

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Государственное образовательное учреждение высшего профессионального
образования

«Ивановский государственный энергетический университет имени В.И.Ленина»

И. А. Левенец

**ТЕХНОЛОГИЯ РАЗРАБОТКИ
ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.
АНАЛИЗ И ПРОЕКТИРОВАНИЕ**

Учебно-методическое пособие

Иваново
2009

УДК 004.6
Л35

Левенец И.А. Технология разработки программного обеспечения. Анализ и проектирование: учеб.-метод. пособие / ГОУВПО «Ивановский государственный энергетический университет имени В. И. Ленина». – Иваново, 2009. – 88 с.

ISBN

Методическое пособие содержит изложение принципов, моделей и методов, используемых в инженерном процессе разработки программного обеспечения. Включает описание методологий IDEF0, IDEF3, DFD, IDEF1X и UML в объеме, необходимом для практической работы.

Рекомендуется для студентов старших курсов и аспирантов инженерных специальностей вузов, а также для специалистов в области информационных технологий (системных аналитиков, проектировщиков, администраторов баз данных).

Табл.13. Ил.35. Библиогр. 25 назв.

Печатается по решению редакционно-издательского совета ГОУВПО «Ивановский государственный энергетический университет имени В. И. Ленина».

Научный редактор
Пантелеев Е.Р.

Рецензент
кафедра программного обеспечения компьютерных систем
ГОУВПО «Ивановский государственный энергетический
университет имени В. И. Ленина»

ISBN

© Левенец И.А., 2009

ВВЕДЕНИЕ

Программное обеспечение как средство автоматизации и улучшения работы сегодня используется практически во всех сферах деятельности человека: государственном управлении, банковском деле и финансах, образовании, транспорте, индустрии, медицине, сельском хозяйстве, юриспруденции и других областях.

С начала развития компьютерных технологий работу по созданию программных продуктов традиционно относили к *разработке*, для которой требуются в основном навыки программирования, а не знания инженерной науки. Со временем процесс разработки программного продукта начал требовать такой же высоты научных знаний, как это необходимо в других инженерных областях – электронике, механике, строительстве. Поэтому в 1968 году было предложено рассматривать *программную инженерию* как новую научную дисциплину.

Технология разработки программного обеспечения (software engineering, программная инженерия) – система инженерных принципов для эффективного создания программного обеспечения, охватывающих все производственные процессы разработки и эксплуатации программного обеспечения.

В процессе работы по обобщению и унификации знаний в области разработки программного обеспечения был выпущен международный стандарт по программной инженерии IEEE/ACM Software Engineering Body of Knowledge SWEBOK и международный стандарт по компьютерному образованию Computing Curricula 2001: Computer Science.

Учебный курс «Технология разработки программного обеспечения» отвечает рекомендациям этих стандартов и базируется на ранее изученных дисциплинах «Информатика», «Объектно-ориентированное программирование», «Базы данных», «Сети ЭВМ и телекоммуникаций», связывает их в единый процесс разработки ПО, показывая место каждой из них в этом процессе.

Предметом изучения в данном курсе являются основные процессы разработки ПО – анализ, проектирование, конструирование (программирование), тестирование, а также

организационные процессы – управление проектами, управление конфигурацией, управление изменениями, управление качеством, внедрение и сопровождение. Для каждого рассматриваемого процесса дается понятийный аппарат, методы и средства поддержки инженерной деятельности.

Данное учебно-методическое пособие охватывает два основных процесса разработки ПО: *анализ* и *проектирование*, знакомя с современными профессиональными методами и средствами поддержки процесса разработки ПО на этих этапах.

Лабораторный практикум состоит из 5 лабораторных работ, связанных сквозным программным проектом. Каждая последующая работа основана на результатах предыдущей работы. Это позволяет пройти весь цикл начальных этапов программного проекта: бизнес-моделирование, анализ требований, системный анализ и проектирование. При этом рассматриваются два подхода: структурный и объектно-ориентированный.

Тема 1. Бизнес-моделирование.

Методологии функционального моделирования IDEF0 и IDEF3

При автоматизации деятельности любого предприятия одним из первых и наиважнейших шагов является анализ этой деятельности. В этот анализ, в частности, входят:

- описание бизнес-процессов, происходящих на предприятии;
- выделение бизнес-процессов или их участков, подлежащих автоматизации.

Бизнес-процесс — это связанный набор повторяющихся действий (функций), которые преобразуют исходный материал и/или информацию в конечный продукт (услугу) в соответствии с предварительно установленными стандартами.

Моделирование бизнес-процессов — документирование, анализ и разработка структуры бизнес-процессов, их взаимосвязей с ресурсами, необходимыми для выполнения процессов, и среды, где эти процессы будут использованы.

Цели моделирования бизнес-процессов:

1. Понять структуру и динамику организации, в которой будет использоваться разрабатываемая система (целевой организации).
2. Осмыслить текущие проблемы целевой организации и определение возможностей улучшения.
3. Обеспечить общее понимание целевой организации заказчиками, конечными пользователями и разработчиками.
4. Определить требования к автоматизированной системе, необходимые для поддержки целевой организации.

Методология функционального моделирования IDEF0

IDEF0 является сегодня основным стандартом моделирования бизнес-процессов. Стандарт IDEF0 был разработан на основе методологии структурного анализа SADT (Structured Analysis and Design Technique), предложенной Дугласом Россом в военных разработках для ВВС США в 1970-х годах.

Описание системы с помощью IDEF0 называется **функциональной моделью**. Функциональная модель предназначена для описания существующих бизнес-процессов. При этом используется как естественный, так и графический языки.

Модель – искусственный объект, представляющий собой образ системы и ее компонентов.

М моделирует А, если М отвечает на вопросы относительно А.

Здесь **М** – модель, **А** – моделируемый объект (оригинал). Модель разрабатывается для понимания, анализа и принятия решений о реконструкции (реинжинеринге) или замене существующей либо о проектировании новой системы. **Система** представляет собой совокупность взаимосвязанных и взаимодействующих частей, выполняющих некоторую полезную работу. Частями (компонентами) системы могут быть любые комбинации разнообразных сущностей, включающие людей, информацию, программное обеспечение, оборудование, изделия, сырье или энергию. Модель описывает, что происходит в системе, как ею управляют, какие сущности она преобразует, какие средства использует для выполнения своих функций и что производит.

При построении модели должна быть поставлена **цель моделирования (Purpose)**, отвечающая на следующие вопросы:

- Почему этот процесс должен быть смоделирован?
- Что должна показывать модель?
- Что может получить читатель?

Примеры определения цели: идентифицировать и определить текущие проблемы, сделать возможным анализ потенциальных улучшений, идентифицировать роли и ответственность служащих для написания должностных инструкций, определить возможность автоматизации, «регламентировать процесс и т.п.

Точка зрения (Viewpoint) также является одним из ключевых элементов при построении модели. Точку зрения можно представить как взгляд человека, который видит систему в нужном для моделирования аспекте. Точка зрения должна соответствовать цели моделирования. Часто при выборе точки зрения на модель важно задокументировать дополнительные альтернативные точки зрения.

В нотации IDEF0 описание системы (модель) организовано в виде иерархически упорядоченных и взаимосвязанных диаграмм. Сначала проводится описание системы в целом и ее взаимодействие с окружающим миром (контекстная диаграмма). Контекстная диаграмма включает только один блок, характеризующий всю совокупность моделируемых процессов, без подробностей (рис.1.1). После этого проводится *функциональная декомпозиция* – блок деятельности (главный процесс) подразделяется на крупные подпроцессы – и каждый подпроцесс описывается отдельно (диаграммы декомпозиции). Затем каждый подпроцесс декомпозируется на более мелкие – и так далее до достижения необходимой детализации описания.

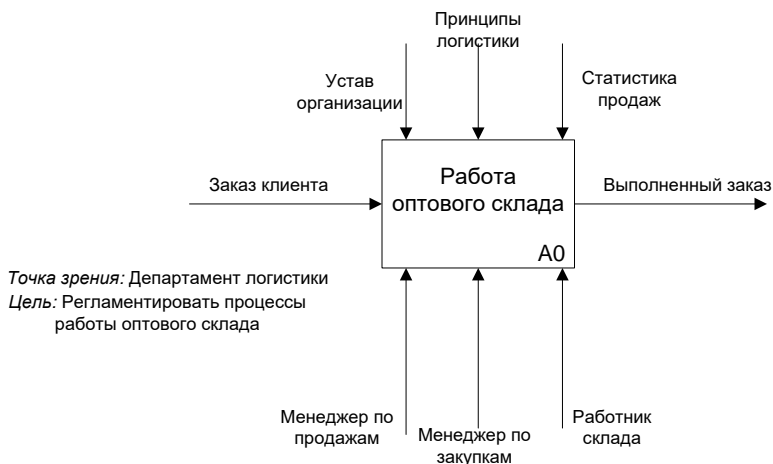


Рис. 1.1. Пример контекстной диаграммы IDEF0

Модель может содержать четыре типа диаграмм:

- контекстную диаграмму (в каждой модели может быть только одна контекстная диаграмма);
- диаграммы декомпозиции;
- диаграммы дерева узлов;
- диаграммы только для экспозиции (FEO).

Декомпозиция позволяет постепенно и структурировано представлять модель системы в виде иерархической структуры отдельных диаграмм, что делает ее менее перегруженной и легко понимаемой (рис.1.2).

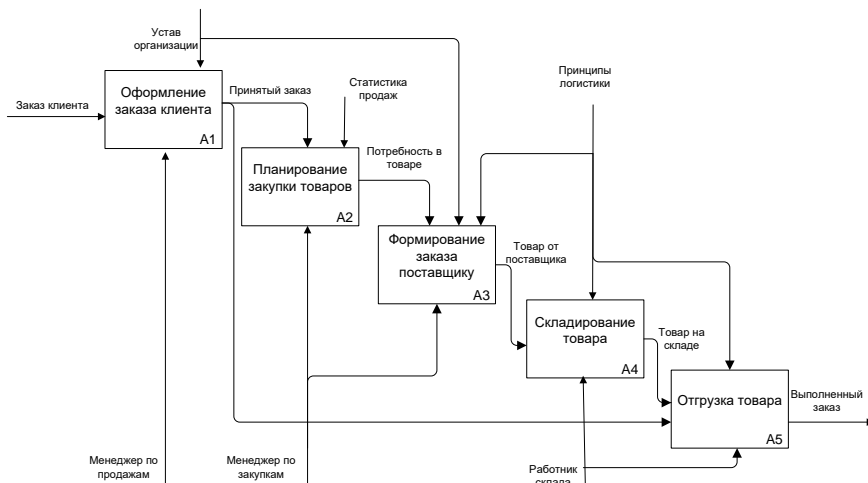


Рис. 1.2. Пример диаграммы декомпозиции IDEF0

Основной концептуальный принцип методологии IDEF – представление любой изучаемой системы в виде набора взаимодействующих и взаимосвязанных блоков, отображающих процессы, операции, действия, происходящие в изучаемой системе. В IDEF0 все, что происходит в системе и ее элементах, принято называть **функциями**. Каждой функции ставится в соответствие **блок**. На **IDEF0 – диаграмме**, основном документе при анализе и проектировании систем, блок представляет собой прямоугольник. Интерфейсы, посредством которых блок взаимодействует с другими блоками или с внешней по отношению к моделируемой системе среде, представляются **стрелками**, входящими в блок или выходящими из него. Входящие стрелки показывают, какие условия должны быть одновременно выполнены, чтобы функция, описываемая блоком, осуществилась.

Функциональные блоки (Activity) на диаграммах изображаются прямоугольниками, означающими именованные процессы, функции, работы или операции, которые происходят в течение определенного времени и имеют распознаваемые результаты. Внутри каждого блока помещается его имя и номер. Имя блока должно быть активным глаголом, глагольным оборотом или отглагольным существительным, обозначающим действие.

IDEF0 требует, чтобы в диаграмме было не менее трех и не более шести блоков. Эти ограничения поддерживают сложность диаграмм и модели на уровне, доступном для чтения, понимания и использования.

Блоки в IDEF0 размещаются по степени важности, как ее понимает автор диаграммы. Этот относительный порядок называется *доминированием*. Доминирование понимается как влияние, которое один блок оказывает на другие блоки диаграммы. Наиболее доминирующий блок обычно размещается в верхнем левом углу диаграммы, а наименее доминирующий – в правом углу.

Каждая сторона функционального блока имеет стандартное значение с точки зрения связи блок/стрелка. В свою очередь, сторона блока, к которой присоединена стрелка, однозначно определяет ее роль (рис.1.3). Стрелки, входящие в левую сторону блока, – это *входы*. Входы преобразуются или расходуются функцией, чтобы создать то, что появится на ее выходе. Стрелки, входящие в блок сверху, – это *управления*. Управления определяют условия, необходимые функции, чтобы произвести правильный выход. Стрелки, покидающие блок справа, – это *выходы*, т.е. данные или материальные объекты, произведенные функцией.

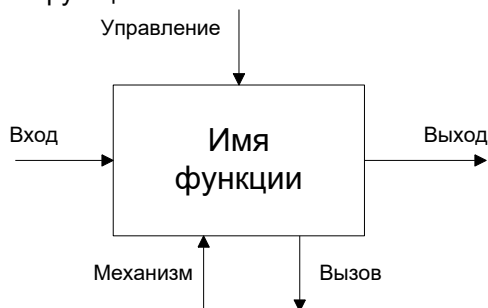


Рис. 1.3. Стандартное расположение стрелок

Стрелки, подключенные к нижней стороне блока, представляют *механизмы*. Стрелки, направленные вверх, идентифицируют средства, поддерживающие выполнение функции. Другие средства могут наследоваться из родительского блока. Стрелки механизма, направленные вниз, являются стрелками вызова. Стрелки вызова обозначают обращение из данной модели или из данной части модели к блоку, входящему в состав другой модели или другой части модели, обеспечивая

их связь, т.е. разные модели или разные части одной и той же модели могут совместно использовать один и тот же элемент (блок).

Стрелки (Arrow) описывают взаимодействие блоков с внешним миром и между собой. Стрелки представляют собой некую информацию и именуются существительными или оборотами существительного.

В IDEF0 различают пять типов стрелок:

1. *Вход* (Input) – материал или информация, которые используются или преобразуется функциональным блоком для получения результата (выхода). Допускается, что работа может не иметь ни одной стрелки входа. Зачастую сложно определить, являются ли данные входом или управлением. В этом случае подсказкой может служить то, перерабатываются/изменяются ли данные в работе или нет. Если изменяются, то скорее всего это вход, если нет – управление.
2. *Управление* (Control) – правила, стратегии, процедуры или стандарты, которыми руководствуется блок. Управление влияет на блок, но не преобразуется им.
3. *Выход* (Output) – материал или информация, которые производятся блоком. Блок без результата не имеет смысла и не должен моделироваться.
4. *Механизм* (Mechanism) – ресурсы, которые выполняют блок, например, персонал предприятия, станки, устройства и т.д. По усмотрению аналитика стрелки механизма могут не изображаться в модели.
5. *Вызов* (Call) – специальная стрелка, указывающая на другую модель работы. Стрелка вызова используется для указания того, что некоторая работа выполняется за пределами моделируемой системы.

Стрелки на контекстной диаграмме служат для описания взаимодействия системы с окружающим миром. *Граничные стрелки* – это стрелки, которые начинаются у границы диаграммы, а заканчиваются у работы или наоборот. *Внутренние стрелки* связывают блоки между собой.

Для идентификации граничных стрелок используются ICOM-коды (**ICOM** – аббревиатура от **I**ntput, **C**ontrol, **O**utput и **M**echanism). Код содержит префикс, соответствующий типу стрелки.

Внутренние стрелки связывают работы между собой. Различают пять видов связей работ.

1. Связь по входу – стрелка выхода вышестоящего блока направляется на вход нижестоящего (рис. 1.4).

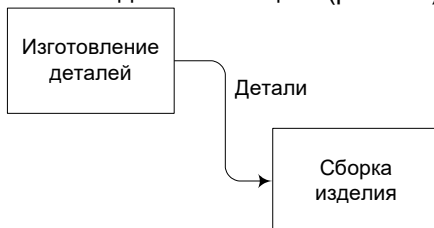


Рис. 1.4. Отношение «выход-вход»

2. Связь по управлению – выход вышестоящего блока направляется на управление нижестоящего (рис. 1.5).

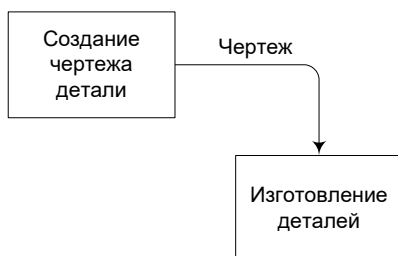


Рис. 1.5. Отношение «выход-управление»

3. Связь выход-механизм – выход одного блока направляется на механизм другого (рис. 1.6).

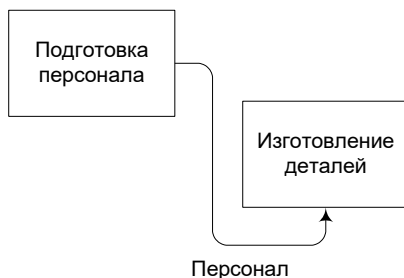


Рис. 1.6. Связь «выход-механизм»

4. Обратная связь по управлению – выход нижестоящего блока направляется на управление вышестоящего (рис. 1.7).

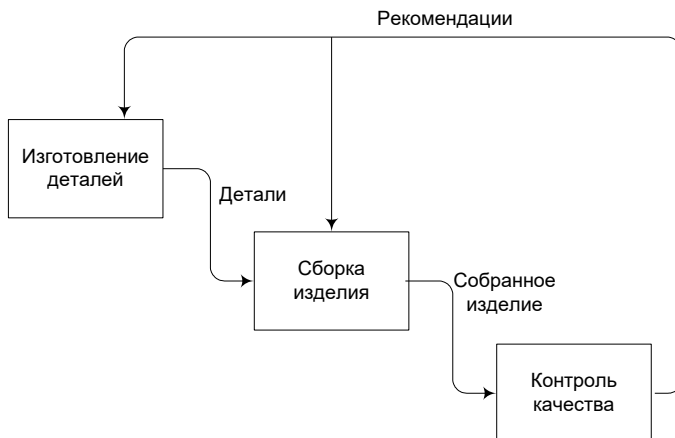


Рис. 1.7. Обратная связь по управлению

5. Обратная связь по входу – выход нижестоящего блока направляется на вход вышестоящего (рис. 1.8).

