МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Озерский технологический институт — филиал НИЯУ МИФИ

Кафедра прикладной математики

Вл. Пономарев

Объектно-ориентированный анализ и проектирование

Учебно-методическое пособие

Озерск, 2019

УДК 681.3.06
П56

Вл. Пономарев. Объектно-ориентированный анализ и проектирование. Учебно-методическое пособие. Редакция 2019-05-20. Озерск: ОТИ НИЯУ МИФИ, 2019. — 26 с.

В пособии излагаются основы объектно-ориентированного анализа и проектирования и использование языка UML2.

В качестве вспомогательного материала пособие предназначено для студентов, обучающихся по направлению подготовки 09.03.01 «Информатика и вычислительная техника» и специальности 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения».

Рецензенты:
1.
2.

УТВЕРЖДЕНО
Редакционно-издательским
Советом ОТИ НИЯУ МИФИ

Содержание

Введение 4
1. Процесс проектирования программных систем 5
1.1. Понятие о проектировании 5
1.1. Процесс проектирования 6
2. Анализ 7
2.1. Цели анализа 7
2.2. Требования 7
2.3. Прецеденты 8
2.3.1. Идентификация актеров 9
2.3.2. Идентификация прецедентов 9
2.3.3. Детализация прецедентов 10
2.3.4. Сценарии 11
2.3.5. Обобщение актеров 12
2.4. Отношения прецедентов 13
2.4.1. Отношение включения 13
2.4.2. Отношение расширения 14
2.5. Классы анализа 15
2.5.1. Признаки хорошего класса анализа 15
2.5.2. Практические правила 16
2.6. Выявление классов анализа 16
2.6.1. Классический подход 17
2.6.2. Анализ существительное-глагол 17
2.6.3. CRC-анализ 18
2.6.4. Стереотипные классы 19
2.7. Отношения 20
2.7.1. Диаграмма объектов 20
2.7.2. Ассоциация 20
2.7.3. Классы-ассоциации 23
2.7.4. Материализованные отношения 24
2.7.5. Квалифицированные ассоциации 25
2.7.6. Зависимость 25
2.7.7. Обобщение 26

Введение

Эта предварительная версия документа находится в стадии редактирования и предназначена для студентов ОТИ НИЯУ МИФИ.

Текст пособия постоянно совершенствуется, поэтому рекомендуется использовать его электронную версию, которая периодически обновляется на сайте http://revol.ponocom.ru.

1. Процесс проектирования программных систем

1.1. Понятие о проектировании

Инженерное проектирование имеет целью разработку промышленного программного обеспечения. Характерные особенности такого ПО:

- сложность;
- долговременность использования;
- необходимость сопровождения при эксплуатации;

Сложность современных программных систем такова, что превышает интеллектуальные возможности одного человека. Вследствие этого необходимы инструменты и механизмы для создания проекта, работа над которым может выполняться различными программистами.

Под проектированием понимается формальный метод, при помощи которого находится решение определенной проблемы, что обеспечивает переход от технического задания к реализации. Цель проектирования можно определить как конструирование системы, которая удовлетворяет явным и неявным требованиям (Mostow):

- функциональной (возможно неформальной) спецификации;
- ограничениям среды;
- к производительности и потребляемым ресурсам;
- к форме продукции;
- к процессу разработки (продолжительность, стоимость, средства).

Страуструп отмечает: «Цель проектирования — создать ясную и относительно простую внутреннюю структуру, иногда называемую архитектурой».

Важнейшим принципом управления сложными системами является *декомпозиция*. Она разделяет систему на части, сложность которых значительно меньше.

Функциональная, или алгоритмическая декомпозиция соответствует принципу проектирования «сверху вниз», в котором каждый модуль системы выполняет один из этапов общего процесса. Эта декомпозиция основана на алгоритмах. ОО декомпозиция основана на объектах, составляющих словарь предметной области.

Преимущества объектной модели следующие.

1. Она позволяет эффективно использовать выразительные возможности объектных и объектно-ориентированных языков.
2. Объектный подход стимулирует повторное использование не только кода, но и проектных решений.
3. Она создает системы с устойчивыми промежуточными формами, что упрощает их изменение.
4. Она уменьшает риски проектирования сложных систем.

5. Она учитывает процесс познания, так как многие люди считают объектно-ориентированные системы естественными.

ОО программирование — метод программирования, основанный на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определенного класса, а классы являются членами определенной иерархии наследования.

ОО проектирование — метод проектирования, сочетающий процесс объектно-ориентированной декомпозиции и систему обозначений для представления логической и физической, а также статической и динамической моделей проектируемой системы.

ОО анализ — метод анализа, исследующий требования к системе с точки зрения классов и объектов, относящихся к словарю предметной области.

Результатами ОО анализа являются модели, лежащие в основе ОО проектирования, которое в свою очередь позволяет разработать схему полной реализации системы с использованием ОО программирования.

1.1. Процесс проектирования

Процесс проектирования ОО системы итеративный.

Сложную систему невозможно создать сразу. Сначала проблема решается приблизительно. По мере накопления опыта обнаруживается, что некоторые решения лучше. Они изучаются, программируются и анализируются. Это позволяет разрабатывать теории, обобщающие найденное решение. Можно выделить следующие шаги разработки ПО.

1. Концептуализация.
2. Анализ.
3. Проектирование.
4. Кодирование.
5. Тестирование.
6. Внедрение.

Концептуализация — это видение продукта, выражаемое предложением или абзацем. Например: «Мы создаем систему для обслуживания клиентов банка (банкомат, АТМ)».

Анализ — это процесс понимания требований к системе. Заключается в первичном исследовании задачи и моделировании реальности с помощью классов и объектов, образующих словарь предметной области.

Проектирование — это процесс создания модели классов, изобретение абстракций и механизмов реализации принятого решения.

Он должен отвечать на вопросы: кто? что? когда? как?

