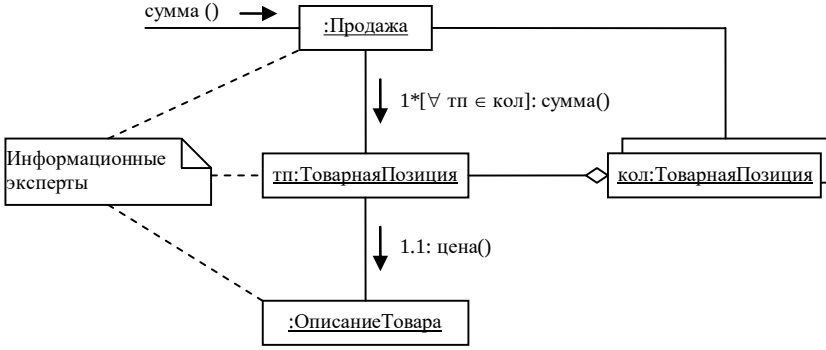


Рис. 97. Распределение обязанностей в соответствии с шаблоном Expert

Распределение обязанностей в соответствии с шаблонном Expert показано на рис. 97. Для выполнения обязанности «знать и предоставлять общую сумму продажи» объектам системы

были назначены следующие обязанности: объектам класса «Продажа» было вменено в обязанность вычисление общей суммы продажи и владение коллекцией товарных позиций; объектам класса «ТоварнаяПозиция» было вменено в обязанность вычисление суммы товарной позиции, знание количества продаваемого товара и его описания; объектам класса «ОписаниеТовара» было вменено в обязанность знание цены товара.

Заметим, что на рис. 97 классы «Продажа», «ТоварнаяПозиция» и «ОписаниеТовара» появились не столько потому, что соответствующие понятия есть на концептуальной модели, сколько потому, что эти классы понадобились для реализации обязанности. Модель спецификации и тем более модель реализации могут существенно отличаться от концептуальной модели.

Шаблон Expert является наиболее часто используемым шаблоном. Этот шаблон поддерживает инкапсуляцию, т.к. используются собственные данные. Кроме того, его применение приводит к определению классов, которые гораздо легче понимать и поддерживать.

Шаблон Creator (Создатель)

Проблема. Кто должен отвечать за создание нового экземпляра некоторого класса?

Решение. Назначить классу А обязанность создавать экземпляры класса Б, если:

- А агрегирует объекты Б;
- А содержит объекты Б;
- А записывает объекты Б;
- А активно использует объекты Б;
- А обладает данными инициализации, которые передаются объектам Б при создании (т.е. А является экспертом).

Если выполняется несколько условий (т.е. есть несколько кандидатов на роль создателя), то предпочтение следует отдать первым двум, т.е. агрегирует и содержит.

Например, кто в системе розничной торговли должен отвечать за создание товарных позиций? Продажа агрегирует товарные позиции и, следовательно, является кандидатом на создателя (рис. 98).

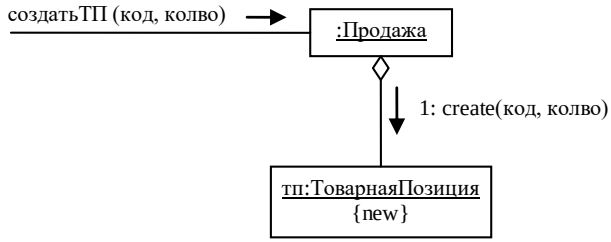


Рис. 98. Создание товарных позиций

При использовании шаблона *Creator* степень связности не повышается (см. шаблон *Low Coupling*), т.к. создаваемый объект, как правило, оказывается видимым для создателя посредством уже имеющихся ассоциаций.

Шаблон Low Coupling (Низкое Связывание)

Дадим определение степени связности (coupling).

Определение. *Степень связности* – это мера того, насколько жестко один класс связан с другими классами; иными словами, каким количеством информации о них он обладает.

Проблема. Наличие классов с высокой степенью связности нежелательно, т.к. приводит к следующим проблемам:

- изменения в связанных классах обусловливают изменениям в данном классе;
- затрудняется понимание каждого класса в отдельности;
- усложняется повторное использование.

Решение. Распределять обязанности таким образом, чтобы степень связности оставалась низкой.