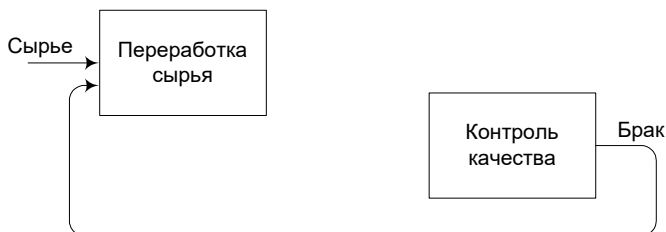


Рис. 1.8. Отношение обратной связи по входу

Словарь стрелок содержит информацию и комментарии к каждой стрелке. Каждому понятию можно дать расширенное, и если это необходимо, формальное определение, чтобы их могли понять эксперты.

В IDEF0 стрелка обычно символизирует набор объектов. Поэтому дуги могут иметь множество начальных точек (источников) и конечных точек (назначений). Таким образом, стрелки могут разветвляться и сливаться различными способами. Вся стрелка или ее часть может выходить из одного блока или нескольких блоков и заканчиваться в одном или

нескольких блоках. Ветвление и слияние стрелок призвано уменьшить загруженность диаграмм графическими элементами.

Разветвление стрелок, изображаемое в виде расходящихся линий, означает, что все содержимое стрелок или его часть может появиться в каждом ответвлении. Дуга всегда помечается до разветвления, чтобы дать название всему «пучку». Кроме того, каждый сегмент стрелки может быть помечен или не помечен в соответствии со следующими правилами:

- непомеченные сегменты содержат все объекты, указанные в метке стрелки перед ветвлением (все объекты принадлежат каждому из сегментов);
- сегменты, помеченные после точки ветвления, содержат все объекты, указанные в метке стрелки перед ветвлением, или их часть, описываемую меткой каждого конкретного сегмента.

Слияние стрелок в IDEF0 изображается в виде сходящихся линий и указывает, что содержимое каждой ветви идет на формирование метки для стрелки, являющейся результатом слияния исходных стрелок. После слияния результирующая стрелка всегда помечается для указания нового набора объектов, возникших после объединения. Каждый сегмент перед слиянием может помечаться или не помечаться в соответствии со следующими правилами:

- при слиянии непомеченных сегментов объединенный сегмент стрелки содержит все объекты, принадлежащие сливаемым сегментам и указанные в общей метке стрелки после слияния;
- при слиянии помеченных сегментов объединенный сегмент содержит все или некоторые объекты, принадлежащие сливаемым сегментам и перечисленные в общей метке после слияния; если общая метка после слияния отсутствует, это означает, что общий сегмент передает все объекты, принадлежащие сливаемым сегментам.

Правила построения диаграмм IDEF0

1. Прямоугольники блоков должны располагаться по диагонали с левого верхнего в правый нижний угол.

2. Следует максимально увеличивать расстояние между входящими или выходящими стрелками на одной грани работы.

3. Следует максимально увеличивать расстояние между блоками, поворотами и пересечениями стрелок.
4. Если две стрелки проходят параллельно (начинаются из одной и той же грани и заканчиваются на одной и той же грани другой работы), то по возможности следует их объединить и назвать единым термином.
5. Обратные связи по входу рисуются "нижней" петлей, обратная связь по управлению – "верхней".
6. Циклические обратные связи следует рисовать только в случае крайней необходимости, когда подчеркивают значение повторно используемого объекта. Принято изображать такие связи на диаграмме декомпозиции.
7. Следует минимизировать число пересечений, петель и поворотов стрелок.
8. Блоки всегда должны иметь хотя бы одну управляющую и одну выходящую стрелку, но могут не иметь входных стрелок.
9. Если одни и те же данные служат и для управления, и для входа, вычерчивается только стрелка управления.
10. Если нужно изобразить связь по входу, необходимо избегать "нависания" блоков друг над другом.
11. При разработке моделей следует избегать изначальной «привязки» функций исследуемой системы к существующей организационной структуре моделируемого объекта.

Методология IDEF3

Нотация IDEF3 была разработана в целях более удобного описания рабочих процессов (WorkFlow), для которых важно отразить логическую последовательность выполнения процедур. С помощью IDEF3 можно описывать сценарии действий сотрудников организации. Например, последовательность обработки товара от поставщика (рис. 1.9).

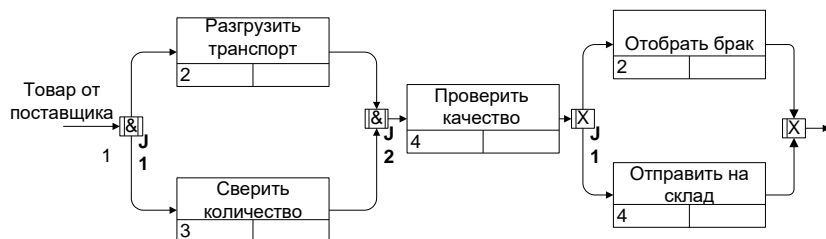
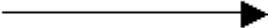




Рис.1.9. Пример диаграммы декомпозиции в IDEF3

Прямоугольники на диаграмме Workflow называются *единицами работы* (Unit of Work, UOW) и обозначают событие, процесс, решение или работу. В правом нижнем углу элемента располагается ссылка для указания на элементы из функциональной модели IDEF0, отделы или конкретных исполнителей, которые будут выполнять указанную работу.

Связи показывают взаимоотношения работ. В IDEF3 различают три типа стрелок (табл. 1.1).

Таблица 1.1. Типы стрелок в IDEF3

Тип стрелки	Определение	Графическое представление
Стрелка предшествования (Precedence)	Соединяет последовательно выполняемые работы. Работа-источник должна закончиться прежде, чем начнется работа-цель	
Стрелка нечеткого отношения (Relational)	Используется для изображения связей между работами и между работами и объектами ссылок	
Стрелка потока объектов (Object Flow)	Показывает передачу объектов от одной работы к другой, причем исходное действие должно завершиться прежде, чем начнет выполняться следующее	

Все связи в IDEF3 однонаправлены и могут начинаться и заканчиваться на любой стороне блока. Но обычно диаграммы IDEF3 организуются таким образом, что связи направлены справа налево. Стрелки начинаются на правой и заканчиваются на левой стороне блока.

Перекрестки (Junction) используются для отображения логики взаимодействия стрелок при слиянии и разветвлении или для отображения множества событий, которые могут или должны быть завершены перед началом следующей работы. Различают

перекрестки для слияния (Fan-in Junction) и разветвления (Fan-out Junction) стрелок. Перекресток не может использоваться одновременно для слияния и для разветвления. При внесении перекрестка в диаграмму необходимо указать тип перекрестка (табл. 1.2).

Таблица 1.2. Типы перекрестков в IDEF3

Обозначение	Наименование	Смысл в случае слияния стрелок (Fan-in Junction)	Смысл в случае разветвления стрелок (Fan-out Junction)
	Asynchronous AND	Все предшествующие процессы должны быть завершены	Все следующие процессы должны быть запущены
	Synchronous AND	Все предшествующие процессы завершены одновременно	Все следующие процессы запускаются одновременно
	Asynchronous OR	Один или несколько предшествующих процессов должны быть завершены	Один или несколько следующих процессов должны быть запущены
	Synchronous OR	Один или несколько предшествующих процессов завершены одновременно	Один или несколько следующих процессов запускаются одновременно
	XOR (Exclusive OR)	Только один предшествующий процесс завершен	Только один следующий процесс запускается

Все перекрестки на диаграмме нумеруются, каждый номер имеет префикс "J". В отличие от IDEF0 и DFD, в IDEF3 стрелки могут сливаться и разветвляться только через перекрестки.

Объект ссылки (Referent) в IDEF3 выражает некую идею, концепцию или данные, на которые аналитик хотел бы обратить внимание, что невозможно выразить стрелкой, работой или перекрестком. Объекты ссылки должны быть связаны с

единицами работ, стрелками или перекрестками пунктирными линиями.

Общие принципы построения модели в IDEF3 сходны с построением модели IDEF0: модель представляет собой совокупность иерархически зависимых диаграмм, прямоугольники изображают работы или процессы, стрелки – это тоже некие данные; построение модели осуществляется сверху вниз путем проведения декомпозиции крупных работ на более мелкие.

Правила построения диаграмм IDEF3

На одной диаграмме IDEF3 может быть создано несколько перекрестков различных типов. Определенные сочетания перекрестков для слияния и разветвления могут приводить к логическим несоответствиям. Чтобы избежать конфликтов, необходимо соблюдать следующие правила.

1. Каждому перекрестку для слияния должен предшествовать перекресток для разветвления.

2. Перекресток для слияния «И» не может следовать за перекрестком для разветвления типа асинхронного или синхронного «ИЛИ».

3. Перекресток для слияния «И» не может следовать за перекрестком для разветвления типа исключающего «ИЛИ».

4. Перекресток для слияния исключающего «ИЛИ» не может следовать за перекрестком для разветвления типа «И».

5. Перекресток, имеющий одну стрелку на одной стороне, должен иметь более одной стрелки на другой.

Список литературы

1. ГОСТ Р 50.1.028-2001. Информационные технологии поддержки жизненного цикла продукции. Методология функционального моделирования М.: Госстандарт России: Изд-во стандартов, 2001.
2. Маклаков, С.В. Моделирование бизнес-процессов с AllFusion Process Modeler (BPWin 4.1) /С.В. Маклаков. – М.: ДИАЛОГ-МИФИ, 2003.
3. Черемных, С.В. Структурный анализ систем: IDEF-технологии /С.В. Черемных, И.О. Семенов, В.С. Ручкин. – М.: Финансы и статистика, 2001.
4. Марка, Д. А. Методология структурного анализа и

- проектирования SADT /Д. А. Марка, К. МакГоуэн. – М.: МетаТехнология, – 1993.
5. Калянов, Г.Н. Консалтинг при автоматизации предприятий /Г.Н. Калянов. – М.: Синтег, 1997.

Задания

1. Определить цель моделирования и точку зрения модели.
2. Сформировать контекстную диаграмму, задав входы, выходы, механизмы и управление.
3. Декомпонировать контекстную диаграмму, связать граничные стрелки и задать внутренние стрелки.
4. Провести декомпозицию следующего уровня для одного-двух блоков.
5. Заполнить словари функциональных блоков и стрелок.
6. Декомпонировать один из блоков диаграммой IDEF3.
7. Оформить отчет по лабораторной работе. Отчет должен включать:
 - постановку задачи, цель моделирования и точку зрения;
 - контекстную диаграмму, диаграммы декомпозиции;
 - диаграмму IDEF3;
 - описания функциональных блоков и стрелок;
 - выводы по работе.

Контрольные вопросы

1. Что представляет собой модель в нотации IDEF0?
2. Что обозначают функциональные блоки в IDEF0?
3. Как именуются блоки в IDEF0?
4. Какое количество блоков должно присутствовать на одной диаграмме?
5. В чем заключается порядок доминирования?
6. Каково назначение сторон функционального блока на диаграмме?
7. Охарактеризуйте типы граничных стрелок.
8. Назовите виды отношений между функциональными блоками.

9. Каково назначение граничных стрелок?
10. Объясните принцип именования разветвляющихся и сливающихся стрелок.
11. Что описывает диаграмма IDEF3?
12. Объясните механизм дополнения диаграммы IDEF0 диаграммой IDEF3.
13. Перечислите основные элементы диаграммы IDEF3.
14. Что показывают связи в диаграммах IDEF3?
15. Перечислите типы стрелок в диаграммах IDEF3.
16. Что называется перекрестком?
17. Назовите типы перекрестков.
18. Назовите правила расположения перекрестков.

Тема 2. Структурный системный анализ. Методологии DFD и IDEF1X

Системный анализ – важный и критический этап в разработке информационных систем. Попытки пропустить этап анализа и сразу приступить к проектированию приводят к неэффективным и негибким проектам. Во время системного анализа в первую очередь изучается проблема, решить которую призвана разрабатываемая система. Крайне важно полностью понять и описать проблему до определения ее решения (табл. 2.1). Если структура решения не вытекает из структуры проблемы, то созданную систему будет трудно модифицировать и сопровождать.

Таблица 2.1. Постановка проблемы

Проблема	<i>[Описание проблемы]</i>
Затрагивает	<i>[Указание заинтересованных лиц, на которых оказывает влияние данная проблема]</i>
Ее следствием является	<i>[Описание воздействия данной проблемы на заинтересованных лиц и бизнес-деятельность]</i>
Успешное решение позволит	<i>[Список основных предоставляемых решением преимуществ]</i>

Требования к программному обеспечению — совокупность утверждений относительно свойств программной системы, подлежащая реализации при ее создании. Выделяют три уровня требований:

- *Бизнес-требования* — определяют цели создания новой системы и следуют из проблемы, которую система призвана решить.
- *Требования пользователей* – описывают цели и задачи, которые система позволит решить пользователям.
- *Функциональные требования* — определяют элементарные операции, которые должна иметь система для реализации требований пользователей.

Результат системного анализа – системные спецификации, включающие концепцию системы и требования к ней (техническое задание).

Системная спецификация – описание системы, которое определяет задачи ее функционирования и эффект действия, не указывая способа достижения эффекта. Спецификации

используются для планирования проекта и оценки людских ресурсов, для разработки тестовых сценариев и планирования тестирования, для проектирования и последующего документирования.

Структурный системный анализ – метод исследования систем, который начинается с общего обзора системы и затем детализируется, приобретая иерархическую структуру с большим числом уровней. Структурный анализ использует метод нисходящей функциональной декомпозиции для определения требований к системе. Данный метод основан на следующих принципах:

- 1) нисходящая иерархическая организация;
- 2) решение проблемы по частям;
- 3) графические средства общения и документирования.

Для поддержки структурного системного анализа разработана методология *диаграмм потоков данных*. Традиционно используются две схожие нотации: Йордона – Де Марко (Yourdon-De Marco) и Гейна-Карсона (Gane-Sarson).

Методология DFD

Диаграммы потоков данных (Data Flow Diagram) – основной инструмент структурного анализа. DFD позволяет определить преобразовательные процессы системы, перемещение (хранение) данных или материалов между процессами, хранилищами и внешним миром.

При моделировании потоков данных для структурного системного анализа используется прием разложения функций, при котором сложные проблемы раскладываются на составляющие, размещаемые по нарастающей по уровням детализации. Этот метод подходит для моделирования приложений, имеющих большой набор функций.

Основное предназначение DFD-моделей – моделирование функциональных требований проектируемой программной системы.

В бизнес-моделировании DFD используется для описания документооборота и обработки информации и может дополнять модель IDEF0 на нижних уровнях декомпозиции.

Диаграммы потоков данных моделируют систему как набор действий (блоков), соединенных друг с другом стрелками. Диаграммы потоков данных содержат также объекты,

собирающие информацию – хранилища данных и внешние сущности – объекты, которые выходят за границы моделируемой системы.

Стрелки в DFD показывают, как объекты (данные) перемещаются от одного действия к другому. Это представление потока обеспечивает отражение в DFD-моделях таких физических характеристик системы, как *движение* объектов (потоки данных), *хранение объектов* (хранилища данных), *источники и потребители* объектов (внешние сущности).

DFD описывает:

- 1) функции обработки информации (блоки, activity);
- 2) документы (стрелки, arrow), объекты, сотрудников или отделы, которые участвуют в обработке информации;
- 3) внешние сущности (external reference), которые обеспечивают интерфейс с внешними объектами, находящимися за границей моделируемой системы;
- 4) таблицы для хранения документов (хранилища, data store).

Потоки данных (Arrow) моделируют передачу информации (или даже физических компонент) из одной части системы в другую. Потоки на диаграммах обычно изображаются именованными стрелками, ориентация которых указывает направление движения информации. Иногда информация может двигаться в одном направлении, обрабатываться и возвращаться назад в ее источник. Такая ситуация может моделироваться либо двумя различными потоками, либо одним – двунаправленным.

Функциональный блок DFD (Activity) моделирует некоторую функцию, которая преобразует вход в выход в соответствии с действием, задаваемым именем блока. Как и действия IDEF3, функциональные блоки DFD имеют входы и выходы, но не имеют управления и механизма исполнения, как IDEF0.

Хранилище данных (data store) позволяет на определенных участках определять данные, которые будут храниться в памяти между процессами. Фактически хранилище представляет «срезы» потоков данных во времени. Информация, которую оно содержит, может использоваться в любое время, при этом данные могут выбираться в любом порядке. Имя хранилища должно идентифицировать его содержимое.

При разработке информационных систем популярен подход, называемый разделением событий (event partitioning), в котором различные диаграммы DFD выстраивают модель системы.

Вначале строится *модель окружения* (environment model), описывающая систему как объект, взаимодействующий с событиями из внешних сущностей. Такая модель обычно содержит описание цели системы, одну контекстную диаграмму и список событий. Контекстная диаграмма содержит один прямоугольник работы, изображающий систему в целом, и внешние сущности (окружение), с которыми взаимодействует система.

Далее создается *модель поведения* (behavior model), которая показывает, как система обрабатывает события (рис. 2.2). Эта модель начинается с диаграммы декомпозиции с одним блоком в ответ системы на каждое событие, описанное в модели окружения. Хранилища данных используются для моделирования данных, которые нужно запоминать между событиями. Потоки показывают движение данных.

