КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И.Раззакова

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра **«ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ КОМПЬЮТЕРНЫХ СИСТЕМ»**

ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРАКТИЧЕСКИХ ЗАНЯТИЯМ

Для студентов специализации
552801.04 «Программное обеспечение вычислительной техники и автоматизированных систем»

Бишкек – 2008

РАССМОТРЕНО
На заседании кафедры
«Программное обеспечение
компьютерных систем»
Прот.№13 от 27.05.2008г.

ОДОБРЕНО
Методическим советом ФИТ
Прот. № 10 от 23.06.2008г.

УДК 681.3.06

Составители – ст. преп. МУСИНА И.Р., КАРИМОВА Г.Т.

Технология разработки программного обеспечения: Методические указания к практическим работам /Кырг. гос. техн. ун-т, Бишкек, 2007, 61с.

Представлены теоретические сведения, примеры применения современных методов проектирования программного обеспечения, задания для выполнения практических работ.

Предназначено для студентов специальности «Программное обеспечение вычислительной техники и автоматизированных систем» всех форм обучения.

Ил. 46. Библиогр. 6 названий.

Рецензент: зав. каф. «Информационные системы в экономике» Кыргызского государственного технического университета им. И.Раззакова, д.т.н. Бабак В.Ф.

Введение

Проектирование программного обеспечения (ПО) - это логически сложная, трудоемкая и длительная работа, требующая высокой квалификации участвующих в ней специалистов. Освоение и правильное применение методов и средств создания ПО позволят повысить его качество, обеспечить управляемость процесса его проектирования и увеличить срок его жизни. Основная идея этих методов и средств заключается в применении инженерного подхода к проектированию программ, которое понимается как процесс создания проекта программного изделия, во многом аналогичный процессу создания промышленной продукции.

Для успешной реализации проекта объект проектирования (ПО) должен быть прежде всего адекватно описан, т.е. должны быть построены полные и непротиворечивые модели архитектуры ПО. Под моделью понимается полное описание системы ПО с определенной точки зрения. Модели представляют собой средства для визуализации, описания, проектирования и документирования архитектуры ПО. По мнению одного из авторитетнейших специалистов в области объектно-ориентированного подхода Гради Буча, моделирование является центральным звеном всей деятельности по созданию качественного ПО. Модели строятся для того, чтобы понять и осмыслить структуру и поведение будущей системы, облегчит управление процессом ее создания и уменьшить возможный риск, а также документировать принимаемые проектные решения. Хорошие модели являются основой взаимодействия участников проекта и гарантируют корректность архитектуры. Поскольку сложность систем повышается, важно располагать эффективными методами моделирования.

На сегодняшний день в программной инженерии существует два основных подхода к проектированию программного обеспечения. Принципиальное различие между ними обусловлено разными способами декомпозиции системы.

Первый подход называется структурным или функционально- модульным. В его основу положен принцип функциональной декомпозиции, при которой структура системы описывается в терминах иерархии ее функций и передачи информации между отдельными функциональными элементами. Система разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, подразделяемые на задачи, и т. д. Процесс разбиения продолжается вплоть до конкретных процедур. При этом создаваемая система сохраняет целостное представление, в котором все составляющие компоненты взаимоувязаны.

Второй, объектно - ориентированный подход использует объектную декомпозицию. При этом структура системы описывается в терминах объектов и связи между ними, а поведение системы описывается в терминах обмена сообщениями между объектами.

Разделение по алгоритмам концентрирует внимание на порядке происходящих событий, а разделение по объектам придает особое значение агентам, которые являются либо объектами, либо субъектами действия. Однако эти способы, по сути, ортогональны, поэтому нельзя сконструировать сложную систему одновременно двумя способами. Необходимо начать разделение системы либо по алгоритмам, либо по объектам, а затем, используя полученную структуру, попытаться рассмотреть проблему с другой точки зрения.

Алгоритмическую декомпозицию можно представить как обычное разделение алгоритмов, где каждый модуль системы выполняет один из этапов общего процесса.

При объектно-ориентированной декомпозиции каждый объект обладает своим собственным поведением и каждый из них моделирует некоторый объект реального мира. С этой точки зрения объект является вполне осязаемой вещью, которая демонстрирует вполне определенное поведение. Объекты что-то делают, и мы можем, послав им сообщение, попросить их выполнить некоторые операции.

Главный недостаток структурного подхода заключается в следующем: процессы и данные существуют отдельно друг от друга (как в модели деятельности организации, так и в модели ПО), причем проектирование ведется от процессов к данным. Таким образом, помимо функциональной декомпозиции существует также структура данных, находящаяся на втором плане.

В объектно – ориентированном подходе основная категория объектной модели – класс - объединяет в себе на элементарном уровне как данные, так и операции, которые над ними выполняются (методы). Разделение процессов и данных преодолено, однако остается проблема сложности системы, которая решается путем использования механизмов компонентов.

Данные по сравнению с процессами являются более стабильной и относительно редко изменяющейся частью системы. Гради Буч отмечает ряд преимуществ объектной декомпозиции перед алгоритмической [1].

- Объектная декомпозиция уменьшает размер программных систем за счет повторного использования общих механизмов, что приводит к существенной экономии выразительных средств.
- Объектно-ориентированные системы более гибки и проще эволюционируют со временем, потому что их схемы базируется на устойчивых промежуточных формах. Действительно, объектная декомпозиция существенно снижает риск при создании сложной программной системы, так как она развивается из меньших систем, в которых мы уже уверены.
- Объектная декомпозиция помогает нам разобраться в сложной программной системе, предлагая нам разумные решения относительно выбора подпространства большого пространства состояний.

С другой стороны, диаграммы, отражающие специфику объектного подхода (диаграммы классов и т.п.), гораздо менее наглядны и плохо понимаемы

непрофессионалами. Поэтому одна из главных целей внедрения CASE – технологий, а именно снабжение всех участков проекта (в том числе и заказчика) общим языком «для передачи понимания», обеспечивается на сегодняшний день только структурными методами [2.].

Структурный анализ может использоваться как основа для объектно – ориентированного программирования. В некоторых ситуациях иной подход просто невозможен. При этом структурный анализ следует прекращать, как только диаграммы потоков данных начнут отражать не только деятельность организации (предметную область), а и ПО.

Для автоматизации процесса разработки сложных информационных систем в настоящее время широко используются CASE-средства, представляющие собой специальные программы, которые поддерживают одну или несколько методологий анализа и проектирования ПО. CASE-технология, в рамках методологии, включает в себя методы, с помощью которых на основе нотаций строятся диаграммы, поддерживаемые конкретным CASE-средством. После выполнения структурного анализа и построения диаграмм потоков данных вместе со структурами данных и другими продуктами анализа можно различными способами приступать к определению классов и объектов. Так, если взять какую – либо отдельную диаграмму, то кандидатами в объекты могут быть внешние сущности накопители данных, а кандидатами в классы – потоки данных.

Современные CASE-средства охватывают обширную область поддержки многочисленных технологий проектирования информационных систем: от простых средств анализа и документирования до полномасштабных средств автоматизации, покрывающих весь жизненный цикл программного обеспечения. Наиболее трудоемкими этапами разработки информационных систем являются этапы анализа и проектирования, в процессе которых CASE-средства обеспечивают качество принимаемых технических решений и подготовку проектной документации. При этом большую роль играют методы визуального представления информации.

Настоящий курс практических работ посвящен средствам структурного и объектно-ориентированного анализа.

Цель данных методических указаний помочь студентам в освоении:

- современных методов и средств проектирования программного обеспечения, основанных на использовании структурного и объектно-ориентированного подходов;
- технологии составления диаграмм по стандартам DFD, IDEF0, IDEF3, IDEF1X, UML;
- технологии построения моделей с помощью CASE – средств как структурного, так и объектно – ориентированного анализа: BPWIN, ERWIN, MS VISIO;

Методические указания направлены не только на обучение студентов методам проектирования ПО, но и формирование навыков их самостоятельного практического применения.

Тема: СТРУКТУРНЫЙ ПОДХОД К ПРОЕКТИРОВАНИЮ

В структурном подходе используются:

- IDEF0 – Icam DEFinition - модели и соответствующие функциональные диаграммы;
- DFD - Data Flow Diagrams – диаграммы потоков данных;
- ERD (Entity- Relationship Diagrams) - диаграммы «сущность-связь».

На стадии формирования требований к ПО IDEF0 и DFD используются для построения модели AS-IS и модели TO-BE, отражая, таким образом, существующую и предлагаемую структуру бизнес-процессов организации и взаимодействия между ними. С помощью ERD выполняется описание используемых в организации данных на концептуальном уровне, не зависимом от реализации базы данных.

На стадии проектирования DFD используют для описания проектируемой системы ПО, при этом они могут уточняться и дополняться новыми конструкциями. Аналогично ERD уточняются и дополняются новыми конструкциями, описывающими представление данных на логическом уровне, пригодном для последующей генерации схемы базы данных.

Цель практических работ по данной теме: Обучить студентов структурному подходу к проектированию ПО, привить студентам навыки абстрагирования и функционального моделирования системы с помощью CASE BPWIN и ERWIN

Практическое занятие №1. Построение модели процессов IDEF0

Построение иерархии диаграмм в IDEF0

1. Процесс моделирования начинается с определения контекста, т.е. наиболее абстрактного уровня описания системы в целом. В контекст входит определение субъекта моделирования, цели и точки зрения на модель. Под субъектом понимается сама система, при этом точно надо установить, что входит в систему, а что лежит за ее пределами. Самое общее описание системы и ее взаимодействие с внешней средой, называется контекстной диаграммой. Таким образом, построение модели начинается с представления всей системы в виде простейшего компонента – одного блока и дуг, изображающих интерфейсы с функциями вне системы.

2. После описания системы в целом производится разбиение ее на крупные фрагменты. Этот процесс называется функциональной декомпозицией, а диаграммы, которые описывают каждый фрагмент и взаимодействие фрагментов, называются диаграммами декомпозиции.

3. После декомпозиции контекстной диаграммы производится декомпозиция каждого большого фрагмента системы на более мелкие и так далее до достижения нижнего уровня подробности описания.

4. После каждого сеанса декомпозиции проводятся сеансы экспертизы – эксперты предметной области указывают на соответствие реальных бизнес - процессов созданным диаграммам. Найденные соответствия исправляются и

только после прохождения экспертизы без замечаний можно приступать к следующему сеансу декомпозиции.

Состав функциональной модели

Синтаксис описания системы в целом и каждого ее фрагмента одинаков во всей модели. **Работы (Activity)**, которые означают некие поименованные процессы, функции или задачи, изображаются в виде прямоугольников. Именем работы должен быть глагол или глагольная форма (например, «Изготовление детали», «Прием заказа»).

Взаимодействие работы с внешним миром и между собой описывается в виде **стрелок.** Стрелка служит для обозначения некоторого носителя или воздействия, которое обеспечивает перенос данных или объектов от одной деятельности к другой. Стрелки представляют собой некоторую информацию и именуются существительными (например, «Заготовка», «Изделие», «Заказ»). В IDEF0 различают 5 видов стрелок:

Вход (Input) – материал или информация, которая используется или преобразовывается работой.

Управление (Control) – правила, стратегии**,** процедуры или стандарты, которыми руководствуется работа (это может быть ограничение на выполнение операции процесса). Каждая работа должна иметь хотя бы одну стрелку управления.

Выход (Output) – материал или информация, которая производится работой. Каждая работа должна иметь хотя бы одну стрелку выхода.

Механизм (Mechanism) – ресурсы, которые выполняют работу; например, персонал предприятия, станки, механизмы, автоматизированная система и т.д.

Вызов – специальная стрелка, указывающая на другую модель работы. (рис.1.1)

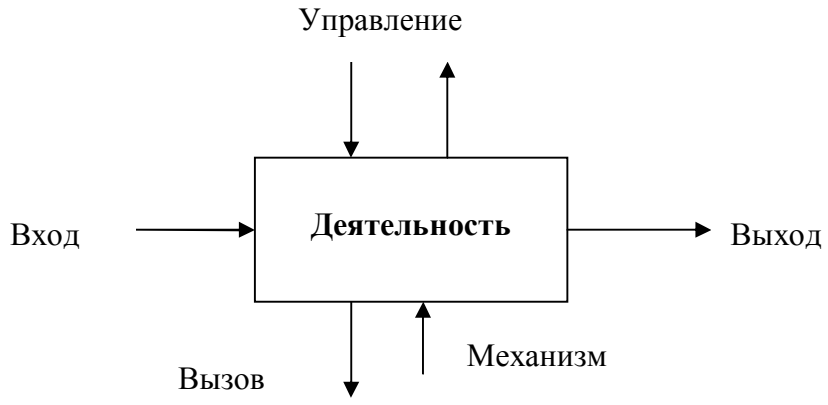


Рис.1.1 Функциональный блок и интерфейсные дуги.

Пример 1.1. Описание процесса оформления кредита в банке**.**

На рис.1.2 представлен контекстная диаграмма процесса оформления кредита в банке.

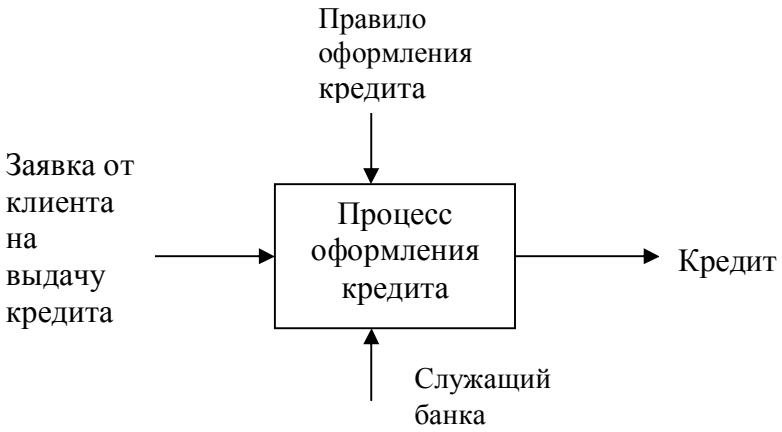


Рис.1.2 Представление процесса оформления кредита в банке на контекстной диаграмме IDEFF0

Пример 1.2. Построение функциональной модели деятельности предприятия торговли.

На рис.1.3 изображена контекстная диаграмма IDEFF0, построенная в среде CASE – средства BPWIN.

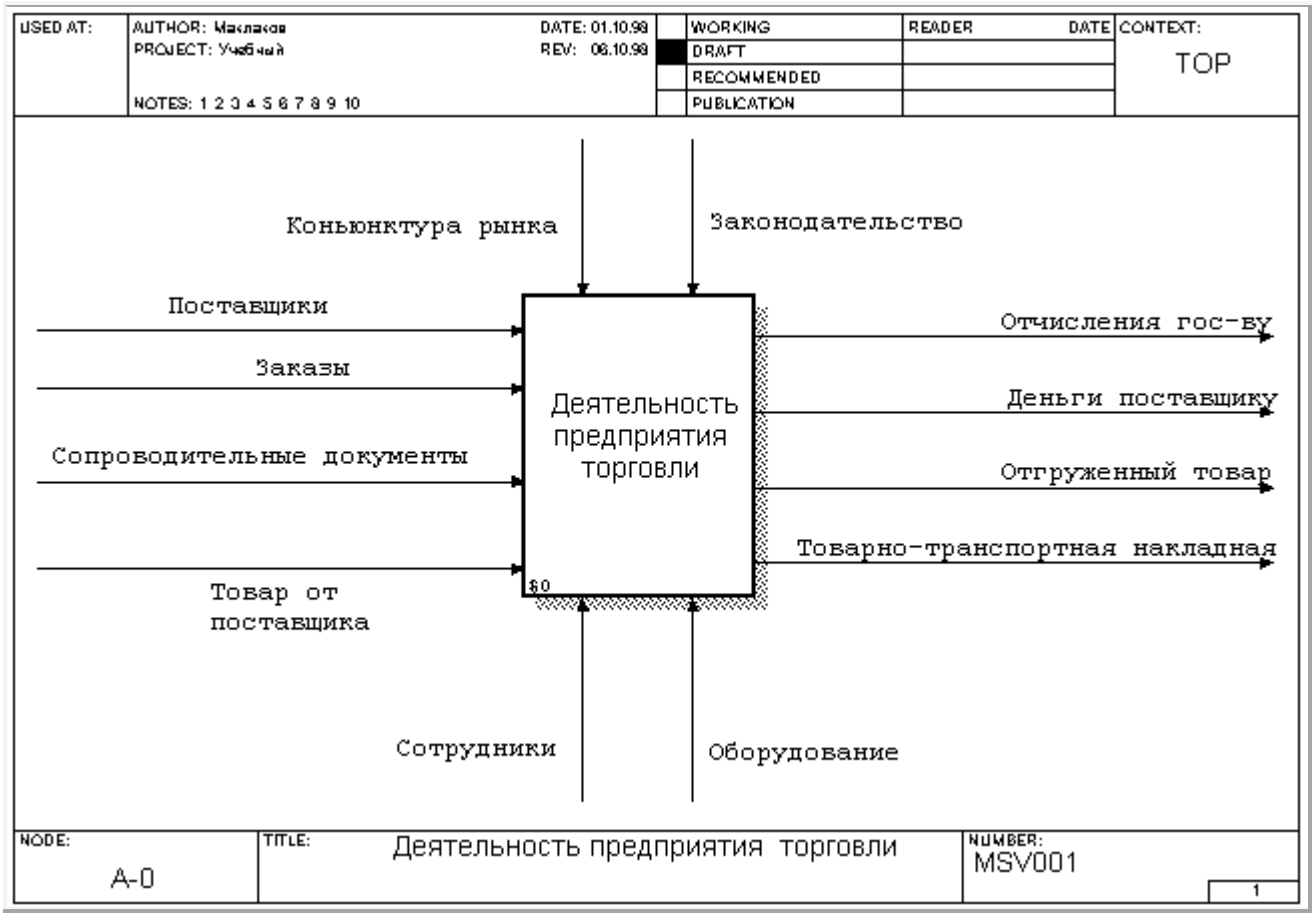


Рис.1.3 Контекстная диаграммы процесса – деятельность предприятия торговли

На рисунке 1.4 представлена декомпозиция контекстной диаграммы.

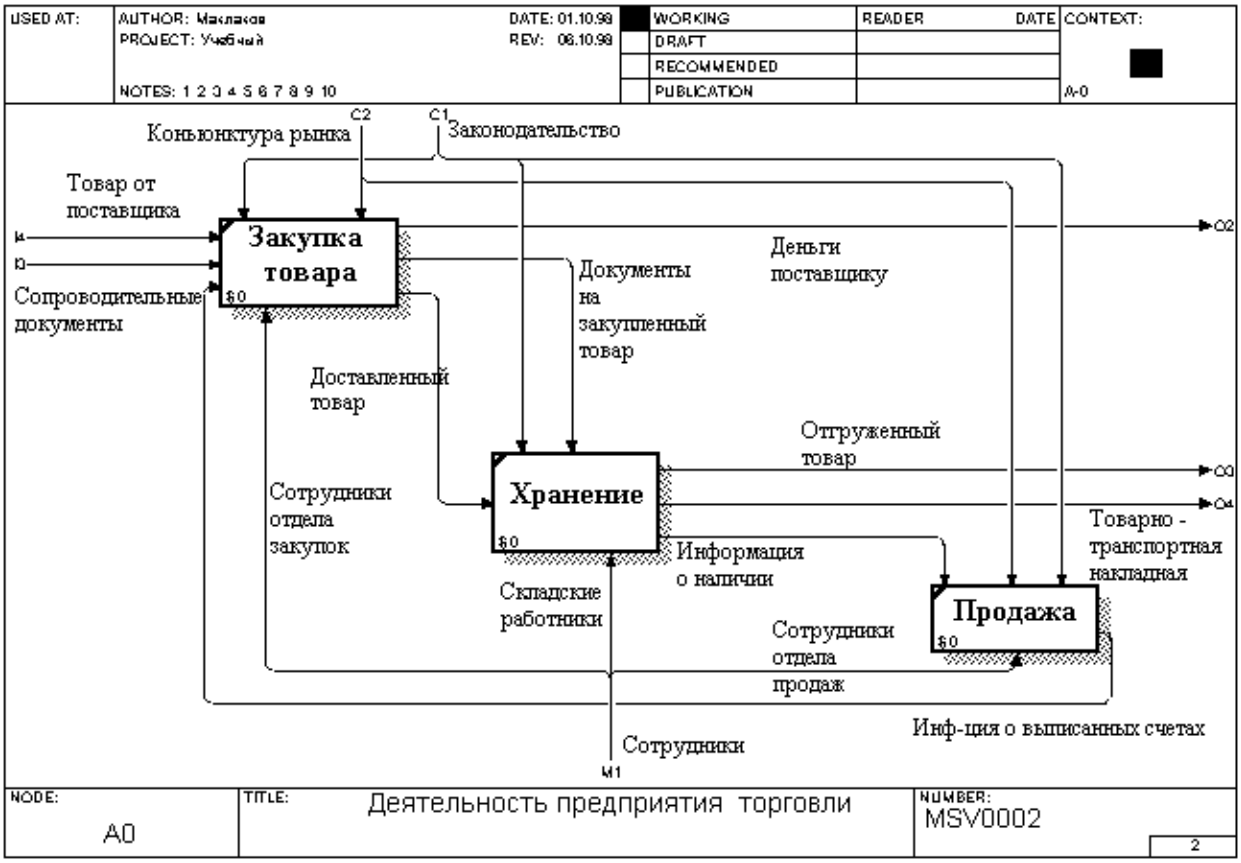


Рис.1.4 Функциональная модель – декомпозиция контекстной диаграммы

Практическое занятие №2. Построение диаграммы потоков данных (DFD)

DFD являются основным средством моделирования функциональных требований к проектируемой системе. С их помощью эти требования представляются в виде иерархии функциональных компонентов (процессов), связанных потоком данных. Главная цель такого представления – продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между этими процессами.

