Методологии проектирования информационных систем

- **Методология функционального моделирования работ SADT (Structured Analysis and Design Technique);**
- **Диаграммы потоков данных DFD (Data Flow Diagrams);**
- **Методология объектного проектирования на языке UML (UML-диаграммы);**

Методология функционального моделирования работ SADT

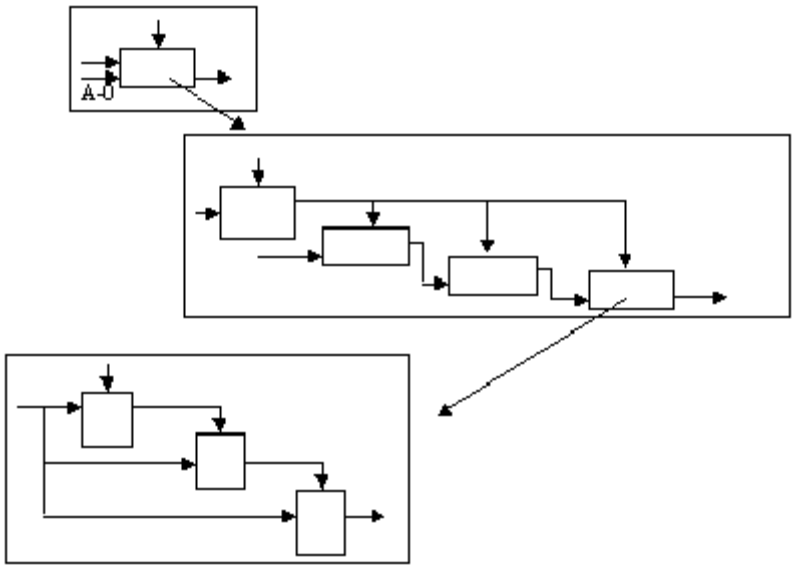
Методология SADT (Structured Analisys and Design Technique - технология структурного анализа и проектирования) разработана Дугласом Т. Россом в 1969-1973 годах. Технология изначально создавалась для проектирования систем более общего назначения по сравнению с другими структурными методами, выросшими из проектирования программного обеспечения. SADT - одна из самых известных и широко используемых методик проектирования. Новое название методики, принятое в качестве стандарта - IDEF0 (Icam DEFinition) - часть программы ICAM (Integrated Computer-Aided Manufacturing - интегрированная компьютеризация производства), проводимой по инициативе ВВС США.

Процесс моделирования в SADT включает сбор информации об исследуемой области, документирование полученной информации и представление ее в виде модели и уточнение модели. Кроме того, этот процесс подсказывает вполне определенный путь выполнения согласованной и достоверной структурной декомпозиции, что является ключевым моментом в квалифицированном анализе системы. SADT уникальна в своей способности обеспечить как графический язык, так и процесс создания непротиворечивой и полезной системы описаний.

В IDEF0 система представляется как совокупность взаимодействующих работ (или функций). Связи между работами определяют технологический процесс или структуру взаимосвязи внутри организации. Модель SADT представляет собой серию диаграмм, разбивающих сложный объект на составные части.

Каждый блок IDEF0-диаграммы может быть представлен несколькими блоками, соединенными интерфейсными дугами, на диаграмме следующего уровня. Эти блоки представляют подфункции (подмодули) исходной функции. Каждый из подмодулей может быть декомпозирован аналогичным образом. Число уровней не ограничивается, зато рекомендуется на одной диаграмме использовать не менее 3 и не более 6 блоков.

Работы (activity) обозначают поименованные процессы, функции или задачи, которые происходят в течение определенного времени и имеют распознаваемые результаты (изображается в виде прямоугольников). Каждая из работ на диаграмме может быть впоследствии декомпозирована.