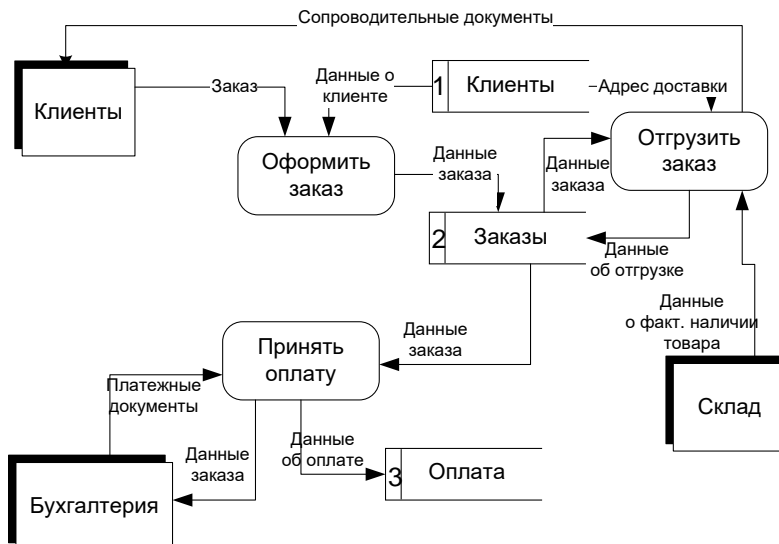


Рис. 2.2. Диаграмма декомпозиции DFD (модель поведения)

Правила построения диаграмм DFD

1. Хранилища данных следует размещать только на уровне 0 и более низких уровнях диаграммы потоков данных, а не на контекстной диаграмме.
2. Процессы взаимодействуют через хранилища данных, а не посредством прямых потоков от одного процесса к другому.
3. Данные не могут передаваться напрямую из одного хранилища в другое, они должны пройти через процесс.
4. Внешние сущности не взаимодействуют напрямую с хранилищами данных и друг с другом.
5. Диаграмма потоков данных не предназначена для отображения последовательности этапов процесса.
6. На одной диаграмме должно присутствовать не более 8-10 процессов. В том случае, если процессов больше, следует создать еще один уровень абстракции, сгруппировав связанные процессы в процесс более высокого уровня.
7. Блоки только с входящими или только выходящими потоками не допускаются. Корректный процесс, изображаемый блоком на диаграмме потоков данных, содержит входящие и исходящие потоки данных.

Методология IDEF1X

Точно так же, как диаграмма потока данных иллюстрирует процессы, происходящие в системе, модель данных отображает взаимоотношения данных (рис.2.3). Широко используется такая модель данных, как *диаграмма сущность — связь* (entity-relationship diagrams, ERD).

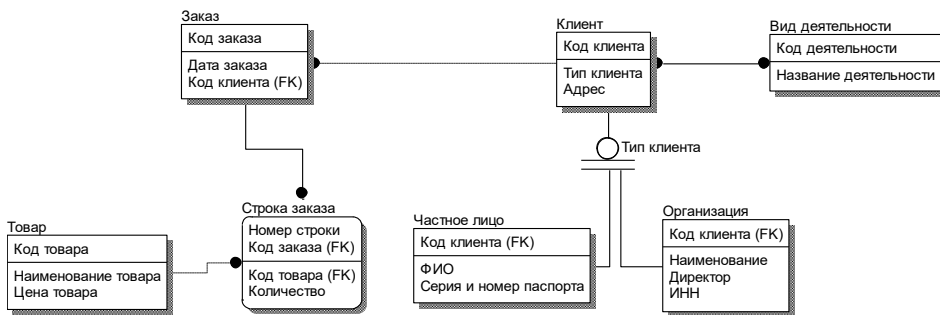


Рис.2.3. Пример модели в нотации IDEF1X

IDEF1X – представитель семейства IDEF, предназначенный для построения статических информационных моделей системы и применяемый для моделирования информации, хранимой в системе. По сути, IDEF1X – это гибридный реляционной модели и модели *сущность-связь*.

Сущности изображаются в виде прямоугольников с прямыми и закругленными углами. Прямоугольники с закругленными углами представляют *зависимые сущности*, уникальные идентификаторы (первичные ключи) которых включают по меньшей мере одну связь с другой сущностью (внешний ключ). *Независимые сущности*, уникальные идентификаторы которых не наследуются из других сущностей, изображаются в виде прямоугольников с прямыми углами. IDEF1X описывает уникальные идентификаторы в терминах первичных ключей, как это будет реализовано в реляционной базе данных. Также требуется точная идентификация внешних ключей, которые, в принципе, отображаются связью.

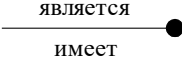
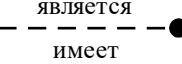
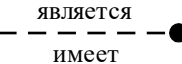
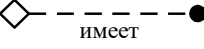
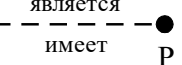
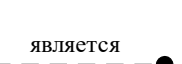

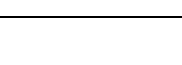
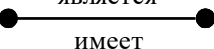
Имя сущности находится вне блока. Блок делится на две части таким образом, чтобы идентифицирующие атрибуты (первичный ключ) оказались в верхней части, а остальные – в нижней.

Связи в IDEF1X – ассиметричные: различные символы используются для изображения обязательности связи в зависимости от ее степени. Каждый набор символов описывает комбинацию обязательности и множественности соответствующей сущности (табл. 2.2).

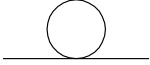
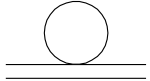
Таблица 2.2. Элементы нотации IDEF1x

Нотация	Название	Значение
ЗАКАЗ <div> <div>Код заказа</div> <div>Код заказчика (FK) Дата заказа</div> </div>	Независимая сущность	Объект, о котором нужно хранить информацию
СТРОКА ЗАКАЗА <div> <div>Код заказа (FK) Код строки</div> <div>Код элемента (FK) Количество Стоимость</div> </div>	Зависимая сущность	Сущность, в уникальный идентификатор которой входит связь с другой сущностью

Продолжение табл. 2.2

Нотация	Название	Значение
	Идентифицирующая связь	Связь между сущностями, являющаяся частью уникального идентификатора
	Неидентифицирующая связь	Связь между сущностями, не являющаяся частью уникального идентификатора
	Ноль, один или более	Каждый экземпляр родительской сущности ассоциируется с нулевым или большим количеством экземпляров другой сущности
	Ограничение необязательности	Зависимая сущность может существовать без соответствующей независимой
	Один или более	Каждый экземпляр родительской сущности ассоциируется с одним или несколькими экземплярами другой сущности
	0 или 1	Каждый экземпляр родительской сущности соответствует нулю или одному экземплярам другой сущности
	Один к 3	Каждый экземпляр родительской сущности ассоциируется точно с тремя экземплярами другой сущности
	Один к 2 -3	Каждый экземпляр родительской сущности ассоциируется с экземплярами другой сущности в количестве от двух до трех
	Многие ко многим	Каждый экземпляр родительской сущности ассоциируется с нулевым или большим количеством экземпляров другой сущности

Продолжение табл. 2.2

Нотация	Название	Значение
	Неполное наследование	Родительская сущность является супертипом для сущностей-потомков
	Полное наследование	Родительская сущность является супертипом для сущностей-потомков и все ее экземпляры должны быть экземплярами одного из подтипов

Если связь является частью уникального идентификатора, то она изображается сплошной линией, иначе – пунктирной. Соответственно блок *сущности*, чей уникальный идентификатор включает *связь*, рисуется с закругленными углами, а не с прямыми.

IDEF1X представляет два вида подтипов. Кругок и две линии под ним изображают полное наследование: все экземпляры родительской сущности должны быть экземплярами одного из подтипов. Например, ЗАКАЗЧИК всегда является либо ОРГАНИЗАЦИЕЙ, либо ЧАСТНЫМ ЛИЦОМ. Кругок только с одной линией демонстрирует неполное наследование, когда экземпляры подтипа не обязательно представляют все экземпляры родителя.

Выбор первичного ключа

При создании сущности необходимо выделить группу атрибутов, которые потенциально могут стать *первичным* ключом (потенциальные ключи), затем произвести отбор атрибутов для включения в состав первичного ключа согласно следующим рекомендациям.

1. Первичный ключ должен однозначно идентифицировать экземпляр сущности.
2. Первичный ключ должен быть компактен, то есть удаление любого атрибута из состава первичного ключа должно приводить к потере уникальности экземпляра сущности.
3. Каждый атрибут из состава первичного ключа не должен принимать NULL-значений.

4. Каждый атрибут первичного ключа не должен менять свое значение в течение всего времени существования экземпляра сущности.

При выборе первичного ключа для сущности разработчики модели часто используют дополнительный (суррогатный) ключ, т.е. произвольный номер, который уникальным образом определяет запись в сущности. Атрибут "Код заказчика" является примером суррогатного ключа. *Суррогатный ключ* лучше всего подходит на роль первичного ключа потому, что является коротким и быстрее всего идентифицирует экземпляры в объекте. К тому же суррогатные ключи могут автоматически генерироваться системой так, чтобы нумерация была сплошной, т.е. без пропусков.

Построение физической модели

На физическом уровне необходимо определить названия таблиц и полей, а также типы данных атрибутов. Также модель данных необходимо дополнить такой информацией, как учет ограничений ссылочной целостности, хранимые процедуры, триггеры, индексы.

Реляционная модель данных требует разрешения связей *многие ко многим* и иерархии наследования и допускает их только на логической модели. На физическом уровне эти связи должны быть преобразованы.

На рис. 2.4 представлен фрагмент физической модели данных с преобразованными связями наследования (миграция первичного ключа и неключевых атрибутов в иерархии от потомков к предку Customer) и *многие ко многим* (создание ассоциативной таблицы).

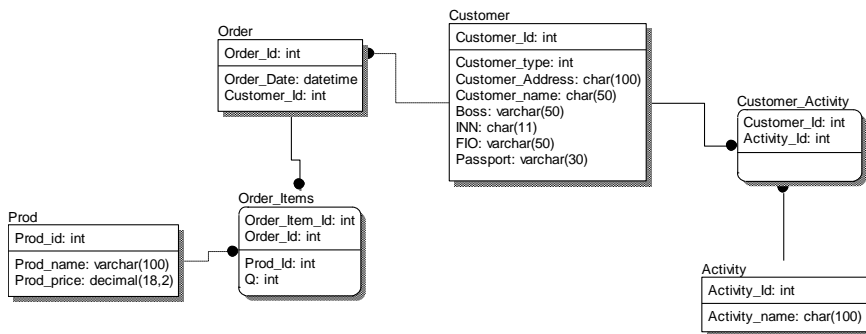


Рис. 2.4. Пример физической модели данных

Список литературы

1. Калянов, Г.Н. Консалтинг при автоматизации предприятий /Г.Н. Калянов. – М.: Синтег, 1997.
2. Вендеров, А.М. CASE-технологии. Современные методы и средства проектирования информационных систем /А.М. Вендеров. – М.: Финансы и статистика, 1998.
3. Маклаков, С.В. BPwin и ERwin: CASE-средства для разработки информационных систем /С.В. Маклаков. – М.: Диалог-МИФИ, 2001.
4. Маклаков, С.В. Создание информационных систем с AllFusion Modeling Suite /С.В. Маклаков. – М.: Диалог–МИФИ, 2003 г.

Варианты проектов

1. Информационная система «Кассовый терминал»
2. Информационная система «Турагентство»
3. Информационная система «Агентство недвижимости»
4. Информационная система «Агентство знакомств»
5. Информационная система «Букмекерская контора»
6. Информационная система «Инфоцентр»
7. Информационная система «Интернет-магазин»
8. Информационная система «Отдел кадров»
9. Информационная система «Гостиничный комплекс»
10. Информационная система «Библиотека»

Задания

1. Для выбранной информационной системы описать проблему, которую она решает, определить список заинтересованных лиц, бизнес-требования (цели системы), требования пользователей и функциональные требования.
2. Построить модель окружения (контекстную диаграмму) и модель поведения (диаграмму декомпозиции) функциональной модели DFD информационной системы.
3. Построить логическую модель данных, определив сущности, атрибуты, первичные ключи и связи.
4. Построить физическую модель данных, определив названия таблиц, полей и типы данных полей.
5. Выполнить необходимые трансформации (многие ко многим, супертип-подтип, нормализация, денормализация).

6. Оформить отчет по лабораторной работе. Отчет должен включать:
- постановку задачи информационной системы (проблема, которую она должна решать);
 - список заинтересованных лиц, бизнес-требования, требования пользователей;
 - диаграммы DFD, описание процессов, потоков данных, хранилищ данных и внешних сущностей;
 - выбранный тип СУБД, логическую и физическую модели данных;
 - выводы по работе.

Контрольные вопросы

1. Что описывает диаграмма DFD?
2. Какая нотация используется в BPWin для построения диаграмм DFD?
3. Перечислите составные части диаграммы DFD.
4. В чем состоит назначение процесса?
5. Что называется внешней сущностью?
6. Что описывают хранилища?
7. Как с помощью диаграмм DFD описать программную систему?
8. Чем логическая модель отличается от физической?
9. Назовите основные части ER-диаграммы.
10. Что показывают связи между сущностями? Назовите основные типы связей.
11. Что называется первичным ключом?
12. Что называется альтернативным ключом?
13. Назовите принципы, по которым формируется первичный ключ.
14. Чем идентифицирующая связь отличается от неидентифицирующей?
15. В каком случае образуются внешние ключи?
16. Что называется процессом нормализации?
17. В чем смысл денормализации?
18. Как осуществляется разрешение связей многие ко многим?

Тема 3. Анализ требований.

Методология UML. Модель вариантов использования

Анализ требований – процесс изучения потребностей и целей пользователей, классификация и преобразование их к требованиям к системе, аппаратуре и ПО, установление и разрешение конфликтов между требованиями, определение приоритетов, границ системы и принципов взаимодействия со средой функционирования.

Требования документируются в виде фраз-утверждений, вариантов использования UML, пользовательских историй (user story) или сценариев взаимодействия.

Методология UML

Унифицированный язык моделирования (Unified Modeling Language – UML) – это язык визуального моделирования для решения задач общего характера, который используется при определении, визуализации, конструировании и документировании артефактов программной системы (рис. 3.1). UML задумывался специально для применения в объектно-ориентированном подходе, хотя подходит и для неobjектно-ориентированных применений, в том числе для моделирования данных и требований.

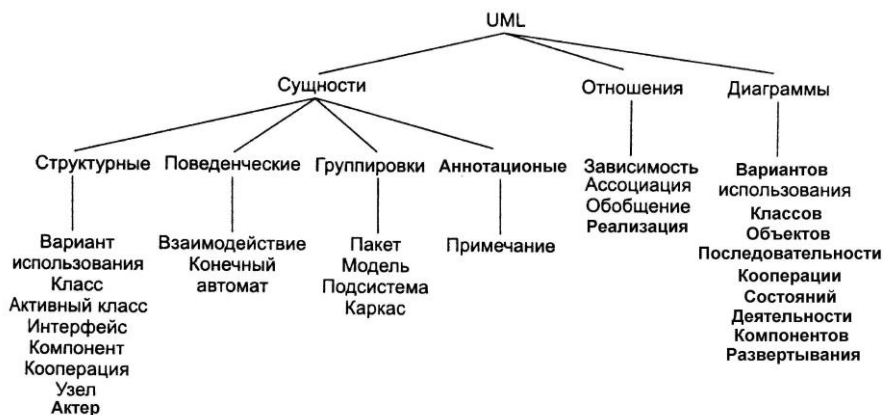

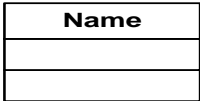


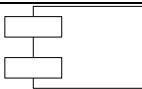
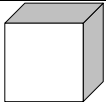


Рис. 3.1. Словарь UML

Сущности – это абстракции, являющиеся основными элементами модели. *Отношения* связывают различные сущности; *диаграммы* группируют представляющие интерес совокупности сущностей.

Таблица 3.1. Сущности в UML

Сущность	Функция	Нотация
Структурные сущности		
Актор (Actor)	Пользователь системы	
Класс (Class)	Концепция моделируемой системы	
Интерфейс (Interface)	Именованный набор операций, характеризующих некоторое поведение	IName ○ —
Вариант использования, прецедент (Use Case)	Спецификация поведения системы или ее части при взаимодействии с актерами	
Кооперация (Collaboration)	Совокупность классов, интерфейсов и других элементов, совместно работающих для достижения определенного поведения	
Компонент (Component)	Физическая часть системы	
Узел (Node)	Вычислительный ресурс	
Поведенческие сущности		
Взаимодействие (Interaction)	Обмен сообщениями (Messages) между объектами	
Конечный автомат (State machine)	Граф состояний и переходов из одного состояния в другое.	


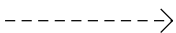


Продолжение табл. 3.1

Сущность	Функция	Нотация
Группировки		
Пакет (Package)	Универсальный механизм группировки элементов	
Модель (Model)	Пакет, в котором содержится полное описание системы, сделанное с определенной точки зрения на определенном уровне абстракции	
Подсистема (Subsystem)	Некий блок системы, обладающий спецификацией, программной реализацией и индивидуальностью. Является разновидностью пакета	
Каркас (framework)	Некий расширяемый шаблон для приложений в определенной предметной области	
Аннотационные сущности		
Примечание (Note)	Комментарий для дополнительного понимания, разъяснения или замечания к любому элементу модели	

В языке UML существует понятие стереотипа (stereotype), с помощью которого создаются новые элементы модели путем расширения функциональности базовых элементов. Таким образом, это понятие позволяет языку UML иметь минимальный набор символов, которые могут быть при необходимости дополнены для создания связующих элементов в разрабатываемой системе. Имя стереотипа заключается в двойные треугольные скобки и помещается рядом с линией связи. Например, стереотипы используются для создания отношений *include* и *extend* между ВИ.

Отношения (связи) в UML изображаются в виде линий различного начертания. Основное значение имеют четыре типа отношений: зависимости, обобщения, ассоциации и реализации.