Например, в системе розничной торговли необходимо создать платеж и связать его с продажей. Кто этим должен заниматься? На рис. 99 представлены два варианта распределения обязанностей для этой задачи. Второй вариант является более предпочтительным, т.к. в нем на одну связь меньше. Кроме того, второй вариант согласуется с шаблоном *Creator*, т.к. продажа обладает данными инициализации платежа.

Крайним случаем применения данного шаблона является полное отсутствие связывания. Это тоже плохо, т.к. базовой идеей объектно-ориентированного подхода является система связанных объектов, взаимодействующих при помощи сообще-

ний. Иначе система сводится к нескольким изолированным сложным активным объектам, которые выполняют все действия, и множеству пассивных объектов, основным назначением которых является хранение данных.

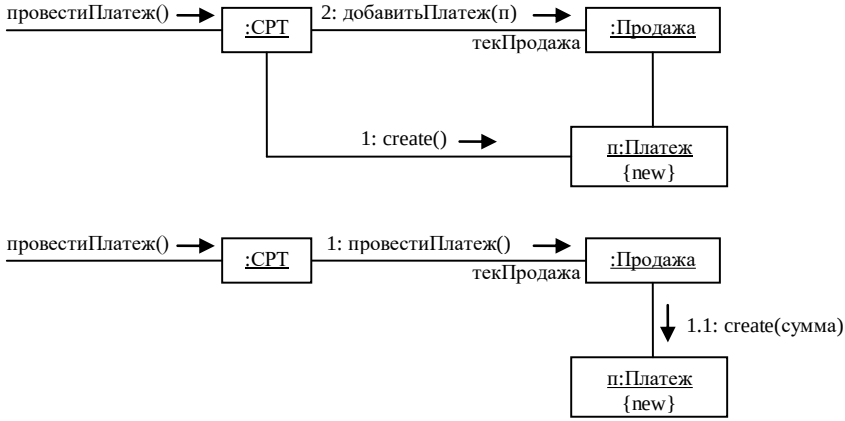


Рис. 99. Два варианта распределения обязанностей при создании платежа

Применение данного шаблона приводит к локализации изменений, простоте понимания и легкости повторного использования классов.

Шаблон High Cohesion (Высокое Зацепление)

Дадим определение зацеплению (cohesion).

Определение. *Зацепление* (а точнее, функциональное зацепление) – это мера связанности и сфокусированности обязанностей класса. Класс обладает высокой степенью зацепления, если его обязанности тесно связаны между собой и он не выполняет работ непомерных размеров.

Проблема. Наличие классов с низкой степенью зацепления, которые выполняют много разнородных обязанностей, приводит к следующим проблемам:

- трудность в понимании;
- сложности при повторном использовании;
- сложности при поддержке;
- постоянная подверженность изменениям.

Решение. Распределять обязанности таким образом, чтобы поддерживалась высокая степень зацепления.

Рассмотрим применение данного шаблона на том же примере, что и шаблон *Low Coupling*. В первом случае на рис. 99 экземпляр системы розничной торговли (`:СРТ`) частично отвечает за проведение платежа. В данном случае это не очень плохо, но если дальше возлагать на него обязанности, то класс будет перегружен, и будет обладать низкой степенью зацепления. Во втором случае обязанность делегирована продаже, что предпочтительнее.

Использование шаблона Low Coupling приводит к легкости в понимании классов и в их поддержке и повторном использовании.

Шаблон Controller (Контроллер)

Проблема. Кто должен отвечать за обработку системных сообщений?

Решение. Делегировать обязанности по обработке системных сообщений классу, удовлетворяющему одному из следующих условий:

- класс представляет всю систему в целом (внешний контроллер), например «`СРТ`».
- класс представляет всю организацию (внешний контроллер), например «`Магазин`».
- класс представляет активный объект из реального мира (котроллер роли), например «`Кассир`».
- класс представляет искусственный обработчик всех системных сообщений в рамках некоторого прецедента (контроллер прецедента), например «`ПродажаТовара`».

В любом случае для всех системных сообщений в рамках одного прецедента должен использоваться один контроллер.

Заметим, что в указанный выше перечень не включены объекты интерфейса пользователя. Обработка системных сообщений на уровне интерфейса пользователя является довольно распространенной ошибкой проектирования, которая приводит к серьезным проблемам при поддержке и повторном использовании. На рис. 100 представлен пример использования шаблона *Controller*.