2. Анализ

2.1. Цели анализа

Целью анализа является уточнение требований к системе в виде диаграмм прецедентов и выявление классов анализа — создание аналитической модели системы.

Аналитическая модель создается на языке предметной области. Диаграммы анализа должны раскрывать некоторые важные части поведения системы, и отображать ее основную картину. Внимание следует направлять на абстракции предметной области. Аналитическая модель должна быть также полезной всем заинтересованным сторонам.

В ходе анализа должен быть создан *глоссарий* проекта. Глоссарий отображает бизнес-терминологию и жаргон. Глоссарий должен содержать синонимы и омонимы для каждого ключевого слова. Синонимы — это разные слова, обозначающие одно и то же. Омонимы — это слова, одинаковые по звучанию, но разные по значению.

2.2. Требования

Требования определяют, **что** должно быть разработано, но не **как**.

Требования следуют из контекста моделируемой системы, в который могут входить:

- пользователи системы;
- другие заинтересованные стороны: руководители, специалисты по обслуживанию, эксплуатации, продаже;
- другие системы, с которыми взаимодействует данная;
- аппаратные устройства, с которыми взаимодействует система;
- правовые и регулирующие ограничения;
- технические ограничения;
- коммерческие цели.

Выработка требований состоит в выявлении и классификации требований, предъявляемых к системе всеми заинтересованными сторонами. Это переговорный процесс, в котором стороны выступают в качестве экспертов, и цель которого — согласовать противоречивость требований разных сторон.

Способы сбора требований:

- интервью;
- анкетирование;
- семинар (обсуждение с заинтересованными сторонами).

Спецификация требований формирует модель требований и модель прецедентов. Общей формы представления требований нет, чаще всего используется бумага и ручка.

В модель требований входят функциональные требования, определяющие, что должна делать система, и нефункциональные требования, выражающие ограничения системы, не относящиеся к ее функциональности.

В качестве примере далее будем рассматривать банкомат (automated teller machine, ATM).

Функциональные требования.

ATM должен (shall):

1. проверять действительность карточки.
2. проверять достоверность PIN-кода.
3. выдавать по одной карточке не более 20000 в сутки.

Нефункциональные требования.

ATM должен (shall):

1. быть написан на C.
2. обмениваться информацией с банком в кодировке ANSI.
3. проверять действительность карточки не дольше 3 секунд.
4. проверять достоверность PIN-кода не дольше 3 секунд.

Требования могут содержать атрибуты:

- обязательное (must have);
- важное (should have);
- необязательное (could have);
- желательное (want to have).

Требования могут быть ранжированы по приоритетам.

Требования могут быть категоризированы по типам.

2.3. Прецеденты

Основным инструментом анализа является прецедент (Use Case, случай или вариант использования). Это другой способ выявления и документирования требований. Моделирование прецедентов обычно происходит следующим образом:

- устанавливаются границы потенциальной системы;
- выявляются актеры (actor);
- выявляются прецеденты;
- шаги повторяются до установления равновесия.

Результатом выявления прецедентов является построение модели прецедентов, состоящей из 4-х компонентов:

- граница системы (прямоугольник);
- актеры — роли, выполняемые людьми или другими системами;
- прецеденты — то, что актеры могут делать с системой;
- отношения — это значимые отношения между актерами и прецедентами.

Модель прецедентов является основным источником объектов и классов и исходными данными для моделирования классов.

Границы системы определяют, что является частью системы, а что находится вне ее. В UML ее называют контекстом системы (subject).

Актеры размещаются вне границ, а прецеденты — внутри.

2.3.1. Идентификация актеров

Актер определяет роль, которую выполняет некоторая внешняя сущность при непосредственном взаимодействии с системой.

Следующие вопросы помогут идентифицировать актеров.

Кто или что использует систему?

Какие роли они играют во взаимодействии?

Кто устанавливает систему?

Кто или что запускает и выключает систему?

Кто обслуживает систему?

Какие системы взаимодействуют с данной системой?

Кто или что получает и предоставляет информацию системе?

Заметим, что один человек или сущность могут играть разные роли по отношению к системе, одновременно, или последовательно. Каждому актеру присваивается короткое имя (существительное).

Анализируя банкомат, можно выявить роли:

- клиент;
- персонал;
- обслуживающая система (back-office);
- ответственный за пополнение банкомата.

2.3.2. Идентификация прецедентов

Прецедент описывает поведение системы с целью получения значимого результата для одного или более актеров. Прецедент всегда инициируется актером, и описывается с его точки зрения.

Для идентификации прецедентов рассматривают, как каждый актер использует систему. Полученный список прецедентов уточняется, при этом могут обнаружиться новые актеры. Каждому прецеденту присваивается короткое имя, представляющее собой глагольную группу.

Следующие вопросы помогут выявить прецеденты.

Как каждый из актеров использует систему?

Какой из актеров инициирует сохранение информации?

Какой из актеров получает уведомление об изменении состояния?

Какие внешние события влияют на систему?

Система взаимодействует с какой-либо внешней системой?

Система генерирует какие-либо отчеты?

Для каждого прецедента рисуется диаграмма. Актер изображается обычно как проволочный человечек, прецедент изображается овалом.

Рассматривая актера «Клиент» банкомата, можно выявить следующие прецеденты:

1. Проверяет баланс.
2. Кладет деньги на счет.
3. Снимает деньги со счета.
4. Переводит деньги между счетами.
5. Открывает счет.
6. Закрывает счет.

Пример прецедента приведен на следующем рисунке.