Таблица 3.2. Отношения в UML

Отношение	Функция	Нотация
Ассоциация (Association)	Семантическая связь между экземплярами классов. Основная задача состоит в обеспечении взаимодействия между объектами, принадлежащими разным классам	
Зависимость (Dependency)	Семантическое отношение между двумя сущностями, при котором изменение одной из них, независимой, влечет изменение другой, зависимой	
Обобщение (Generalization)	Выражает отношение между общим описанием и более специфическими его разновидностями; используется при наследовании между родителем (суперклассом) и потомками (подклассами)	
Реализация (Realization)	Связывает спецификацию и ее реализацию в программном коде. Встречается в двух случаях: 1) между интерфейсами и реализующими их классами или между компонентами; 2) между вариантами использования и реализующими их кооперациями	

Диаграммы UML группируются в *модели* программной системы. Модели сопровождают все этапы жизненного цикла процесса разработки ПО.

Модель вариантов использования

Модель вариантов использования состоит из набора всех ВИ для системы (или части системы) и набора всех актеров,

взаимодействующих с этими ВИ и их текстовых описаний. Таким образом, модель вариантов использования полностью описывает функциональные возможности системы. Она представляет модель функций, ожидаемых от системы, и среду этой системы. Для визуализации модели вариантов использования (в том числе для отображения возможных отношений между вариантами использования) в языке UML и процессе Rational Unified Process используют диаграммы вариантов использования (ДВИ) и диаграммы деятельности. Для текстового описания используются спецификации вариантов использования.

Диаграмма вариантов использования (Use Case Diagram)

Актер – это роль объекта или объектов вне системы, напрямую взаимодействующих с ней.

Вариант использования (ВИ, прецедент, сценарий использования системы, СИС, use case) — это последовательность выполняемых системой действий (включая варианты), которые приводят к видимому, значимому для актера результату. **Ради выполнения именно этих действий актер взаимодействует с системой!**

Диаграмма вариантов использования (ДВИ) иллюстрирует всех актеров системы и все варианты ее использования, а также указывает, какие актеры в каких вариантах использования фигурируют.

В диаграмму вариантов использования следует включать варианты использования *уровня целей (требований) пользователя* (метафора уровня моря А. Коберна [3]). При этом необходимо следить за тем, чтобы не опускаться намного ниже уровня моря и не подниматься очень высоко. Например, «Оформить заказ», «Аннулировать заказ» – это варианты использования уровня целей пользователя (уровень моря), «Контролировать заказы» – это обобщенный вариант использования уровня бизнес-требований (выше уровня моря), «Найти клиента» – это вариант использования уровня подфункций (ниже уровня моря).

Варианты использования уровня подфункций допустимо добавлять в ДВИ в следующих случаях:

- для более лёгкого прочтения диаграммы вариантов использования;
- при их большой сложности и важности для проекта;
- при частой повторяемости этого варианта использования;
- если такой вариант использования единственный для определённого актёра.

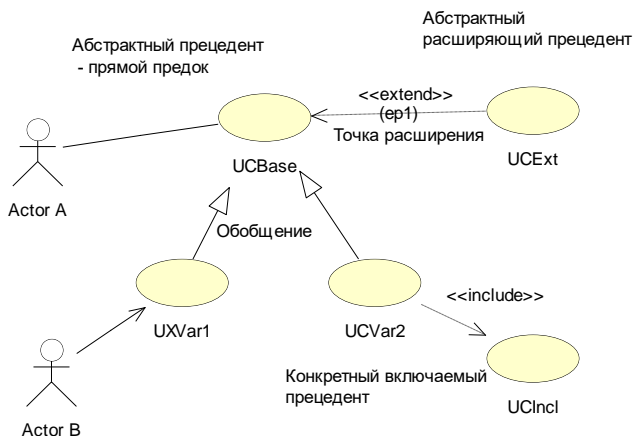

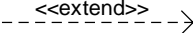
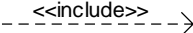


Рис. 3.2. Нотация диаграммы вариантов использования

Между актером и ВИ может существовать связь, которую называют ассоциацией (коммуникативной ассоциацией). Ассоциативная связь между актером и вариантом использования может быть либо двусторонней (от актёра к ВИ и от ВИ к актёру), либо односторонней (от актёра к ВИ или от ВИ к актёру). Направление связи показывает, кто является ее инициатором (актер или вариант использования). Такой тип отношений изображается в виде линий, соединяющих взаимодействующие элементы. Направление связи обозначается стрелками на концах линии связи.

В диаграмме вариантов использования существуют следующие типы отношений (табл.3.3).

Таблица 3.3. Виды отношений вариантов использования

Отношение	Функция	Нотация
Ассоциация (Association)	Отношение, указывающее на связь между актером и вариантом использования	
Расширение (Extend)	Включение добавочного поведения в исходный ВИ без изменения последнего	
Включение (Include)	Включение добавочного поведения в исходный ВИ, который явно описывает включение	
Обобщение (Generalization)	Отношение между общим и более специфичным (второй наследует черты первого и добавляет к ним свои)	

Правила разработки диаграмм вариантов использования

1. Не моделировать связи между актерами. По определению актеры находятся вне системы и связи между ними не относятся к сфере нашей компетенции.
2. Не соединять связью непосредственно два ВИ, кроме случаев связи включения и расширения. Диаграммы данного типа описывают только, какие варианты использования есть у системы, а не порядок их выполнения. Для этого используются диаграммы деятельности.
3. Каждый ВИ должен быть инициирован актером. Это означает, что всегда должна быть стрелка, начинающаяся на актере и заканчивающаяся на ВИ. Исключение – связи расширения и включения.
4. База данных – это слой под диаграммой. Один ВИ вводит данные в базу, другой читает. Для изображения потока информации не нужно рисовать стрелки от одного ВИ к другому.
5. Каждый актер может взаимодействовать со своим набором вариантов использования системы, в том числе только с одним из них, или со всеми (рис.3.3). Не требуется, чтобы каждый актер был связан с каждым вариантом

использования, т.е. не нужно, чтобы диаграмма была полностью связана.

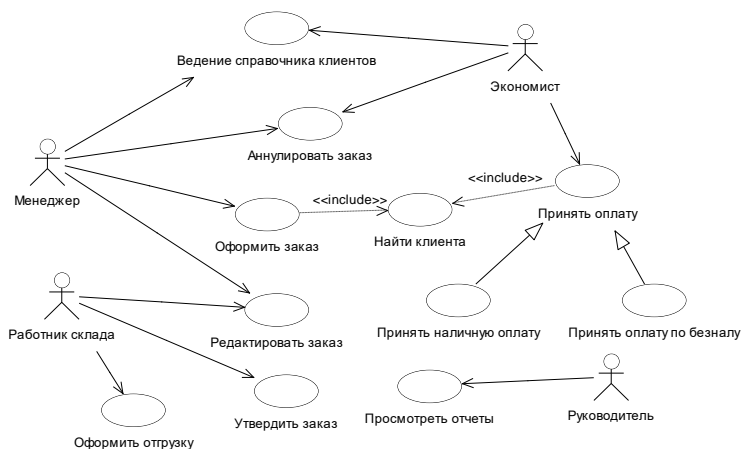


Рис. 3.3. Пример диаграммы вариантов использования

Спецификация варианта использования

Вариант использования (ВИ) – это текстовое описание на высоком уровне некоторого фрагмента поведения системы. В текстовом описании варианта использования указывается, что делает система при его выполнении. Овал, в виде которого ВИ изображается в UML, служит вспомогательным средством, помогающим визуально смоделировать систему и отобразить взаимодействия между ВИ и актерами.

Формат спецификации варианта использования

Название варианта использования

Краткое описание

...текст...

Участвующие актеры

...текст...

Поток событий

Основной поток

...текст...

Альтернативные потоки

Условие 1

...Текст...

Условие 2

...Текст...

Предусловия

...Текст...

Постусловия

...Текст...

Основной поток событий (flow of events) для варианта использования – наиболее простая последовательность шагов, при которых достигается цель действующего лица, и все заинтересованные лица остаются удовлетворены.

Каждый шаг описывает три вида действий:

1. Взаимодействие актера и системы («Клиент вводит адрес»).
2. Проверка достоверности, подтверждение для защиты интереса участника («Система проверяет пароль»).
3. Изменение внутреннего состояния от имени участника («Система рассчитывает общую сумму заказа»).

Правила описания основного потока событий.

1. Структура предложения должна быть предельно простой: 1) *субъект*; 2) *глагол*; 3) *объект*; 4) *предложное дополнение*. Например, *система (1); вычисляет (2); процентную ставку (3); по счету (4)*.

2. Текст составляется с внешней точки зрения. Нельзя описывать шаги с точки зрения системы, изнутри. Пример: *"Считать информацию с банковской карточки и PIN-код"* – неверно, *"Клиент вставляет банковскую карточку и вводит PIN-код"* – верно.

3. Необходимо писать компактные и достаточно общие фразы, показывая продвижение процесса. *«Пользователь нажимает Enter»* – неверно. *«Пользователь выбирает товар, нажимая Enter»* – верно.

4. Необходимо показывать намерение, а не движения актера. Не следует опускаться до уровня элементов пользовательского дизайна. Пример: 5 шагов («система запрашивает имя, клиент вводит имя; система запрашивает адрес, клиент вводит адрес и т.п.») удлинняют документ, ухудшая его качество, и на самом деле показывают не требования, а то, как автор представляет себе пользовательский интерфейс. Лучше вариант *"оператор вводит имя, фамилию, отчество, паспортные данные и адрес проживания клиента"*.

5. «Подтвердить», а не «проверить». Приведем два похожих варианта описания одного и того же шага. 1) *Система проверяет, является ли введенный пользователем пароль корректным*, 2) *Система проверяет введенный пользователем пароль*. Первый вариант лучше, поскольку он описывает основной успешный сценарий и побуждает писать к нему альтернативные потоки. Второй вариант подталкивает к описанию альтернатив непосредственно в теле основного успешного сценария.

6. Наличие цикла полезно выносить за пределы основных шагов. «Делать шаги x-у, пока не возникнет условие». Иногда мы хотим отметить, что некоторые шаги могут повторяться. Если повторяется только один шаг, то можно записать указание о повторении прямо в шаге. Пример: *пользователь выбирает один или несколько продуктов*.

Если повторяется несколько шагов, можно указать на повторение до или после повторения шагов. «1. *Пользователь выбирает товар и отмечает его для покупки*. 2. *Система помещает товар в корзину*. Клиент повторяет шаги 1-2, пока клиент не укажет, что выбор окончен».

7. Шаги необходимо нумеровать.

Альтернативный поток запускается при возникновении условия отклонения, принимающего значение true. Он содержит последовательность шагов, описывающих, что происходит при этом условии, и заканчивается достижением цели или отказом от нее.

Формат описания альтернативных потоков

<шагN a> <условие **a**, при котором возникает отклонение от основного поведения шага N основного сценария>

<шагN a 1> <шаг 1 новой ветви>

<шагN a 2> <шаг 2>

...

<шагN a m> <шаг **m** – переход к некоторому шагу L основного сценария, достижение цели или аварийное завершение>

<шагN b> <условие **b**, при котором возникает отклонение от основного поведения>

...

<шагN+m a> <условие, при котором возникает отклонение от основного поведения>...

Правила описания альтернативных потоков

1. Условие отклонения должно быть описано в терминах событий, которые регистрирует система. *"Клиент забыл пароль"* – неверно. *"Превышение заданного времени ожидания ввода пароля"* – верно

2. Если одно и то же отклонение от основного сценария может возникнуть на нескольких этапах, то можно их перечислить. Например, *2-5, 7а. Истекло время ожидания ввода пользователя*. Если отклонение может возникнуть на любом этапе, его помещают в начале раздела "альтернативные потоки" и обозначают символом "***". Например, **а. Пропала связь с сервером*.

3. Необходимо анализировать все дерево вариантов использования. В некоторых случаях можно перенести отклонение из нескольких вариантов использования низкого уровня в один ВИ более высокого уровня.

4. Следует ограничивать вложенность альтернативного потока двумя-тремя уровнями. Если требуется большее количество уровней, то, вероятнее всего, следует описать новый ВИ.

5. Обязательно нужно фиксировать результат выполнения альтернативного потока событий – переход к некоторому шагу основного сценария, успешное или неудачное завершение ВИ.

6. Необходимо делать отступы в тексте обработки условий.

Пример спецификации варианта использования

Название ВИ. Оформление заказа (фрагмент)

Краткое описание. Клиент определяет набор товаров, которые бы он хотел приобрести с требуемым количеством (по телефону, по факсу или лично). Менеджер находит нужные товары на складе и формирует заказ.

Участвующие актеры. Менеджер.

Поток событий

Основной поток

1. Менеджер открывает новый заказ и выбирает клиента. Система определяет условия скидок для данного клиента.
2. Менеджер добавляет новую строку в заказ. Система открывает окно выбора товара.
3. Менеджер выбирает товар. Система записывает в строку заказа его наименование, вычисляет цену в соответствии со скидкой.

4. Менеджер вводит требуемое количество. Система подтверждает, есть ли товар в необходимом количестве, и переводит требуемое количество товара в резерв.
5. Система рассчитывает сумму строки, НДС и общую сумму заказа.

.....

Альтернативные потоки

1а. Клиент заблокирован.

1а1. Система выдает предупреждение и предлагает выбрать другого клиента.

4а. На складе нет требуемого количества товара.

4а1. Система выдает предупреждение и записывает в строку доступное количество товара.

4а2. Система переводит доступное количество товара в резерв. Переход к шагу 5.

....

Предусловия

Менеджер идентифицирован в системе. Клиент существует в справочнике клиентов.

Постусловия

Информация о заказе сохранена в БД.

Диаграмма деятельности (Activity Diagram)

Диаграммы деятельности отражают динамику системы и представляют собой схемы потоков управления в системе от действия к действию, а также параллельные действия и альтернативные потоки.

Процесс выполнения последовательно переходит от одного *действия* (Activity) к другому. Когда действие в некотором состоянии завершается, поток управления сразу переходит в следующее действие. Для описания этого потока используются *переходы* (Transitions), показывающие путь от одного действия к другому (рис. 3.4).

Точка ветвления (Decision) представляется ромбом. В точку ветвления может входить ровно один переход, а выходить – два или более. Для каждого исходящего перехода задается булевское выражение, называемое сторожевым условием Guard Condition, которое вычисляется только один раз при входе в точку ветвления. Для удобства разрешается использовать ключевое слово *else* для пометки того из исходящих переходов, который должен быть выбран в случае, если условия, заданные для всех остальных переходов, не выполнены. Элементы

выбора и условия позволяют задавать альтернативные потоки (табл. 3.4).

Таблица 3.4. Элементы диаграммы деятельности

Понятие	Определение	Пиктограмма
Исходное состояние (Start State)	Начало работы	
Конечное состояние (End State)	Окончание работы	
Действие (Activity)	Имеющий некоторую протяженность во времени процесс выполнения внутренней структуры	
Переход (Transition)	Путь от одного действия к другому	
Сторожевое условие (Guard Condition)	Условие, которое должно быть выполнено, чтобы запустился ассоциированный с ним переход	[Условие1]
Выбор (Decision)	Разветвление или слияние альтернативных потоков	
Линии синхронизации (Synchronization)	Развилка или слияние параллельных потоков	

Реализовать цикл можно, если ввести два состояния действия – в первом устанавливается значение счетчика, во втором оно увеличивается – и точку ветвления, вычисление в которой показывает, следует ли прекратить цикл (рис.3.5).

В потоке обычно существуют действия, выполняемые параллельно. *Линия синхронизации* (synchronization bar) позволяет указать на необходимость их одновременного выполнения, а также обеспечивает единое выполнение действий в потоке (то есть указывает на необходимость завершения определенных действий для перехода к следующему). С концептуальной точки зрения имеется в виду истинный параллелизм, то есть одновременное выполнение, но в реальной системе это может как выполняться, так и не выполняться. Таким образом, линия слияния может иметь

несколько входящих линий перехода и одну исходящую, а линия разделения – одну входящую и несколько исходящих.

Должен поддерживаться баланс между точками разделения и слияния. Это означает, что число потоков, исходящих из точки разделения, должно быть равно числу потоков, приходящих в соответствующую ей точку слияния.

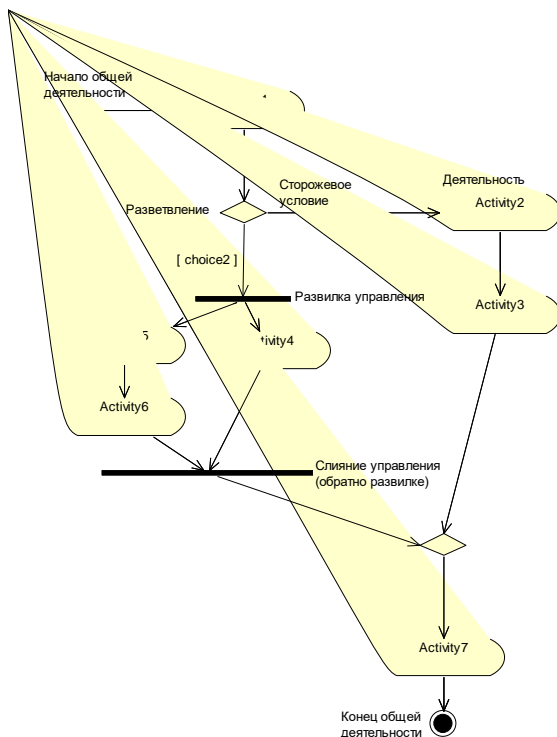


Рис.3.4. Нотация диаграммы деятельности

При моделировании течения бизнес-процессов иногда бывает полезно разбить диаграмму деятельности на участки, чтобы показать, кто отвечает за выполнение действий на каждом участке. В UML такие группы называются дорожками (Swimlanes).