Как отмечалось выше, в терминах IDEF0 система представляется в виде комбинации блоков и дуг. Блоки представляют функции системы, дуги представляют множество объектов (физические объекты, информация или действия, которые образуют связи между функциональными блоками). Взаимодействие работ с внешним миром и между собой описывается в виде стрелок или дуг. С дугами связываются метки на естественном языке, описывающие данные, которые они представляют. Дуги показывают, как функции системы связаны между собой, как они обмениваются данными и осуществляют управление друг другом. Дуги могут разветвляться и соединяться. Ветвление означает множественность (идентичные копии одного объекта) или расщепление (различные части одного объекта). Соединение означает объединение или слияние объектов. Пять типов стрелок допускаются в диаграммах:

Вход (Input) - материал или информация, которые используются работой для получения результата (стрелка, входящая в левую грань).

Управление (Control) - правила, стратегии, стандарты, которыми руководствуется работа (стрелка, входящая в верхнюю грань). В отличии от входной информации управление не подлежит изменению.

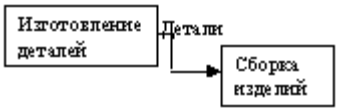
Выход (Output) - материал или информация, которые производятся работой (стрелка, исходящая из правой грани). Каждая работа должна иметь хотя бы одну стрелку выхода, т.к. работа без результата не имеет смысла и не должна моделироваться.

Механизм (Mechanism) - ресурсы, которые выполняют работу (персонал, станки, устройства - стрелка, входящая в нижнюю грань).

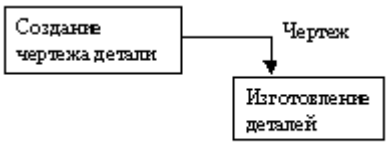
Вызов (Call) - стрелка, указывающая на другую модель работы (стрелка, исходящая из нижней грани).

Различают в IDEF0 пять типов связей работ.

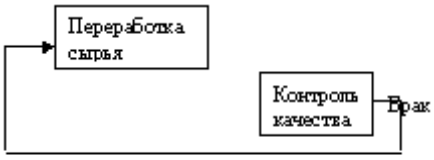
Связь по входу (input-output), когда выход вышестоящей работы направляется на вход следующей работы.



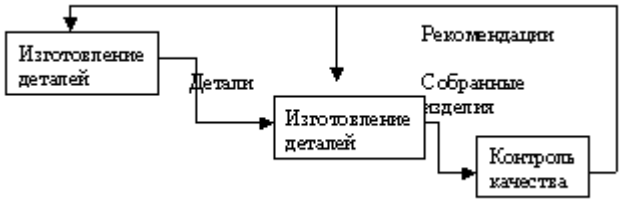
Связь по управлению (output-control), когда выход вышестоящей работы направляется на управление следующей работы. Связь показывает доминирование вышестоящей работы.



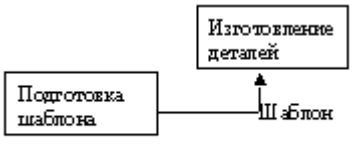
Обратная связь по входу (output-input feedback), когда выход нижестоящей работы направляется на вход вышестоящей. Используется для описания циклов.



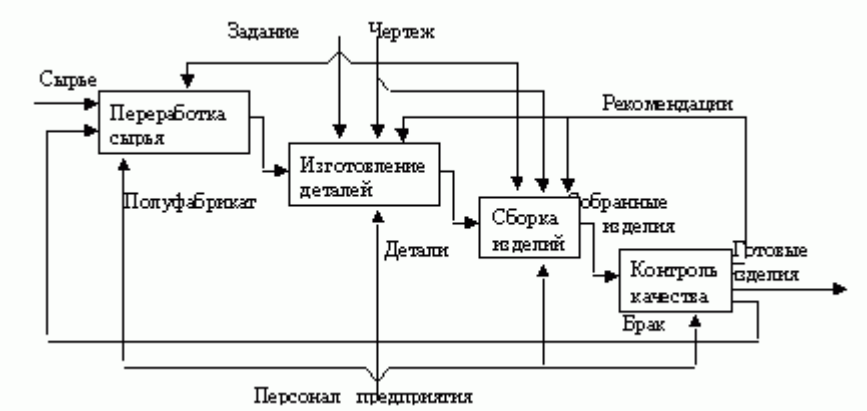
Обратная связь по управлению (output-control feedback), когда выход нижестоящей работы направляется на управление вышестоящей. Является показателем эффективности бизнес-процесса.