Применение шаблона Controller позволяет контролировать состояние прецедента, т.к. все системные сообщения прецедента обрабатываются в одном месте. Кроме того, система получается независимой от уровня представления, что позволяет легко изменять интерфейс пользователя и поддерживать сразу несколько интерфейсов (например, *WinForms*, Web-интерфейс, интерфейс для мобильных устройств и т.д.).

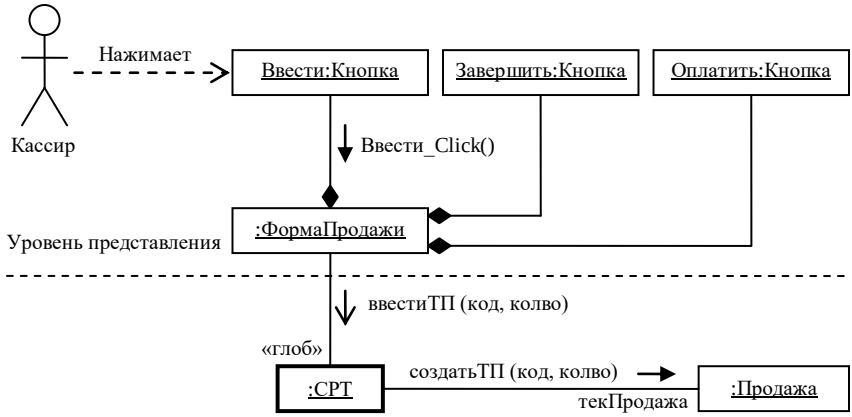


Рис. 100. Использование шаблона Controller

На практике рекомендуется использовать контроллеры прецедентов, а если системных сообщений мало, то любой внешний контроллер.

Вопросы для самоконтроля

1. Опишите каждый из шаблонов.
2. Какой шаблон является наиболее общим?
3. Что такое связывание и зацепление?
4. Какие бывают контроллеры и зачем они нужны?

Задания для самостоятельной работы

Переделайте Ваши диаграммы сотрудничества и диаграммы классов с учетом рекомендаций, описанных выше в шаблонах GRASP.

Учебное задание

Информационная система для аптеки.

У аптеки имеется несколько поставщиков, поставляющих лекарства, различных производителей. Каждое лекарство относится к некоторой категории (от насморка, болеутоляющее, от аллергии и пр.). Лекарства хранятся в различных шкафах и на полках, в зависимости от условий реализации (требуется рецепт, спец. рецепт) и условий хранения. Полки могут находиться в шкафах.

Перед разрабатываемой системой ставятся следующие основные задачи:

1. Оперативный учет остатков товаров.
2. Анализ спроса.
3. Формирование отчетов о приходе/расходе товара.

Примерный перечень вопросов к зачету по всему курсу

1. Жизненный цикл информационной системы. Понятие жизненного цикла, процессы жизненного цикла, его виды.
2. Структурный и объектный подходы к разработке информационных систем. Суть подходов, различия между ними, достоинства и недостатки.
3. Моделирование. Понятие и назначение моделирования, понятие модели, классификация моделей.
4. Метамоделирование. Классификация метамоделей. Классификация информационных систем по количеству, уровню и способу использования моделей.
5. Модель прецедентов. Цель и принцип построения модели прецедентов, понятия прецедента и актора, диаграмма прецедентов.
6. Модель бизнес-процессов. Цель и принцип построения модели бизнес-процессов, понятие активности, диаграмма активностей.
7. Концептуальная модель. Цель и принцип построения концептуальной модели, диаграмма понятий (понятия,

атрибуты, ассоциации, роли).

8. Концептуальная модель. Цель и принцип построения концептуальной модели, диаграмма понятий (обобщение, правило is_a, правило 100%, агрегация).
9. Диаграммы последовательностей. Цель и принцип построения диаграмм последовательностей, понятия системного сообщения и системной операции, связь между ними.
10. Диаграммы сотрудничества. Цель и принцип построения диаграмм сотрудничества, понятия сотрудничества и взаимодействия, связь между ними.
11. Шаблоны проектирования. Шаблоны GRASP. Примеры.
12. Диаграммы классов. Цель и принцип построения диаграмм классов.
13. Модель реализации. Понятие компонента, диаграмма компонентов.
14. Модель развертывания. Понятие узла, диаграмма развертывания.