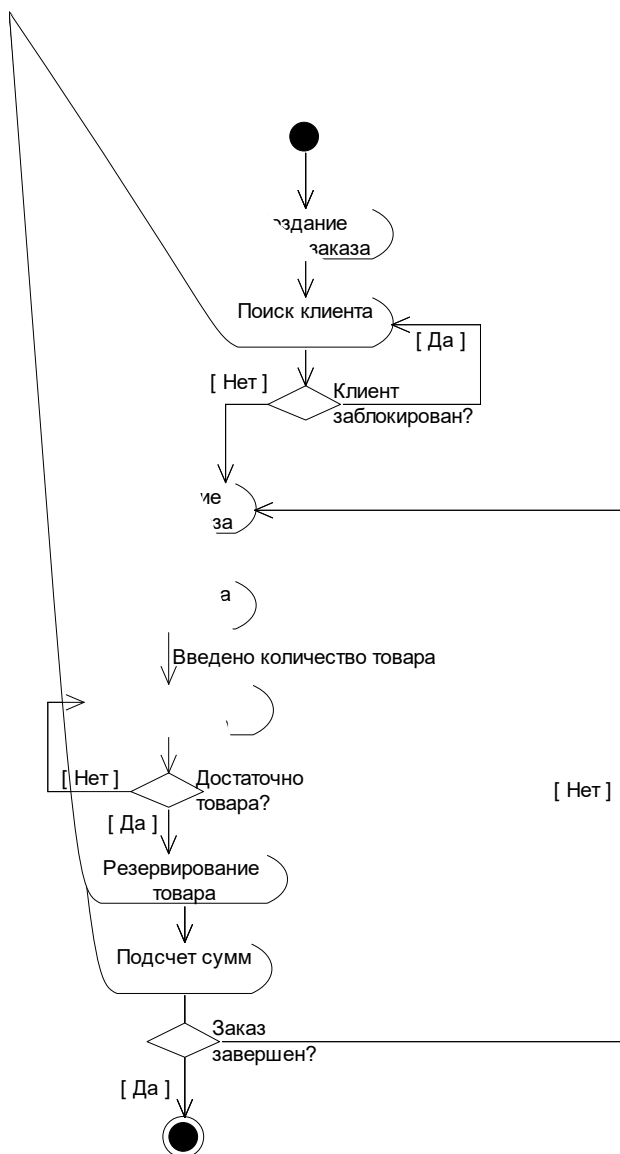


Рис. 3.5. Пример диаграммы деятельности для варианта использования «Оформить заказ» (фрагмент)

Список литературы

1. Леффингуэлл, Д. Принципы работы с требованиями к программному обеспечению. Унифицированный подход /Д. Леффингуэлл, Д. Уидриг – М.: Издат. дом «Вильямс», 2002.
2. Мацяшек, Л.А. Анализ требований и разработка информационных систем с использованием UML /Л.А. Мацяшек. – М.: Издат. дом «Вильямс», 2002.
3. Коберн, А. Современные методы описания функциональных требований к системам /А. Коберн. – М.: Лори. – 2002.
4. Вигерс К. Разработка требований к программному обеспечению /К.Вигерс. – М.: Русская редакция Microsoft. – 2004.
5. Фаулер, М. UML. Основы /М. Фаулер. – СПб.: Символ Плюс, 2009.
6. Ларман, К. Применение UML 2.0 и шаблонов проектирования /К. Ларман. – М.: «Вильямс», 2006.
7. Скотт, К. UML. Основные концепции /К. Скотт. – М. «Вильямс», 2002.
8. Буч, Г. Язык UML. Руководство пользователя /Г. Буч, Д. Рамбо, А. Джекобсон. – М.: ДМК-Пресс, 2007.
9. Рамбо, Дж. UML: Специальный справочник /Дж. Рамбо, Г. Буч, А. Джекобсон. – СПб.: Питер, – 2002.
10. Кватрани, Т. Rational Rose 2000 и UML. Визуальное моделирование /Т. Кватрани. – М.: ДМК Пресс, 2001.
11. Боггс, Ч. UML и Rational Rose /Ч. Боггс, М. Боггс. – М.: Лори, - 2000.
12. Розенберг, Д. Применение объектного моделирования с использованием UML и анализ прецедентов /Д. Розенберг, К. Скотт. – М.: ДМК Пресс, 2002.
13. Коннален, Дж. Разработка Web-приложений с использованием UML /Дж. Коннален. – М. «Вильямс», 2001.

Задания

1. Построить диаграмму вариантов использования программной системы. Добавить краткие описания актеров и вариантов использования.

2. Выбрать один из вариантов использования и написать его спецификацию согласно приведенному шаблону.

3. Построить диаграмму деятельности для выбранного варианта использования или системы в целом.

4. Оформить отчет по лабораторной работе. Отчет должен включать:

- постановку задачи, краткое описание проектируемой программной системы;
- диаграмму вариантов использования;
- описание актеров и вариантов использования;
- спецификацию варианта использования;
- диаграмму деятельности;
- выводы по работе.

Контрольные вопросы

1. Какие нотации объектно-ориентированного моделирования вы знаете?
2. Поясните назначение UML.
3. Какие блоки образуют словарь UML. Охарактеризуйте их.
4. Какие сущности UML вы знаете? Каково их назначение?
5. Перечислите диаграммы UML.
6. В чем суть механизма стереотипов UML?
7. Назовите цели модели вариантов использования.
8. Что такое актер? Назовите типы актеров.
9. Как определяются границы системы?
10. Что такое вариант использования?
11. Каковы основные свойства вариантов использования?
12. В чем заключается метафора уровня моря?
13. Как определяют набор вариантов использования?
14. Из каких элементов состоит диаграмма вариантов использования?
15. Какие отношения разрешены между элементами диаграммы вариантов использования?
16. Чем отличаются друг от друга отношения включения и расширения?
17. Каково назначение спецификации варианта использования и как она оформляется?
18. Охарактеризуйте элементы и возможности диаграммы деятельности.

Тема 4. Объектно-ориентированный анализ.

Методология UML. Модель анализа

Целью этапа *анализа* является преобразование требований к системе в форму, понятную разработчику программного обеспечения. Иными словами, требования необходимо выразить через набор классов и систем. Анализ подразумевает работу с вариантами использования и функциональными требованиями, что в результате приводит к модели анализа системы.

Далее на этапе проектирования происходит переход от абстракций предметной области к реальным программным сущностям. Таким образом, выполняя анализ, добиваются разделения задач, готовят и упрощают их для последующей деятельности по проектированию и реализации.

Модель анализа

Модель анализа – это абстракция, концептуальное обобщение проекта. Модель анализа иллюстрирует структуру проектируемой системы на достаточно высоком уровне, никак не связанном с физической реализацией системы. Она состоит из взаимодействующих классов, выражающих динамическое поведение, подробно описанное в вариантах использования и требованиях (рис. 4.1).



Рис. 4.1. Составные части модели анализа

Диаграмма классов

Диаграмма классов определяет типы объектов системы и различного рода статические связи между ними (рис. 4.2).

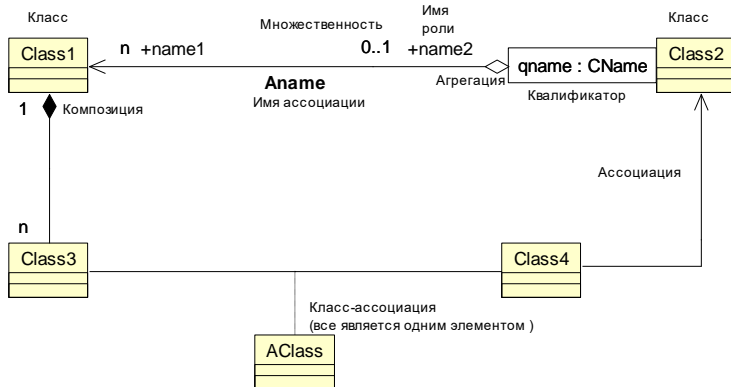


Рис. 4.2. Нотация диаграммы классов

Существует три различных уровня диаграмм классов.

1. Концептуальная модель (входит в модель анализа).
2. Модель спецификации (входит в модель проектирования).
3. Модель реализации.

Обычно для описания системы создают несколько диаграмм классов. На одной показывается некоторое подмножество классов и их отношения между классами подмножества. На других изображают то же подмножество, но вместе с атрибутами и операциями классов. Третьи соответствуют только пакетам классов и отношениям между ними. Для представления полной картины можно разработать столько диаграмм классов, сколько требуется.

Класс (class) – это некоторая сущность, инкапсулирующая данные и поведение.

Класс графически изображается в виде прямоугольника с прямыми углами, разделенного на три части (рис. 4.3).

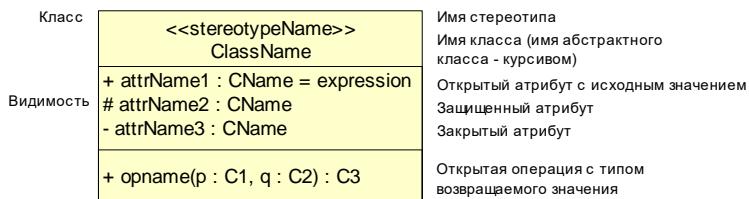


Рис. 4.3. Нотация класса

Средняя и нижняя секции прямоугольника класса содержат списки его атрибутов и операций. При описании класса необязательно сразу заполнять эти поля. Это возможно лишь в том случае, когда имеется четкое представление о том, какие операции должен выполнять данный класс и какие атрибуты для этого необходимы. На этапе анализа это может быть еще неясно, поэтому для начала содержимое этих полей может быть опущено.

Ассоциации представляют собой связи между экземплярами классов «Кого я знаю».

Зависимость (Dependency) – семантическое отношение между двумя классами, при котором изменение в спецификации одной из них, независимой, может повлиять на другую, которая ее использует.

Агрегация (Aggregation) – форма ассоциации, описывающая отношение «часть–целое». Графически отмечается с помощью ромба рядом с сущностью–целым.

Композиция (Composition) – объект часть может принадлежать только единственному целому и жизненный цикл частей совпадает с циклом целого: живут и умирают вместе. Любое удаление целого влечет удаление всех его частей.

Обобщение (Generalization) – это отношение между общей сущностью, которую называют супертипом или родителем и ее подтипами или потомками. Графически изображается в виде линии с незакрашенной стрелкой, указывающей на сущность-супертип.

Каждый конец ассоциации называется *ролью*. Роль может иметь следующие дополнительные характеристики:

- Имя.
- Кратность.
- Направление связи (navigable).
- Агрегирование.

Роль обладает кратностью, которая показывает, сколько объектов может участвовать в данной связи (табл. 4.1). Кратность определяется в виде одного или нескольких диапазонов:

нижняя граница .. верхняя граница

Таблица 4.1. Общая нотация UML для задания множественности (кратности) связи

Множественность	Значение
<i>Стандартные значения множественности (находятся в выпадающем списке)</i>	
n (по умолчанию)	Много
0..0	Ноль
0..1	Ноль или один
0..n	Ноль или больше
1..1	Ровно один
1..n	Один или больше
<i>Формат задания нестандартной множественности</i>	
<число>	Ровно число
<число1>.. <i>число2</i> >	Между числом 1 и числом 2
<число>.. <i>n</i>	Число или больше
<число1>,<число2>	Число 1 или число 2
<число1>,<число2>.. <i>число3</i> >	Ровно число 1 или между числом 2 и числом 3
<число1>.. <i>число2</i> >,<число3>.. <i>число4</i> >	Между числом 1 и числом 2 или между числом 3 и числом 4

Атрибут (attribute)– фрагмент информации, связанной с классом.

Видимость атрибута показывает, какие классы имеют право читать и изменять атрибуты (табл. 4.2).

Таблица 4.2. Видимость атрибутов

Видимость	Определение	UML
Public (Открытый)	Атрибут виден всем остальным классам. Любой класс может просмотреть или изменить значение атрибута.	+
Private (Закрытый)	Атрибут виден только внутри класса или классам friends	-
Protected (Защищенный)	Атрибут доступен только самому классу и его наследникам.	#
Implementation (Реализации)	Атрибут видим только классам внутри того же пакета	

Концептуальная модель

Основной составляющей объектно-ориентированного анализа или исследования является декомпозиция проблемы на отдельные понятия или объекты (рис. 4.4). *Концептуальная модель* – это представление понятий в терминах предметной области.

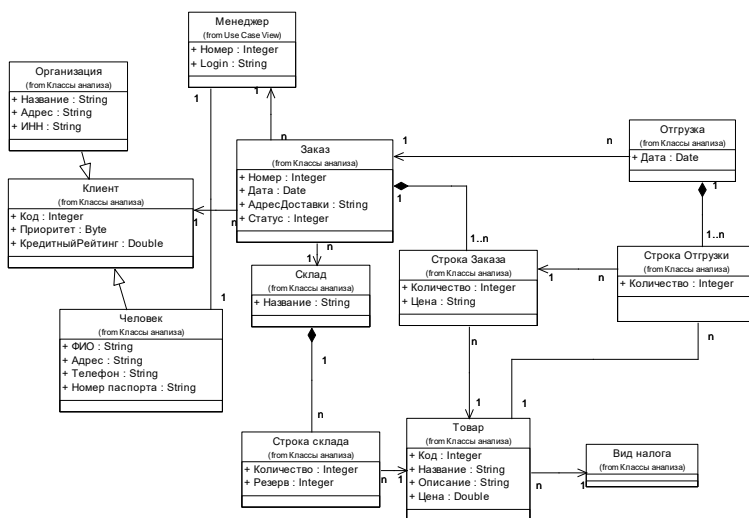


Рис. 4.4. Диаграмма классов уровня анализа (концептуальная модель) для системы заказов

На языке UML концептуальная модель представлена в виде набора статических структурных диаграмм классов, на которых не определены никакие операции.

Концептуальная модель может отображать:

- Классы.
- Ассоциации между классами.
- Атрибуты классов.

Построение модели предметной области начинается с выявления абстракций, существующих в реальном мире, то есть тех основных концептуальных объектов, которые встречаются в системе. При проектировании объектно-ориентированного программного обеспечения необходимо стремиться структурировать программу так, чтобы в центре оказались именно эти объекты из пространства задачи. Это вызвано тем,

что требования к системе меняются намного быстрее, чем реальный мир. Таким образом, концептуальная модель – это не структура программы, а декомпозиция предметной области. Лучше излишне детализировать концептуальную модель, чем недоопределить ее, поэтому в концептуальную модель включаются все понятия предметной области, даже если они не будут использоваться при написании кода.

Существуют разные подходы к выявлению классов (class discovery). Подход на основе использования общих категорий для классов позволяет вывести потенциальные классы на основе теории родовой классификации объектов, т.е. разделения мира на удобные группы, что позволяет более эффективно изучать их (рис. 4.5).

Основные категории классов (по Питеру Коду)

1. Типы актеров (*Люди и Организации*).
2. Актеры (роли актеров, участники) *Кассир, Продавец, Покупатель, Менеджер*.
3. Физические или материальные объекты (реальные вещи) *Товар, Счет, Расписание, Платежный документ*.
4. Дискриптивные вещи (спецификации и описания). *Ставка налога*.
5. Транзакции (события, процессы) *Договор, Заказ, Продажа, Прокат, Регистрация, Поставка, подписка, Оплата*.
6. Элементы транзакций *Строка заказа*.
7. Места (Контейнеры) *Магазин, Банк, Сервисный или Торговый центр, Склад, Ангар, Гараж, Больница*.

Некоторые понятия могут одновременно относиться к нескольким категориям, это подчеркивает их важность для объектной модели.

Шаблоны стандартных ассоциаций (по Питеру Коду)

1. Тип актера–актер. *Человек-Менеджер. Организация–Клиент*.
2. Актер-транзакция. *Заказ оформляется Менеджером (М:1)*.
3. Место–транзакция. *Склад является местом, хранящим товары для Заказа (1:М)*.
4. Транзакция–экземпляр строки транзакции. Здесь также используется связь агрегирования. *Заказ–СтрокаЗаказа*.

5. Транзакция – Следующая транзакция. *Заказ* – это транзакция. Какое важное событие должно произойти вслед за ним? Это *Отгрузка* заказанного товара. Аналогично в магазине за *Продажей* последует *Оплата*.
6. Строка транзакции – Строка следующей транзакции. *СтрокаЗаказа* – *СтрокаОтгрузки*.
7. Вещь–*СтрокаТранзакции*. *Товар*–*СтрокаЗаказа*.
8. Вещь–ОписаниеВещи. *Товар*–*ОписаниеТовара*.

Агрегаты. Далее используются связи агрегирования.

9. Контейнер – содержимое. *Магазин* – *Кассиры*, *Кассы*.
10. Контейнер – экземпляр строки контейнера. *Склад содержит строки-Экземпляры товаров с определенным количеством*.
11. Целое – часть.

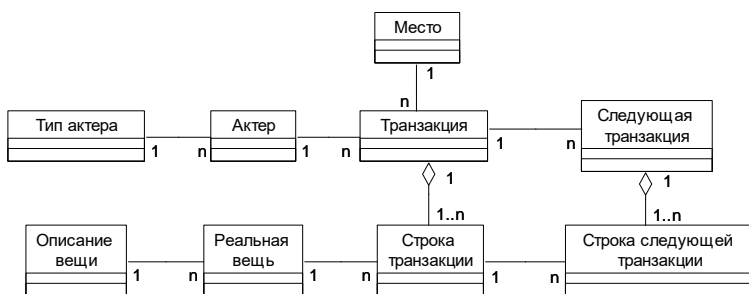


Рис. 4.5. Шаблоны стандартных ассоциаций

Принципы создания концептуальной модели

1. Составить список понятий-кандидатов на основе списка категорий и метода анализа требований. Проанализировать список.
2. Отобразить отобранные классы в концептуальной модели.
3. Добавить необходимые ассоциации между классами.
4. Добавить атрибуты, необходимые для выполнения информационных требований.

В концептуальную модель включаются только те атрибуты, для которых определены соответствующие требования или для которых предполагается, что нужно хранить определенную информацию.

В концептуальной модели атрибуты должны быть простых типов (Boolean, Date, Integer, String и т.п.). Если возникли сомнения, то лучше создать отдельное понятие. Стандартной ошибкой является моделирование сложного понятия предметной области в форме атрибута.

В концептуальной модели атрибуты не должны использоваться для связи понятий. Для этого используются ассоциации. Наиболее распространенным нарушением данного принципа является добавление некоторой разновидности атрибута внешнего ключа, как при разработке реляционных баз данных.