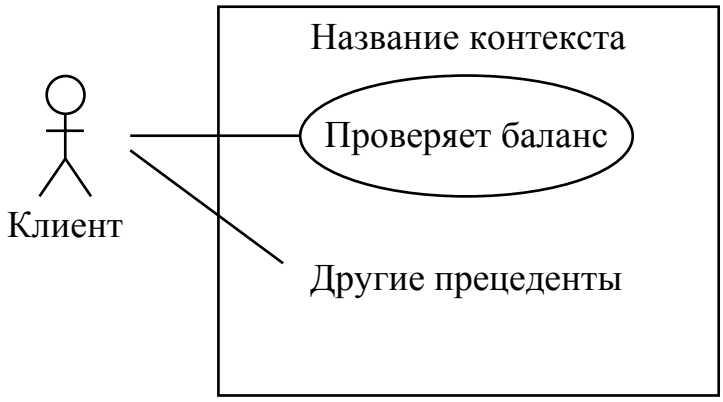


Рисунок 1 — Изображение прецедента

Исследование актеров и прецедентов может выявить дополнительные прецеденты. Например, задаваясь вопросом, что должен сделать клиент, чтобы начать пользоваться банкоматом, мы обнаружим, что он должен авторизоваться.

2.3.3. Детализация прецедентов

Каждый прецедент должен быть детально описан. Единой спецификации для этого не предусмотрено, рекомендуемые элементы описания следующие.

1. Название.
2. Идентификатор.
3. Краткое описание.
4. Главные актеры.
5. Второстепенные актеры.
6. Сценарии.
7. Предусловия.
8. Постусловия.
9. Другие необходимые сведения.

Название прецедента оформляется как имя функции. Идентификатор — это целое число. Краткое описание — это максимум один абзац.

Главные актеры те, которые инициируют прецедент. Второстепенные актеры взаимодействуют с прецедентом после его инициации.

Предусловия и постусловия — это ограничения. Предусловия указывают состояние системы, необходимое для запуска прецедента. Постусловия указывают состояние системы после выполнения прецедента.

2.3.4. Сценарии

Сценарии (или потоки) — это возможные варианты выполнения прецедента, потока его событий. Сценарии могут быть основными и альтернативными.

Основной сценарий описывает последовательность действий в случае идеальной ситуации, когда не возникает ошибок, отклонений, прерываний и ответвлений. Отклонения от основного потока моделируются как простые и сложные. Простые отклонения разветвляют основной сценарий. Сложные отклонения создают альтернативные потоки.

Основной сценарий начинается с действий главного актера, которые направлены на инициацию. Поток событий состоит из последовательности коротких декларативных этапов, пронумерованных и упорядоченных во времени.

Примеры сценариев прецедента «Снимает деньги со счета».

Основной сценарий:

1. Клиент запрашивает выдачу 1000 р.
2. Система кладет деньги в слот выдачи.
3. Система печатает квитанцию.

Альтернативный сценарий 1:

1. Клиент запрашивает выдачу 1000 р.
2. Система уведомляет клиента, что баланс меньше 1000 р.

Альтернативный сценарий 2:

1. Клиент запрашивает выдачу 1000 рублей.
2. Система уведомляет клиента, что лимит составляет 1000 р./день.

Для простых ответвлений в сценариях можно использовать ключевое слово «Если», например:

Основной сценарий:

1. Клиент запрашивает выдачу 1000 р.
2. Если на счете меньше 1000 р.:
    2.1. Система уведомляет клиента, что баланс меньше 1000 р.
3. Если на счете не меньше 1000 р.:
    3.1. Система кладет деньги в слот выдачи.
    3.2. Система печатает квитанцию.

Для повторений (итераций) сценарий может использовать ключевые слова «Для» и «Пока».

Один альтернативный сценарий записывается непосредственно в описании прецедента. Если альтернативных сценариев несколько, они именуются и описываются отдельно. При этом к идентификатору прецедента добавляется номер, например, 1.1.

Альтернативные сценарии инициируются тремя способами.

1. Вместо основного сценария. В этом случае он инициируется главным актером и полностью замещает прецедент.

2. После определенного этапа основного сценария. В этом случае он начинается с этапа: «1. Альтернативный сценарий начинается после шага X основного сценария».

3. В любой момент времени в выполнении основного сценария. В этом случае он начинается с этапа: «1. Альтернативный сценарий начинается в любой момент времени».

Альтернативные сценарии обычно не возвращаются в основной сценарий. Если же возвращаются, то последний этап должен быть: «Альтернативный сценарий возвращается в шаг M основного сценария».

2.3.5. Обобщение актеров

Часто разные актеры инициируют одни и те же прецеденты. В этом случае диаграмму прецедентов можно упрощать, вводя обобщенного, абстрактного актера (актера-предка, актера-родителя). С другой стороны, одни актеры могут выступать как предки других актеров.

Например, в системе банкомата актер-клиент является обобщением актера, пополняющего банкомат, поскольку он может быть также клиентом (рисунок 2).

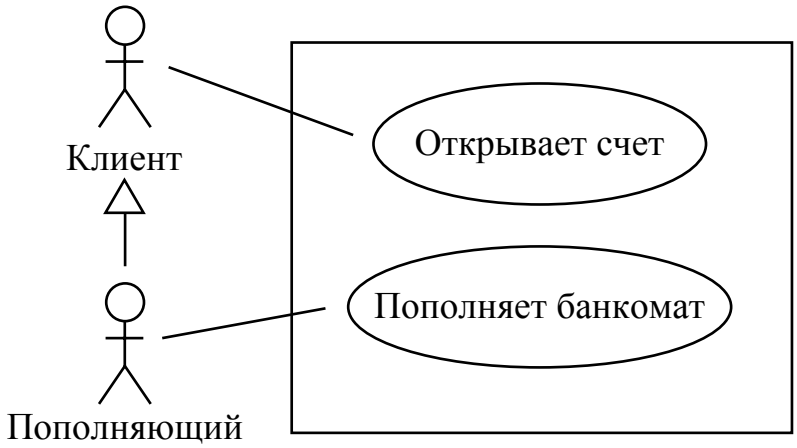


Рисунок 2 — Обобщение актеров

2.4. Отношения прецедентов

Между прецедентами могут существовать отношения, показываемые на диаграмме пунктирными линиями с названием отношения.