Связь выход-механизм (output-mechanism), когда выход одной работы направляется на механизм другой и показывает, что работа подготавливает ресурсы для проведения другой работы.



Из перечисленных блоков, как из отдельных кирпичиков, строится диаграмма. Пример SADT-диаграммы:



Диаграммы потоков данных DFD (Data Flow Diagrams)

Диаграммы потоков данных используются для описания движения документов и обработки информации как дополнение к IDEF0. В отличие от IDEF0, где система рассматривается как взаимосвязанные работы и стрелки представляют собой жесткие взаимосвязи, стрелки в DFD показывают лишь то, как объекты (включая данные) движутся от одной работы к другой. DFD отражает функциональные зависимости значений, вычисляемых в системе, включая входные значения, выходные значения и внутренние хранилища данных. DFD - это граф, на котором показано движение значений данных от их источников через преобразующие их процессы к их потребителям в других объектах.

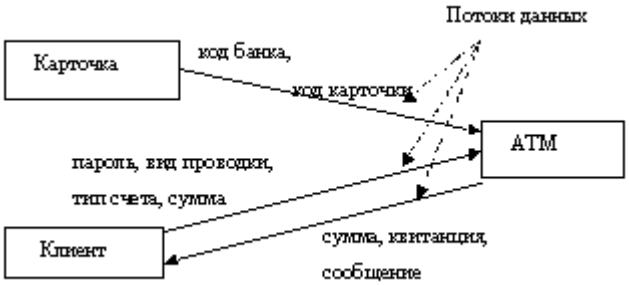
DFD содержит процессы, которые преобразуют данные, потоки данных, которые переносят данные, активные объекты, которые производят и потребляют данные, и хранилища данных, которые пассивно хранят данные.

Процессы. Процесс преобразует значения данных. Процессы самого нижнего уровня представляют собой функции без побочных эффектов (примерами таких функций являются вычисление суммы двух чисел, вычисление комиссионного сбора за выполнение проводки с помощью банковской карточки и т.п.). Весь граф потока данных тоже представляет собой процесс (высокого уровня). Процесс может иметь побочные эффекты, если он содержит нефункциональные компоненты, такие как хранилища данных или внешние объекты. На DFD процесс изображается в виде эллипса, внутри которого помещается имя процесса; каждый процесс имеет фиксированное число входных и выходных данных, изображаемых стрелками.

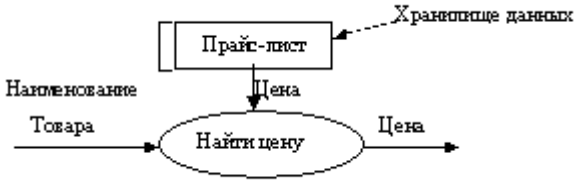
Потоки данных. Поток данных соединяет выход объекта (или процесса) с входом другого объекта (или процесса). Он представляет промежуточные данные

вычислений. Поток данных изображается в виде стрелки между производителем и потребителем данных, помеченной именами соответствующих данных. Дуги могут разветвляться или сливаться, что означает, соответственно, разделение потока данных на части, либо слияние объектов.

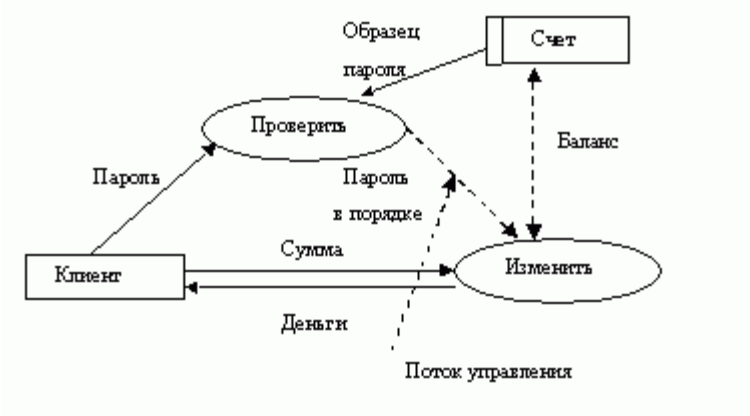
Активные объекты. Активным называется объект, который обеспечивает движение данных, поставляя или потребляя их. Активные объекты обычно бывают присоединены к входам и выходам DFD.