Таким образом, модель системы в контексте **DFD** представляется в виде некоторой информационной модели, основными компонентами которой являются различные потоки данных, которые переносят информацию от одной подсистемы к другой.

Основными компонентами диаграмм потоков данных являются:

- Внешние сущности;
- Системы и подсистемы;
- Процессы;
- Накопители данных;
- Потоки данных.

Внешняя сущность (Рис.1.5) представляет собой материальный объект или физическое лицо, представляющее собой источник или приемник информации (заказчик, персонал, поставщики, клиенты, склад). Определение некоторого объекта или системы в качестве внешней

сущности указывает на то, что они находятся за пределами границ анализируемой системы.

Рис.1.5 Представление внешней сущности

При построении модели сложного ПО она может быть представлена в самом общем виде на контекстной диаграмме в виде одной системы как единого целого, либо может быть декомпозирована на ряд подсистем.

Процесс – преобразование входных потоков данных в выходные в соответствие с определенным алгоритмом. Физически процесс может быть реализован различными способами: это может быть отдел, выполняющий обработку входных документов, программа, устройство и т.д.

Накопитель данных – это абстрактное устройство для хранения информации. Это – прообраз будущей базы данных, и описание хранящихся в нем данных должно быть увязано с информационной моделью ERD.

Поток данных определяет информацию, передаваемую через некоторое соединение от источника к приемнику. Поток данных изображается линией, заканчивающей стрелкой, которая показывает направление потока.

В отличие от стрелок в IDEF0, которые иллюстрируют отношения, стрелки в DFD показывают, как объекты (включая и данные) реально 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).

Построение DFD-диаграмм в основном ассоциируется с разработкой программного обеспечения, поскольку нотация DFD изначально была разработана для этих целей. В частности, графическое изображение объектов на DFD-диаграммах этой главы соответствует принятому Крисом Гейном (Chris Gane), Тришем Сарсоном (Trish Sarson) — авторами DFD-метода, известного как метод Гейна-Сарсона. Другой распространенной нотацией DFD является так называемый метод Иордана-Де Марко (Yourdon-DeMarco).

Пример 1.3. Построение DFD – диаграммы для детализации процесса **Хранение** для диаграммы 1.4.

На рисунке 1.6 представлена диаграмма потоков данных при сочетании DFD и IDEF0: Процесс **Хранение,** представленный на рис.1.4, был детализирован в виде диаграммы DFD на следующем уровне.

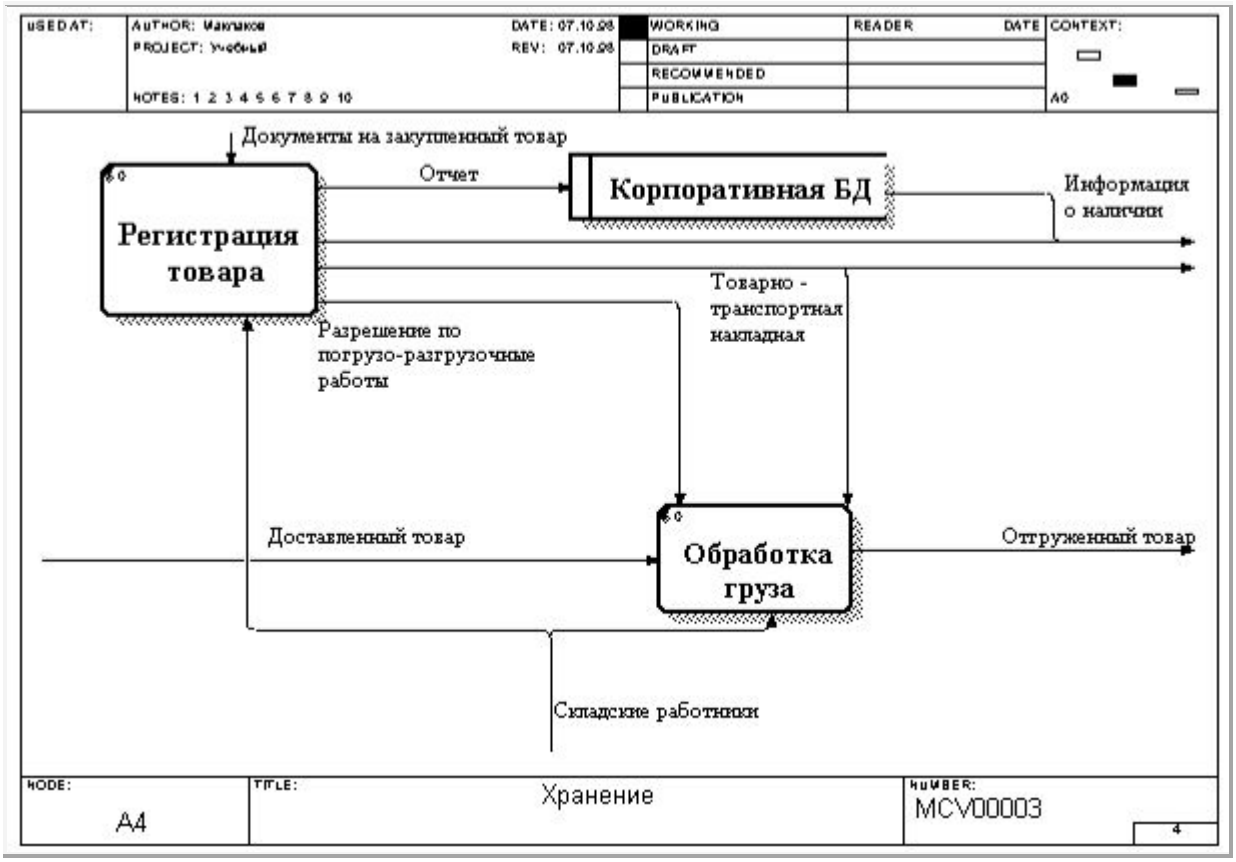


Рис. 1.6 Пример диаграммы DFD для процесса **Хранение**

Практическое занятие №3. Построение модели IDEF3

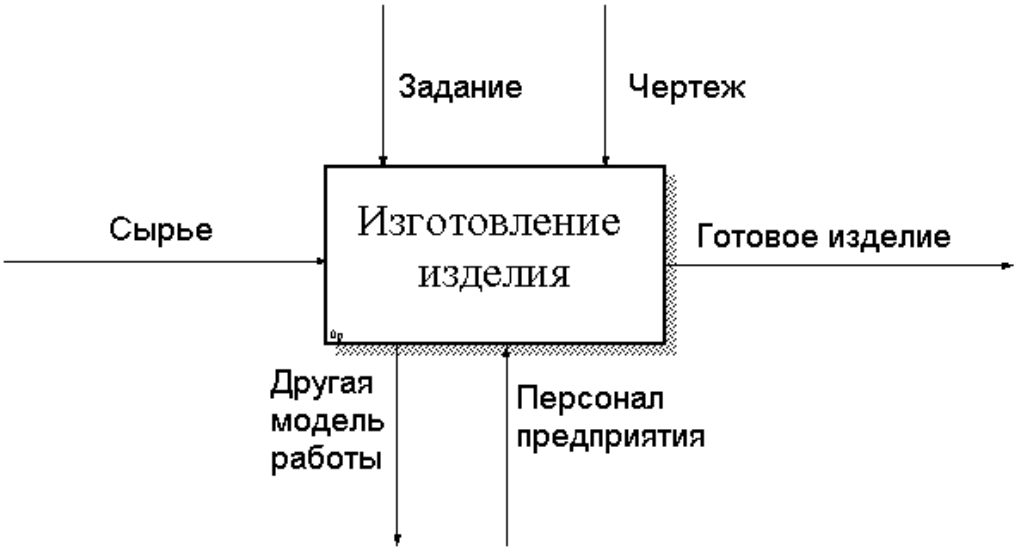
Наличие в диаграммах DFD элементов для описания источников, приемников и хранилищ данных позволяет более эффективно и наглядно описать процесс документооборота. Однако для описания логики взаимодействия информационных потоков более подходит IDEF3, называемая также workflow diagramming, - методология моделирования, использующая графическое описание информационных потоков, взаимоотношений между процессами обработки информации и объектов, являющихся частью этих процессов. Диаграммы Workflow могут быть использованы в моделировании бизнес - процессов для анализа завершенности процедур обработки информации. С их помощью можно описывать сценарии действий сотрудников организации, например, последовательность обработки заказа или события, которые необходимо обработать за конечное время. Каждый сценарий сопровождается описанием процесса и может быть использован для документирования каждой функции.

Прямоугольники на диаграмме Workflow называются единицами работы (Unit of Work, UOW) и обозначают событие, процесс, решение или работу. Для редактирования диаграммы используются примерно те же диалоги, что и для IDEF0. В палитре инструментов на диаграмме Workflow имеются кнопки для новых элементов, например, перекресток – специфический для IDEF3 элемент – позволяет описать последовательность выполнения работ, очередность их запуска и завершения. Различают перекрестки для слияния (Fan-in Junction) и разветвления (Fan-out Junction) стрелок. Перекресток не

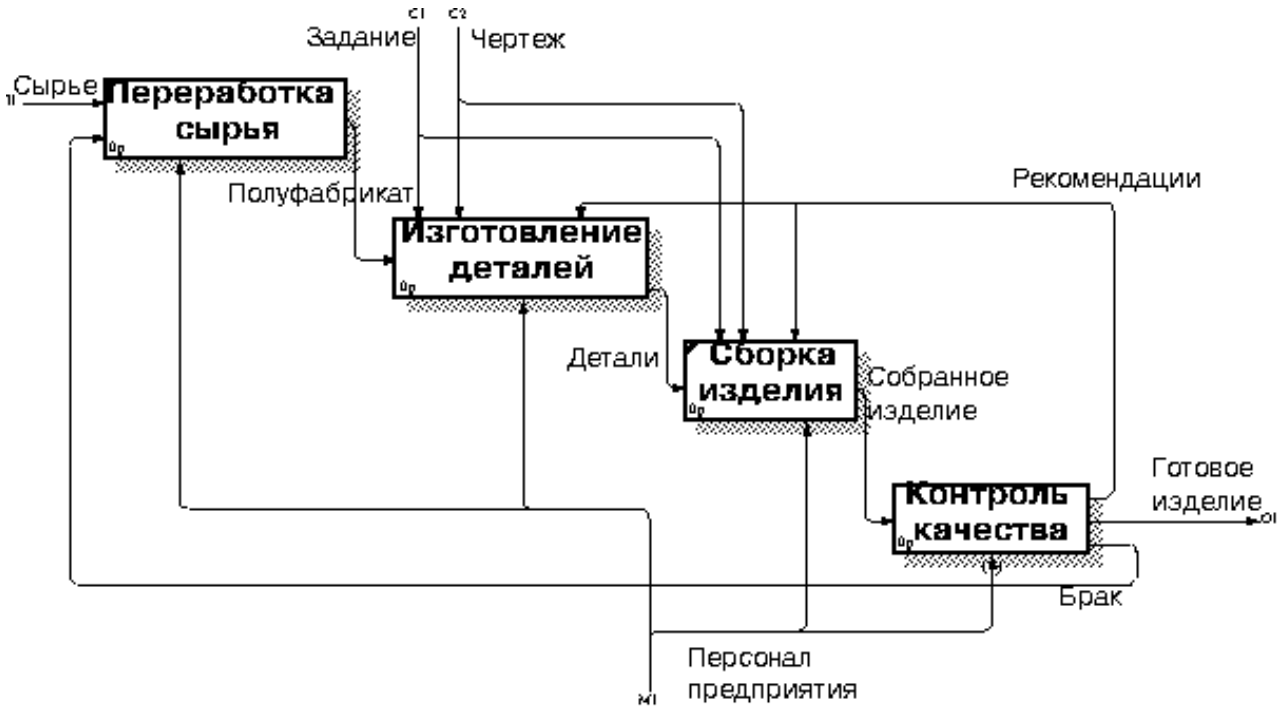
может использоваться одновременно для слияния и для разветвления. С помощью диаграмм Workflow можно описывать сценарии действий сотрудников организации, например, последовательность обработки заказа или события, которые необходимо обработать за конечное время. Каждый сценарий сопровождается описанием процесса и может быть использован для документирования каждой функции, моделируемой в методологии IDEF0. Если в одной модели необходимо осветить специфические стороны бизнес-процессов предприятия, BPwin позволяет переключиться на любой ветви модели с IDEF0 на нотацию IDEF3 или DFD и создать смешанную модель.

Пример 1.4. Сочетание диаграмм IDEF0, DFD, IDEF3 при описании процесса изготовления изделия (моделирование предметной области).

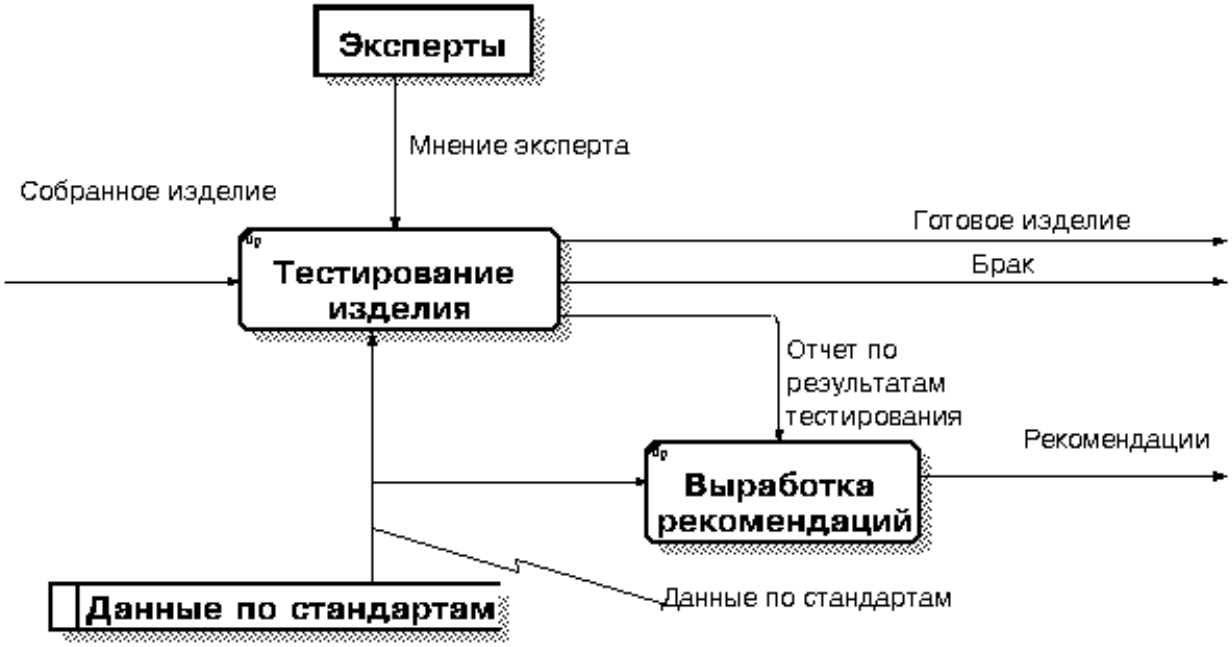
1. Контекстная диаграмма.



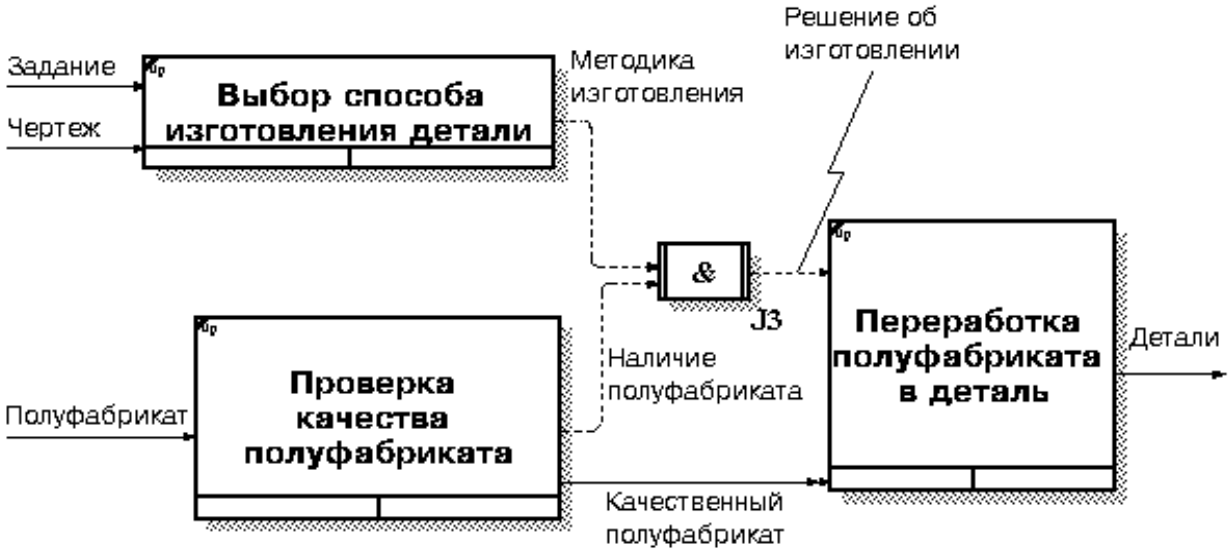
2. Детализация процесса в виде IDEF0



3. Детализация процесса **Контроль качества** в виде диаграммы DFD



4. Детализация процесса **Изготовление деталей** в виде диаграммы IDEF3.



Практическое занятие №4. Моделирование данных

Цель моделирования данных – обеспечить разработчика ПО концептуальной схемой базы данных в форме одной модели или нескольких локальных моделей, которые относительно легко могут быть отображены в любую систему баз данных.

Наиболее распространенным средством моделирования данных являются диаграммы «Сущность- связь» (ERD), нотация которых была введена Питером Ченом в 1976г.. Базовыми понятиями ERD являются:

Сущность (Entity) - реальный или воображаемый объект, имеющий существенное значение для рассматриваемой предметной области.

Каждая сущность должна обладать уникальным идентификатором. Каждая сущность должна обладать некоторыми свойствами:

- Иметь уникальное имя;
- Обладать одним или несколькими атрибутами;
- Обладать одним или несколькими атрибутами, которые однозначно идентифицируют каждый экземпляр сущности.

Каждая сущность может обладать любым количеством связей с другими сущностями модели.

Связь (*Relationship)* – это ассоциация между сущностями, при которой каждый экземпляр одной сущности ассоциирован с произвольным (в том числе нулевым) количеством экземпляров второй сущности, и наоборот.

Атрибут (Attribute) – любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности. Атрибут представляет тип характеристик или свойств, ассоциированных с множеством реальных или абстрактных объектов. На диаграмме «сущность – связь» атрибуты ассоциируются с конкретными сущностями.

С точки зрения БД (физическая модель) сущности соответствует таблица, экземпляру сущности – строка в таблице, а атрибуту – колонка таблицы.

Существуют различные нотации ERD. Например,

- нотация, предложенная Ричардом Баркером, используемая в CASE – средстве Oracle Designer,
- одна из вариантов нотации Чена используется в CASE – средстве Silverrun для концептуального моделирования данных (на стадии формирования требований),
- метод IDEF1X (основан на подходе Чена) позволяет построить модель данных, эквивалентную реляционной модели в третьей нормальной форме. Этот метод используется в ERWIN.

Сущность в методе IDEF1X является не зависимой от идентификаторов или просто независимой, если каждый экземпляр сущности может быть однозначно идентифицирован без его отношений с другими сущностями (служащий/44). Сущность называется зависимой от идентификаторов или просто зависимой, если однозначная идентификация экземпляра сущности зависит от его отношения к другой сущности (проектное задание/56).

Связь может определяться с помощью указания степени или мощности (количества экземпляров сущности-потомка, которое может существовать для каждого экземпляра сущности – родителя). В IDEF1X могут быть выражены следующие мощности связей:

- каждый экземпляр сущности – родителя может иметь ноль, один или более одного связанного с ним экземпляра сущности – потомка;
- каждый экземпляр сущности – родителя должен иметь не менее одного связанного с ним экземпляра сущности – потомка;

- каждый экземпляр сущности – родителя должен иметь не более одного связанного с ним экземпляра сущности – потомка;
- каждый экземпляр сущности – родителя связан с фиксированным числом экземпляров сущности – потомка;

Если экземпляр сущности потомка однозначно определяется своей связью с сущностью – родителем, то связь называется *идентифицирующей*, в противном случае – *неидентифицирующей (*служит для связывания независимых сущностей, например, экземпляр сущности СОТРУДНИК может существовать безотносительно к какому-либо экземпляру сущности ОТДЕЛ, т.е. сотрудник может работать в организации, не числясь в каком либо отделе).

Связь изображается линией, проводимой между сущностью – родителем и сущностью – потомком, с точкой на конце линии у сущности – потомка. Мощность связи может принимать следующие значения: N – ноль, один или более, Z – ноль или один, P – один или более. По умолчанию мощность связи принимается равной N.