2.4.1. Отношение включения

Иногда в прецедентах присутствует многократное повторение одних и тех же действий. Например, в системе банкомата практически любое действие включает в себя проверку баланса. Отношение «include» устанавливаемое между прецедентами, включает поведение одного прецедента в сценарий другого прецедента. Включающий прецедент называют базовым, тот, который включают, — включенным. Место, в котором выполняется включенный прецедент, указывается в спецификации базового прецедента *точкой включения* (inclusion point). На следующем рисунке показан включаемый прецедент «Проверяет баланс».

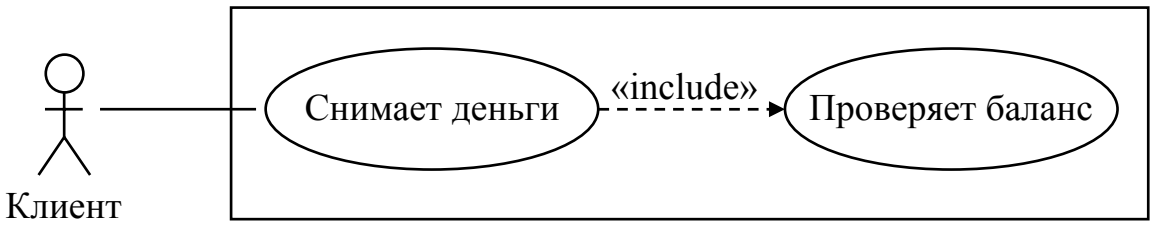


Рисунок 3 — Отношение «include»

Пример описания базового и включаемого прецедентов.
Прецедент: Снимает деньги.
. . .
Основной сценарий:
1. Клиент авторизуется.
2. Клиент запрашивает выдачу 1000 р.
3. include(Проверяет баланс).
4. Если баланс не менее 1000 р.
. . .
Прецедент: Проверяет баланс.
. . .
Основной сценарий:
1. Система возвращает баланс счета.
. . .

Включаемый прецедент называется *полным*, если он может выступать как обычный прецедент, то есть инициироваться актером. Иначе он называется *неполным*, и не может использоваться самостоятельно.

Заметим, что другим претендентом на включаемый прецедент является авторизация клиента.

2.4.2. Отношение расширения

Отношение «extend» дает возможность ввести новое поведение в существующие прецедент. Для этого базовый прецедент предоставляет точки расширения (extension points), в которые может быть добавлено новое поведение. Расширяющий прецедент предоставляет ряд сегментов вставки, которые можно ввести в точках расширения.

Заметим, что базовый прецедент ничего не знает о расширяющих прецедентах, он просто предоставляет точки входа. Базовый прецедент является полным и без расширений. Этим отношение расширения отличается от отношения включения, в котором базовый прецедент является неполным в том смысле, что его нельзя выполнить без включаемых прецедентов.

Буч отмечает, что расширяющие прецеденты являются необязательной частью базового прецедента.

Точки расширения не составляют часть сценария, они не нумеруются, а как бы накладываются поверх потока событий. В сценарии они записываются между двумя нумерованными пунктами. На диаграмме точке расширения указываются под горизонтальной чертой.

Нас следующем рисунке показан расширяющий прецедент «Стандартные суммы», который помогает клиенту быстро ввести сумму.

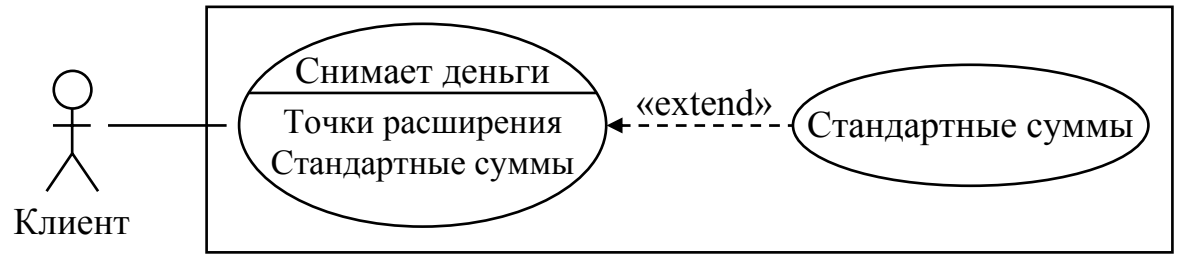


Рисунок 4 — Отношение расширения

Пример описания базового и расширяющего прецедентов.

Прецедент: Снимает деньги.

Основной сценарий:

1. Клиент авторизуется.
2. Клиент запрашивает выдачу денег.

точка расширения: Стандартные суммы

3. Клиент запрашивает выдачу 1000 р.

Расширяющий прецедент обычно является неполным, то есть не может создаваться самостоятельно. Он состоит из стольких сегментов, сколько есть точек вставки. Сегменты нумеруются и используются как описания сценариев.

2.5. Классы анализа

Входными данными для выявления классов анализа являются:
- бизнес-модель (предметная область);
- модель требований;
- модель прецедентов;
- описание архитектуры.

Классы анализа, это классы, которые:
- представляют четкую абстракцию предметной области;
- проецируются на реальные бизнес-понятия.

Самое важное свойство класса анализа — он должен четко и однозначно проецироваться в некоторое реальное прикладное понятие, такое, как покупатель, продукт или счет. Это предполагает четкость самих бизнес-понятий. Сложность ОО анализа как раз и заключается в уточнении и упорядочении бизнес-понятий.

Классы анализа не должны вытекать из проектных соображений, то есть не должны лежать в области решения. Соотношение между классами анализа и проектными классами не однозначное. Классы анализа могут объединяться в проектные классы или наоборот, разбиваться.

Классы анализа определяют набор высокоуровневых элементов, атрибутов и операций. На стадии проектирования эти высокоуровневые элементы, как правило, разбиваются на несколько элементов уровня проекта. Следует избегать излишней детализации, так как цель анализа — создать общее представление и уловить суть абстракции.