Хранилища данных. Хранилище данных - это пассивный объект в составе DFD, в котором данные сохраняются для последующего доступа. Хранилище данных допускает доступ к хранимым в нем данным в порядке, отличном от того, в котором они были туда помещены. Агрегатные хранилища данных, как, например, списки и таблицы, обеспечивают доступ к данным в порядке их поступления, либо по ключам.



Потоки управления. DFD показывает все пути вычисления значений, но не показывает в каком порядке значения вычисляются. Решения о порядке вычислений связаны с управлением программой, которое отражается в динамической модели. Эти решения, вырабатываемые специальными функциями, или предикатами, определяют, будет ли выполнен тот или иной процесс, но при этом не передают процессу никаких данных, так что их включение в функциональную модель необязательно. Тем не менее, иногда бывает полезно включать указанные предикаты в функциональную модель, чтобы в ней были отражены условия выполнения соответствующего процесса. Функция, принимающая решение о запуске процесса, будучи включенной в DFD, порождает в DFD поток управления и изображается пунктирной стрелкой.



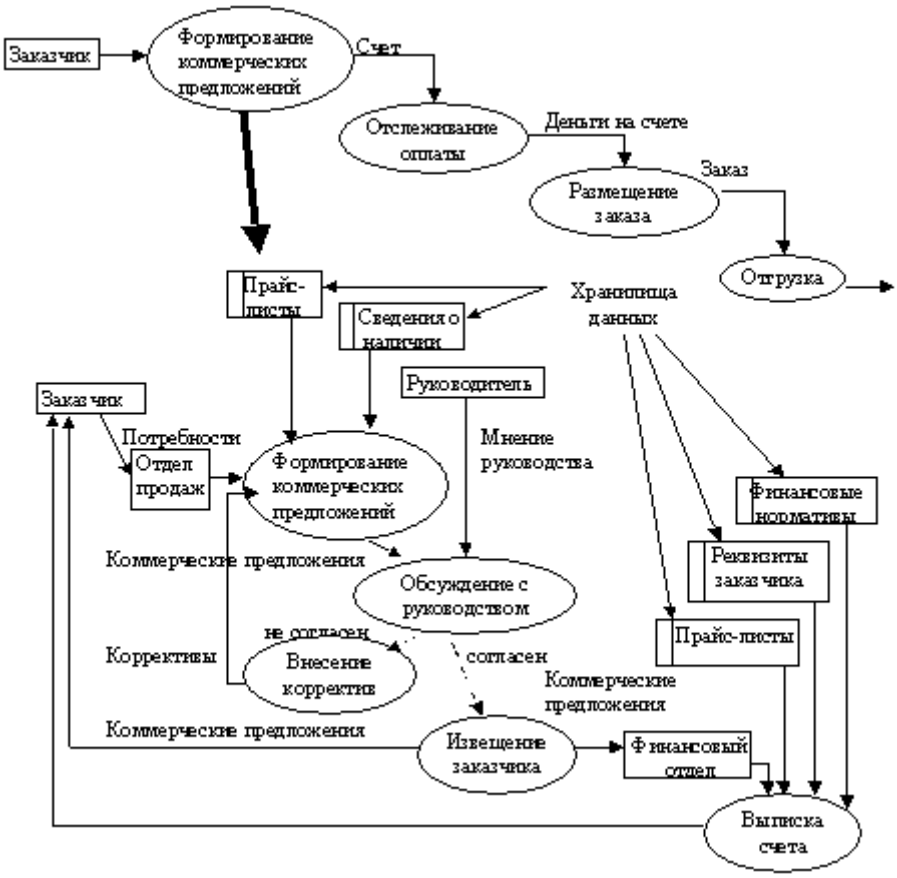
Первым шагом при построении иерархии DFD является построение контекстных диаграмм. Обычно при проектировании относительно простых информационных систем строится единственная контекстная диаграмма со звездообразной топологией, в центре которой находится так называемый главный процесс, соединенный с приемниками и источниками информации, посредством которых с системой взаимодействуют пользователи и другие внешние системы.