Атрибуты изображаются в виде списка имен внутри блока сущности. Атрибуты, определяющие первичный ключ, размещаются наверху списка и отделяются от других атрибутов горизонтальной чертой.

Сущности могут иметь также внешние ключи (Foreign key), рядом с внешним ключом следуют буквы FK в скобках.

ERWIN имеет два уровня представления модели – логический и физический.

Логический уровень – это абстрактный взгляд на данные, на нем данные представляются так, как выглядят в реальном мире, и могут называться так, как они называются в реальном мире, например, «Постоянный клиент», «Отдел», «Фамилия сотрудника» и т.д. Объекты модели, представленные на логическом уровне, называются сущностями и атрибутами. Логическая модель данных является универсальной и никак не связана с конкретной реализацией СУБД.

Физический уровень данных, напротив, зависит от конкретной СУБД. Одной и той же логической модели могут соответствовать несколько разных физических моделей. Создание модели данных начинается, как правило, с создания логической модели. После описания логической модели, проектировщик может выбрать необходимую СУБД и ERWIN автоматически создаст соответствующую физическую модель. На основе физической модели ERWIN может сгенерировать системный каталог СУБД или соответствующий SQL – скрипт. Этот процесс называется прямым проектированием (Forward Engineering). С другой стороны, ERWIN способен по содержимому системного каталога или SQL – скрипту воссоздать физическую и логическую модель и логическую модель данных (Reverse Engineering). На основе полученной логической модели данных можно сгенерировать физическую модель для другой СУБД и затем сгенерировать ее системный каталог. Следовательно, ERWIN позволяет решить задачу по переносу структуры данных с одного сервера на другой.

Создание логической модели

ERWIN имеет несколько уровней отображения диаграммы: уровень сущностей, уровень атрибутов, уровень определений, уровень первичных ключей и уровень иконок. Переключиться на другие уровни отображения можно при помощи контекстного меню.

По глубине представления информации о данных в ERWIN различают три уровня логической модели:

- диаграмма сущность – связь (Entity Relationship Diagram, ERD);
- модель данных, основанная на ключах (Key Based model, KB);
- полная атрибутивная модель (Fully Attributed model, FA).

Сущности и атрибуты

С точки зрения БД (физическая модель) сущности соответствует таблица, экземпляру сущности – строка в таблице, а атрибуту – колонка в таблице. Каждая сущность должна быть полностью определена с помощью текстового описания в закладке Definition. Имя атрибута должно быть уникально в рамках модели (Unique name). ERWin, при попытке внесения уже существующего имени, дает сообщение об этом

Связи

Каждая связь должна именоваться глаголом или глагольной фразой (Relationship Verb Phrases). Например, Каждое Изделие <состоит из> ДЕТАЛей. Для отображения имени связи в контекстном меню, которое появляется, если щелкнуть левой кнопкой мыши по любому месту на диаграмме, не занятому объектами модели, выбрать пункт Display Options/Relationship и затем включить опцию Verb Phrase.

Иерархия наследования

Она представляет собой особый тип объединения сущностей, которые разделяют общие характеристики. Например, в организации работают служащие, занятые полный рабочий день (постоянные служащие) и совместители. Из их общих свойств можно сформировать обобщенную сущность (родовой предок) Сотрудник, чтобы представить информацию, общую для всех типов служащих.

Создание физической модели

Различают два уровня физической модели:

- трансформационная модель (Transformation Model);
- модель СУБД (DBMS).

Физическая модель содержит всю информацию, необходимую для реализации конкретной БД. Трансформационная модель содержит информацию, для реализации отдельного проекта, который может быть частью общей ИС и описывать подмножество предметной области.ERWIN поддерживает ведение отдельных проектов, позволяя проектировщику выделять подмножество модели в виде предметных областей (Subject Area).

Пример 1.5. Использования структурного подхода

1. Постановка задачи

Разработка автоматизированной системы поддержки функции регистрации и учета налогоплательщиков – организаций для гос. налоговой инспекции.

2. Описание предметной области

В качестве предметной области рассматривается работа одного из подразделений гос. налоговой инспекции (ГНИ), а именно подразделения учета налогоплательщиков - организаций (юридических лиц).

2.1. Реализация функции учета включает в себя следующие действия:

- Первичную постановку на налоговый учет (налогоплательщик первый раз становится на учет);
- Повторную постановку на налоговый учет (налогоплательщик уже имеет ИНН);
- Снятие с налогового учета (без ликвидации юридического лица);
- Снятие с налогового учета (при ликвидации юридического лица);
- Ведение гос. реестра налогоплательщиков;
- Учет сведений об открытии и закрытии банковских счетов налогоплательщиков;
- Сверку данных по расчетным счетам налогоплательщиков с коммерческими банками;
- Прием заявлений налогоплательщиков об изменении учетной политики, организации учета и отчетности.

Налогоплательщик должен представить следующие документы;

- Заявление о постановке на учет;
- Устав организации;
- Свидетельство о гос. регистрации юридического лица;
- Протокол собрания учредителей.

2.2. Описание последовательности действий

Заявление регистрируется в журнале. Формы и документы проверяются на соответствие законодательству, полноту заполнения и точность информации. Если документы в порядке, налогоплательщику присваивается ИНН и код причины постановки на учет (КПП). Данные из заявления о постановке на учет вводятся в базу данных ГНИ с последующим занесением в Гос.реестр. Заносимые данные проверяются на правильность по соответствующим справочникам. После выполнения всех формальных процедур налогоплательщику выдается свидетельство о постановке на учет в налоговом органе. Об открытии счета банк и налогоплательщик должны известить налоговую инспекцию. После того как информация о расчетном счете введена в базу данных ГНИ, налогоплательщик может платить налоги.

По каждому налогоплательщику в БД должны храниться следующие данные реестра;

- ИНН;
- КПП;

- Наименование плательщика;
- Юридический адрес;
- Фактический адрес;
- Номер расчетного счета и атрибуты банка, его обслуживающего;
- Полные атрибуты учредителей плательщика;
- Дата регистрации;
- Размер уставного фонда;
- Код формы собственности;
- Вид деятельности;
- Место регистрации;
- Регистрационный номер;
- Информация о всех счетах предприятия.

Получаемая в результате БД является основой для последующих проверок.

3. Построение моделей деятельности организации

3.1. Модель DFD

Начальная контекстная диаграмма представлена на рисунке 1.9.

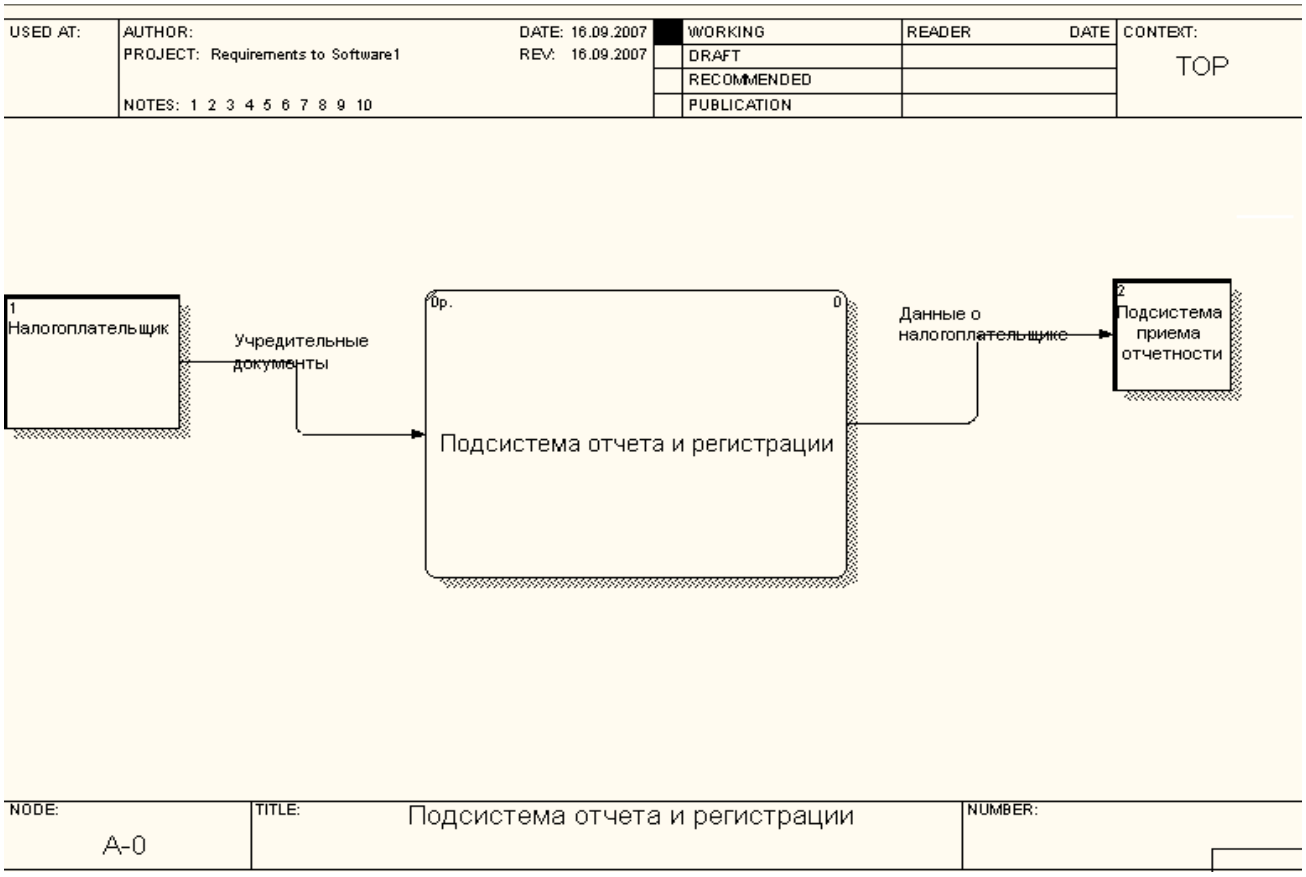


Рис.1.9. Начальная контекстная диаграмма

Концептуальная модель данных в виде ERD строится из следующих соображений:

- Сущности могут быть распознаны как структуры данных в DFD.Чтобы рассматривать объект в качестве сущности, он должен обладать более чем одним атрибутом;

- Связи должны отражать взаимодействия между сущностями, причем в системе должна сохраняться информация об этом взаимодействии.

Декомпозиция контекстной диаграммы представлена на рис.1.10.

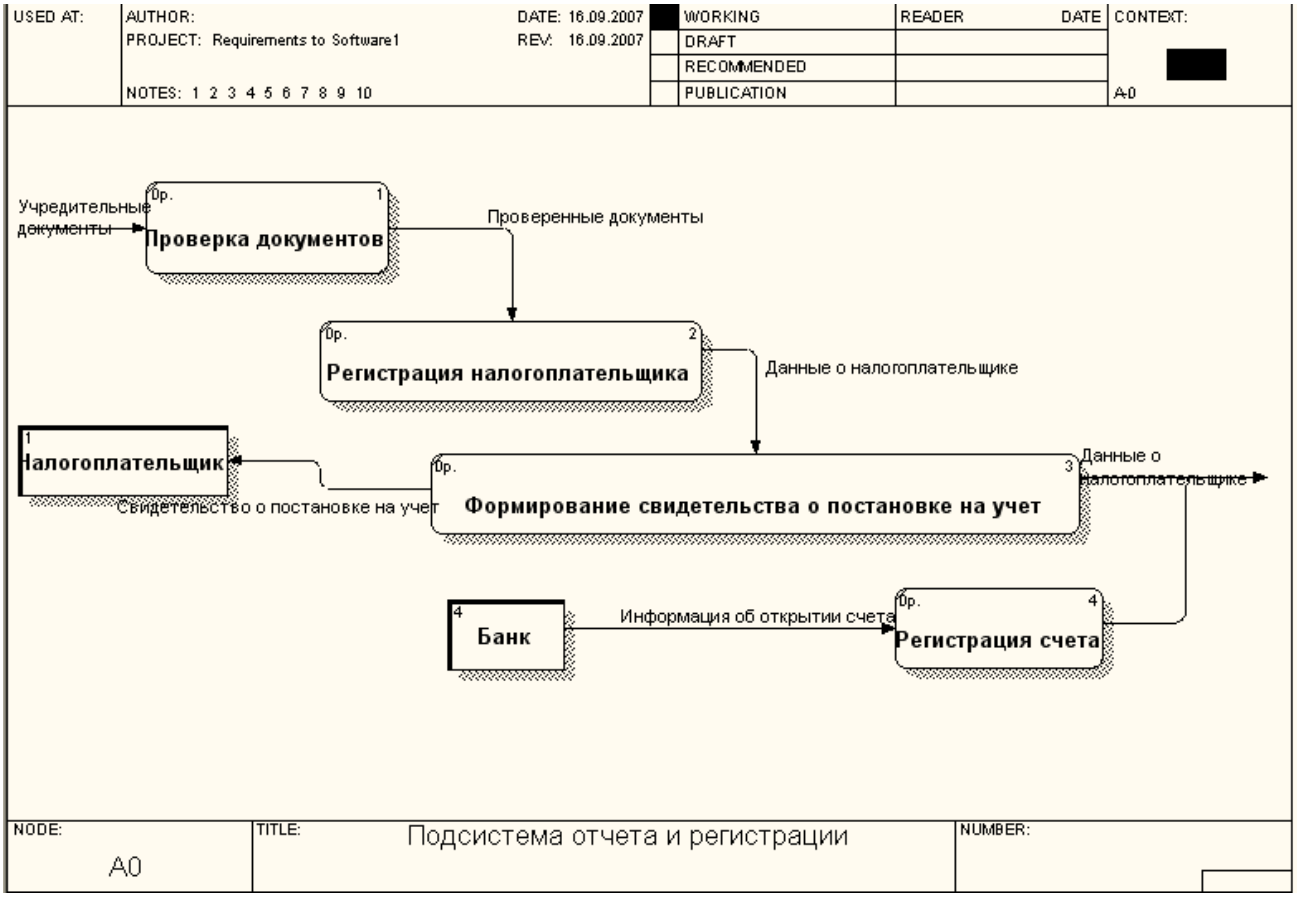


Рис.1.10 Декомпозиция контекстной диаграммы

С использованием структур данных определяются атрибуты каждой сущности и изображаются на диаграмме (1.11). Проверяется соответствие между описанием структур данных и концептуальной моделью (все элементы данных должны быть использованы в качестве атрибутов). Для представления концептуальной модели данных будет использовать один из вариантов нотации Чена.

На диаграмме (1.11) числа над линиями обозначают степень и обязательность связи. Так, на примере связи **Банк – Счет налогоплательщика** пара (0,N) означает:

- Банк может не иметь счетов налогоплательщика (необязательная связь) либо иметь много счетов (степень связи N);
- Каждый счет налогоплательщика принадлежит одному (обязательная связь) и только одному банку (степень связи – 1).

Поскольку учредитель может быть либо юридическим, либо физическим лицом, то имеет место связь типа «супертип-подтип». В этом виде связи экземпляр подтипа существует только при условии существования определенного экземпляра супертипа.

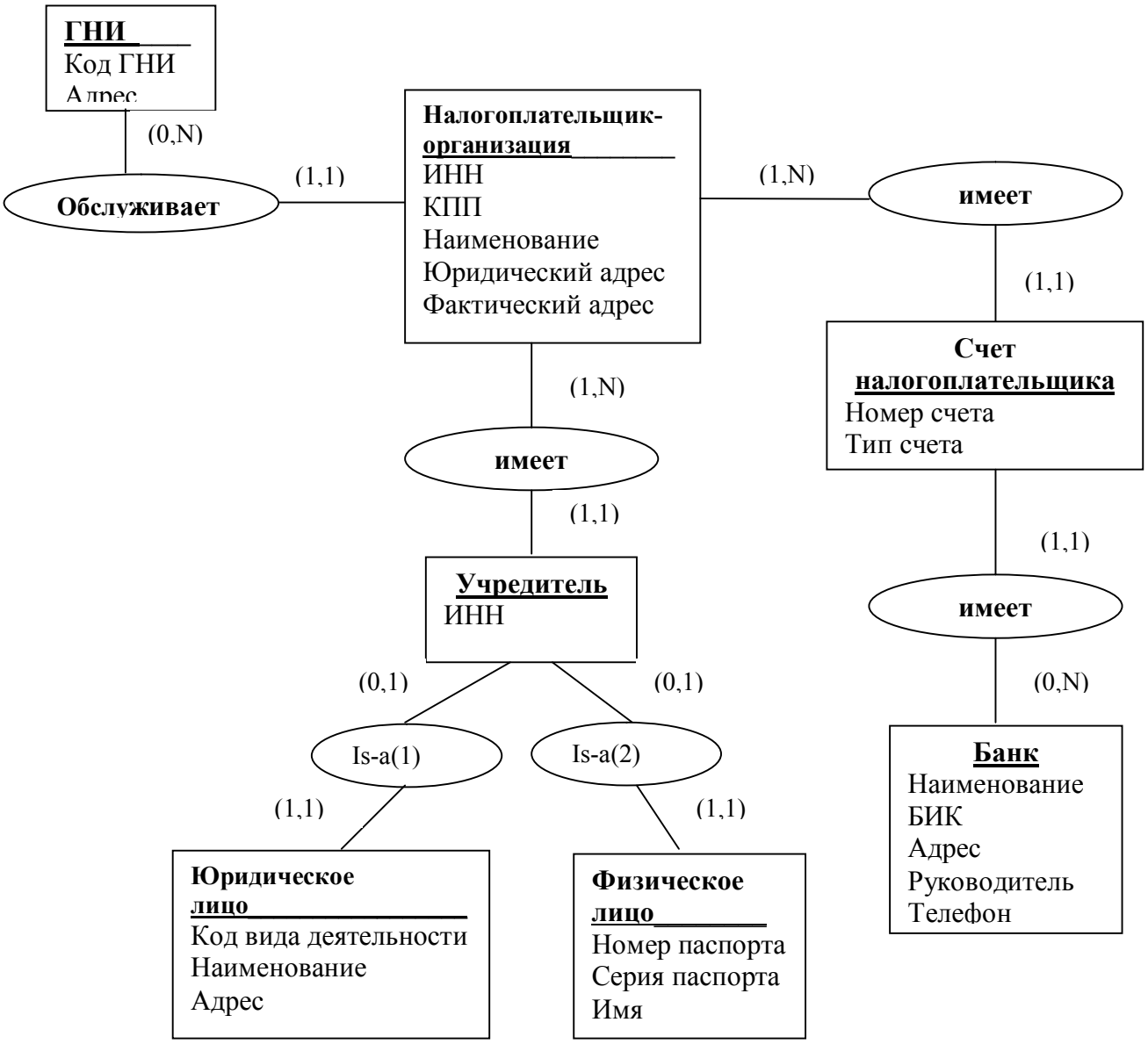


Рис.1.11 Диаграмма «сущность – связь»

На рис.1.12 описанная выше диаграмма «сущность – связь» представлена в CASE –средстве ERWIN.

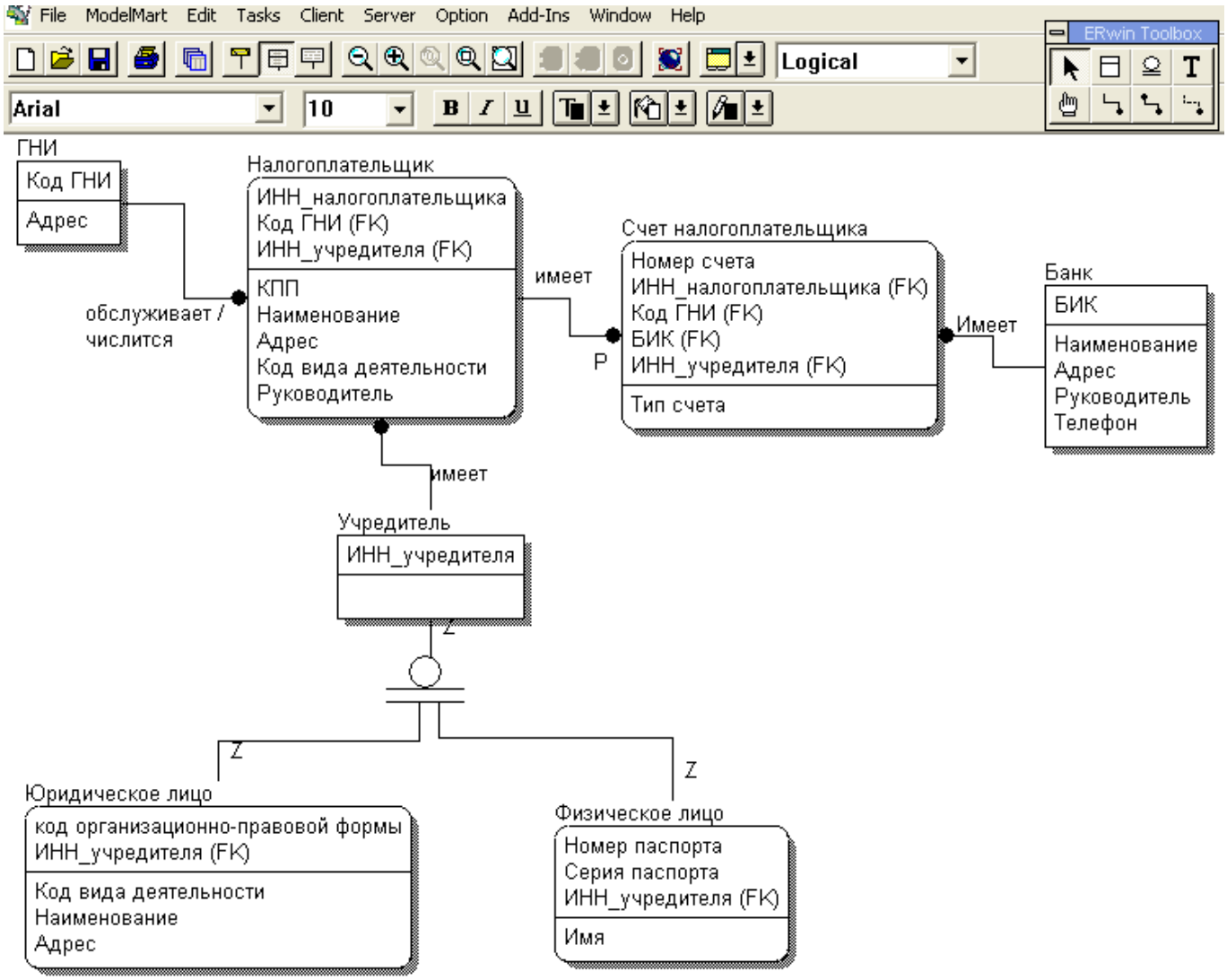


Рис.1.12 Представление в ERWIN

Вопросы для самопроверки по теме1:

1. В чем заключаются основные принципы структурного программирования?
2. Что общего и в чем различие между методом SADT и моделированием потоков данных?
3. В чем заключаются достоинства и недостатки структурного подхода?
4. Что представляет собой модель в нотации IDEF0?
5. Что обозначают работы в IDEF0?
6. Перечислите типы стрелок в IDEF0.
7. Что описывает диаграмма DFD?
8. Перечислите составные части диаграммы DFD.
9. Что описывает диаграмма IDEF3?
10. Что называется внешней сущностью?
11. Что описывают хранилища?
12. Объясните механизм дополнения диаграммы IDEFO диаграммой DFD.