Обязательные элементы класса анализа:
- имя;
- имена атрибутов;
- имена операций;

Другие элементы, такие, как видимость, типы или стереотипы, указываются только в случае необходимости для понимания модели.

2.5.1. Признаки хорошего класса анализа

Следующие признаки отличают классы анализа:
- имя отражает назначение класса;
- он является четкой абстракцией, моделирующей один конкретный элемент предметной области;
- он проецируется на четко определяемую возможность предметной области;
- он имеет небольшой, четко определенный набор обязанностей;
- он обладает высокой внутренней связностью (cohesion);
- он обладает низкой связанностью с другими классами (coupling).

Высокая внутренняя связность означает направленность обязанностей класса на реализацию одной цели. Например, класс, описывающий покупательскую корзину, может иметь следующие операции:

- добавить продукт в корзину;
- удалить продукт из корзины;
- показать продукты в корзине.

Все операции направлены на обслуживание корзины. С другой стороны, следующие операции не связаны с этой целью:

- проверить действительность кредитной карты;
- принять платеж;
- распечатать квитанцию.

Связанность с другими классами измеряется количеством классов, с которыми данный класс имеет отношения. Низкую связанность обеспечивает равномерное распределение обязанностей между классами.

2.5.2. Практические правила

Практические правила для создания правильно сформированных классов анализа следующие.

1. В каждом классе должно быть 3-5 обязанностей.
2. Ни один класс не является изолированным. Каждый класс должен быть связан с небольшим количеством других классов. Классы могут делегировать свои обязанности вспомогательным классам, предназначенным для выполнения конкретной функции.
3. Множество мелких классов нарушает баланс распределения обязанностей. Класс редко имеет одну обязанность.
4. Несколько больших классов со множеством обязанностей также является признаком неправильной декомпозиции.
5. Классы не должны быть функтоидами. Функтиоид — это обычная процедурная функция, выдаваемая за класс. Такие классы появляются вследствие привычки думать в структурном, процедурном стиле.
6. Не должно быть классов, способных делать все сразу.
7. Глубокие деревья наследования встречаются крайне редко. Каждый уровень абстракции должен иметь четко определенное назначение. Два-три уровня наследования в большинстве случаев достаточно точно описывают реальное соответствие понятий предметной области.

2.6. Выявление классов анализа

Алгоритма выявления классов анализа не существует. Если бы такой алгоритм существовал, существовал бы и универсальный способ разработки ПО. Есть проверенные и протестированные методы. Они включают анализ текста и интервьюирование пользователей.

Перед выполнением анализа нужно собрать как можно больше информации о предметной области. Источниками информации являются:

- модель требований;
- модель прецедентов;
- глоссарий проекта.

2.6.1. Классический подход

Классический подход имеет целью выявить только классы на основе классической категоризации (классификации) предметной области. Есть несколько авторов, предлагающих выделять в качестве предполагаемых объектов разные сущности.

Sclaer и Mellor:

- материальные предметы (автомобиль, телеметрические данные, датчики давления);
- роли (мать, учитель, политик);
- события (посадка, прерывание, запрос);

взаимодействие (заем, встреча, пересечение).

Ross (моделирование баз данных):

- люди;
- места;
- предметы;
- организации;
- понятия;
- события.

Coad и Yourdon:

- структуры;
- другие системы;
- устройства;
- памятные события;
- роли;
- места;
- организационные единицы.

2.6.2. Анализ существительное-глагол

Анализ существительное-глагол — простой способ анализа текста с целью выявления классов, атрибутов и обязанностей. Существительные и именные группы указывают на классы или атрибуты класса, а глаголы и глагольные группы — на обязанности и операции класса.

Самый сложный аспект анализа существительное-глагол заключается в выявлении скрытых классов. Это классы, свойственные предметной области, но не упоминающиеся в тексте явно.

Например, в системе резервирования для компании, занимающейся организацией отдыха, будут говорить о резервировании, бронировании, но более важная абстракция, Заказ, может вообще не упоминаться.

После сбора информации производится ее анализ и выделяются:

- существительные, например, рейс;
- именные группы, например, номер рейса;
- глаголы, например, размещать;
- глагольные группы, например, проверить действительность кредитной карточки.

После создания этого списка выполняется предварительное распределение атрибутов и обязанностей по классам.

2.6.3. CRC-анализ

CRC (class - responsibilities - collaborators, класс - обязанности - участники) — это техника мозгового штурма, в которой важные моменты записывают на стикерах (или карточках небольшого размера).

Рисунок 5 — CRC-карточка

На каждый предполагаемый класс заводится отдельная карточка, поделенная на три части. В верхней записывается имя класса, в левой части — обязанности, в правой — участники. Участники, или сотрудники — это другие классы, которые могут взаимодействовать с данным для реализации функциональности системы.

Процедура CRC-анализа состоит из двух этапов:

- сбор информации (мозговой штурм);
- анализ информации (выявление классов).

Процедура первого этапа.

Участники: ОО аналитики, заинтересованные стороны и эксперты.

1. Объясните участникам, что это мозговой штурм.
    1.1. Все идеи принимаются как хорошие.
    1.2. Идеи записываются, но не обсуждаются.
2. Попросите членов команды назвать сущности и области их деятельности, например, покупатель, продукт.

2.1. Каждую сущность запишите на карточку.

2.2. Приклейте карточку на доску.

3. Попросите команду сформулировать обязанности этих сущностей и запишите их в ячейках обязанностей.

4. Попытайтесь установить классы, которые могут работать совместно. Нарисуйте между этими классами линии.

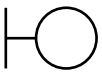
Процедура второго этапа.

Участники: ОО аналитики и эксперты.

Выявляется, какие из карточек станут классами, а какие — атрибутами. Если карточка является частью другой карточки, она — атрибут.

2.6.4. Стереотипные классы

Стереотипные классы — часть унифицированного процесса компании Rational Rose. Три типа классов анализа обозначены стереотипами.