Вопрос для итоговой аттестации

Понятие модели информационной системы. Статическая, динамическая и функциональная модели информационной системы; связь между ними; относительная важность. Концептуальная модель, модель спецификации и модель реализации; различия в интерпретации. Классификация Буча. Понятие метамодели. Язык UML. Краткий обзор диаграмм языка UML.

III. ПРИМЕРНОЕ РАСПРЕДЕЛЕНИЕ ЧАСОВ КУРСА ПО ФОРМАМ И ВИДАМ РАБОТ

№ п/п	Наименование разделов и тем	Всего часов	Аудиторные занятия		Самостоятельная работа
			лекционные	лабораторные	
Раздел 1	Введение	**2**	**2**	**0**	**0**
Тема 1.1	Понятие информационной системы	1	1	0	0
Тема 1.2	Проблемы сложных задач	1	1	0	0
Раздел 2	Введение в теорию моделирования	**8**	**5**	**0**	**3**
Тема 2.1	Понятие моделирования и модели. Принципы моделирования и классификация моделей	3	2	0	1
Тема 2.2	Метамоделирование	3	2	0	1
Тема 2.3	Классификация информационных систем по уровню и составу моделей	2	1	0	1
Раздел 3	Жизненный цикл программного обеспечения	**4**	**2**	**0**	**2**
Тема 3.1	Понятие жизненного цикла. Процессы жизненного цикла	2	1	0	1
Тема 3.2	Модели жизненного цикла.	2	1	0	1

№ п/п	Наименование разделов и тем	Всего часов	Аудиторные занятия		Самостоятельная работа
			лекционные	лабораторные	
Раздел 4	Структурный подход к разработке информационных систем	**8**	**4**	**0**	**4**
Тема 4.1	Сущность и основные принципы структурного подхода	2	1	0	1
Тема 4.2	Метод функционального моделирования SADT	2	1	0	1
Тема 4.3	Моделирование потоков данных	2	1	0	1
Тема 4.4	Моделирование структур данных	2	1	0	1
Раздел 5	Объектно-ориентированный подход к разработке информационных систем	**4**	**2**	**0**	**2**
Тема 5.1	Сущность и основные принципы объектно-ориентированного подхода	2	1	0	1
Тема 5.2	Пример объектно-ориентированного анализа и проектирования	2	1	0	1
Раздел 6	Унифицированный язык моделирования UML	**46**	**18**	**0**	**28**
Тема 6.1	Обзор языка UML	4	2	0	2
Тема 6.2	Моделирование функциональных требований и диаграмма прецедентов	6	2	0	4

№ п/п	Наименование разделов и тем	Всего часов	Аудиторные занятия		Самостоятельная работа
			лекционные	лабораторные	
Тема 6.3	Моделирование бизнес-процессов и диаграмма активностей	4	2	0	2
Тема 6.4	Концептуальное моделирование и диаграмма понятий	10	4	0	6
Тема 6.5	Моделирование поведения системы и диаграмма последовательностей	4	2	0	2
Тема 6.6	Проектирование поведения системы и диаграмма сотрудничества	6	2	0	4
Тема 6.7	Проектирование статической структуры системы и диаграмма классов	6	2	0	4
Тема 6.8	Модель реализации и диаграмма компонентов	3	1	0	2
Тема 6.9	Модель и диаграмма развертывания	3	1	0	2
Раздел 7	Шаблоны проектирования	**9**	**3**	**0**	**6**
Тема 7.1	Введение в шаблоны проектирования	3	1	0	2
Тема 7.2	Шаблоны проектирования GRASP	6	2	0	3
	Итого:	**80**	**36**	**0**	**44**

IV. ФОРМА ИТОГОВОГО КОНТРОЛЯ

Изучение курса завершается зачетом, выставляемым по результатам выполнения практических заданий, и ответа на теоретический вопрос.

V. УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ КУРСА

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА (ОБЯЗАТЕЛЬНАЯ)

1. *Буч Г.* Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, А. Джекобсон: пер. с англ. М.: ДМК Пресс; СПб.: Питер, 2004. (Сер. «Объектно-ориентированные технологии в программировании»).
2. *Вендров А.М*. Проектирование программного обеспечения экономических информационных систем / А.М. Вендров. М.: Финансы и статистика, 2000.
3. *Калянов Г.Н*. Современные CASE-технологии / Г.Н. Калянов. М.: ИПУ, 1992.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА (ДОПОЛНИТЕЛЬНАЯ)

1. *Боэм Б*. Инженерные проектирование программного обеспечения / Б. Боэм. М.: Радио и связь, 1985.
2. *Гантер Р*. Методы управления проектированием программного обеспечения / Р. Гантер. М.:Мир, 1981.
3. *Гласс. Р.* Сопровождение программного обеспечения / Р. Гласс, Р. Нуазо. М.: Мир, 1983.
4. *Зиглер К*. Методы проектирования программных систем / К. Зиглер. М.: Мир, 1985.
5. *Золотухина Е.Б*. Пример описания предметной области с использованием Unified Modeling Language (UML) при разработке программных систем // Е.Б. Золотухина, Р.В. Алфимов. Interface Ltd., 2001.
6. *Калянов Г.Н*. CASE структурный системный анализ (ав-

томатизация и применение) / Г.Н. Калянов. М.: Изд-во «Лори», 1996.
7. *Калянов Г.Н.* Методы и средства системного структурного анализа и проектирования / Г.Н. Калянов. М.: НИВЦ МГУ, 1996.
8. *Кватрани Т.* Rational Rose 2000 и UML. Визуальное моделирование / Т. Кватрани. Изд-во ДМК, 2001.
9. *Кумсков М.* Унифицированный язык моделирования (UML) и его поддержка в Rational Rose 98i – CASE-средстве визуального моделирования / М. Кумсков [Электронный ресурс] (www.interface.ru/public/990804/uml4b.htm).
10. *Ларман К.* Применение UML и шаблонов проектирования.: пер. с англ.: учеб. пособие / К. Ларман. М.: Изд. дом «Вильямс», 2001.
11. *Липаев В.В.* Управление разработкой программных комплексов / В.В. Липаев. М.: Финансы и статистика, 1993.
12. *Лядова Л.Н.* Основы информатики и информационных технологий / Л.Н. Лядова, Б.И. Мызникова, Н.В. Фролова. Пермь: Перм. ун-т, 2004.
13. *Маклаков С.В.* BPwin и ERwin. CASE-средства разработки информационных систем / С.В. Маклаков. М.: ДИАЛОГ-МИФИ, 2000.
14. *Фаулер М.* UML в кратком изложении. Применение стандартного языка объектного моделирования / М. Фаулер, К. Скотт. М.: Мир, 1999.
15. Фергюсон Я. Язык моделирования UML – новая веха в истории Rational / Я. Фергюсон // Computerworld Today, Австралия, 27.11.1997, Computerworld, #44/1997
16. *Фокс Д.* Программное обеспечение и его разработка / Д. Фокс. М.: Мир, 1985.
17. *Штрик А.А.* CASE: машинное проектирование программного обеспечения / А.А. Штрик. М.: МЦИЭ ИнтерЭВМ, 1990.
18. *Юдицкий С.А.* Технология проектирования архитектуры информационно-управляющих систем / С.А. Юдицкий, А.Т. Кутанов. М.: ИПУ, 1993.
19. UML – новый стандарт языка объектно-

ориентированного моделирования. Квинтэссенция успешного опыта // SoftScribe International [Электронный ресурс] (http://www.ci.ru/inform22_97/sscrb.htm).
20. UML. Метаязык проектирования и моделирования программного обеспечения [Электронный ресурс] (http://it.metod.ru/reviews/UML).

Список адресов в Интернет

1. www.interface.ru
2. www.citforum.ru
3. www.rif.ru
4. www.osp.ru
5. www.uml.ru
6. www.uml.org
7. www.omg.org
8. Материалы сайтов фирм-производителей CASE-средств.

Учебное издание

Шаврин Сергей Михайлович

Лядова Людмила Николаевна

Чуприна Светлана Игоревна

Моделирование и проектирование информационных систем

Учебно-методическое пособие

Редактор *Л.А. Богданова*

Корректор *Л.И. Семицветова*

Подписано в печать 15.12.2007. Формат 60×84/16.
Усл. печ. л. 8,83. Уч.-изд. л. 6,5.
Тираж 100 экз.
Заказ

Редакционно-издательский отдел Пермского государственного университета
614990. Пермь, ул. Букирева, 15

Типография Пермского государственного университета
614990. Пермь, ул. Букирева, 15