Тема: Объектно-ориентированный подход к проектированию

Как отмечалось во введении, принципиальное различие между структурным и объектно-ориентированным подходом заключается в способе декомпозиции системы. Объектно-ориентированный подход использует объектную декомпозицию. При этом структура системы описывается в терминах объектов и связей между ними, а поведение системы описывается в терминах обмена сообщениями между объектами. В отличие от функционально - ориентированного подхода декомпозиция системы на основе объектно-ориентированного подхода обладает лучшей способностью отражать динамическое поведение системы от возникающих событий. В этом плане модель проблемной области рассматривается как совокупность взаимодействующих во времени объектов. Тогда конкретный процесс обработки информации формируется в виде последовательности взаимодействий объектов. Одна операция обработки данных может рассматриваться как результат одного взаимодействия объектов.

В настоящее время для объектно-ориентированного моделирования широко используется язык UML (Unified Modeling Language). Система объектно-ориентированных моделей в соответствии с нотациями UML включает в себя следующие диаграммы:

1. диаграммы вариантов использования (Use case Diagrams) – для моделирования бизнес – процессов организации (требований к системе);
2. диаграммы классов (Class Diagram)- для моделирования статической структуры классов системы и связи между ними;
3. диаграммы поведения:
   - диаграммы состояний (Statechart diagrams) – для моделирования поведения объектов системы при переходе из одного состояния в другое;
   - диаграммы деятельности (Activity diagrams) – для моделирования поведения системы в рамках различных вариантов использования (отображают потоки работ во взаимосвязанных прецедентах использования);
   - диаграммы взаимодействия (interaction diagrams)– для моделирования процесса обмена сообщениями между объектами. Существуют два вида диаграмм взаимодействия:
     - диаграммы последовательности (sequence diagrams)– последовательность сообщений;
     - кооперативные диаграммы (collaboration diagrams)– пространственное расположение объектов, чтобы показать их статическое взаимодействие;
4. диаграммы реализации (implementation diagrams):
   - диаграммы компонентов (component diagrams) – для моделирования иерархии компонентов (подсистем) системы (отображаются физические модули программного кода);
   - диаграммы размещения или развертывания (deployment

diagrams) – для моделирования физической архитектуры системы (отображает распределение объектов по узлам вычислительной сети).

Цель практических работ по данной теме: состоит в отображении функциональных возможностей системы и требований в UML.

Практическое занятие №5. Построение модели вариантов использования.

О вариантах использования

Варианты использования объясняют определенную часть функциональных возможностей системы, компонента или даже класса. Каждый вариант использования должен иметь название, которое состоит обычно из нескольких слов, описывающее требуемые функциональные возможности, типа «просмотр журнала регистрации ошибок».

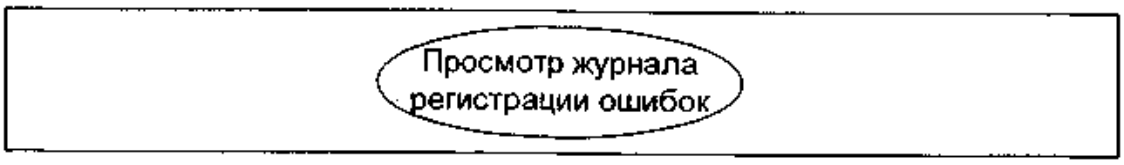


Рис. 2.1. Простой вариант использования

О действующих лицах

По определению UML вариант использования должен быть инициирован кем-то или чем-то вне содержания варианта использования. Эта заинтересованная сторона называется действующее лицо. Действующее лицо не обязательно должно быть человеком-пользователем; любая внешняя система, время, или элемент вне варианта использования могут вызвать начало варианта использования (или получателя результатов варианта использования) и должны быть смоделированы в качестве действующего лица (актера). Например, это очень типично моделировать системные часы как действующее лицо, которое вызывает вариант использования в заданное время или интервал.

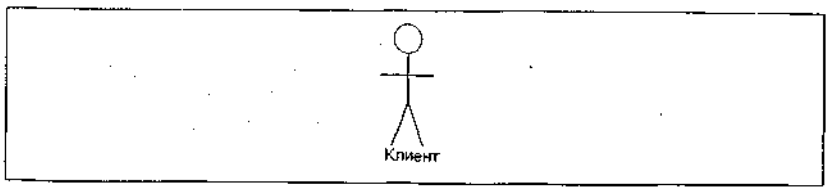


Рис. 2.2**.** Действующее лицо, использующее представление фигуры человечка

Об ассоциациях действующее лицо / вариант использования

Обычно действующее лицо ассоциируется с одним или большим количеством варианта использования. Отношения между действующим лицом и вариантом использования указывают на то, что действующее лицо инициирует вариант использования, вариант использования обеспечивает действующее лицо результатами, или оба случая. Ассоциация между

действующим лицом и вариантом использования обозначается в виде сплошной линии. В дополнение к связям между действующими лицами и вариантами использования существуют два других типа связей: «использование» (uses) и «расширение» (extends) между вариантами использования.

Связь типа «расширение» применяется тогда, когда один вариант использования подобен другому, но несет несколько большую нагрузку. Эту связь используют при описании изменений в нормальном состоянии объекта. Связь «использование» применяется в тех ситуациях, когда имеется какой-либо фрагмент поведения системы, который повторяется в более, чем одном варианте использования, и нет необходимости копировать его описание в каждом из этих вариантов.

Обычно диаграммы варианта использования читаются слева направо, с действующими лицами, инициирующими варианты использования, слева и действующими лицами, которые получают результаты варианта использования, справа. Однако, в зависимости от модели или уровня сложности, может иметь смысл группировать действующие лица по-другому. Рисунок 3.3 показывает действующее лицо, поддерживающее связь с вариантом использования.

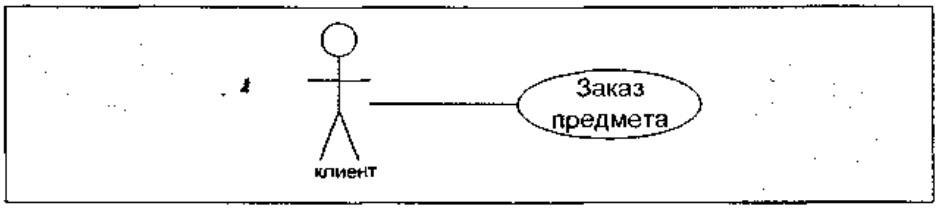


Рис. 2.3. Действующее лицо, ассоциированное с вариантом использования

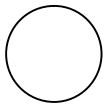
Заказ товара (Order Item)

Интерфейсы

Интерфейс служит для спецификации параметров модели, которые видимы извне без указания их внутренней структуры. В языке UML интерфейс является классификатором и характеризует только ограниченную часть поведения моделируемой сущности. Применительно к диаграммам вариантов использования, интерфейсы определяют совокупность операций, которые обеспечивают необходимый набор сервисов и функциональности для актеров. Интерфейсы не могут содержать ни атрибутов, ни состояний, ни направленных ассоциаций. Они содержат только операции без указания особенностей их реализации. Формально интерфейс эквивалентен абстрактному классу без атрибутов и методов с наличием только абстрактных операций.

На диаграмме вариантов использования интерфейс изображается в виде маленького круга, радом с которым записывается его имя. В качестве имени может служить только существительное, которое характеризует

соответствующую информацию или сервис (например, «датчик», «сирена», «видеокамера»), но чаще «запрос к базе данных», «форма ввода», «устройство подачи звукового сигнала». Если имя записывается на английском, то оно должно начинаться с заглавной буквы I, например, ISensor.



ISensor

Графический символ отдельного интерфейса может соединяться на диаграмме сплошной линией с тем вариантом использования, который его поддерживает. Сплошная линия в этом случае указывает на тот факт, что связанный с интерфейсом вариант использования должен реализовать все операции, необходимые для данного интерфейса, а возможно и больше. Кроме того, интерфейсы могут соединяться с вариантами использования пунктирной линией со стрелкой, означающей, что вариант использования предназначен для спецификации только сервиса, который необходим для реализации данного интерфейса (рис.2.4).

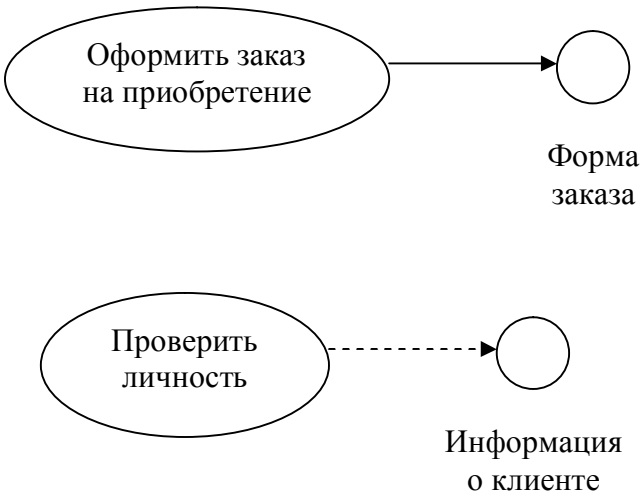


Рис.2.4 Примеры интерфейсов

С системно – аналитической точки зрения интерфейс не только отделяет спецификацию операций системы от их реализации, но отделяет общие границы проектируемой системы. В последующем интерфейс может быть уточнен явным указанием тех операций, которые специфицируют отдельный аспект поведения системы. В этом случае он изображается в форме прямоугольника класса с ключевым словом «интерфейс» в секции имени, с пустой секции атрибутов и с непустой секцией операций. Однако подобное графическое представление используется на диаграмме классов или диаграммах, характеризующих поведение моделируемой системы.

Важность интерфейсов заключается в том, что они определяют стыковочные узлы в проектируемой системе, что совершенно необходимо для организации коллективной работы над проектом.

Определение границ системы

Варианты использования отображают функциональные возможности определенного субъекта. Все, что не относится к субъекту, считается вне границ системы и должно моделироваться в виде действующего лица. Эта методика очень полезна при определении содержания и назначения ответственностей при проектировании системы, подсистемы или компонента. Например, если вы моделируете систему банкоматов, ваши обсуждения проектирования часто уходят от центральной темы подробностей серверной части банковской системы, модель варианта использования с ясно определенными границами системы будет идентифицировать банковскую систему в качестве действующего лица и поэтому за пределами содержания проблемы.
Границы системы в общем смысле обозначаются с помощью простого прямоугольника с названием системы сверху. Рисунок 2.5 представляет границы системы для банкомата.

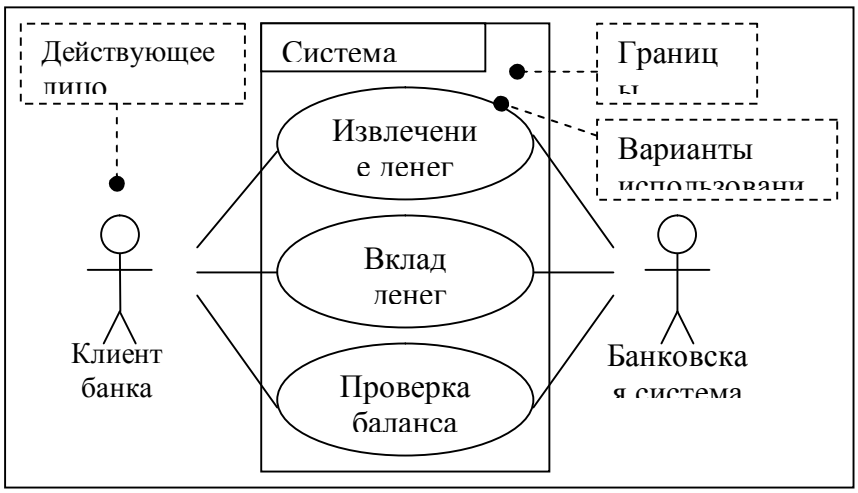


Рис. 2.5. Диаграмма варианта использования, демонстрирующая границы системы для банкомата

Пример 2.1. Построение диаграммы вариантов использования для моделирования системы продажи товаров по каталогу.
Рассмотрим процесс моделирования системы продаж товаров по каталогу, который может быть использован при создании соответствующей информационной системы. В качестве актеров данной системы могут выступать два субъекта, один из которых является продавцом, а другой - покупателем. Каждый из этих актеров взаимодействует с рассматриваемой системой продажи товаров по каталогу и является ее пользователем, т.е. они оба обращаются к сервису «Оформить заказ на покупку». Как следует из

более детального анализа процесса продажи товаров, можно выделить в качестве отдельных сервисов такие действия, как обеспечить покупателя информацией о товаре, согласовать условия оплаты товара и заказать товар со склада. Вполне очевидно, что указанные действия раскрывают поведение исходного варианта использования в смысле его конкретизации, и поэтому между ними будет иметь место отношение включения. С другой стороны, продажа товара по каталогу предполагает наличие самостоятельного информационного объекта – каталога товаров, который не зависит от реализации сервиса по обслуживанию покупателей. В нашем случае, каталог товаров может запрашиваться покупателем или продавцом при необходимости выбора товара и уточнения деталей его продажи. Вполне резонно представить сервис «Запросит каталог товаров» в качестве самостоятельного варианта использования. Полученная в результате последующей детализации уточненная диаграмма вариантов использования будет содержать 5 вариантов использования и 2 актеров (рис. 2.6), между которыми установлены отношения включения и расширения.

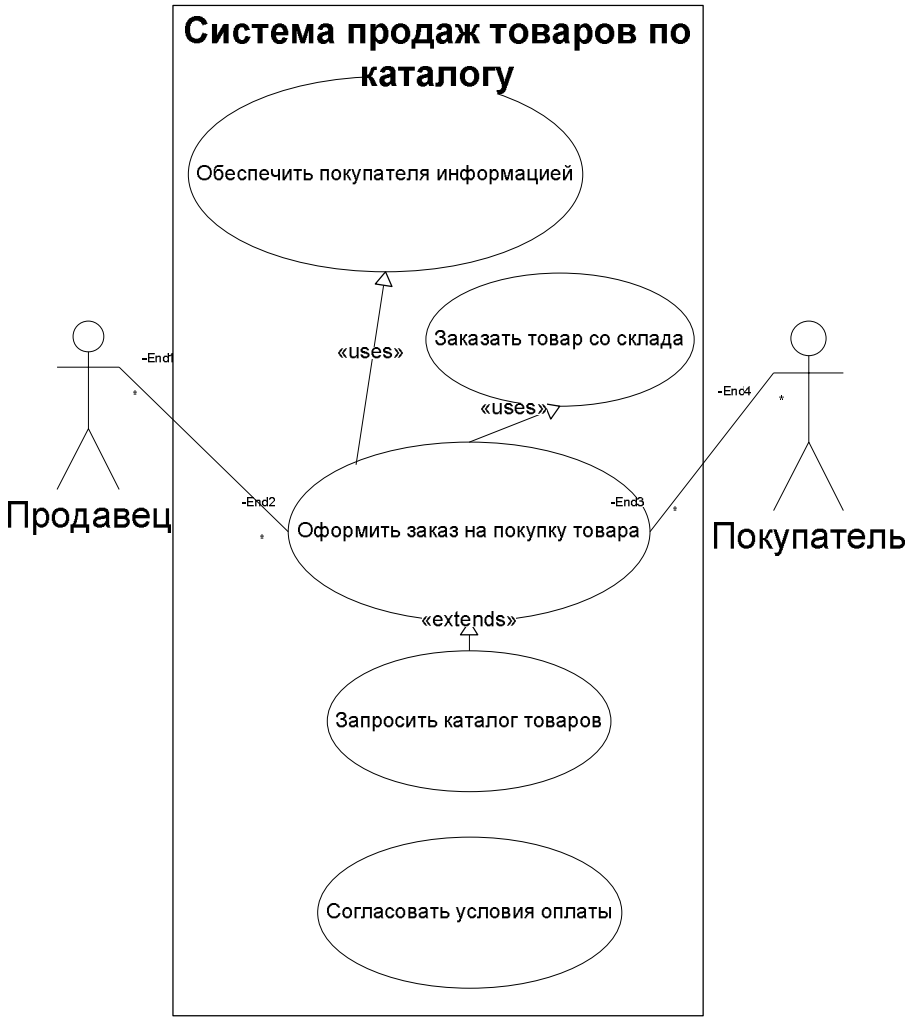


Рис.2.6.

Пример 2.2. Построение диаграммы вариантов для процесса моделирования системы учета в налоговой инспекции.

В качестве предметной области, как и в примере применения структурного подхода, рассматривается учета налогоплательщиков- организаций. На стадии формирования требований строится диаграмма вариантов использования (рис.2.7).

Рис.2.7 Начальная диаграмма использования

Практическое занятие №6. **Построение диаграммы классов.**

Диаграмма классов (class diagram) служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы.

С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы.

Построение диаграмм классов можно рассматривать в различных аспектах:

- *концептуальный аспект* – диаграммы классов отображают понятия изучаемой предметной области (моделируемой организации). Концептуальную модель можно рассматривать как не зависимую от средств реализации (языка программирования);
- *аспект спецификации* – модель спускается на уровень ПО, но рассматриваются только интерфейсы, а не программная реализация классов (под интерфейсом здесь понимается набор операций класса, видимых извне);
- *аспект реализации* – модель действительно определяет реализацию классов ПО. Этот аспект наиболее важен для программистов.

  При построении диаграммы необходимо выбрать единственный аспект. При чтении диаграммы следует выяснить, в соответствие с каким аспектом она строилась.

Класс

Класс (class) в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов.

Классы объектов могут иметь различные стереотипы поведения: объекты-сущности, управляющие объекты, интерфейсные объекты.

Интерфейсный объект – активный объект, форма взаимодействия информационной системы с пользователем (экранная форма, меню, командная строка, кнопка).

Управляющий объект – активный объект, координирующий выполнение функций.

Сущность – пассивный объект, над которым выполняется обработка процесса.

Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции (рис. 2.8). В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

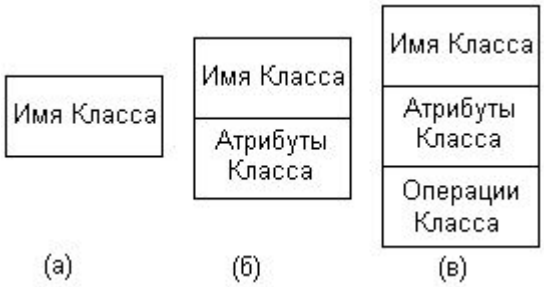


Рис. 2.8**.** Графическое изображение класса на диаграмме классов

Обязательным элементов обозначения класса является его имя. На начальных этапах разработки диаграммы отдельные классы могут обозначаться простым прямоугольником с указанием только имени соответствующего класса (рис. 2.8, а). По мере проработки отдельных компонентов, диаграммы описания классов дополняются атрибутами (рис. 2.8, б) и операциями (рис. 2.8, в).
Предполагается, что окончательный вариант диаграммы содержит наиболее полное описание классов, которые состоят из трех разделов или секций. Иногда в обозначениях классов используется дополнительный четвертый раздел, в котором приводится семантическая информация справочного характера или явно указываются исключительные ситуации.
Даже если секция атрибутов и операций является пустой, в обозначении класса она выделяется горизонтальной линией, чтобы сразу отличить класс от других элементов языка UML. Примеры графического изображения классов на диаграмме классов приведены на рис. 2.9. В первом случае для класса "Прямоугольник" (рис. 2.9, а) указаны только его атрибуты - точки на координатной плоскости, которые определяют его расположение. Для класса "Окно" (рис. 2.9, б) указаны только его операции, секция атрибутов оставлена пустой. Для класса "Счет" (рис. 2.9, в) дополнительно изображена четвертая секция, в которой указано исключение - отказ от обработки просроченной кредитной карточки.