Стереотип	Значок	Семантика
«boundary»		класс, который является посредником во взаимодействии между системой и ее окружением
«control»		класс, инкапсулирующий характерное для прецедента поведение
«entity»		класс, используемый для моделирования постоянной информации о чем-то

Любой класс анализа может быть отнесен к стереотипному классу.

Классы типа «boundary» встречаются на границе системы и общаются с внешними актерами. Есть три типа граничных классов:

- классы пользовательского интерфейса (система-пользователь);
- классы системного интерфейса (система-система);
- классы аппаратного интерфейса (система-устройство).

Классы типа «control» являются управляющими, они координируют поведение системы, соответствующее одному или более прецедентам.

Простое поведение часто может быть распределено между граничными классами и классами сущностей. Более сложное поведение лучше обозначить управляющим классом, таким, как «менеджер заказов», «регистратор», «контроллер».

Классы сущностей моделируют информацию о чем-то, имеют простое поведение (получение и задание значений). Примерами являются классы «адрес», «персона». Классы сущностей:

- пересекают несколько прецедентов;
- с ними оперируют управляющие классы;
- предоставляют и принимают информацию от граничных классов;
- представляют ключевые сущности, которыми управляет система.

2.7. Отношения

Отношения — это семантические (значимые) связи между элементами модели. Ранее эти отношения были выявлены между актерами и прецедентами, между прецедентами и между актерами.

Здесь мы рассматриваем отношения между объектами и классами.

Чтобы создать ОО программу, объекты должны общаться друг с другом. Объекты обмениваются сообщениями через соединения, называемые связями. Связь — это семантическое соединение между двумя объектами, которое обеспечивает им возможность обмена сообщениями.

Связи реализуются в языках программирования как ссылки, указатели, или как включение одних объектов в другие. Минимальное требование для установления связи — как минимум один объект должен иметь объектную ссылку на другой.

2.7.1. Диаграмма объектов

Диаграмма объектов представляет объекты и их отношения в некоторый момент времени. Соединенные связями объекты исполняют различные роли по отношению друг к другу. Эти роли записывают на соответствующем конце связи.

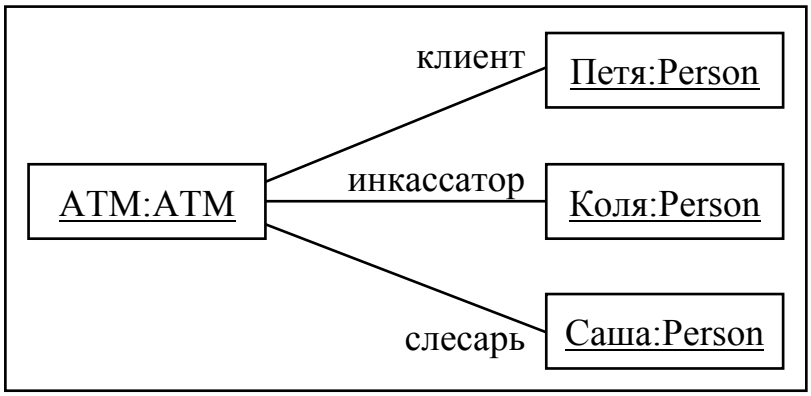


Рисунок 6 — Связи между объектами

Связь обычно соединяет два объекта. Если несколько объектов соединяются одной связью, она изображается при помощи ромба, из которого выходит несколько линий. Встречается очень редко.

Движение сообщений по связи определяет атрибут «возможность навигации», рассматриваемый далее.

2.7.2. Ассоциация

Ассоциация — это отношение между классами. Для того, чтобы между объектами существовала связь, между классами этих объектов должна существовать ассоциация. Связь — это экземпляр ассоциации.

Поскольку связи зависят от ассоциаций, это моделируется как отношение зависимости, обозначаемое пунктирной стрелкой. Соотношение между связями и ассоциациями, и классами и объектами поясняет следующий рисунок.

Рисунок 7 — Отношения и связи между классами и объектами

2.7.2.1. Синтаксис ассоциации

Ассоциация может иметь:
- имя,
- имена ролей,
- кратность,
- возможность навигации.

На следующем рисунке приведен пример ассоциации.

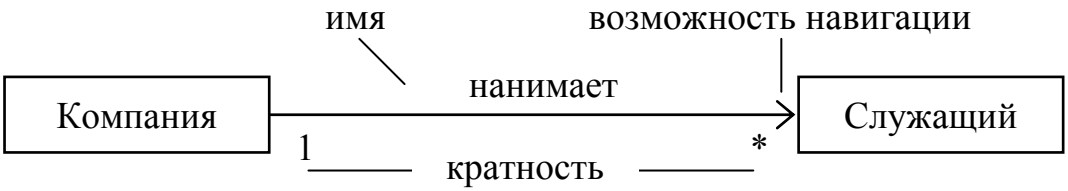


Рисунок 8 — Структура ассоциации

Здесь нет имен ролей. Роли используются вместо имени, располагаются у краев. Имена ролей — это именные группы, например, слева «работодатель», справа «работник». Принято использовать либо имя ассоциации, либо имена ролей, но не одновременно и то, и другое.

В качестве имени используется глагольная группа, так как ассоциация — это действие над целевым объектом, например, «компания *нанимает* служащего».

Кратность ограничивает число объектов класса, которые могут быть вовлечены в отношение *в любой момент времени*. Фраза «в любой момент времени» имеет решающее значение для понимания кратности.

Из кратности, показанной на рисунке, можно сделать вывод, что служащий не может быть безработным.

Ассоциация читается следующим образом: «компания нанимает много служащих». Чтение в обратном направлении: «служащий в любой момент времени нанят только одной компанией».

Кратность задается в виде «минимум..максимум», где минимум и максимум — целые числа или выражение, возвращающее целое число.