Позднее, на стадиях проектирования и программирования ассоциации между объектами зачастую реализуются как атрибуты, указывающие на другие сложные типы. Однако это не единственная возможность реализации ассоциации, и соответствующее решение нужно отложить на стадию проектирования.

Диаграмма состояний

Диаграмма состояний (statechart diagram) иллюстрирует важные события и состояния объекта, а также поведение объекта в ответ на реализацию событий. Диаграмма состояний отражает жизненный цикл объекта, начиная с момента его создания и заканчивая уничтожением. На ней изображаются события, в которых участвует данный объект, переходы и состояния объекта между ними (рис. 4.6). В основу диаграммы состояний положена теория конечных автоматов.

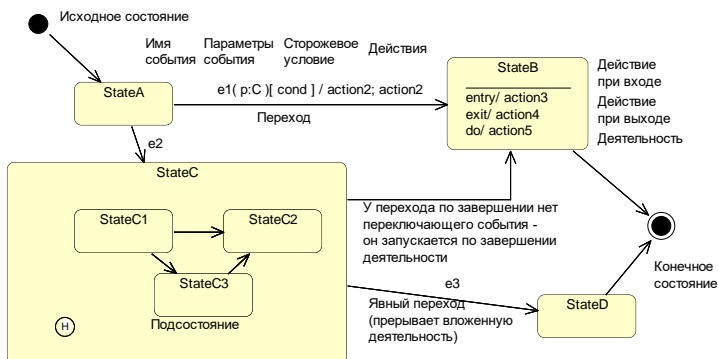


Рис. 4.6. Нотация диаграммы состояний

Состояние (state) – это некое положение в жизни объекта, при котором он удовлетворяет определённому условию, выполняет некоторое действие или ожидает события. В языке UML состояние изображается в виде прямоугольника с закругленными углами.

Переходы между состояниями (state transitions) представляют собой смену исходного состояния последующим (которое может быть тем же, что и исходное).

С переходом между состояниями может быть связано событие, условие, действие и посылаемые события.

Синтаксически метка перехода состоит из трех частей, каждая из которых является необязательной.

<Событие>(Параметры)[<Условие>]/<Действие>

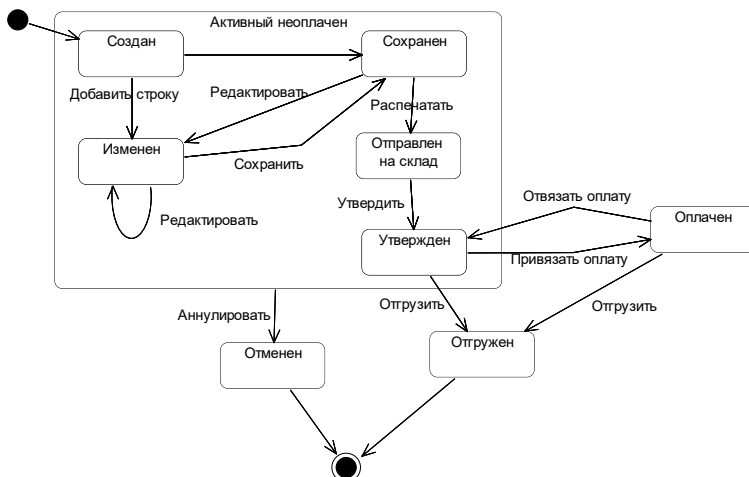


Рис. 4.7. Пример диаграммы состояний для сущности «Заказ»

Внешние события (системные) вызываются внешними по отношению к системе условиями (например, актерами). *Актер нажимает кнопку Новый заказ или Сохранить.*

Внутренние события инициируются внутренними по отношению к системе условиями, например при выполнении операции после получения сообщения от другого объекта.

Временные события возникают в определенный день и час или через определенный интервал. Например, по таймеру происходит автосохранение в текстовом редакторе.

Диаграммы состояний создаются для зависимых от состояния типов со сложным поведением (рис. 4.7). Диаграммы состояний следует использовать для иллюстрации внешних и временных событий, а также реакции на них, а не для проектирования поведения объектов на основе внутренних событий.

Анализ реализации варианта использования

Реализация варианта использования (кооперация) содержит текстовое описание потока событий, диаграммы классов, описывающие участвующие в нем классы анализа и диаграммы взаимодействия, которые описывают реализацию конкретного потока событий в терминах взаимодействующих объектов. Связь "реализует" (realizes) связывает вариант использования с его реализацией (рис. 4.8).

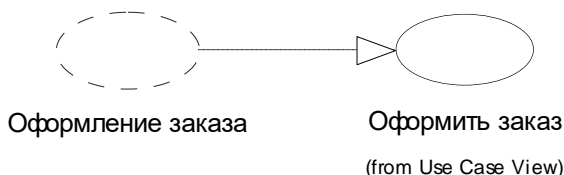
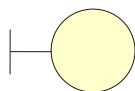


Рис. 4.8. Связь между вариантом использования и его реализацией

Классы анализа

Классы анализа (analysis classes) не зависят от языка программирования. Они позволяют увидеть структуру системы, не углубляясь в специфические особенности конкретного языка программирования. На этапе проектирования все эти классы могут быть преобразованы в классы проектирования (design classes).

При моделировании классов анализа применяется методология Boundary-Entity-Control, которая делит все классы на три группы – граничные (boundary), сущности (entity) и управляющие (control).



Граничные классы (boundary) обеспечивают интерфейс между системой и ее актерами (пользователями и внешними системами). Они находятся на границе системы с окружающим миром и моделируют

взаимодействие с ним. Взаимодействие включает в себя получение и передачу информации, а также запросы пользователей и внешних систем к ним. Обычно экземпляры этих классов представляют собой диалоговые окна, формы, панели, коммуникационные интерфейсы, интерфейсы принтера, датчиков, терминалов, а также формы и отчеты. В Web-приложениях такие объекты могут представлять целые Web-страницы.



Классы-сущности (entity) — используются для моделирования долгоживущей, нередко сохраняемой информации. Это тип классов может представлять сущности реального мира или внутренние элементы системы, необходимость которых проявляется в результате анализа (концептуальная модель). Сущностные объекты хранят данные и предоставляют ограниченный доступ к ним с помощью своих операций. Сущностные классы часто отображаются на таблицы базы данных.



Управляющие классы – контроллеры (control) отвечают за координацию, порядок последовательности, взаимодействия и управления другими объектами и часто используются для хранения управления, относящегося к некоторому варианту использования. Именно в них заключается бизнес-логика и стратегия приложения. Идея состоит в том, чтобы локализовать изменения в этих классах, не трогая интерфейса пользователя и схемы базы данных.

Диаграмма коммуникации

Диаграмма взаимодействия объектов, построенная в терминах классов анализа для варианта использования, в UML 2.0 называется *диаграммой коммуникации* (*communication diagram*). В более ранних версиях UML эта диаграмма называлась диаграммой кооперации (*collaboration diagram*).

Диаграмма коммуникации для варианта использования выполняется путем исследования его спецификации, изображения актеров, граничных, сущностных и управляющих классов, а также связей между ними. И главная, и все

альтернативные последовательности должны уместиться на одной диаграмме (рис. 4.9).

Ассоциации между классами анализа подчиняются следующим *правилам*.

- Актеры могут взаимодействовать только с граничными классами.
- Граничные классы могут общаться только с контроллерами и актерами.
- Классы-сущности могут взаимодействовать только с классами-контроллерами.
- Классы-контроллеры могут взаимодействовать с граничными классами, сущностными и другими контроллерами, но не с актерами.

Диаграмма коммуникации является средством контроля корректности текстов вариантов использования с учетом того набора объектов, с которыми придется работать, и выполняет функции предварительного проектирования.

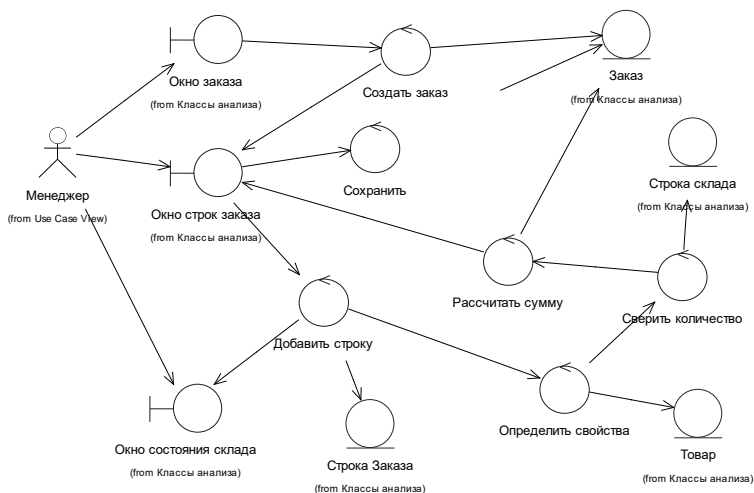


Рис. 4.9. Пример диаграммы коммуникации для варианта использования «Оформление заказа»

Диаграмма последовательности

Диаграмма последовательности (sequence diagram) представляет собой взаимодействие – множество сообщений между объектами, упорядоченное по временной оси (рис. 4.10).

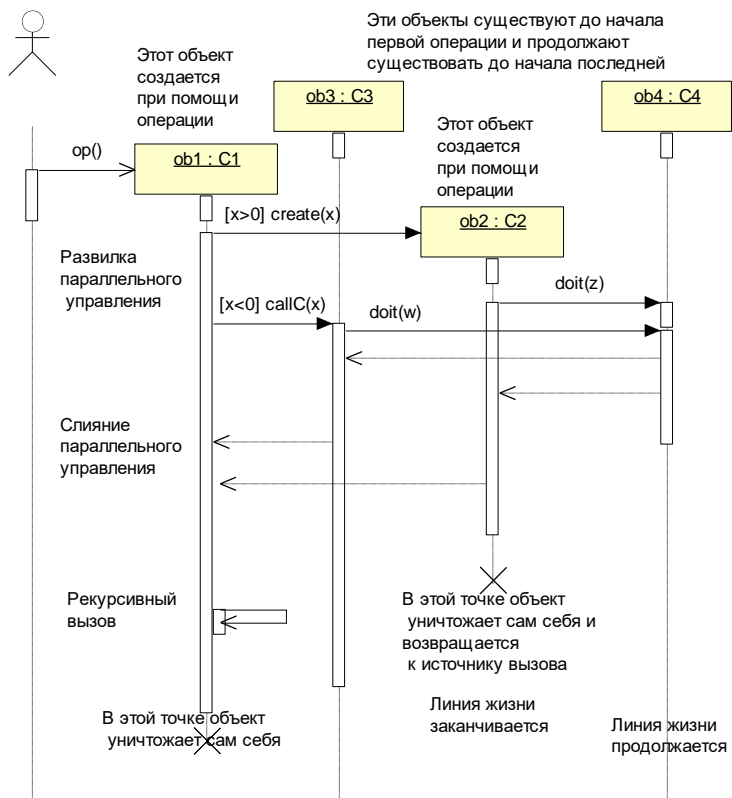


Рис. 4.10. Нотация диаграммы последовательности

Линия жизни (life line) объекта – это вертикальная пунктирная линия, отражающая существование объекта во времени. Каждый объект изображается в виде отдельной вертикальной колонки. Символ объекта (прямоугольник, внутри которого располагается подчеркнутое имя объекта) помещается сверху колонки. Объекты могут существовать до

взаимодействия, создаваться и уничтожаться во время взаимодействий.

Сообщение (message) представляется в виде стрелки между линиями жизни объектов. Сообщения появляются в порядке, как они показаны на странице (сверху вниз). Каждое сообщение помечается как минимум именем сообщения; при желании можно добавить также аргументы и некоторую управляющую информацию и, кроме того, можно показать сообщение, которое объект посылает самому себе, при этом стрелка сообщения указывает на ту же самую линию жизни. Из управляющей информации особое значение имеет условие, показывающее, в каком случае посылается сообщение (например, [нуженПовторныйЗаказ=true]). Сообщение посылается только при выполнении данного условия.

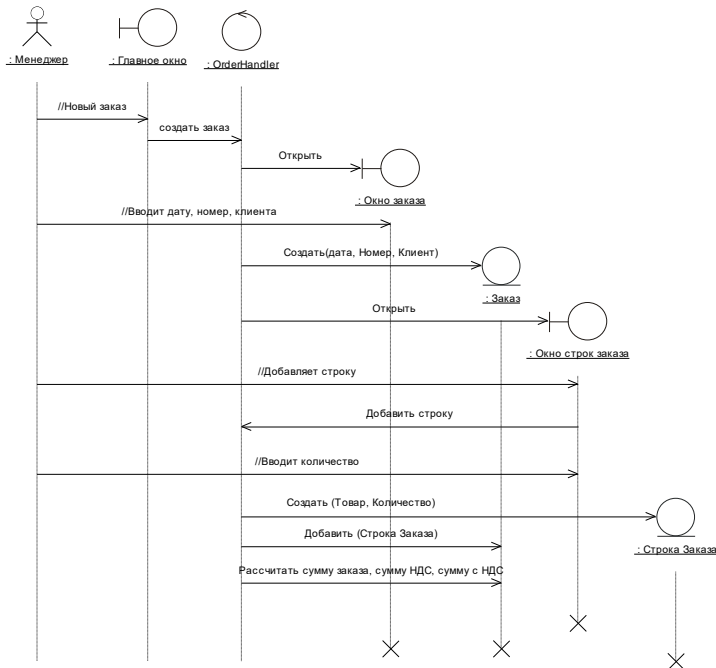


Рис. 4.11. Пример диаграммы последовательности уровня анализа для варианта использования «Оформить заказ» (фрагмент)

Фокус управления (focus of control) изображается в виде вытянутого прямоугольника, показывающего промежуток времени, в течение которого объект выполняет какое-либо действие.

При построении диаграмм последовательности необходимо следить, чтобы прослеживался точный путь передачи управления (рис.4.11).

Список литературы

1. Розенберг, Д. Применение объектного моделирования с использованием UML и анализ прецедентов /Д. Розенберг, К. Скотт. – М.: ДМК Пресс, 2002.
2. Коуд, П. Объектные модели. Стратегии, шаблоны и приложения / П. Коуд, Д. Норт, М. Мейнфилд. – М.: «Лори», 1999.
3. Фаулер, М. UML. Основы /М. Фаулер. – СПб.: Символ Плюс, 2009.
4. Ларман, К. Применение UML 2.0 и шаблонов проектирования /К. Ларман. – М.: «Вильямс», 2006.
5. Скотт, К. UML. Основные концепции /К. Скотт. – М. «Вильямс», 2002.
6. Буч, Г. Язык UML. Руководство пользователя /Г. Буч, Д. Рамбо, А. Джекобсон. – М.: ДМК-Пресс, 2007.
7. Рамбо, Дж. UML 2.0. Объектно-ориентированное моделирование и разработка /Дж. Рамбо, М.Блаха. СПб.: . – Питер, 2007.
8. Рамбо, Дж. UML: Специальный справочник /Дж. Рамбо, Г. Буч, А. Джекобсон. – СПб.: Питер, – 2002.
9. Арлоу Дж. UML 2 и Унифицированный процесс /Дж Арлоу, А. Нейштадт. – СПб.: Символ Плюс, 2008.
10. Кватрани, Т. Rational Rose 2000 и UML. Визуальное моделирование /Т. Кватрани. – М.: ДМК Пресс, 2001.
11. Боггс, Ч. UML и Rational Rose /Ч. Боггс, М. Боггс. – М.: Лори, - 2000.

Задания

1. Построить концептуальную модель предметной области.

2. Для одной из сущностей концептуальной модели построить диаграмму состояний.

3. Создать реализацию (кооперацию) выбранного варианта использования.

4. Создать классы анализа для выбранного варианта использования и разделить их на группы согласно методологии Boundary-Entity-Control.

5. Построить диаграмму коммуникации и диаграмму последовательности уровня анализа выбранного варианта использования.

6. Оформить отчет по лабораторной работе. Отчет должен включать:

- постановку задачи;
- концептуальную модель;
- диаграмму состояний;
- диаграмму коммуникации;
- диаграмму последовательности;
- выводы по работе.

Контрольные вопросы

1. С какой целью строится модель анализа при разработке программной системы?
2. Какие составные части входят в модель анализа?
3. Какие существуют уровни построения диаграмм классов?
4. Как изображается класс на диаграмме UML?
5. Какие секции входят в графическое обозначение класса? Какие секции класса можно не показывать?
6. Какие отношения между классами отражаются в UML и каким образом?
7. Как показывается видимость атрибутов на диаграмме классов?
8. В чем суть механизма стереотипов в UML?
9. Какие элементы входят в концептуальную модель?
10. Как применяется подход на основе использования общих категорий для выявления классов?
11. Что лежит в основе диаграммы состояний?
12. Для каких объектов могут быть построены диаграммы состояний?
13. Какие причины могут повлечь изменение состояний, и как это отражается на диаграмме состояний?

14. В чем состоит суть методологии Boundary-Entity-Control. Как она может быть использована при анализе разрабатываемой системы?
15. Как связаны текст спецификации варианта использования и диаграмма коммуникации UML?
16. Каковы правила взаимодействия объектов на диаграмме коммуникации?
17. Что показывает диаграмма последовательности UML?
18. Как отображается порядок передачи сообщений в диаграмме последовательности?
19. В какой форме записываются сообщения в UML? Поясните смысл сообщения.
20. Что общего и в чем различия между диаграммами последовательности и коммуникации (кооперации)? Какие из них, на ваш взгляд, ближе к структурным, а какие – к поведенческим?

Тема 5. Объектно-ориентированное проектирование.

Методология UML. Модель проектирования

Проектирование – стадия в разработке системы, на которой описывается, как система будет реализована на логическом уровне, еще до написания программного кода. На этом этапе принимаются стратегические решения относительно того, как воплотить в жизнь функциональные и нефункциональные требования к системе.