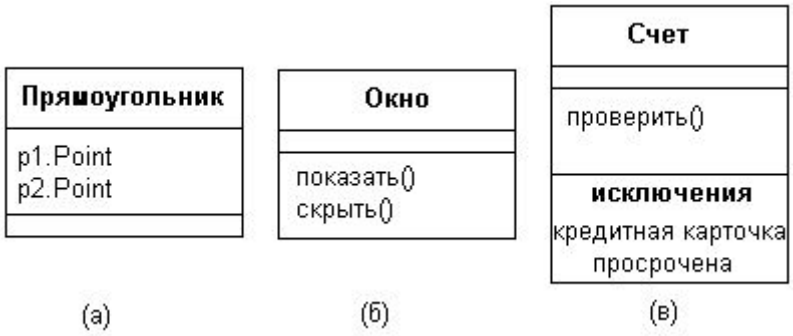


Рис.2.9**.** Примеры графического изображения классов на диаграмме

Имя класса

Имя класса должно быть уникальным в пределах пакета, который описывается некоторой совокупностью диаграмм классов (возможно, одной диаграммой). Оно указывается в первой верхней секции прямоугольника. В первой секции обозначения класса могут находиться ссылки на стандартные шаблоны или абстрактные классы, от которых образован данный класс и, соответственно, от которых он наследует свойства и методы. В этой секции может приводиться информация о разработчике данного класса и статус состояния разработки, а также могут записываться и другие общие свойства этого класса, имеющие отношение к другим классам диаграммы или стандартным элементам языка UML.
Примерами имен классов могут быть такие существительные, как "Сотрудник", "Компания", "Руководитель", "Клиент", "Продавец", "Менеджер", "Офис" и многие другие, имеющие непосредственное отношение к моделируемой предметной области и функциональному назначению проектируемой системы.

Класс может не иметь экземпляров или объектов. В этом случае он называется абстрактным классом, а для обозначения его имени используется наклонный шрифт (курсив). В языке UML принято общее соглашение о том, что любой текст, относящийся к абстрактному элементу, записывается курсивом. Данное обстоятельство является семантическим аспектом описания соответствующих элементов языка UML.

Атрибуты класса

Во второй сверху секции прямоугольника класса записываются его атрибуты (attributes) или свойства. Рассмотрим класс Клиент.

Клиент
Имя Адрес
Кредитный рейтинг()

На *концептуальном уровне* наличие атрибута «Имя клиента» указывает на то, что Клиенты обладают именами. На уровне спецификаций этот атрибут указывает на то, что объект Клиент может сообщить свое имя и обладает некоторым способом его определения. На уровне реализации Клиент содержит поле (называемое также переменной или элементом данных), соответствующее его имени.

В зависимости от степени детальности программы обозначение атрибута может включать имя атрибута, тип и значение, присваиваемое по умолчанию.

В языке UML принята определенная стандартизация записи атрибутов класса, которая подчиняется некоторым синтаксическим правилам. Каждому атрибуту класса соответствует отдельная строка текста, которая состоит из квантора видимости атрибута, имени атрибута, его кратности, типа значений атрибута и, возможно, его исходного значения:

<квантор видимости><имя атрибута>[кратность]:

<тип атрибута> = <исходное значение>{строка-свойство}

Квантор видимости может принимать одно из трех возможных значений и, соответственно, отображается при помощи специальных символов:

- Символ "+" обозначает атрибут с областью видимости типа общедоступный (public). Атрибут с этой областью видимости доступен или виден из любого другого класса пакета, в котором определена диаграмма.
- Символ "#" обозначает атрибут с областью видимости типа защищенный (protected). Атрибут с этой областью видимости недоступен или невиден для всех классов, за исключением подклассов данного класса.
- И, наконец, знак "-" обозначает атрибут с областью видимости типа закрытый (private). Атрибут с этой областью видимости недоступен или невиден для всех классов без исключения.

Квантор видимости может быть опущен. В этом случае его отсутствие просто означает, что видимость атрибута не указывается. Эта ситуация отличается от принятых по умолчанию соглашений в традиционных языках программирования, когда отсутствие квантора видимости трактуется как public или private. Однако вместо условных графических обозначений можно записывать соответствующее ключевое слово: public, protected, private.

Имя атрибута представляет собой строку текста, которая используется в качестве идентификатора соответствующего атрибута и поэтому должна быть уникальной в пределах данного класса. Имя атрибута является единственным обязательным элементом синтаксического обозначения атрибута.

Кратность атрибута характеризует общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса. В общем случае кратность записывается в форме строки текста в квадратных скобках после имени соответствующего атрибута:

```
[нижняя_граница1 .. верхняя_граница1, нижняя_граница2..
верхняя_граница2,      ...,      нижняя_границаk      ..
верхняя_границаk],
```

где нижняя_граница и верхняя_граница являются положительными целыми числами, каждая пара которых служит для обозначения отдельного замкнутого интервала целых чисел, у которого нижняя (верхняя) граница равна значению нижняя_граница (верхняя_граница). В целом данное условное обозначение кратности соответствует теоретико-множественному объединению соответствующих интервалов. В качестве верхней_границы может использоваться специальный символ "*", который означает произвольное положительное целое число. Другими словами, это означает неограниченное сверху значение кратности соответствующего атрибута.

Значения кратности из интервала следуют в монотонно возрастающем порядке без пропуска отдельных чисел, лежащих между нижней и верхней границами. При этом придерживаются следующего правила: соответствующие нижние и верхние границы интервалов включаются в значение кратности. Если в качестве кратности указывается единственное число, то кратность атрибута принимается равной данному числу. Если же указывается единственный знак "*", то это означает, что кратность атрибута может быть произвольным положительным целым числом или нулем.

В качестве примера рассмотрим следующие варианты задания кратности атрибутов.

- [0..1] означает, что кратность атрибута может принимать значение 0 или 1. При этом 0 означает отсутствие значения для данного атрибута.
- [0..*] означает, что кратность атрибута может принимать любое положительное целое значение большее или равное 0. Эта кратность может быть записана короче в виде простого символа - [*].
- [1.:*] означает, что кратность атрибута может принимать любое положительное целое значение большее или равное 1.

- [1..5] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 4, 5.
- [1..3,5,7] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 5, 7.
- [1..3,7.. 10] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 7, 8, 9, 10.
- [1..3,7..*] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, а также любое положительное целое значение большее или равное 7.

Если кратность атрибута не указана, то по умолчанию принимается ее значение равное 1..1, т. е. в точности 1.

Тип атрибута представляет собой выражение, семантика которого определяется языком спецификации соответствующей модели. В нотации UML тип атрибута иногда определяется в зависимости от языка программирования, который предполагается использовать для реализации данной модели. В простейшем случае тип атрибута указывается строкой текста, имеющей осмысленное значение в пределах пакета или модели, к которым относится рассматриваемый класс.

Можно привести следующие примеры задания имен и типов атрибутов классов:

- цвет: Color - здесь цвет является именем атрибута, Color - именем типа данного атрибута. Указанная запись может определять традиционно используемую RGB-модель (красный, зеленый, синий) для представления цвета. В этом случае имя типа Color как раз и характеризует семантическую конструкцию, которая применяется в большинстве языков программирования для представления цвета.
- имя_сотрудника [1..2]: String - здесь имя_сотрудника является именем атрибута, который служит для представления информации об имени, а возможно, и отчестве конкретного сотрудника. Тип атрибута String (Строка) как раз и указывает на тот факт, что отдельное значение имени представляет собой строку текста из одного или двух слов (например, "Кирилл" или "Дмитрий Иванович"). Поскольку во многих языках программирования существует тип данных String, использование соответствующего англоязычного термина не вызывает недоразумения у большинства программистов. Однако, хотя в языке UML все термины даются в англоязычном представлении, использование в качестве типа атрибута Строка в данной ситуации не исключается и определяется только соображениями удобства.
- видимость:Boolean - здесь видимость есть имя абстрактного атрибута (курсив здесь не случаен), который может характеризовать наличие визуального представления соответствующего класса на экране монитора. В этом случае тип Boolean означает, что возможными значениями данного атрибута является одно из двух логических значений: истина (true) или ложь (false). При этом значение истина может соответствовать наличию графического изображения на экране

монитора, а значение ложь - его отсутствию, о чем дополнительно указывается в пояснительном тексте. Поскольку кратность атрибута видимость не указана, она принимает значение 1 по умолчанию. В этой ситуации англоязычное имя типа атрибута вполне оправдано наличием соответствующего базового типа в языках программирования. Абстрактный характер данного атрибута обозначается курсивным текстом в записи данного атрибута.

- форма:Многоугольник - здесь имя атрибута форма может характеризовать такой класс, который является геометрической фигурой на плоскости. В этом случае тип атрибута Многоугольник указывает на тот факт, что отдельная геометрическая фигура может иметь форму треугольника, прямоугольника, ромба, пятиугольника и любого другого многоугольника, но не окружности или эллипса. Вполне очевидно, что в данной ситуации использование соответствующего англоязычного термина вряд ли целесообразно, поскольку тип Многоугольник не является базовым для языков программирования.

Исходное значение служит для задания некоторого начального значения для соответствующего атрибута в момент создания отдельного экземпляра класса. Здесь необходимо придерживаться правила принадлежности значения типу конкретного атрибута. Если исходное значение не указано, то значение соответствующего атрибута не определено на момент создания нового экземпляра класса. С другой стороны, конструктор соответствующего объекта может переопределять исходное значение в процессе выполнения программы, если в этом возникает необходимость.

В качестве примеров исходных значений атрибутов можно привести следующие дополненные выше варианты задания атрибутов:

- цвет:Color = (255, 0, 0) - в RGB-модели цвета это соответствует чистому красному цвету в качестве исходного значения для данного атрибута.
- имя_сотрудника[1..2]:String = Иван Иванович - возможно, это нетипичный случай, который, скорее, соответствует ситуации имя_руководителя[2]:81пп§ = Иван Иванович.
- видимость:Boolean = истина - может соответствовать ситуации, когда в момент создания экземпляра класса создается видимое на экране монитора окно, соответствующее данному объекту.
- форма:Многоугольник = прямоугольник - вряд ли требует комментариев, поскольку здесь речь идет о геометрической форме создаваемого объекта.

При задании атрибутов могут быть использованы две дополнительные синтаксические конструкции - это подчеркивание строки атрибута и пояснительный текст в фигурных скобках.

Подчеркивание строки атрибута означает, что соответствующий атрибут может принимать подмножество значений из некоторой области значений атрибута, определяемой его типом. Эти значения можно рассматривать как

набор однотипных записей или массив, которые в совокупности характеризуют каждый объект класса.
Например, если некоторый атрибут задан в виде форма: Прямоугольник. то это будет означать, что все объекты данного класса могут иметь несколько различных форм, каждая из которых является прямоугольником. Другим примером может служить задание атрибута в виде номер_счета:Integer. что может означать для объекта Сотрудник наличие некоторого подмножества счетов, общее количество которых заранее не фиксируется.
Строка-свойство служит для указания значений атрибута, которые не могут быть изменены в программе при работе с данным типом объектов. Фигурные скобки как раз и обозначают фиксированное значение соответствующего атрибута для класса в целом, которое должны принимать все вновь создаваемые экземпляры класса без исключения. Это значение принимается за исходное значение атрибута, которое не может быть переопределено в последующем. Отсутствие строки-свойства по умолчанию трактуется так, что значение соответствующего атрибута может быть изменено в программе. Например, строка-свойство в записи атрибута заработная_плата:Currency = = {$500} может служить для обозначения фиксированной заработной платы для каждого объекта класса "Сотрудник" определенной должности в некоторой организации. С другой стороны, запись данного атрибута в виде заработная_плата: Currency = $500 означает уже нечто иное, а именно - при создании нового экземпляра Сотрудник (аналогия - прием на работу нового сотрудника) для него устанавливается по умолчанию заработная плата в $500. Однако для отдельных сотрудников могут быть сделаны исключения как в большую, так и в меньшую сторону, о чем необходимо позаботиться дополнительно в программе.
Операция
В третьей сверху секции прямоугольника записываются операции или методы класса. Операция (operation) представляет собой некоторый сервис, предоставляющий каждый экземпляр класса по определенному требованию (процессы, реализуемые классом) Совокупность операций характеризует функциональный аспект поведения класса.
На уровне *спецификаций* операции соответствуют общим методам над типом. В модели реализации может потребоваться отражение уровней секретности и защиты операций.
Запись операций класса в языке UML также стандартизована и подчиняется определенным синтаксическим правилам. При этом каждой операции класса соответствует отдельная строка, которая состоит из квантора видимости операции, имени операции, выражения типа возвращаемого операцией значения и, возможно, строка-свойство данной операции:
<квантор видимости><имя операции>(список параметров):
<выражение типа возвращаемого значения>{строка-свойство}
Квантор видимости, как и в случае атрибутов класса, может принимать одно из трех возможных значений и, соответственно, отображается при помощи специального символа. Символ "+" обозначает операцию с областью

видимости типа общедоступный (public). Символ "#" обозначает операцию с областью видимости типа защищенный (protected). И, наконец, символ "-" используется для обозначения операции с областью видимости типа закрытый (private).
Квантор видимости для операции может быть опущен. В этом случае его отсутствие просто означает, что видимость операции не указывается. Вместо условных графических обозначений также можно записывать соответствующее ключевое слово: public, protected, private.
Имя операции представляет собой строку текста, которая используется в качестве идентификатора соответствующей операции и поэтому должна быть уникальной в пределах данного класса. Имя оперции является единственным обязательным элементом синтаксического обозначения операции.
Список параметров является перечнем разделенных запятой формальных параметров, каждый из которых может быть представлен в следующем виде:
<вид параметра><имя параметра>:<выражение типа>=<значение параметра по умолчанию>.
Здесь вид параметра - есть одно из ключевых слов in, out или inout со значением in по умолчанию, в случае если вид параметра не указывается. Имя параметра есть идентификатор соответствующего формального параметра. Выражение типа является зависимой от конкретного языка программирования спецификацией типа возвращаемого значения для соответствующего формального параметра. Наконец, значение по умолчанию в общем случае представляет собой выражение для значения формального параметра, синтаксис которого зависит от конкретного языка программирования и подчиняется принятым в нем ограничениям.
Выражение типа возвращаемого значения также является зависимой от языка реализации спецификацией типа или типов значений параметров, которые возвращаются объектом после выполнения соответствующей операции. Двоеточие и выражение типа возвращаемого значения могут быть опущены, если операция не возвращает никакого значения. Для указания кратности возвращаемого значения данная спецификация может быть записана в виде списка отдельных выражений.
Строка-свойство служит для указания значений свойств, которые могут быть применены к данному элементу. Строка-свойство не является обязательной, она может отсутствовать, если никакие свойства не специфицированы.
Операция с областью действия на весь класс показывается подчеркиванием имени и строки выражения типа. По умолчанию под областью операции понимается объект класса. В этом случае имя и строка выражения типа операции не подчеркиваются.
Операция, которая не может изменять состояние системы и, соответственно, не имеет никакого побочного эффекта, обозначается строкой-свойством "{запрос}" ("{query}"). В противном случае операция может изменять состояние системы, хотя нет никаких гарантий, что она будет это делать.
Для повышения производительности системы одни операции могут выполняться параллельно или одновременно, а другие - только

последовательно. В этом случае для указания параллельности выполнения операции используется строка-свойство вида "{concurrency = имя}", где имя может принимать одно из следующих значений: последовательная (sequential), параллельная (concurrent), охраняемая (guarded). При этом придерживаются следующей семантики для данных значений:

- последовательная (sequential) - для данной операции необходимо обеспечить ее единственное выполнение в системе, одновременное выполнение других операций может привести к ошибкам или нарушениям целостности объектов класса.
- параллельная (concurrent) - данная операция в силу своих особенностей может выполняться параллельно с другими операциями в системе, при этом параллельность должна поддерживаться на уровне реализации модели.
- охраняемая (guarded) - все обращения к данной операции должны быть строго упорядочены во времени с целью сохранения целостности объектов данного класса, при этом могут быть приняты дополнительные меры по контролю исключительных ситуаций на этапе ее выполнения.

С целью сокращения обозначений допускается использование одного имени в качестве строки-свойства для указания соответствующего значения параллельности. Отсутствие данной строки-свойства означает, что семантика параллельности для операции не определена. Поэтому следует предположить худший с точки зрения производительности случай, когда данная операция требует последовательного выполнения.

Появление сигнатуры операции на самом верхнем уровне объявляет эту операцию на весь класс, при этом данная операция наследуется всеми потомками данного класса. Если в некотором классе операция не выполняется (т. е. некоторый метод не применяется), то такая операция может быть помечена как абстрактная "{abstract}". Другой способ показать абстрактный характер операции - записать ее сигнатуру курсивом. Подчиненное появление записи данной операции без свойства {абстрактная} указывает на тот факт, что соответствующий класс-потомок может выполнять данную операцию в качестве своего "метода.

Если для некоторой операции необходимо дополнительно указать особенности ее реализации (например, алгоритм), то это может быть сделано в форме примечания, записанного в виде текста, который присоединяется к записи операции в соответствующей секции класса. Если объекты класса принимают и реагируют на некоторый сигнал, то запись данной операции помечается ключевым словом "сигнал" ("signal"). Это обозначение равнозначно обозначению некоторой операции. Реакция объекта на прием сигнала может быть показана в виде некоторого автомата. Кроме других случаев эта нотация может быть использована, чтобы показать реакцию объектов класса на ошибочные ситуации или исключения, которые могут моделироваться как сигналы или сообщения.

Список формальных параметров и тип возвращаемого значения могут не указываться. Квантор видимости атрибутов и операций может быть указан в виде специального значка или символа, которые используются для графического представления моделей в некотором инструментальном средстве. Имена операций, так же как и атрибутов, записываются со строчной (малой) буквы, а их типы - с заглавной (большой) буквы. При этом обязательной частью строки записи операции является наличие имени операции и круглых скобок.

В качестве примеров записи операций можно привести следующие обозначения отдельных операций:

- +создать() - может обозначать абстрактную операцию по созданию отдельного объекта класса, которая является общедоступной и не содержит формальных параметров. Эта операция не возвращает никакого значения после своего выполнения.
- +нарисовать(форма: Многоугольник = прямоугольник, цвет_заливки: Color = (O, O, 255)) - может обозначать операцию по изображению на экране монитора прямоугольной области синего цвета, если не указываются другие значения в качестве аргументов данной операции.
- запросить_счет_клиента (номер_счета:Integer):Currency - обозначает операцию по установлению наличия средств на текущем счете клиента банка. При этом аргументом данной операции является номер счета клиента, который записывается в виде целого числа (например, "123456"). Результатом выполнения этой операции является некоторое число, записанное в принятом денежном формате (например, $1,500.00).
- выдать_сообщение():{"Ошибка деления на ноль"} - смысл данной операции не требует пояснения, поскольку содержится в строке-свойстве операции. Данное сообщение может появиться на экране монитора в случае попытки деления некоторого числа на ноль, что недопустимо.

Отношения между классами

Кроме внутреннего устройства или структуры классов на соответствующей диаграмме указываются различные отношения между классами. При этом совокупность типов таких отношений фиксирована в языке UML и предопределена семантикой этих типов отношений. Базовыми отношениями или связями в языке UML являются:

- Отношение зависимости (dependency relationship)
- Отношение ассоциации (association relationship)
- Отношение обобщения (generalization relationship)
- Отношение реализации (realization relationship)

Каждое из этих отношений имеет собственное графическое представление на диаграмме, которое отражает взаимосвязи между объектами соответствующих классов.

Отношение зависимости

Отношение зависимости в общем случае указывает некоторое семантическое отношение между двумя элементами модели или двумя множествами таких элементов, которое не является отношением ассоциации, обобщения или реализации. Оно касается только самих элементов модели и не требует множества отдельных примеров для пояснения своего смысла. Отношение зависимости используется в такой ситуации, когда некоторое изменение одного элемента модели может потребовать изменения другого зависимого от него элемента модели.

Отношение зависимости графически изображается пунктирной линией между соответствующими элементами со стрелкой на одном из ее концов ("->" или "<-"). На диаграмме классов данное отношение связывает отдельные классы между собой, при этом стрелка направлена от класса-клиента зависимости к независимому классу или классу-источнику (рис. 2.10). На данном рисунке изображены два класса: Класс_А и Класс_Б, при этом Класс_Б является источником некоторой зависимости, а Класс_А - клиентом этой зависимости.

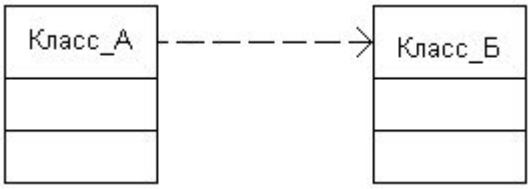


Рис. 2.10. Графическое изображение отношения зависимости на диаграмме классов

В качестве класса-клиента и класса-источника зависимости могут выступать целые множества элементов модели. В этом случае одна линия со стрелкой, выходящая от источника зависимости, расщепляется в некоторой точке на несколько отдельных линий, каждая из которых имеет отдельную стрелку для класса-клиента. Например, если функционирование Класса_С зависит от особенностей реализации Класса_А и Класса_Б, то данная зависимость может быть изображена следующим образом (рис. 2.11).