Примеры задания кратности:

1	ровно 1
1..6	от одного до шести
0..1	нуль или 1
0..*	нуль или более
*	нуль или более

Если кратность не задана, значит, решение о ней еще не принято. Никаких умолчаний для кратности не предусмотрено.

2.7.2.2. Рефлексивные ассоциации

Ассоциация называется рефлексивной, если она устанавливает отношение класса с самим собой. Это означает, что объект класса имеет связи с другими объектами этого же класса (рисунок 9).

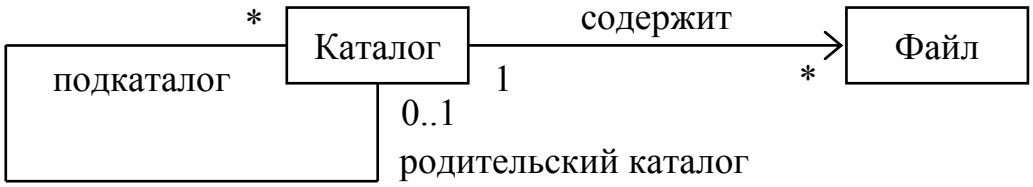


Рисунок 9 — Рефлексивная ассоциация

Если кратность рефлексивной ассоциации с одной стороны 0..1, а с другой 0..*, объекты образуют иерархию в виде дерева. Если кратность с обеих сторон равна 0..*, объекты образуют сеть, такую, как web.

2.7.2.3. Возможность навигации

Возможность навигации (navigability) показывает возможность прохода от объекта исходного класса к одному или более объектам целевого класса, в зависимости от кратности. Объекты могут посылать сообщения только в направлении, в котором есть возможность навигации.

Возможность навигации указывается стрелкой к целевому объекту.

Невозможность навигации указывается крестом у целевого объекта.

Есть три стиля обозначения возможности навигации.

1. Явная навигация. Обозначены все стрелки и кресты.

2. Скрытая навигация. Нет ни стрелок, ни крестов.

3. Компромисс. В этом случае кресты опускаются, а отсутствие стрелок обозначает двунаправленную возможность навигации.

Рекомендуемым стилем является третий (компромисс).

Его недостаток — если у линии ассоциации нет стрелок, невозможно определить, имеется ввиду двунаправленная возможность навигации, или возможность навигации отсутствует в обоих направлениях, или решение о возможности навигации еще не принято.

Однонаправленная возможность навигации реализуется в виде ссылки на целевой объект в исходном объекте, через которую исходный объект может посылать сообщения целевому объекту. Двунаправленная навигация требует, чтобы у целевого объекта была ссылка на исходный объект (обычно родительский объект). Это дает возможность целевому объекту посылать сообщения исходному объекту.

2.7.2.4. Ассоциации и атрибуты

Существует тесная связь между ассоциациями класса и его атрибутами. Ассоциация означает, что объект исходного класса имеет объектную ссылку на объект целевого класса. Поскольку в языках программирования нет языковых конструкций для обозначения ассоциаций, они моделируются как атрибуты классов. С точки зрения UML такие атрибуты являются псевдо атрибутами, поскольку они никак не характеризуют собственно класс.

Атрибутом исходного класса становится роль целевого класса. При кратности ассоциации «один к одному» атрибут моделируется как скалярное значение. При большей кратности целевого класса атрибут моделируется как массив или как коллекция. При кратности «многие ко многим» возникают проблемы, которые чаще всего требуют введения вспомогательных классов.

Если кратность целевого класса равна 1, целевой объект, вероятно, является частью исходного, в этом случае не стоит показывать отношение с ним как ассоциацию.

2.7.3. Классы-ассоциации

Часто встречается проблема размещения атрибутов классов, когда кратность отношения между классами многие ко многим. В качестве примера рассмотрим модель, в которой компания может нанимать много служащих, а служащий может работать во многих компаниях. В этой модели кратность ассоциации с обеих сторон равна 0..*.

Пусть служащий получает зарплату в нанявшей его компании.

Нельзя сделать зарплату атрибутом класса служащего, поскольку в разных компаниях служащий может получать разную зарплату. Аналогично нельзя сделать зарплату атрибутом класса компании.

Решение следует из того факта, что зарплата является частью самой ассоциации. Подобная ситуация может быть смоделирована при помощи класса-ассоциации. Класс-ассоциация — это ассоциация, являющаяся одновременно классом. Он состоит из:

- линии ассоциации,
- пунктирной линии от линии ассоциации к прямоугольнику,
- прямоугольника класса на конце пунктирной линии.

Следующий рисунок показывает класс-ассоциацию.

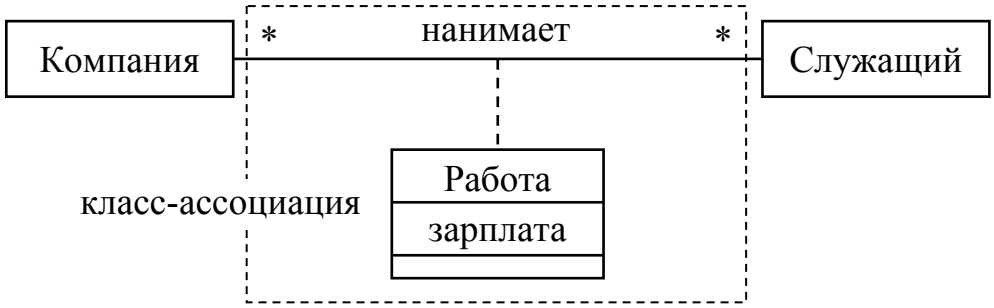


Рисунок 10 — Класс-ассоциация

Экземпляры класса-ассоциации являются связями, у которых есть атрибуты и операции.

Для выявления класса-ассоциации следует установить уникальную индивидуальность объектов, находящихся на каждом конце связи.

2.7.4. Материализованные отношения

Класс-ассоциация используется только тогда, когда между двумя объектами в любой момент времени может быть установлена одна и только одна связь. Иначе говоря, служащий может работать в компании только на одной должности. Если служащий может работать в одной компании на многих должностях, вместо класса-ассоциации следует использовать материализованное отношение, которое моделируется в виде обычного класса. Это показывает следующий рисунок.