Цель проектирования состоит в согласовании результатов анализа с ограничениями, навязанными нефункциональными требованиями, средой реализации, требованиями к производительности и т.д. Таким образом, проектирование – это уточнение анализа, направленное на оптимизацию проекта системы при обязательном удовлетворении всех требований. Основными исходными данными для проектирования будут являться результаты анализа, в частности, модель анализа.

Как и на стадии анализа, работы на этапе проектирования в основном связаны с диаграммами классов и взаимодействий (class & interaction diagram). Классы становятся более определенными, с полностью уточненными свойствами (именем и типом) и операциями (с известной сигнатурой). В процессе проектирования зачастую добавляются новые дополнительные классы, которые являются вспомогательными по отношению к классам реализации. В конечном итоге должна быть разработана модель проектирования (design model), которую можно напрямую преобразовать в программный код.

Модель проектирования

Модель проектирования — это объектная модель, которая описывает физическую реализацию вариантов использования. В этой модели основное внимание уделяется тому, каким образом функциональные и нефункциональные требования вместе с другими ограничениями, относящимися к среде реализации, составляют систему. Кроме того, модель проектирования представляет собой абстракцию реализации системы и используется в качестве исходных данных для реализации.

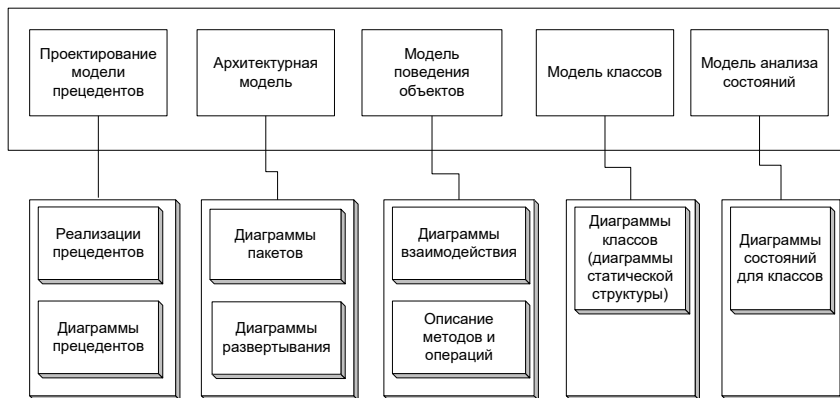


Рис. 5.1. Составные части модели проектирования

В модели проектирования варианты использования реализуются в виде классов проектирования и их объектов. Эта реализация представляется в виде кооперации и называется проектом реализации варианта использования. При этом проект реализации варианта использования отличается от анализа реализации варианта использования. Первый описывает реализацию варианта использования в терминах взаимодействующих объектов проектирования, а другой – в терминах объектов анализа.

Архитектурное проектирование

Архитектурное проектирование – процесс описания системы в терминах ее модулей: подсистем и их интерфейсов.

Архитектура информационных систем, включающая интерфейс пользователя и хранение данных на постоянном носителе, называется *трехуровневой архитектурой* (tree-tier architecture). При такой архитектуре приложение делится по вертикали на три уровня.

1. Уровень представления.
2. Уровень логики приложения.
3. Уровень данных.

Неотъемлемой частью трехуровневой архитектуры является вынесение логики приложения на отдельный уровень. Уровень представления является относительно независимым от выполнения основных задач приложения. Его объекты лишь отправляют запросы на средний уровень логики приложения, а

средний уровень, в свою очередь, взаимодействует с самым нижним уровнем хранения данных.

При выполнении декомпозиции уровня логики приложения трехуровневая архитектура системы превращается в многоуровневую (multi-tiered). Например, уровень логики приложения можно разделить следующим образом:

- Концептуальные объекты (domain objects) – классы, представляющие понятия предметной области.
- Службы (services) – служебные объекты для взаимодействия с базой данных, создания отчетов, обеспечения безопасности и т.д.

Для изображения архитектуры в UML используется механизм пакетов.

Пакет (package) – это группа элементов модели любого вида, например классов, вариантов использования, коопераций или других пакетов (вложенных пакетов).

На этапе проектирования модель системы становится очень громоздкой и возникает необходимость разделения элементов на более мелкие подмножества.

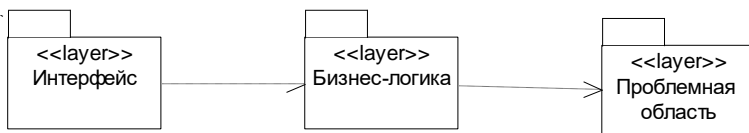


Рис. 5.2. Диаграмма пакетов-слоев

Объединять классы в пакеты можно как угодно, однако существует несколько наиболее распространенных подходов. Во-первых, можно группировать классы по стереотипу. В этом случае получается один пакет с классами-сущностями, один – с граничными классами, один – с управляющими и т.д. (рис. 5.2). Таким образом, структура пакетов будет соответствовать многоуровневой архитектуре системы и можно использовать для пакетов стереотип «layer» (слой).

Второй подход заключается в объединении классов по их функциональности. Например, пакеты Безопасность, Подготовка отчетов, Обработка ошибок и т.д. В этом случае структура пакетов может соответствовать структуре подсистем и можно использовать стереотип «subsystem» (подсистема) для этих пакетов (рис. 5.3).

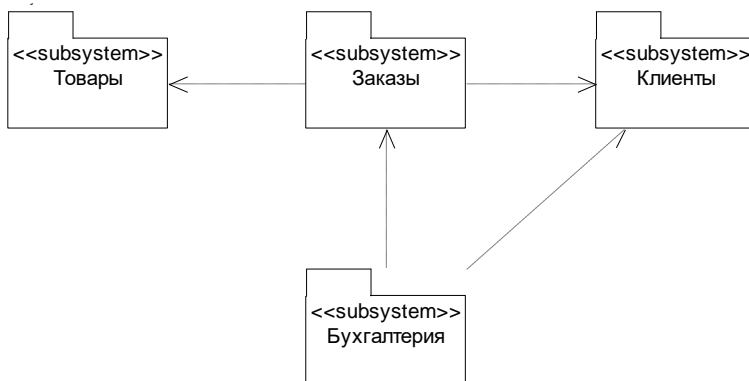


Рис. 5.3. Диаграмма пакетов-подсистем

Также применяют комбинацию этих двух подходов. На высоком уровне можно сгруппировать классы по функциональности, а внутри каждого пакета создать другие пакеты, сгруппировав соответствующие классы по стереотипу.

Класс проектирования описывается на том же языке, на котором будет реализован. Соответственно, операции, параметры, атрибуты, типы и другие подробности определяются с использованием синтаксиса выбранного языка программирования. Задается видимость атрибутов и операций класса проектирования. Например, в языке C++ для этого обычно используются ключевые слова `public`, `protected` и `private`.

Отношения с другими классами получают явное выражение, отображаясь на соответствующие атрибуты классов (рис. 5.4). Класс проектирования часто задается стереотипом, который напрямую отображается в конструкцию соответствующего языка программирования.

Методы (то есть реализации операций) класса проектирования прямо отображаются на соответствующий код. Методы, определяемые в ходе проектирования, часто определяются на естественном языке или на псевдокоде и могут быть в таком виде использованы в аннотации к реализации этих методов (модель реализации).

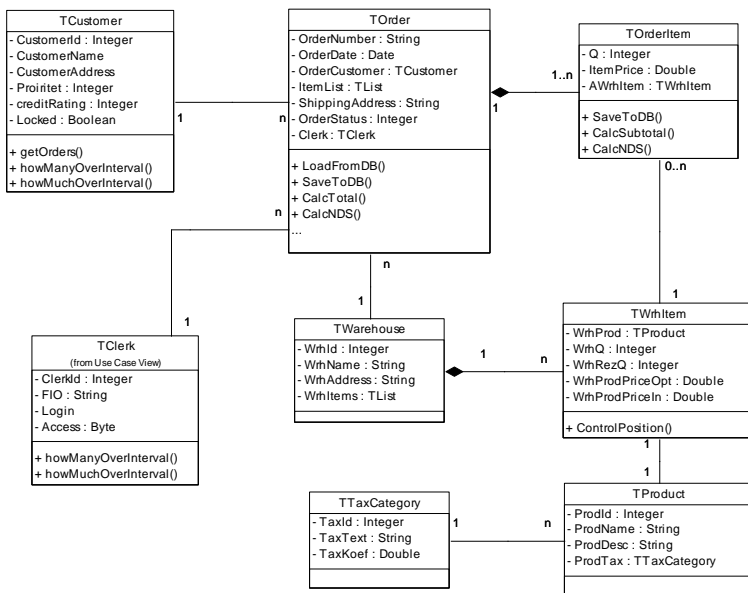


Рис. 5.4. Диаграмма классов уровня проектирования (модель спецификации) для системы приема заказов

Определение атрибутов

На этом этапе определяются атрибуты, необходимые классу проектирования и описываются, используя синтаксис языка программирования. Атрибут определяет свойство класса проектирования и часто участвует в операциях класса.

Атрибут (attribute)– фрагмент информации, связанной с классом.

При определении атрибутов класса проектирования следует учитывать следующие общие соображения.

- Атрибуты соответствующего класса анализа соответствуют одному или нескольким атрибутам класса проектирования.
- Доступные типы атрибутов выбираются из языка программирования.
- Часто имеется однозначное соответствие между таблицами базы данных и классами-сущностями, и соответственно между атрибутами и полями.

- Если класс из-за своих атрибутов становится слишком сложным для понимания, некоторые из этих атрибутов можно выделить и переопределить в виде отдельных классов.
- Если у класса имеется множество атрибутов или они чересчур сложны, его можно проиллюстрировать отдельной диаграммой класса, на которой изображается только та информация, которая относится к атрибутам.

Определение операций

На этом шаге определяются операции, которые должен выполнять класс проектирования. Эти операции необходимо описать с использованием синтаксиса языка программирования. Сюда входит и определение уровня видимости каждой из операций (например, `public`, `protected` и `private` в C++). Исходными данными для этого послужат диаграммы взаимодействия модели анализа.

Операция – связанное с классом поведение. Операция состоит из трех частей – имени, параметров и возвращаемого значения. Имя операции совместно с ее параметрами и типом возвращаемого значения (если таковое имеется) называют *сигнатурой* операции.

Операция изображается текстовой строкой, имеющей следующую грамматику:

<признак видимости><имя>(список параметров):<тип возвращаемого значения>

На некоторых диаграммах полезно показывать полную сигнатуру операций, но можно ее скрыть и оставить только имена.

Типы операций

Операции реализации – реализуют некоторую бизнес-функциональность. Такие операции можно найти, исследуя диаграммы взаимодействия. Эти диаграммы сфокусированы на бизнес-функциональности и каждое сообщение можно соотнести с соответствующей операцией реализации. Необходимо, чтобы каждую операцию реализации можно было проследить до соответствующего требования.

Операции управления – управляют созданием и уничтожением объектов (конструкторы и деструкторы).

Операции доступа – позволяют безопасно инкапсулировать атрибуты внутри класса, защитив их от других классов, но все же осуществлять контролируемый доступ к ним. Атрибуты бывают обычно закрытыми или защищенными, но другие классы иногда должны их просматривать или изменять. Создание операций Get и Set для каждого атрибута класса является промышленным стандартом.

Вспомогательные операции – необходимы для выполнения обязанностей класса, но другим классам о них знать не нужно. Это закрытые и защищенные операции.

Описание операции

Описание операции выполняется в окне спецификации операции на соответствующих вкладках (рис.5.5) и определяет изменение состояния всей системы при ее выполнении. Обычно это выполняется в описательном стиле, и внимание акцентируется на том, *что* должно произойти, а не *как* этого достичь.

Семантика (semantics) – описание того, что делает операция. По сути, это комментарий к операции.

Постусловия (Postconditions)– декларируются изменения, происходящие в системе в результате выполнения операции, т.е. определение того, как будет выглядеть окружающий мир после того, как будет выполнена операция.

Основные категории постусловий:

- создание и удаление экземпляра;
- модификация атрибута;
- формирование и разрыв ассоциаций.

Постусловие – это хороший способ выразить, что должно быть сделано, не говоря при этом, как это сделать, – другими словами, отделить интерфейс от реализации.

Предусловие (precondition)– это предположения о состоянии системы до начала операции. Для каждой операции можно определить множество предусловий, однако наиболее целесообразно декларировать следующие типы предусловий.

- Факторы, которые необходимо проверять в программе при выполнении операции.
- Непроверяемые факторы, от которых зависит успешное выполнение операций (такие предположения отмечаются в описании для будущих пользователей для оповещения их о значимости для данной операции).

Описание классов

Проектная модель передается программистам для фактического создания исходного кода. Поэтому классы проектирования моделируются с высокой степенью детализации и должны быть достаточно подробно документированы (табл. 5.1).

Таблица 5.1. Формат описания классов

Класс <Название класса>			
<Назначение класса>			
Атрибуты			
Видимость	Наименование атрибута	Тип	Описание
Операции			
Видимость	Наименование операции	Описание	

При описании операции указывается ее полная сигнатура – имена и типы всех параметров, а также тип возвращаемого значения, если необходимо.

Если в процессе определения и описания всех атрибутов и операций класса он становится слишком большим, то его необходимо разбить на два или более меньших класса с четко сфокусированными обязанностями.

В процессе описания определяется также, соответствует ли класс основным характеристикам правильно сформированного класса проектирования. Контролируются:

- полнота и достаточность;
- простота;
- высокое внутреннее сцепление;
- низкая связанность с другими классами.

Диаграммы взаимодействия

На этапе проектирования диаграммы взаимодействия становятся более конкретными и приближенными к программной реализации (рис. 5.5).

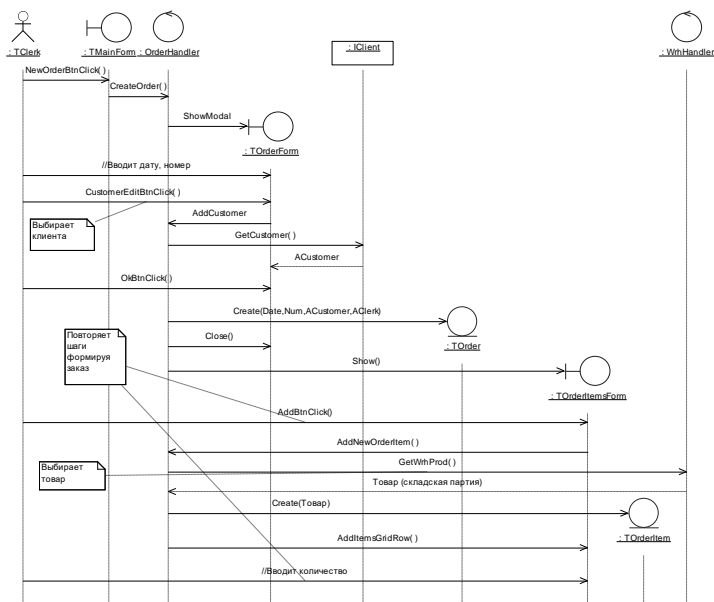


Рис. 5.5. Пример диаграммы последовательности уровня проектирования для варианта использования «Оформить заказ» (фрагмент)





Каждому объекту на диаграмме взаимодействия должно быть дано уникальное имя. В отличие от имен классов, которые, как правило, носят общий характер (Клиент, Заказ), имена объектов конкретны (Фирма «Луч», Заказ №374). На диаграмме взаимодействия может быть два и более объекта, являющихся экземплярами одного и того же класса. Для каждого объекта можно задать его устойчивость (табл. 5.2).

Таблица 5.2. Варианты устойчивости объектов

Тип	Определение
Устойчивый (Persistent)	Устойчивый объект сохраняется в базе данных или другим способом, обеспечивающим постоянное хранение. Такой объект будет существовать даже после прекращения работы программы
Статичный (Static)	Статичный объект сохраняется в памяти компьютера в течение всего времени работы программы. Он продолжает существование после выполнения отраженных на диаграмме взаимодействия действий, но не сохраняется после прекращения работы программы
Временный (Transient)	Временный объект сохраняется в памяти в течение очень короткого времени (например, пока не закончится выполнение изображенных на диаграмме взаимодействия действий)

На диаграмме взаимодействия уровня проектирования все операции соответствуют вызову операции соответствующего класса. При этом можно показать синхронизацию посылаемых сообщений (табл. 5.3).

Таблица 5.3. Синхронизация сообщений

Тип	Определение	Обозначение
Простое (Simple)	Используется по умолчанию. Все сообщения выполняются в одном потоке управления	
Синхронное (Synchronous)	Объект-отправитель посылает сообщение и ждет ответа объекта-получателя	
С отказом становиться в очередь (Balking)	Объект-отправитель посылает сообщение объекту-получателю. Если получатель не может немедленно принять сообщение, оно отменяется	
С лимитированным временем ожидания (Timeout)	Объект-отправитель посылает сообщение объекту-получателю, затем ждет указанное время. Если в течение этого времени объект-получатель не принимает сообщение, оно отменяется	

Продолжение табл. 5.3

Тип	Определение	Обозначение
Процедурный вызов (Procedure call)	Вызывает операцию, применимую к объекту. При этом объект передает управление до момента возврата. Вся вложенная последовательность действий должна завершиться, прежде чем возобновится внешняя последовательность. Объект может послать сообщение самому себе, что приведет к локальному вызову операции	
Асинхронное (Asynhronous)	Объект-отправитель посылает сообщение объекту-получателю, и продолжает свою работу, не ожидая подтверждения о получении	
Возврат (Return)	Возврат из процедурного вызова. Хвост стрелки заканчивает активацию или означает уничтожение объекта	

Обычно *простые последовательности* применяют только при моделировании взаимодействий в контексте вариантов использования, относящихся к системе в целом, включая актеров вне системы. Такие последовательности зачастую оказываются простыми, поскольку управление передается от шага к шагу без учета вложенных потоков управления. Почти во всех остальных случаях применяют *процедурные последовательности*, поскольку с их помощью представляются вложенные вызовы операций, примеры которых можно найти в большинстве языков программирования.