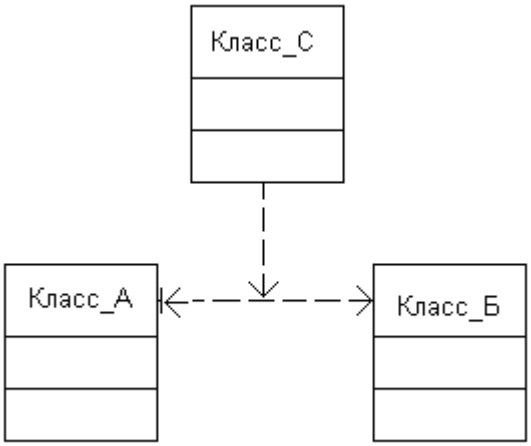


Рис. 2.11. Графическое представление зависимости между классом-клиентом (Класс_С) и классами-источниками (Класс_А и Класс_Б)

Стрелка может помечаться необязательным, но стандартным ключевым словом в кавычках и необязательным индивидуальным именем. Для отношения зависимости предопределены ключевые слова, которые обозначают некоторые специальные виды зависимостей. Эти ключевые слова (стереотипы) записываются в кавычках рядом со стрелкой, которая соответствует данной зависимости. Примеры стереотипов для отношения зависимости представлены ниже:

- "access" - служит для обозначения доступности открытых атрибутов и операций класса-источника для классов-клиентов;
- "bind" - класс-клиент может использовать некоторый шаблон для своей последующей параметризации;
- "derive" - атрибуты класса-клиента могут быть вычислены по атрибутам класса-источника;
- "import" - открытые атрибуты и операции класса-источника становятся частью класса-клиента, как если бы они были объявлены непосредственно в нем;
- "refine" - указывает, что класс-клиент служит уточнением класса-источника в силу причин исторического характера, когда появляется дополнительная информация в ходе работы над проектом.

Отношение ассоциации

Отношение ассоциации соответствует наличию некоторого отношения между экземплярами классов (личность работает в компании, компания имеет ряд офисов).

С *концептуальной точки зрения*, ассоциации представляют собой концептуальные связи между классами. Данное отношение обозначается сплошной линией с дополнительными специальными символами, которые характеризуют отдельные свойства конкретной ассоциации. В качестве дополнительных специальных символов могут использоваться имя ассоциации, а также имена и кратность классов-ролей ассоциации. Имя ассоциации является необязательным элементом ее обозначения. Если оно задано, то записывается с заглавной (большой) буквы рядом с линией соответствующей ассоциации.

Наиболее простой случай данного отношения - бинарная ассоциация. Она связывает в точности два класса и, как исключение, может связывать класс с самим собой. Для бинарной ассоциации на диаграмме может быть указан порядок следования классов с использованием треугольника в форме стрелки рядом с именем данной ассоциации. Направление этой стрелки указывает на порядок классов, один из которых является первым (со стороны треугольника), а другой - вторым (со стороны вершины треугольника). Отсутствие данной стрелки рядом с именем ассоциации означает, что порядок следования классов в рассматриваемом отношении не определен.

В качестве простого примера отношения бинарной ассоциации рассмотрим отношение между двумя классами - классом "Компания" и классом "Сотрудник" (рис. 2.12). Они связаны между собой бинарной ассоциацией Работа, имя которой указано на рисунке рядом с линией ассоциации. Для

данного отношения определен порядок следования классов, первым из которых является класс "Сотрудник", а вторым - класс "Компания". Отдельным примером или экземпляром данного отношения может являться пара значений (Петров И. И., "Рога&Копыта"). Это означает, что сотрудник Петров И. И. работает в компании "Рога&Копыта".

Рис. 2.12 Графическое изображение отношения бинарной ассоциации между классами.

Тернарная ассоциация и ассоциации более высокой арности в общем случае называются N-арной ассоциацией (читается - "эн арная ассоциация"). Такая ассоциация связывает некоторым отношением 3 и более классов, при этом один класс может участвовать в ассоциации более чем один раз. Класс ассоциации играет определенную роль в соответствующем отношении, что может быть явно указано на диаграмме. Каждый экземпляр N-арной ассоциации представляет собой N-арный кортеж значений объектов из соответствующих классов. Бинарная ассоциация является частным случаем N-арной ассоциации, когда значение N=2, и имеет свое собственное обозначение.

N-арная ассоциация графически обозначается ромбом, от которого ведут линии к символам классов данной ассоциации. В этом случае ромб соединяется с символами соответствующих классов сплошными линиями. Обычно линии проводятся от вершин ромба или от середины его сторон. Имя N-арной ассоциации записывается рядом с ромбом соответствующей ассоциации.

Порядок классов в N-арной ассоциации, в отличие от порядка множеств в отношении, на диаграмме не фиксируется. Некоторый класс может быть присоединен к ромбу пунктирной линией. Это означает, что данный класс обеспечивает поддержку свойств соответствующей N-арной ассоциации, а сама N-арная ассоциация имеет атрибуты, операции и/или ассоциации. Другими словами, такая ассоциация, в свою очередь, является классом с соответствующим обозначением в виде прямоугольника и является самостоятельным элементом языка UML - ассоциацией-классом (Association Class). N-арная ассоциация не может содержать символ агрегации ни для какой из своих ролей.

В качестве примера конкретной тернарной ассоциации рассмотрим отношение между тремя классами: "Футбольная команда", "Год" и "Игра". Данная ассоциация указывает на наличие отношения между этими тремя

классами, которое может представлять информацию об играх футбольных команд в национальном чемпионате в течение нескольких последних лет (рис. 2.13).

Как уже упоминалось, отдельный класс ассоциации имеет собственную роль в отношении. Эта роль может быть изображена графически на диаграмме классов. С этой целью в языке UML вводится в рассмотрение специальный элемент - конец ассоциации (Association End), который графически соответствует точке соединения линии ассоциации с отдельным классом. Конец ассоциации является частью ассоциации, но не класса. Каждая ассоциация имеет два или больше концов ассоциации. Наиболее важные свойства ассоциации указываются на диаграмме рядом с этими элементами ассоциации и должны перемешаться вместе с ними.

Рис. 2.13. Графическое изображение тернарной ассоциации между тремя классами.

Одним из таких дополнительных обозначений является имя роли отдельного класса, входящего в ассоциацию. Имя роли представляет собой строку текста рядом с концом ассоциации для соответствующего класса. Она указывает специфическую роль, которую играет класс, являющийся концом рассматриваемой ассоциации. Имя роли не является обязательным элементом обозначений и может отсутствовать на диаграмме.

Следующий элемент обозначений - **кратность отдельных классов**, являющихся концами ассоциации. Кратность отдельного класса обозначается в виде интервала целых чисел, аналогично кратности атрибутов и операций классов. Интервал записывается рядом с концом ассоциации и для N-арной ассоциации означает потенциальное число отдельных экземпляров или значений кортежей этой ассоциации, которые могут иметь место, когда остальные N-1 экземпляров или значений классов фиксированы. Так, для рассмотренного ранее примера (см. рис. 2.12) кратность "1" для класса "Компания" означает, что каждый сотрудник может работать только в одной компании. Кратность "1..*" для класса "Сотрудник" означает, что в каждой компании могут работать несколько сотрудников, общее число которых заранее неизвестно и ничем не ограничено. Заметим, что вместо кратности "1..*" записать только символ "*" нельзя, поскольку последний означает кратность "0..*". Для данного примера это означало бы, что отдельные компании могут совсем не иметь сотрудников в своем штате. Но такая кратность вполне приемлема в других ситуациях, как это видно из рассмотренного выше примера (рис. 2.13). Что касается других свойств отношения, ассоциации, то в случае их наличия, они могут рассматриваться в качестве атрибутов класса ассоциации и могут быть указаны на диаграмме

обычным для класса способом в соответствующей секции прямоугольника класса.

Частным случаем отношения ассоциации является так называемая исключающая ассоциация (Xor-association). Семантика данной ассоциации указывает на тот факт, что из нескольких потенциально возможных вариантов данной ассоциации в каждый момент времени может использоваться только один ее экземпляр. На диаграмме классов исключающая ассоциация изображается пунктирной линией, соединяющей две и более ассоциации, рядом с которой записывается строка-ограничение "{xor}". Например, счет в банке может быть открыт для клиента, в качестве которого может выступать физическое лицо (индивидуум) или компания, что изображается с помощью исключающей ассоциации (рис. 2.14).

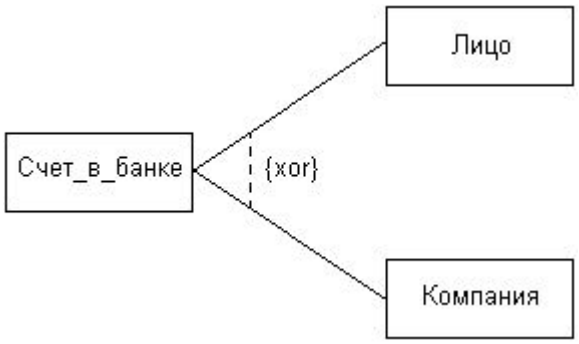


Рис. 2.14. Графическое изображение исключающей ассоциации между тремя классами.

Специальной формой или частным случаем отношения ассоциации является отношение агрегации, которое, в свою очередь, тоже имеет специальную форму – отношение композиции. Поскольку эти отношения имеют свои специальные обозначения и относятся к базовым понятиям языка UML, рассмотрим их последовательно.

Отношение агрегации

Отношение агрегации имеет место между несколькими классами в том случае, если один из классов представляет собой некоторую сущность, включающую в себя в качестве составных частей другие сущности. Данное отношение имеет фундаментальное значение для описания структуры сложных систем, поскольку применяется для представления системных взаимосвязей типа "часть-целое". Раскрывая внутреннюю структуру системы, отношение агрегации показывает, из каких компонентов состоит система и как они связаны между собой. С точки зрения модели отдельные части системы могут выступать как в виде элементов, так и в виде подсистем, которые, в свою очередь, тоже могут образовывать составные компоненты или подсистемы. Это отношение по своей сути описывает декомпозицию или разбиение сложной системы на более простые составные части, которые также могут быть подвергнуты декомпозиции, если в этом возникнет необходимость в последующем. Очевидно, что рассматриваемое в таком аспекте деление системы на составные части представляет собой некоторую иерархию ее компонентов, однако данная иерархия принципиально

отличается от иерархии, порождаемой отношением обобщения. Отличие заключается в том, что части системы никак не обязаны наследовать ее свойства и поведение, поскольку являются вполне самостоятельными сущностями. Более того, части целого обладают своими собственными атрибутами и операциями, которые существенно отличаются от атрибутов и операций целого.

В качестве примера отношения агрегации рассмотрим взаимосвязь типа "часть-целое", которая имеет место между сущностью "Грузовой автомобиль" и такими компонентами, как "Двигатель", "Шасси", "Кабина", "Кузов". Не претендуя на точное соответствие терминологии данной предметной области, нетрудно представить себе, что грузовой автомобиль состоит из двигателя, шасси, кабины и кузова. Именно это отношение между классом "Грузовой_автомобиль" и классами "Двигатель", "Шасси", "Кабина", "Кузов" описывает отношение агрегации. Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой незакрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой "целое". Остальные классы являются его "частями" (рис. 2.15).

Рис. 2.15. Графическое изображение отношения агрегации в языке UML.

Еще одним примером отношения агрегации может служить известное каждому из читателей деление персонального компьютера на составные части: системный блок, монитор, клавиатуру и мышь. Используя обозначения языка UML, компонентный состав ПК можно представить в виде соответствующей диаграммы классов (рис. 2.16), которая в данном случае иллюстрирует отношение агрегации.

Рис. 2.16. Диаграмма классов для иллюстрации отношения агрегации на примере ПК

Отношение композиции

Отношение композиции, как уже упоминалось ранее, является частным случаем отношения агрегации. Это отношение служит для выделения специальной формы отношения "часть-целое", при которой составляющие части в некотором смысле находятся внутри целого. Специфика взаимосвязи между ними заключается в том, что части не могут выступать в отрыве от целого, т. е. с уничтожением целого уничтожаются и все его составные части.

Возможно, не самый лучший, но наверняка понятный всем пример этого отношения представляет собой живая клетка в биологии. Другой пример - окно интерфейса программы, которое может состоять из строки заголовка, кнопок управления размером, полос прокрутки, главного меню, рабочей области и строки состояния. Нетрудно понять, что подобное окно представляет собой класс, а его компоненты являются как классами, так и атрибутами или свойствами окна. Последнее обстоятельство весьма характерно для отношения композиции, поскольку отражает различные способы представления данного отношения. Графически отношение композиции изображается сплошной линией, один из концов которой представляет собой закрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой класс-композицию или "целое". Остальные классы являются его "частями" (рис. 2.17).

Рис. 2.17. Графическое изображение отношения композиции в языке UML

В качестве дополнительных обозначений для отношений композиции и агрегации могут использоваться дополнительные обозначения, применяемые для отношения ассоциации. А именно, указание кратности класса ассоциации и имени данной ассоциации, которые не являются обязательными. Применительно к описанному выше примеру класса "Окно_программы" его диаграмма классов может иметь следующий вид (рис. 2.18).

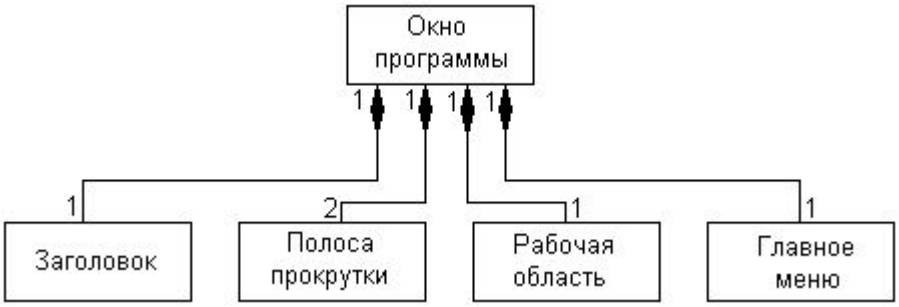


Рис. 2.18. Диаграмма классов для иллюстрации отношения композиции на примере класса окна программы

Данный пример может иллюстрировать и другие особенности разрабатываемой компьютерной программы, которые не указывались в явном виде при описании этого примера. Так, в частности, указание кратности 1 рядом с классом "Рабочая_область" характерно для однодокументных приложений.

Отношение обобщения

Отношение обобщения является обычным таксономическим отношением между более общим элементом (родителем или предком) и более частным или специальным элементом (дочерним или потомком). Данное отношение может использоваться для представления взаимосвязей между пакетами, классами, вариантами использования и другими элементами языка UML.

Применительно к диаграмме классов данное отношение описывает иерархическое строение классов и наследование их свойств и поведения. При этом предполагается, что класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет свои собственные свойства и поведение, которые отсутствуют у класса-предка. На диаграммах отношение обобщения обозначается сплошной линией с треугольной стрелкой на одном из концов (рис. 2.19). Стрелка указывает на более общий класс (класс-предок или суперкласс), а ее отсутствие - на более специальный класс (класс-потомок или подкласс).

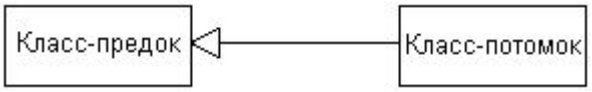


Рис. 2.19. Графическое изображение отношения обобщения в языке UML.

Как правило, на диаграмме может указываться несколько линий для одного отношения обобщения, что отражает его таксономический характер. В этом случае более общий класс разбивается на подклассы одним отношением Обобщения. Например, класс Геометрическая_фигура_на_плоскости (курсив обозначает абстрактный класс) может выступать в качестве суперкласса для подклассов, соответствующих конкретным геометрическим фигурам, таким как, Прямоугольник, Окружность, Эллипс и др. Данный факт может быть представлен графически в форме диаграммы классов следующего вида (рис. 2.20).

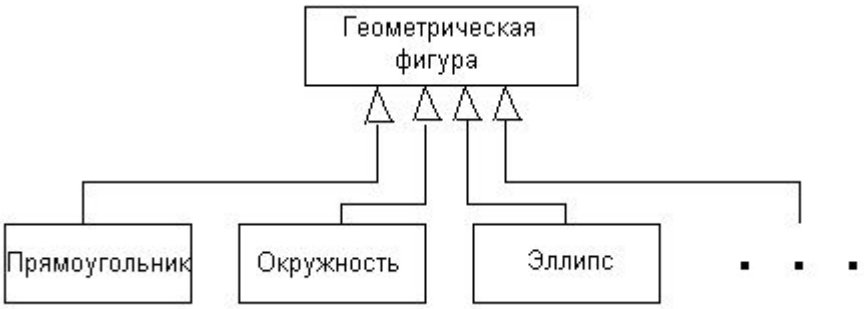


Рис. 2.20. Пример графического изображения отношения обобщения классов

С целью упрощения обозначений на диаграмме классов совокупность линий, обозначающих одно и то же отношение обобщения, может быть объединена в одну линию. В этом случае данные отдельные линии изображаются сходящимися к единственной .стрелке, имеющей с ними общую точку пересечения (рис. 2.21).

Рис. 2.21. Вариант графического изображения отношения обобщения классов для случая объединения отдельных линий

Это обозначение по форме соответствует графу специального вида, именно - иерархическому дереву. В этом случае класс-предок является корнем этого дерева, а классы-потомки - его листьями. Отличие заключается в возможности указания на диаграмме классов потенциальной возможности наличия других классов-потомков, которые не включены в обозначения представленных на диаграмме классов (многоточие вместо прямоугольника).
Рядом со стрелкой обобщения может размещаться строка текста, указывающая на некоторые дополнительные свойства этого отношения. Данный текст будет относиться ко всем линиям обобщения, которые идут к классам-потомкам. Другими словами, отмеченное свойство касается всех подклассов данного отношения. При этом текст следует рассматривать как ограничение, и тогда он записывается в фигурных скобках. В качестве ограничений могут быть использованы следующие ключевые слова языка UML:

- {complete} - означает, что в данном отношении обобщения специфицированы все классы-потомки, и других классов-потомков у данного класса-предка быть не может. Пример - класс Клиент_банка является предком для двух классов: Физическое_лицо и Компания, и других классов-потомков он не имеет. На соответствующей диаграмме классов это можно указать явно, записав рядом с линией обобщения данную строку-ограничение;
- {disjoint} - означает, что классы-потомки не могут содержать объектов, одновременно являющихся экземплярами двух или более классов. В приведенном выше примере это условие также выполняется, поскольку предполагается, что никакое конкретное физическое лицо не может являться одновременно и конкретной компанией. В этом случае рядом с линией обобщения можно записать данную строку-ограничение;
- {incomplete} - означает случай, противоположный первому. А именно, предполагается, что на диаграмме указаны не все классы-потомки. В последующем возможно восполнить их перечень, не изменяя уже построенную диаграмму. Пример - диаграмма класса "Автомобиль", для которой указание всех без исключения моделей автомобилей соизмеримо с созданием соответствующего каталога. С другой стороны, для отдельной задачи, такой как разработка системы продажи автомобилей конкретных моделей, в этом нет необходимости. Но указать неполноту структуры классов-потомков все же следует;
- {overlapping} - означает, что отдельные экземпляры классов-потомков могут принадлежать одновременно нескольким классам. Пример - класс "Многоугольник" является классом-предком для класса "Прямоугольник" и класса "Ромб". Однако существует отдельный класс "Квадрат", экземпляры которого одновременно являются объектами первых двух классов. Вполне естественно такую ситуацию указать явно с помощью данной строки-ограничения. С учетом возможности использования строк-ограничений диаграмма классов

(рис. 2.21) может быть изображена без многоточий и без потери информации (рис. 2.22).

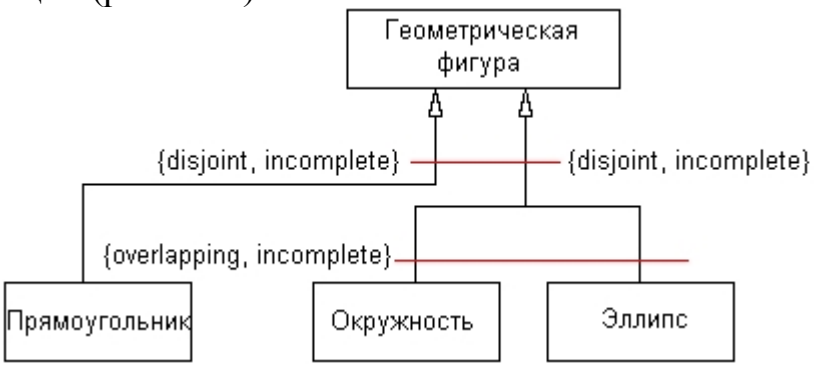


Рис. 2.22. Вариант графического изображения отношения обобщения классов с использованием строки-ограничения

Интерфейсы

Интерфейсы являются элементами диаграммы вариантов использования. Однако при построении диаграммы классов отдельные интерфейсы могут уточняться и в этом случае для их изображения используется специальный графический символ - прямоугольник класса с ключевым словом или стереотипом "interface" (рис. 2.23). При этом секция атрибутов у прямоугольника отсутствует, а указывается только секция операций.