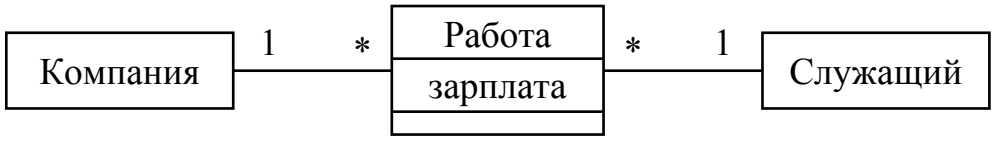


Рисунок 11 — Материализованное отношение

Материализованное отношение делает возможным существование более одной связи между любыми двумя объектами в любой момент времени.

2.7.5. Квалифицированные ассоциации

Квалифицированные ассоциации могут использоваться для понижения арности, например, для превращения ассоциации «многие к одному» в ассоциацию «один к одному». Она выбирает один объект из множества объектов целевого класса. Рассмотрим модель, приведенную на следующем рисунке слева.

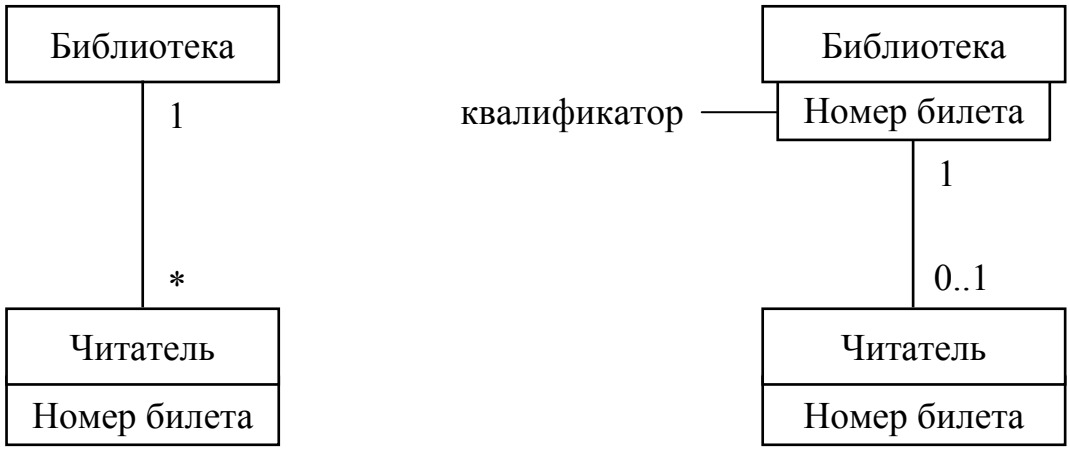


Рисунок 12 — Квалифицированная ассоциация (справа)

Квалифицированная ассоциация выбирает один объект из целевого набора, используя квалификатор — уникальный идентификатор (номер читательского билета). Квалификатор принадлежит концу ассоциации, а не классу библиотеки. Он задает уникальный ключ и превращает отношение один ко многим в отношение один к одному. Обычно квалификаторы ссылаются на атрибут целевого класса, но могут быть также использованы выражения.

2.7.6. Зависимость

Зависимость обозначает отношение между двумя или более элементами модели, при котором изменение одного элемента (поставщика) может повлиять на другой элемент (клиента). Например, объект одного класса передается как параметр в операцию объекта другого класса.

UML 2 определяет три основных типа зависимости:

1. Usage (использование) — клиент использует некоторые из доступных средств поставщика для реализации собственного поведения.

2. Abstraction (абстракция) — обозначает отношение между клиентом и поставщиком, где поставщик более абстрактный, чем клиент.

3. Permission (доступ) — поставщик разрешает клиенту доступ к своему содержимому, это дает ему возможность контролировать доступ.

Зависимости могут устанавливаться между классами, между пакетами, между классами и объектами, между операцией и классом.

2.7.6.1. Зависимости использования

Это наиболее распространенный тип зависимости. Зависимость со стереотипом «use» обозначает, что клиент каким-то образом использует поставщика. Обозначается пунктирной стрелкой от клиента к поставщику, при этом стереотип может быть опущен. Эта зависимость возникает, когда операция класса A использует объект класса B как параметр, как возвращаемое значение, или в ином качестве (вспомогательный объект).

Стереотипы «call», «parameter», «send» и «instantiate» уточняют вариант использования. Используются редко.

2.7.6.2. Зависимости абстракции

Стереотип «trace» определяет отношение, когда поставщик и клиент представляют одно понятие, но находятся в разных моделях. Например, поставщик — аналитическое представление класса, клиент — детальное представление класса.

Стереотип «substitute» показывает, что клиент во время выполнения может заменять поставщика, когда нет отношения обобщения.

Стереотип «refine» устанавливается между элементами одной модели с целью уточнения, например, версии класса.

Стереотип «derive» используется, когда необходимо явно показать возможность получения одной сущности как производной от другой.

2.7.6.3. Зависимости доступа

Стереотип «access» устанавливается между пакетами, и разрешает доступ к открытому содержимому. При этом пространства имен пакетов изолированы друг от друга.

Стереотип «import» концептуально аналогичен «access», но пространства имен поставщика и клиента объединяются.

Стереотип «permit» дает возможность управляемого нарушения инкапсуляции. В C++ этой зависимости соответствуют друзья.

2.7.7. Обобщение

Обобщение — это отношение между более общим элементом и более специальным элементом, когда более специальный элемент полностью совместим с более общим элементом, но содержит больше информации. Эти два элемента подчиняются принципу замещаемости: более специальный элемент может использоваться везде, где предполагается использование более общего элемента, без нарушения системы. Обобщаться могут актеры, классы, компоненты, интерфейсы, узлы, сигналы, прецеденты.