Интерфейсы

При создании программного обеспечения важно строить системы с четким разделением задач так, чтобы при развитии системы изменения в одной ее части не затронули другие. Для достижения этой цели необходимо специфицировать стыковочные узлы системы, к которым присоединяются эти

независимо изменяемые части. В UML для моделирования стыковочных узлов в системе используются интерфейсы.

Интерфейс (interface) – это набор операций, специфицирующих услуги, предоставляемые классом или компонентом. Объявляя интерфейс, декларируется желаемое поведение класса (компонента), не зависящее от его реализации. Другие разработчики будут полагаться на объявленный интерфейс, а впоследствии, создав лучшую реализацию, можно будет заменить старую.

Зачастую интерфейс описывает только небольшую часть поведения класса. Класс, в свою очередь, может поддерживать несколько интерфейсов, причем как независимых, так и пересекающихся между собой. У интерфейсов нет ни реализации, ни атрибутов, ни состояний, ни ассоциаций, только операции.

Все операции интерфейса имеют открытую видимость. Включать в интерфейс операции с другой видимостью не имеет смысла, так как у него нет никакой внутренней структуры, где бы они могли использоваться.

Интерфейсы – это открытые функции, которые могут быть вызваны из-за пределов реализующих их классов (компонентов). Интерфейс определяет имя функции, ее параметры (входные или выходные), соответствующие им типы данных, необязательные параметры, а также тип возвращаемого значения.

Существует два способа изображения интерфейсов на диаграммах.

1. *Компактная форма* – маленький кружок, под которым написано его имя (рис. 5.6). Интерфейс редко существует сам по себе – обычно он присоединяется к реализующему его классу или компоненту.

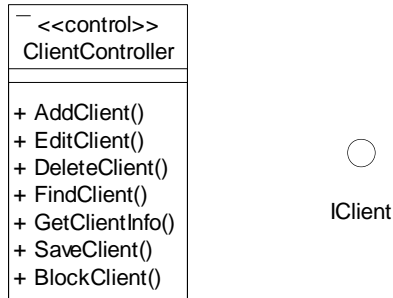


Рис. 5.6. Компактная нотация интерфейса класса

2. *Развернутая форма* – класс со стереотипом «interface», в соответствующем разделе которого указаны операции (рис. 5.7). При этом можно показывать только имя операции или же предоставить развернутое описание с сигнатурой и другими свойствами. Интерфейс связывается с реализующим его классом или компонентом отношением реализации. В UML отношение реализации изображают в виде пунктирной линии с большой незакрашенной стрелкой, направленной в сторону интерфейса. В этой нотации суммируются обобщение и зависимость.

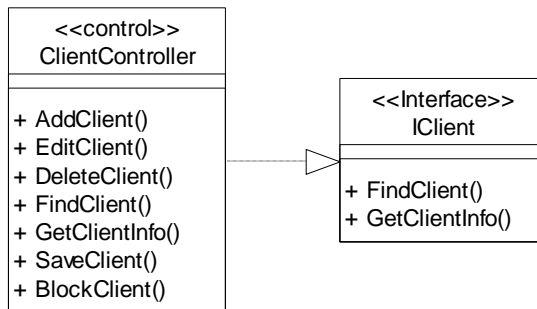


Рис. 5.7. Развернутая нотация интерфейса класса.

В обоих случаях класс или компонент, *использующий* интерфейс, соединяют с ним *отношением зависимости* (рис. 5.8).

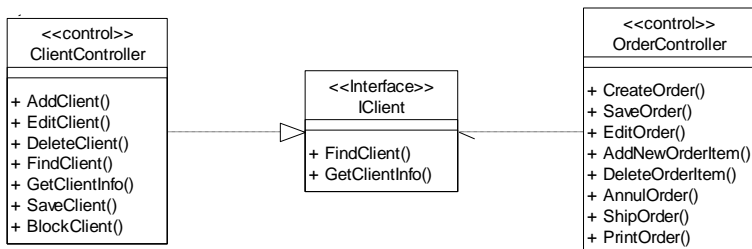


Рис. 5.8. Класс, использующий интерфейс, связывается с ним отношением зависимости

У любого интерфейса должно быть имя, отличающее его от остальных и уникальное внутри пакета, в который входит данный интерфейс. Имя интерфейса может состоять из любого числа букв, цифр и некоторых знаков препинания. Обычно для именования интерфейсов используют одно или несколько коротких существительных, взятых из словаря моделируемой предметной области. Чтобы отличить интерфейс от класса, принято добавлять к его имени начальную букву I, например, *IUnknown* или *ISpelling*.

Чаще всего с помощью интерфейсов моделируют стыковочные узлы в системе, состоящей из программных компонентов.

Диаграмма компонентов

Компонент (Component) – это физически заменяемая часть системы, реализующая некоторый набор интерфейсов.

Фактически компонент представляет собой физическую упаковку логических элементов, таких как классы и интерфейсы. Компоненты используются для моделирования физических сущностей: исполняемых модулей, библиотек, таблиц, файлов и документов. Обычно компоненты преобразуются в исполняемые файлы, классы Java, статические или динамически связываемые библиотеки (DLL – dynamic link library). В системе могут быть различные компоненты, такие как COM или Java Beans, а также компоненты, являющиеся артефактами процесса разработки, например файлы исходного кода.

Для отображения интерфейсов, реализуемых компонентом, обычно используют свернутую компактную форму (рис. 5.9).

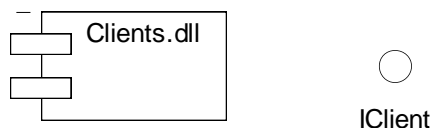


Рис. 5.9. Компонент с интерфейсом

Интерфейс, реализуемый компонентом, называется *экспортируемым* интерфейсом. Это означает, что компонент через данный интерфейс предоставляет ряд услуг другим компонентам. Компонент может экспортировать много интерфейсов. Интерфейс, которым компонент пользуется, называется *импортируемым* интерфейсом. Это означает, что компонент совместим с таким интерфейсом и зависит от него при выполнении своих функций. Компонент может импортировать различные интерфейсы, причем ему разрешается одновременно импортировать и экспортировать интерфейсы.

Диаграмма компонентов (Component diagram) – физическое представление модели, содержит набор компонентов, интерфейсов и отношения между ними (рис. 5.10).

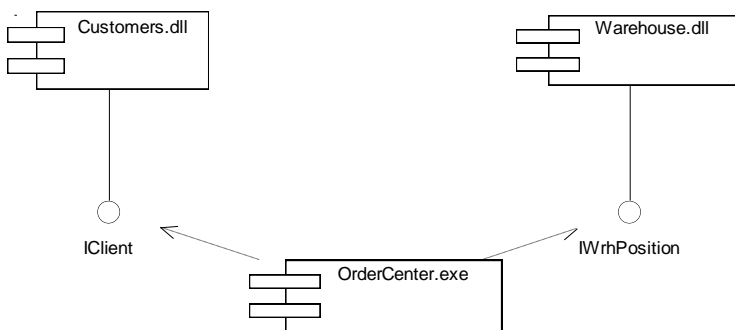


Рис. 5.10. Диаграмма компонентов

Отношения между компонентами показываются с помощью связей зависимости.

При моделировании статического вида с точки зрения реализации диаграммы компонентов, как правило, используются в трех случаях.

1. Моделирование исходного кода – визуализации зависимостей между различными файлами при

компиляции, а также для контроля версий и управления конфигурацией.

2. Моделирование исполняемых версий – визуализация компонентов развертывания, составляющих дистрибутив системы (исполняемые файлы, файлы данных, файлы оперативной подсказки, скрипты на каком-нибудь интерпретируемом языке, файлы журналов, инициализационные файлы и протоколы установки или удаления системы).
3. Моделирование физических баз данных

Диаграмма развертывания

Диаграмма развертывания (Deployment diagram) показывает конфигурацию узлов, на которых выполняется система и компонентов, размещенных в этих узлах.

Диаграмма развертывания представляет собой схему, состоящую из символов узлов, соединенных маршрутами коммуникационных каналов-ассоциаций.



Узел (Node) – это физический элемент, существующий во время выполнения приложения и представляющий собой тип вычислительного устройства – в большинстве случаев часть аппаратуры.

Эта аппаратура может быть и простым устройством или датчиком, а может быть и большим компьютером.

Графически узел изображается в виде куба. Это каноническое обозначение позволяет визуализировать узел, не конкретизируя стоящей за ним аппаратуры. С помощью стереотипов – одного из механизма расширения UML – можно адаптировать эту нотацию для представления конкретных процессоров и узлов.

Процессор (Processor) – это узел, способный обрабатывать данные, то есть исполнять компонент или процесс (серверы, рабочие станции и другие устройства, содержащие физические процессоры).

Устройство (Device) – это узел, не способный обрабатывать данные (различные датчики, модемы, терминалы и других периферийные устройства).

В узлах можно показывать развертывание

- программных компонентов;

- процессов.

Процесс – в данном контексте, это группа функций программной системы, обычно соответствующая одному из вариантов использования. Несколько процессов могут параллельно выполняться на разных узлах с помощью одной и той же программы.

Процессы можно показывать или не показывать на диаграмме развертывания. В первом случае они отображаются непосредственно под процессором, на котором выполняются (рис. 5.12).

Диаграмма развертывания позволяет разработчикам архитектуры визуализировать топологию системы и отобразить компоненты на исполняемые процессы. При этом учитываются следующие вопросы: процессорная архитектура, скорость, емкость, пропускная способность каналов для взаимодействия процессов, физическое расположение аппаратных ресурсов, технология распределенной обработки.

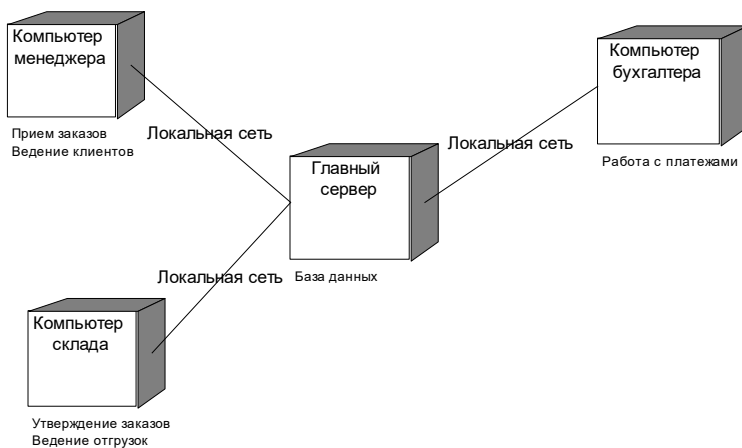


Рис. 5.11. Диаграмма развертывания с отображением процессов

Список литературы

1. Фаулер, М. UML. Основы /М. Фаулер. – СПб.: Символ Плюс, 2009.
2. Ларман, К. Применение UML 2.0 и шаблонов проектирования /К. Ларман. – М.: «Вильямс», 2006.

3. Скотт, К. UML. Основные концепции /К. Скотт. – М. «Вильямс», 2002.
4. Буч, Г. Язык UML. Руководство пользователя /Г. Буч, Д. Рамбо, А. Джекобсон. – М.: ДМК-Пресс, 2007.
5. Рамбо, Дж. UML 2.0. Объектно-ориентированное моделирование и разработка /Дж. Рамбо, М.Блаха. СПб.: . – Питер, 2007.
6. Рамбо, Дж. UML: Специальный справочник /Дж. Рамбо, Г. Буч, А. Джекобсон. – СПб.: Питер, – 2002.
7. Арлоу Дж. UML 2 и Унифицированный процесс /Дж Арлоу, А. Нейштадт. – СПб.: Символ Плюс, 2008.
8. Кватрани, Т. Rational Rose 2000 и UML. Визуальное моделирование /Т. Кватрани. – М.: ДМК Пресс, 2001.
9. Боггс, Ч. UML и Rational Rose /Ч. Боггс, М. Боггс. – М.: Лори, - 2000.

Задания

1. Структурировать модель, сгруппировав классы по пакетам.
2. Построить диаграммы классов уровня проектирования, определив атрибуты и операции. Описать классы.
3. Определить необходимые интерфейсы классов.
4. Создать реализацию (кооперацию) выбранного варианта использования.
5. Построить диаграмму последовательности уровня проектирования выбранного варианта использования.
6. Построить диаграмму компонентов и распределить классы по компонентам.
7. Построить диаграмму развертывания, определив процессы в каждом узле.
8. Оформить отчет по лабораторной работе. Отчет должен включать:
 - постановку задачи, краткое описание проектируемой программной системы;
 - диаграмму пакетов;
 - диаграммы классов уровня проектирования по пакетам;
 - описания классов;
 - диаграмму последовательности;
 - диаграмму компонентов;

- диаграмму развертывания;
- выводы по работе.

Контрольные вопросы

1. С какой целью строится модель проектирования при разработке программной системы?
2. Какие составные части входят в модель проектирования?
3. Чем модель проектирования отличается от модели анализа?
4. В чем суть архитектурного проектирования?
5. С помощью каких средств UML может быть показана архитектура разрабатываемой системы?
6. Какие виды операций вам известны?
7. Поясните общий синтаксис представления операции?
8. Какие уровни видимости вы знаете? В чем их смысл?
9. Что такое компонент? Чем он отличается от пакета?
10. Что такое интерфейс? Чем он полезен?
11. Какие существуют формы представления интерфейса?
12. Какой может быть связь между классом и интерфейсом?
13. Что представляет собой сообщение на диаграмме взаимодействия уровня проектирования?
14. Как показывается объект класса на диаграмме взаимодействия?
15. Какие диаграммы относятся к физическому представлению системы? Чем они отличаются от логического представления?
16. Для чего используются диаграммы компонентов?
17. Какими отношениями связаны элементы на диаграмме компонентов?
18. Какие элементы входят в диаграмму развертывания?
19. Что представляют собой связи между элементами на диаграммах развертывания?
20. Как применяют диаграммы развертывания?
21. Как диаграммы компонентов могут быть связаны с диаграммами развертывания?

Заключение

Современная программная инженерия – молодая и динамично развивающаяся научная дисциплина. Главное ее назначение – поставка заказчику высококачественных программных продуктов с заданными потребительскими и эксплуатационными свойствами. Поэтому особое внимание уделяется систематическим, управляемым и эффективным методам анализа, оценки, спецификации, проектирования и конструирования программ.

В рамках данного учебно-методического пособия были рассмотрены современные методы и средства, применяемые на этапах анализа и проектирования ПО. В качестве дополнительного источника информации по каждой теме рекомендуется использовать предложенную литературу.

Содержание

ВВЕДЕНИЕ.....	3
ТЕМА 1. БИЗНЕС-МОДЕЛИРОВАНИЕ. МЕТОДОЛОГИИ ФУНКЦИОНАЛЬНОГО МОДЕЛИРОВАНИЯ IDEF0 И IDEF3.....	5
Методология функционального моделирования IDEF0	5
<i>Правила построения диаграмм IDEF0.....</i>	<i>13</i>
Методология IDEF3.....	14
<i>Правила построения диаграмм IDEF3.....</i>	<i>17</i>
СПИСОК ЛИТЕРАТУРЫ	17
ЗАДАНИЯ	18
КОНТРОЛЬНЫЕ ВОПРОСЫ.....	18
ТЕМА 2. СТРУКТУРНЫЙ СИСТЕМНЫЙ АНАЛИЗ. МЕТОДОЛОГИИ DFD И IDEF1X	20
Методология DFD	21
<i>Правила построения диаграмм DFD.....</i>	<i>25</i>
Методология IDEF1X	25
<i>Выбор первичного ключа.....</i>	<i>28</i>
<i>Построение физической модели.....</i>	<i>29</i>
СПИСОК ЛИТЕРАТУРЫ	30
ВАРИАНТЫ ПРОЕКТОВ	30
ЗАДАНИЯ	30
КОНТРОЛЬНЫЕ ВОПРОСЫ.....	31
ТЕМА 3. АНАЛИЗ ТРЕБОВАНИЙ. МЕТОДОЛОГИЯ UML. МОДЕЛЬ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ	32
Методология UML	32
Модель вариантов использования.....	35
<i>Диаграмма вариантов использования (Use Case Diagram).....</i>	<i>36</i>
<i>Спецификация варианта использования</i>	<i>39</i>
<i>Диаграмма деятельности (Activity Diagram).....</i>	<i>43</i>
СПИСОК ЛИТЕРАТУРЫ	47
ЗАДАНИЯ	47
КОНТРОЛЬНЫЕ ВОПРОСЫ.....	48
ТЕМА 4. ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ АНАЛИЗ. МЕТОДОЛОГИЯ UML. МОДЕЛЬ АНАЛИЗА	49
Модель анализа	49
<i>Диаграмма классов</i>	<i>50</i>

<i>Диаграмма состояний</i>	56
<i>Анализ реализации варианта использования</i>	58
<i>Диаграмма коммуникации</i>	59
<i>Диаграмма последовательности</i>	61
СПИСОК ЛИТЕРАТУРЫ	63
ЗАДАНИЯ	63
КОНТРОЛЬНЫЕ ВОПРОСЫ	64
ТЕМА 5. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ.	
МЕТОДОЛОГИЯ UML. МОДЕЛЬ ПРОЕКТИРОВАНИЯ	66
МОДЕЛЬ ПРОЕКТИРОВАНИЯ	66
<i>Диаграммы взаимодействия</i>	74
<i>Диаграмма компонентов</i>	79
<i>Диаграмма развертывания</i>	81
СПИСОК ЛИТЕРАТУРЫ	82
ЗАДАНИЯ	83
КОНТРОЛЬНЫЕ ВОПРОСЫ	84
ЗАКЛЮЧЕНИЕ	85

Левенец Ирина Анатольевна
Технология разработки программного обеспечения.
Анализ и проектирование
Учебно-методическое пособие

Редактор Т.В. Соловьева

Подписано в печать. Формат 60х84 1/16.
Печать офсетная. Усл. печ. л. 5,1. Уч. – изд. л. 5,9. Тираж 100 экз. Заказ
ГОО ВПО “Ивановский государственный энергетический университет
имени В.И. Ленина”
153003, г. Иваново, ул. Рабфаковская, 34