Рис. 2.23. Пример графического изображения интерфейса на диаграмме классов

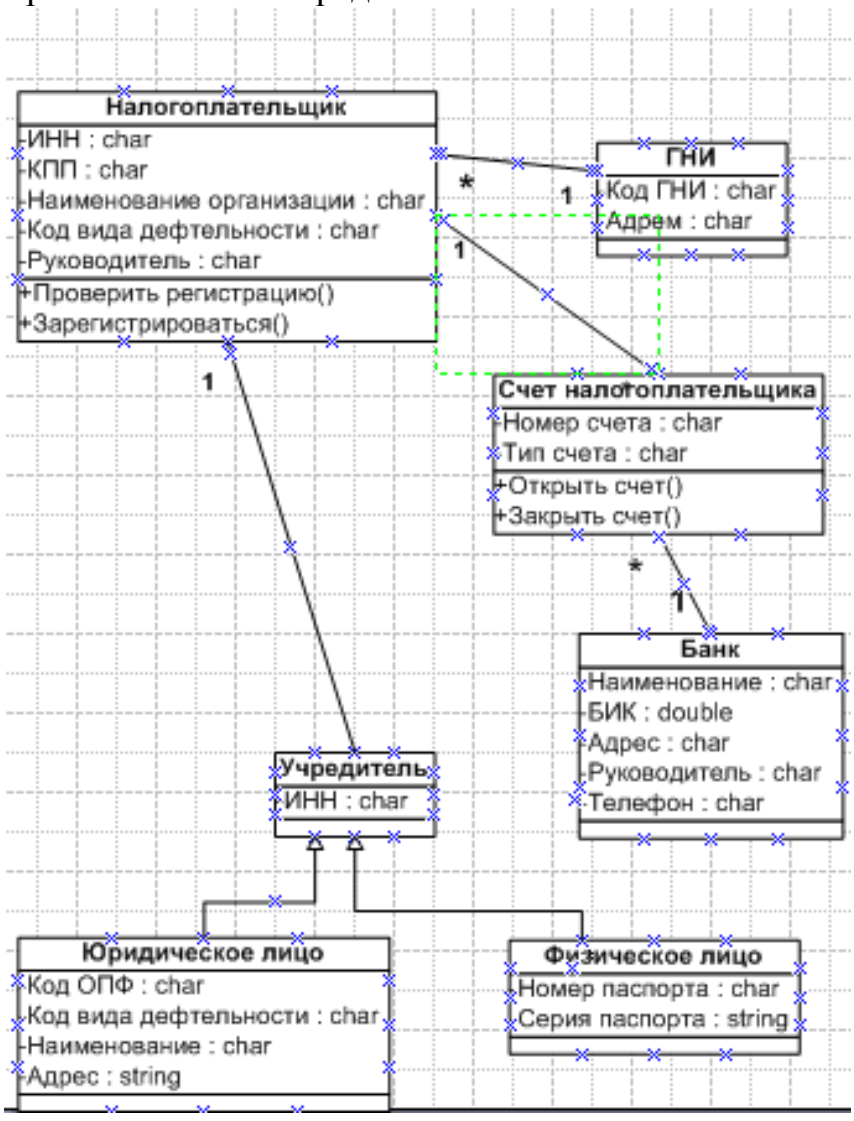
Объекты

Объект (object) является отдельным экземпляром класса, который создается на этапе выполнения программы. Он имеет свое собственное имя и конкретные значения атрибутов. В силу самых различных причин может возникнуть необходимость показать взаимосвязи не только между классами модели, но и между отдельными объектами, реализующими эти классы. В данном случае может быть разработана диаграмма объектов, которая, хотя и не является канонической в метамодели языка UML, но имеет самостоятельное назначение. Для графического изображения объектов используется такой же символ прямоугольника, что и для классов. Отличия проявляются при указании имен объектов, которые в случае объектов обязательно подчеркиваются (рис. 2.24). При этом запись имени объекта представляет собой строку текста "имя объекта:имя класса", разделенную двоеточием (рис. 2.24 а, б). Имя объекта может отсутствовать, в этом случае предполагается, что объект является анонимным, и двоеточие указывает на данное обстоятельство (рис. 2.24, г). Отсутствовать может и имя класса. Тогда указывается просто имя объекта (рис. 2.24, в). Атрибуты объектов принимают конкретные значения.

При изображении диаграммы объектов нужно помнить, что каждый объект представляет собой экземпляр соответствующего класса, а отношения между объектами описываются с помощью связей (links), которые являются экземплярами соответствующих отношений. При этом все связи изображаются сплошными линиями.

Рис. 2.24. Пример графического изображения объектов на диаграммах языка UML

Пример 2.3. Ниже приведена диаграмма классов, построенная в MS VISIO, для описания работы ГНИ как предметной области.



Практическое занятие №6. Диаграмма последовательности (sequence diagram)

Для моделирования взаимодействия объектов в языке UML используются соответствующие диаграммы взаимодействия (interaction diagrams).

Говоря об этих диаграммах, имеют в виду два аспекта взаимодействия. Во-первых, взаимодействие объектов можно рассматривать во времени, и тогда для представления временных особенностей передачи и приема сообщений между объектами используются *диаграммы последовательности*. Во – вторых, можно рассматривать структурные особенности взаимодействия объектов. Для представления структурных особенностей передачи и приема сообщений между объектами используют *диаграммы кооперации.*

Объекты

На диаграмме последовательности изображаются исключительно те объекты, которые непосредственно участвуют во взаимодействии и не показывают возможные статические ассоциации с другими объектами. Для диаграммы последовательности ключевым моментом является именно динамика взаимодействия объектов во времени. При этом диаграмма последовательности имеет как бы два измерения. Одно – слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии. Графически каждый объект изображается прямоугольником и располагается в верхней части своей линии жизни. Внутри прямоугольника записывается имя объекта и имя класса, разделенные двоеточием. При этом вся запись подчеркивается, что является признаком объекта, который, как известно, представляет собой экземпляр класса (Рис.2.25).

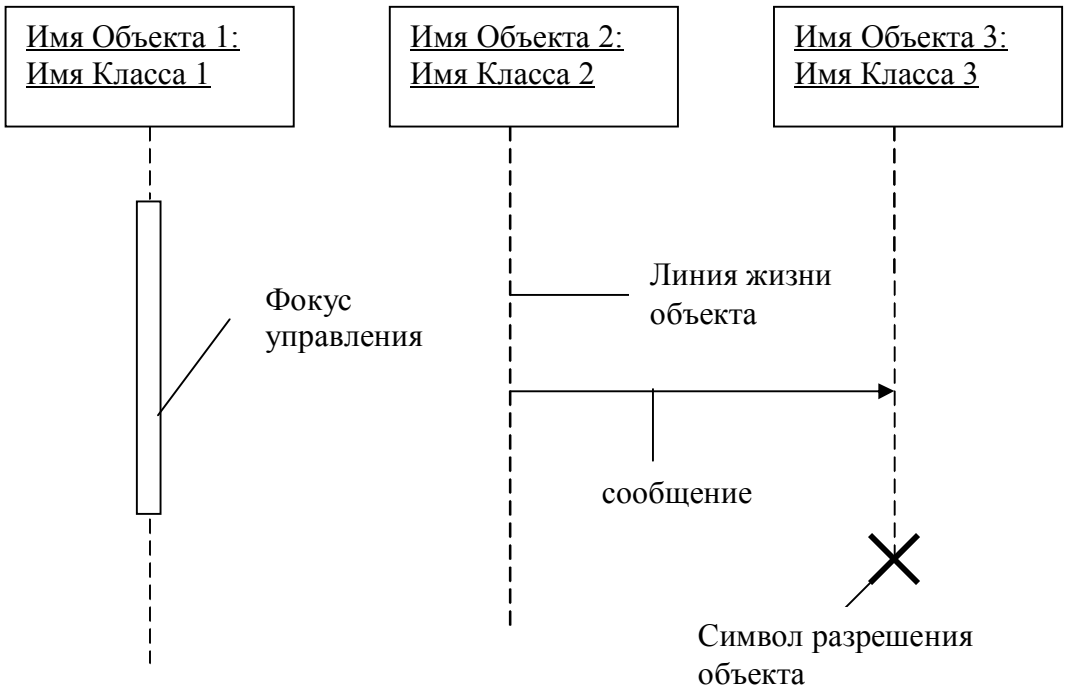


Рис.2.25. Различные графические примитивы диаграммы последовательности

Крайним слева на диаграмме изображается объект, который является инициатором взаимодействия. (объект 1 на рис.2.25). Правее изображается другой объект, который непосредственно взаимодействует с первым. Таким образом, все объекты на диаграмме последовательности образуют некоторый порядок, определяемый степенью активности этих объектов при взаимодействии друг с другом.

Второе измерение диаграммы последовательности – вертикальная временная ось, направленная сверху вниз. Начальному моменту времени соответствует самая верхняя часть диаграммы. При этом взаимодействия объектов реализуются посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения и также образуют порядок во времени своего возникновения. Другим словами, сообщения, расположенные на диаграмме последовательности выше, инициируются раньше тех, которые расположены ниже. При этом масштаб на оси времени не указывается, поскольку диаграмма последовательности моделирует лишь временную упорядоченность взаимодействий типа «раньше – позже».

Линия жизни объекта

Линия жизни объекта (object lifeline) изображается пунктирной вертикальной линией, ассоциированной с единственным объектом на диаграмме последовательности. Линия жизни служит для обозначения периода времени, в течение которого объект существует в системе и, следовательно, может потенциально участвовать во всех ее взаимодействиях. Если объект существует в системе постоянно, то его линия жизни должна продолжаться по всей плоскости диаграммы последовательности от самой верхней ее части до самой нижней (объекты 1 и 2 на рис.2.25).Отдельные объекты, выполнив свою роль в системе, могут быть уничтожены (разрушены), чтобы освободить занимаемые ими ресурсы. Для таких объектов линия жизни обрывается в момент его уничтожения.

Вовсе не обязательно создавать все объекты в начальный момент времени. Отдельные объекты могут создаваться по мере необходимости.

Фокус управления

В процессе функционирования объектно – ориентированных систем одни объекты могут находиться в активном состоянии, непосредственно выполняя определенные действия или в состоянии пассивного ожидания сообщений от других объектов. Чтобы явно выделить подобную активность объектов, в языке UML применяется специальное понятие, получившее название фокуса управления (focus of control). Фокус управления изображается в форме вытянутого узкого прямоугольника. Это прямоугольник располагается ниже обозначения соответствующего объекта и может заменить его линию жизни, если на всем ее протяжении он является активным. С другой стороны периоды активности объекта могут чередоваться с периодами его пассивности или ожидания. В этом случае у такого объекта имеются несколько фокусов управления (см.рис.2.26).

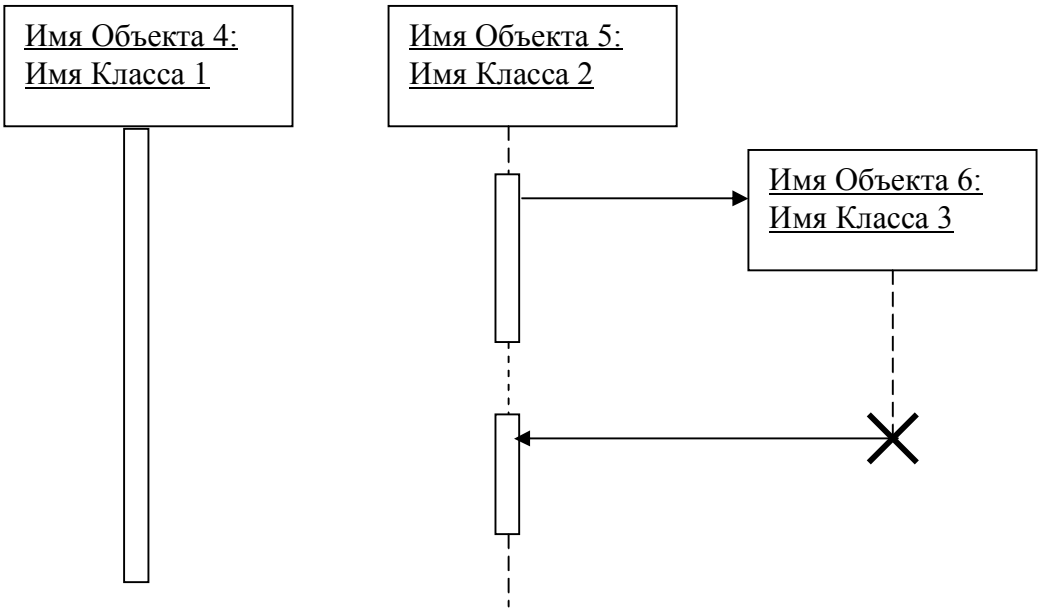


Рис.2.26. Графическое изображение различных вариантов линий жизни и фокусов управления объектов.

Важно понимать, что получить фокус управления может только существующий объект, у которого в этот момент имеется линия жизни. Если же некоторый объект был уничтожен, то вновь возникнуть в системе он уже не может. Вместо него лишь может быть создан другой экземпляр этого же класса, который, строго говоря, будет являться другим объектом.

В отдельных случаях инициатором взаимодействия в системе может быть актер или внешний пользователь.

Сообщения

Каждое взаимодействие объектов описывается совокупностью сообщений, которыми участвующие в нем объекты обмениваются между собой. В этом смысле каждое сообщение (message) представляет собой законченный фрагмент информации, который отправляется одним объектом другому. При этом прием сообщения инициирует выполнение определенных действий, направленных на решение отдельной задачи тем объектом, которому это сообщение отправлено.

Иногда отправителя сообщения называют клиентом, а получателя – сервером. При этом сообщение от клиента имеет форму запроса некоторого сервиса, а реакция сервера на запрос после получения сообщения может быть связана с выполнением определенных действий или передачи клиенту необходимой информации тоже в форме сообщения.

В языке UML могут встречаться несколько разновидностей сообщений, каждое из которых имеет свое графическое изображение (Рис.2.27). Разновидность сообщения а) используется для вызова процедур, выполнения операций или обозначения различных вложенных потоков управления. Разновидность сообщения б) используется для обозначения простого (не вложенного) потока управления. Каждая такая стрелка указывает на прогресс одного шага потока. При этом соответствующие сообщения обычно являются асинхронными, т.е. могут возникать в произвольные моменты времени.

Рис.2.27. Графическое изображение различных видов сообщений между объектами

Разновидность сообщения в) явно обозначает асинхронное сообщение между двумя объектами в некоторой процедурной последовательности. Примером такого сообщения может служить прерывание операции или возникновение исключительной ситуации. В этом случае информация о такой ситуации передается вызывающему объекту для продолжения процесса дальнейшего взаимодействия. Разновидность сообщения г) используется для возврата из вызова процедуры. Примером может служить простое сообщение о завершении некоторых вычислений без предоставления результата расчетов объекту-клиенту.

В отдельных случаях объект может посылать сообщение самому себе, инициируя так называемые *рефлексивные сообщения* (см. рис.5). Подобные ситуации возникают, например, при обработке нажатий клавиши клавиатуры при вводе текста в редактируемый документ, при наборе номера телефона абонента.

Таким образом, в языке UML каждое сообщение ассоциируется с некоторым действием, которое должно быть выполнено принявшим его объектом. При этом действие может иметь некоторые параметры, в зависимости от конкретных действий которых может быть получен различный результат. Соответствующие параметры будет иметь и вызывающее это действие сообщение. Значения параметров отдельных сообщений могут содержать условные выражения, образуя ветвление или альтернативные пути основного потока управления.

Пример 2.4. Моделирование взаимодействия программной системы обслуживания клиентов в банке (Рис.2.28).

Условием ветвления может служить сумма снимаемых клиентом средств со своего текущего счета. Если эта сумма превышает $1000, то могут потребоваться дополнительные действия, связанные с созданием и последующим разрушением объекта4. Если эта сумма превышает $50, но не превышает $1000, то управление передается объекту 3. И, наконец, если сумма не превышает $50, то управление получает объект 2.

При этом объекты 1,2 и 3 постоянно существуют в системе. Объект 4 создается только, если справедливо первое из альтернативных условий. В противном случае он может быть никогда не создан. После выполнения требуемых действий объекты 2 и 3 просто информируют объект 1 о завершении соответствующих операций, не требуя от него никаких действий

(пунктирная стрелка). Объект 4 после завершения своих действий уничтожается, передавая управление объекту 3, делая его активным (фокус управления).

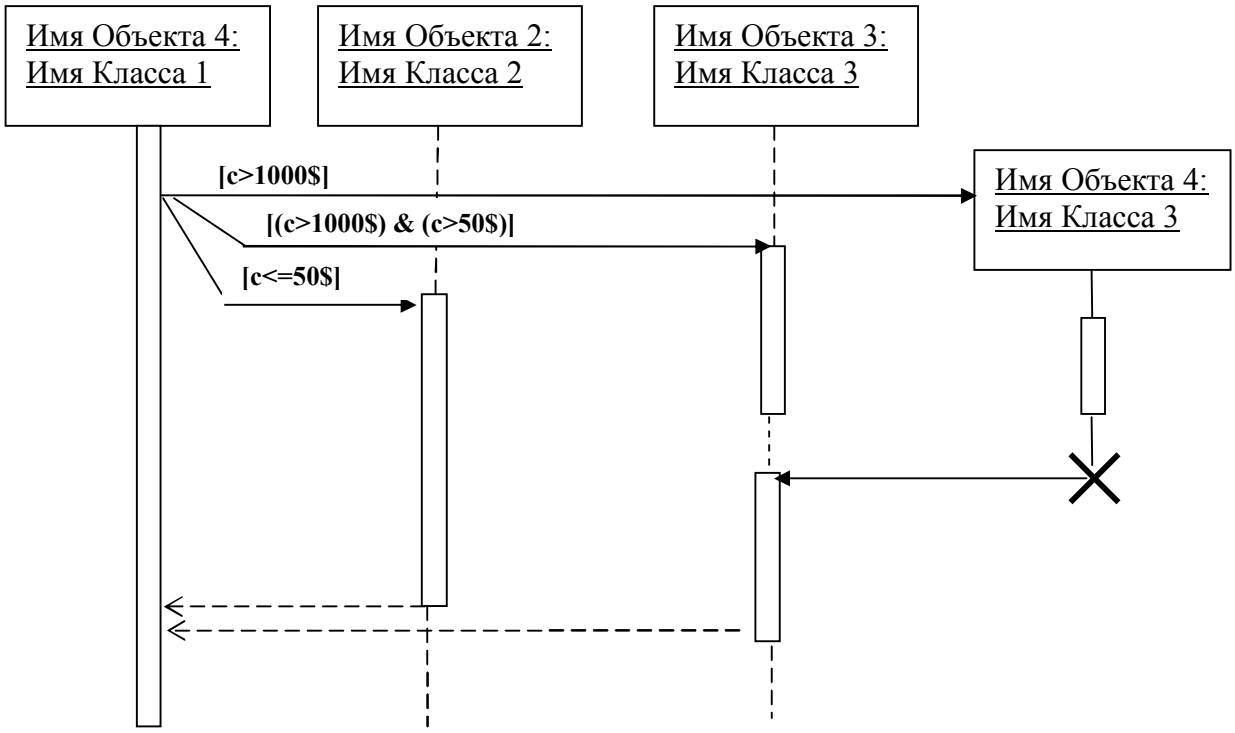


Рис.2.28

Стереотипы сообщений

В языке UMLпредусмотрены некоторые стандартные действия, выполняемые в ответ на получение соответствующего сообщения. Они могут быть явно указан на диаграмме последовательности в форме стереотипа рядом с сообщением, к которому они относятся. В этом случае они записываются в кавычках. Используются следующие обозначения для моделирования действий:

“call” (вызвать) - сообщение, требующее вызова операции или процедуры принимающего объекта;

“return” (возвратить) – сообщение, возвращающее значение выполненной операции или процедуры вызвавшему ее объекту. Значение результата может инициировать выполнение потока управления;

“create” (создать) – сообщение, требующее создание другого объекта для выполнения определенных действий. Созданный объект может получить фокус управления, а может и не получить;

“destroy” (уничтожить) – сообщение с явным требованием уничтожить соответствующий объект;

“send” (послать) – обозначает посылку другому объекту некоторого сигнала, который асинхронно инициируется одним объектом и принимается (перехватывается) другим. Отличие сигнала от сообщения заключается в том, что сигнал должен быть явно описан в том классе, объект которого инициирует его передачу. Ниже представлена диаграмма последовательности для случая ветвления со стереотипным значениями.

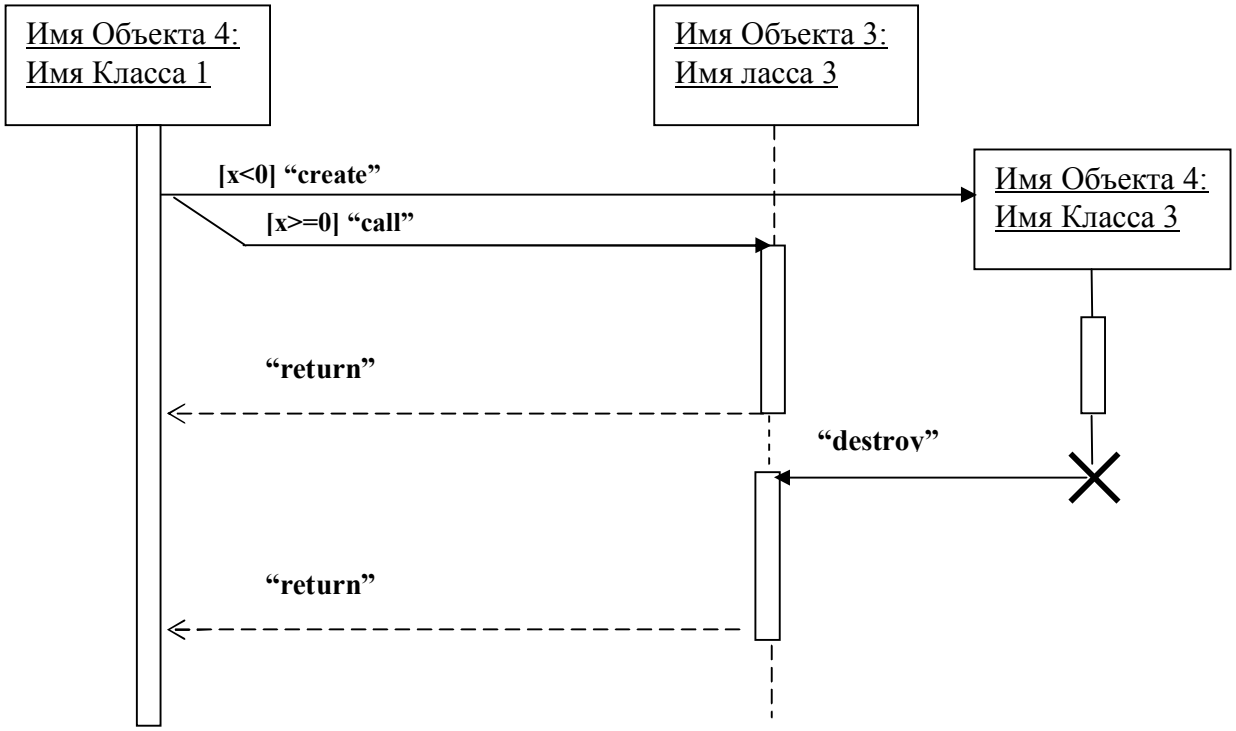


Рис.2.29

Кроме стереотипов, сообщения могут иметь собственное обозначение операции, вызов которой они инициируют у принимающего объекта. В этом случае рядом со стрелкой записывается имя операции с круглыми скобками, в которых могут указываться параметры или аргументы соответствующей операции. Если параметры отсутствуют, то скобки все равно должны присутствовать после имени операции. Примерами таких операций могут служить следующие: «выдать клиенту сумму (n)», «подать звуковой сигнал тревоги()».

Практическое занятие №6. Диаграмма деятельности (activity diagram)

При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической или логической реализации. Важно подчеркнуть то обстоятельство, что с увеличением сложности системы строгое соблюдение последовательности выполняемых операций приобретает все более важное значение.

Для моделирования процесса выполнения операций в языке UML используются так называемые *диаграммы деятельности*.

Состояние деятельности

Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой операции в предыдущем состоянии. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действий, а дугами – переходы от одного состояния действия к другому.

Именно диаграммы деятельности позволяют реализовать в языке UML особенности процедурного и синхронного управления, обусловленного завершением внутренних деятельностей и действий. Основным элементом диаграммы является деятельность. Интерпретация этого термина зависит от той точки зрения, с которой стоится данная диаграмма. На концептуальной диаграмме деятельность – это некоторая задача, которую необходимо выполнить вручную или автоматизированным способом. На диаграмме, построенной в аспекте спецификации или реализации, деятельность представляет собой некоторый метод над классом. Рекомендуется в качестве простого действия использовать глагол с пояснительными словами (рис.2.30)

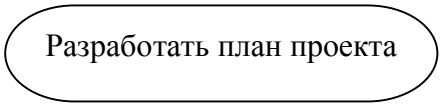


Рис.2.30. Деятельность

Начальное и конечное состояние

Начальное состояние представляет собой такое состояние, которое не содержит никаких внутренних действий. В этом состоянии находится объект по умолчанию в начальный момент времени. Оно служит для указания на диаграмме графической области, от которой начинается процесс изменения состояний. Графически начальное состояние в языке UML обозначается в виде закрашенного кружка (Рис.2).

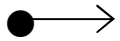


Рис.2.31. Начальное состояние

Конечное состояние представляет собой такое состояние, которое также не содержит никаких внутренних действий. В этом состоянии находится объект по умолчанию в конечный момент времени. Оно служит для указания на диаграмме графической области, от которой завершается процесс изменения состояний. Графически начальное состояние в языке UML обозначается в виде закрашенного кружка, помещенного в окружность (Рис.2.32).

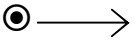


Рис.2.32. Конечное состояние

Каждая диаграмма деятельности должна иметь единственное начальное и единственное конечное состояния.

Переходы

Простой переход представляет собой отношение между двумя последовательными состояниями, которые указывают на факт смены одного состояния другим. Переход осуществляется при наступлении некоторого события: окончания выполнения деятельности. При построении диаграммы используются только нетригерные переходы, т.е. такие, которые срабатывают сразу после завершения деятельности или выполнения соответствующего действия и не связано с событием – триггером .Этот переход переводит деятельность в последующее состояние сразу, как только

закончится действие в предыдущем состоянии. На диаграмме такой переход изображается сплошной линией со стрелкой.

Если из состояния действий выходит единственный переход, то он может быть никак не помечен. Если же таких переходов несколько, то сработать может только один из них. Именно в этом случае для каждого из таких переходов должно быть записано сторожевое условие в прямых скобках. При этом для всех выходящих из некоторого состояния переходов должно выполняться требование истинности только одного из них. Подобный случай встречается тогда, когда последовательно выполняемая деятельность должна разделиться на альтернативные ветви в зависимости от значения некоторого промежуточного результата. Такая ситуация получила название ветвления, а для ее обозначения применяется специальный символ. Графически ветвление на диаграмме деятельности обозначается небольшим ромбом, внутри которого нет никакого текста. В этот ромб может входить только одна стрелка от того состояния, после выполнения которого поток управления должен быть продолжен по одной из взаимно исключающих ветвей. Выходящих стрелок может быть две или более, но для каждой из них явно указывается соответствующее сторожевое условие в форме булевского выражения.

Пример 2.5. Фрагмент известного алгоритма нахождения корней квадратного уравнения (Рис.2.33).

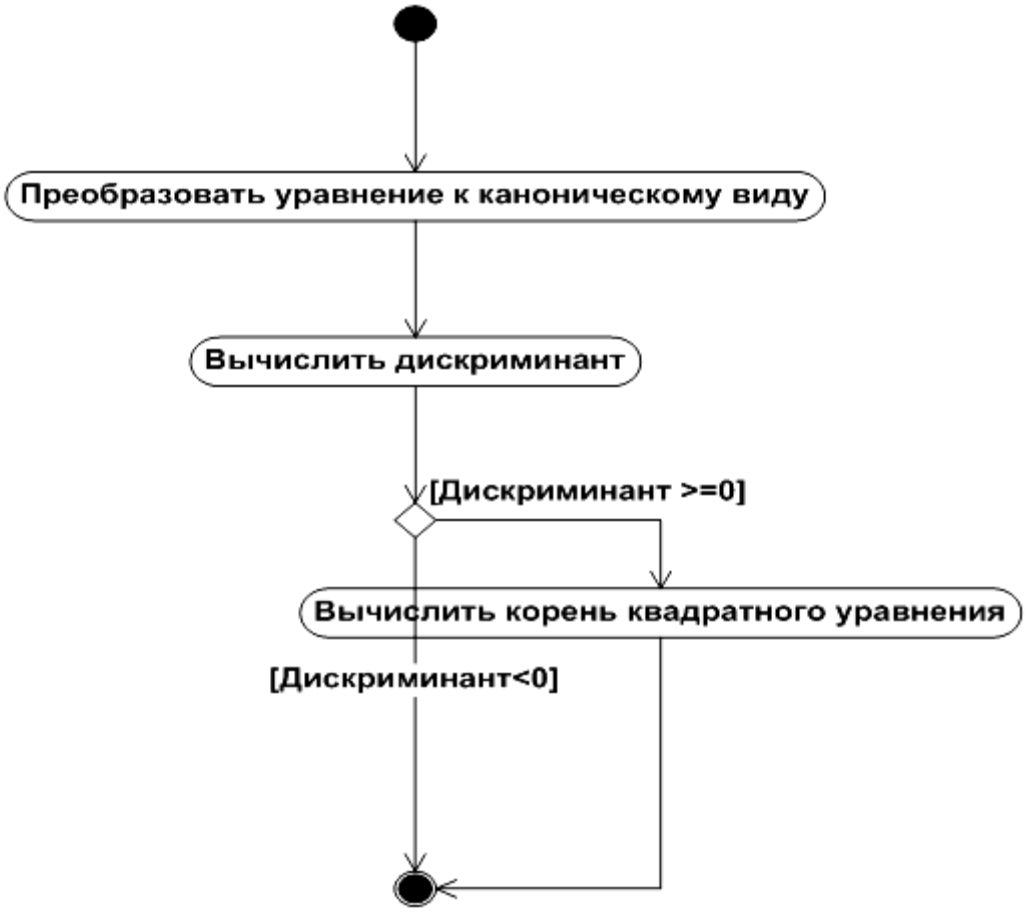


Рис.2.33. Вычисление корня квадратного уравнения

Один из наиболее значимых недостатков обычных блок-схем или структурных схем алгоритмов связан с проблемой изображения параллельных ветвей отдельных вычислений. Поскольку распараллеливание вычислений существенно повышает общее быстродействие программных систем, необходимы графические примитивы для представления параллельных процессов. В языке UML для этой цели используется специальный символ для разделения и слияния параллельных вычислений или потоков управления - прямая черточка (Рис.2.34).

а) разделение **б) слияние**

Рис.2.34

Пример 2.6. Для иллюстрации особенностей параллельных процессов выполнения действий рассмотрим ставший уже классическим пример с приготовлением напитка, представленного на рис.2.35.

На диаграмме представлены простые последовательности деятельностей (методов).для класса *Личность* (например, за деятельностью «Положить кофе в фильтр» следует деятельность «Вставить фильтр в автомат»), а также действия, содержащие условия (например, личность осуществляет деятельность «Поискать напиток», выбирая между кофе и колой).

Предположим, что мы отыскали кофе и идем вниз по «кофейному маршруту». Этот путь идет к линейке синхронизации, с которой связана активизация трех деятельностей: «Положить кофе в фильтр», «Добавить воду в емкость» и «достать чашки». Диаграмма указывает на то, что три деятельности могут выполняться параллельно. По существу это означает, что порядок их выполнения их деятельности не играет роли. Можно также выполнять эти деятельности, чередуя их друг с другом или одновременно. Если при описании поведения системы имеются параллельные деятельности, то их необходимо синхронизовать. Так, кофейный автомат не будет включаться до тех пор, пока в него не вставлен фильтр и не добавлена вода в емкость. Именно поэтому на диаграмме результаты этой деятельности сведены вместе к одной линейке синхронизации. Далее выполняется еще одна синхронизация: кофе должен быть готов и чашки должны стоять на месте перед тем, как мы сможем налить кофе.

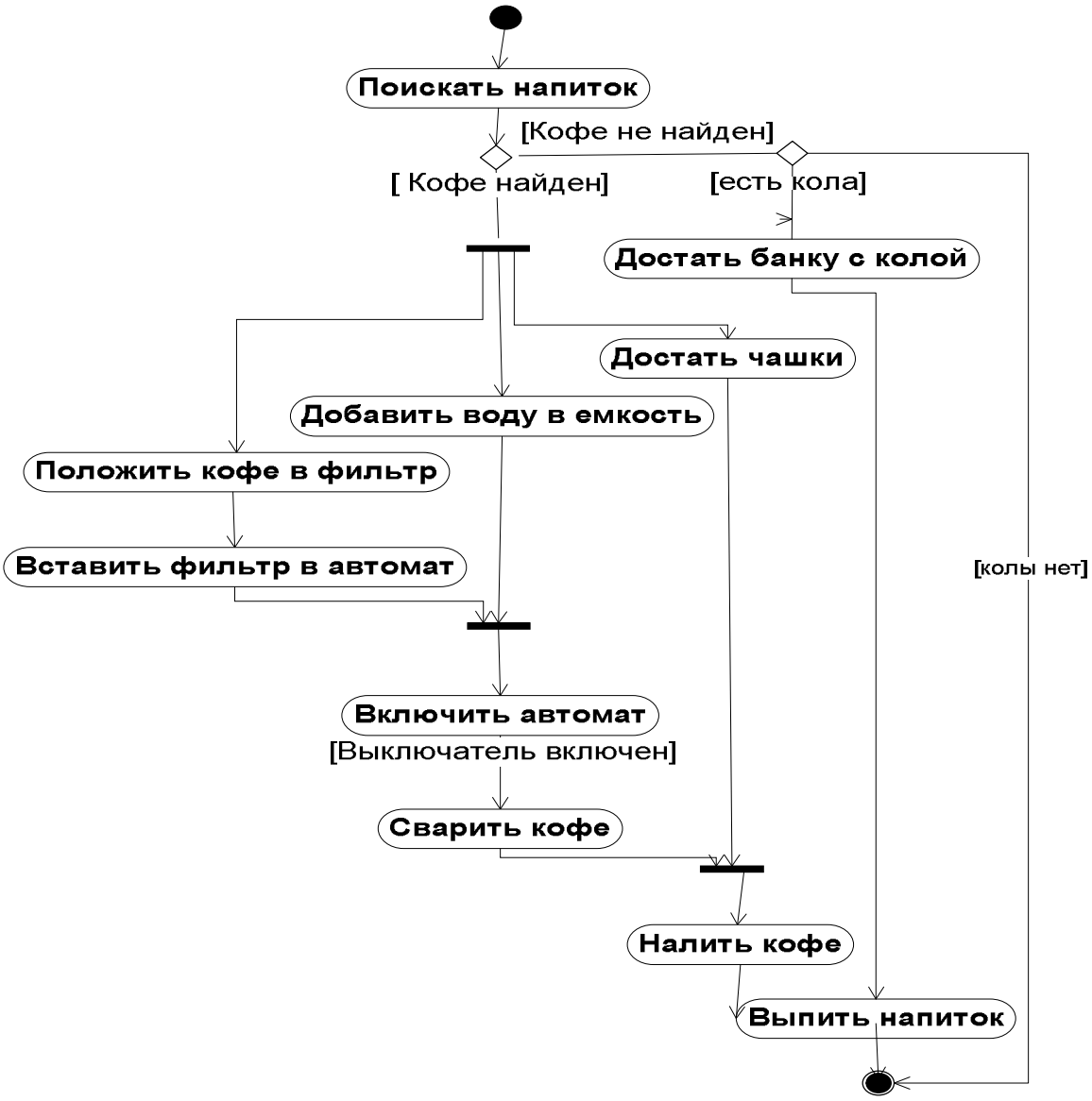
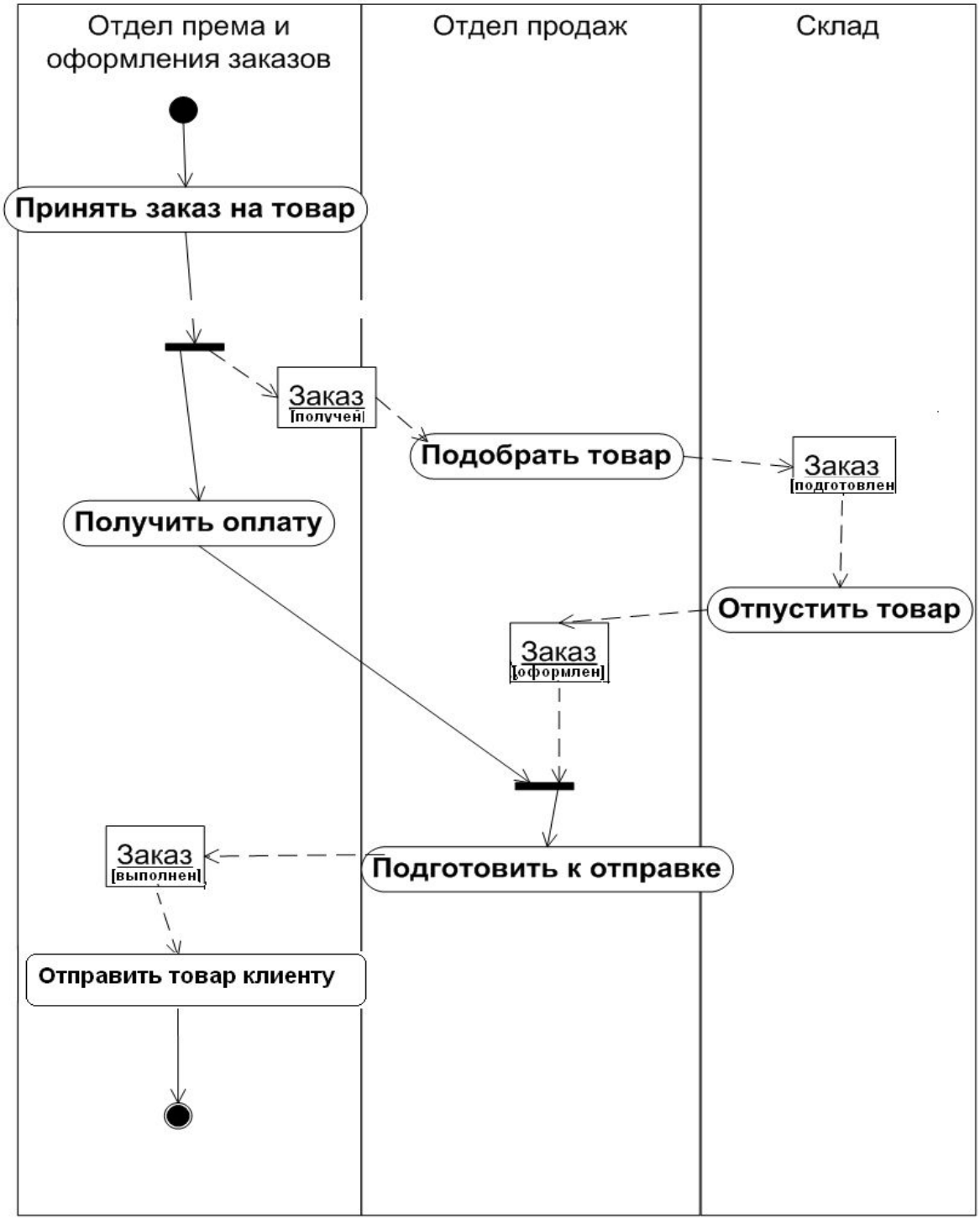


Рис.2.35

Диаграммы деятельности могут быть использованы не только для спецификации алгоритмов вычислений или потоков управления в программных системах. Не менее важная область их применения связана с моделирование бизнес-процессы. Действительно, деятельность любой компании также представляет собой не что иное, как совокупность отдельных действий, направленных на достижение требуемого результата. Однако применительно к бизнес-процессам желательно выполнение каждого действия ассоциировать с конкретным подразделением компании. В этом случае подразделение несет ответственность за реализацию отдельных действий, а сам бизнес-процесс представляется в виде переходов действий из одного подразделения к другому. Для моделирования этих особенностей в UML используются **дорожки** (swimlanes).

Пример 2.7. Ниже приведена диаграмма деятельности торговой компании, обслуживающей клиентов по телефону, с использованием дорожек.



Вопросы для самопроверки по теме 2:

1. В чем заключаются основные принципы объектно-ориентированного подхода?
2. В чем состоят достоинства и недостатки объектно-ориентированного подхода?

3. Каковы принципиальные отличия и что общего между структурным и объектно-ориентированным подходом?
4. В чем смысл варианта использования?
5. Назначение диаграмм вариантов использования, классов, деятельности, последовательности.
6. Назовите основные компоненты диаграмм вариантов использования.
7. Что такое действующее лицо? Какую роль он может играть по отношению к варианту использования?
8. Для чего используется диаграмма классов на стадии анализа?
9. Назовите основные компоненты диаграммы классов.
10. Что собой представляет ассоциация?

Примеры заданий для самостоятельной работы

1. Автоматизация работы деканата (учебные группы, кураторы, сессия);
2. Распределение учебной нагрузки кафедры и учет ее выполнения;
3. Составление учебных планов;
4. Тестирование студентов;
5. Научно-исследовательская работа преподавателей и студентов кафедры;
6. Учет материальных ценностей кафедры;
7. Составление расписания занятий;
8. Работа библиотеки;
9. Служба занятости;
10. Интернет-магазин.

Требуется провести моделирование предметной области, используя диаграмму IDEF0; построить DFD –и ERD- диаграммы для разрабатываемой программы (использовать BPWIN) Построить диаграммы вариантов использования, классов, деятельности и последовательности для описания требований к системе в MS Visio.

Список используемой литературы:

1. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. Пер.с англ.-М.:Издательство Бином, СПб, 1999.
2. Рамбо Дж., Якобсон А., Буч Г. UML: специальный справочник. СПб.: Питер, 2002. 656 с.
3. Вендров А. Практикум по проектированию программного обеспечения экономических информационных систем: Учеб. пособие. М.: Финансы и статистика, 2002. 192 с.
4. Маклаков С. BPwin и Erwin. CASE – средства разработки информационных систем. М.: Диалог-МИФИ, 2001. 304 с.
5. Леоненков А.В. Самоучитель UML. – СПб .: БХВ – Петербург, 2002. – 304с.
6. Федотова Д. CASE – технологии: Практикум. М.: Горячая линия – Телеком, 2003. 160 с.