Если же для сложной системы ограничиться единственной контекстной диаграммой, то она будет содержать слишком большое количество источников и приемников информации, которые трудно расположить на листе бумаги нормального формата, и, кроме того, главный единственный процесс не раскрывает структуры распределенной системы.

Для сложных информационных систем строится иерархия контекстных диаграмм. При этом контекстная диаграмма верхнего уровня содержит не главный единственный процесс, а набор подсистем, соединенных потоками данных. Контекстные диаграммы следующего уровня детализируют контекст и структуру подсистем.

При построении иерархии DFD переходить к детализации процессов следует только после определения содержания всех потоков и накопителей данных, которое описывается при помощи структур данных. Структуры данных конструируются из элементов данных и могут содержать альтернативы, условные вхождения и итерации. Условное вхождение означает, что данный компонент может отсутствовать в структуре. Альтернатива означает, что в структуру может входить один из перечисленных элементов. Итерация означает вхождение любого числа элементов в указанном диапазоне. Для каждого элемента данных может указываться его тип (непрерывные или дискретные данные). Для непрерывных данных может указываться единица измерения (кг, см и т.п.), диапазон значений, точность представления и форма физического кодирования. Для дискретных данных может указываться таблица допустимых значений.

Ниже приведена диаграмма потоков данных верхнего уровня с ее последующим уточнением:



Методология объектного проектирования на языке UML (UML-диаграммы)

Унифицированный язык моделирования (Unified Modeling Language - UML) - это язык для специфицирования, визуализации, конструирования и документирования на основе объектно-ориентированный подхода разные виды систем: программных, аппаратных, программно-аппаратных, смешанных, явно включающие деятельность людей и т. д.

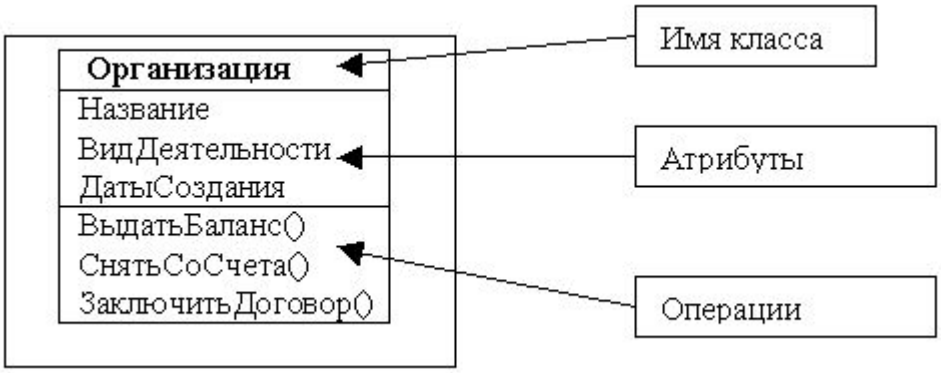
Помимо прочего, язык UML применяется для проектирования реляционных БД. Для этого используется небольшая часть языка (диаграммы классов), да и то не в полном объеме. С точки зрения проектирования реляционных БД модельные возможности не слишком отличаются от возможностей ER-диаграмм

Диаграммой классов в терминологии UML называется диаграмма, на которой показан набор классов (и некоторых других сущностей), не имеющих явного отношения к проектированию БД), а также связей между этими классами. Ограничения могут неформально задаваться на естественном языке или формулироваться на языке объектных ограничений OCL (Object Constraints Language).

Классом называется именованное описание совокупности объектов с общими атрибутами, операциями, связями и семантикой. Графически класс изображается в виде прямоугольника. Имя (текстовая строка), служит для идентификации класса.

Атрибутом класса называется именованное свойство класса, описывающее множество значений, которые могут принимать экземпляры этого свойства. Класс может иметь любое число атрибутов (в частности, не иметь ни одного атрибута).

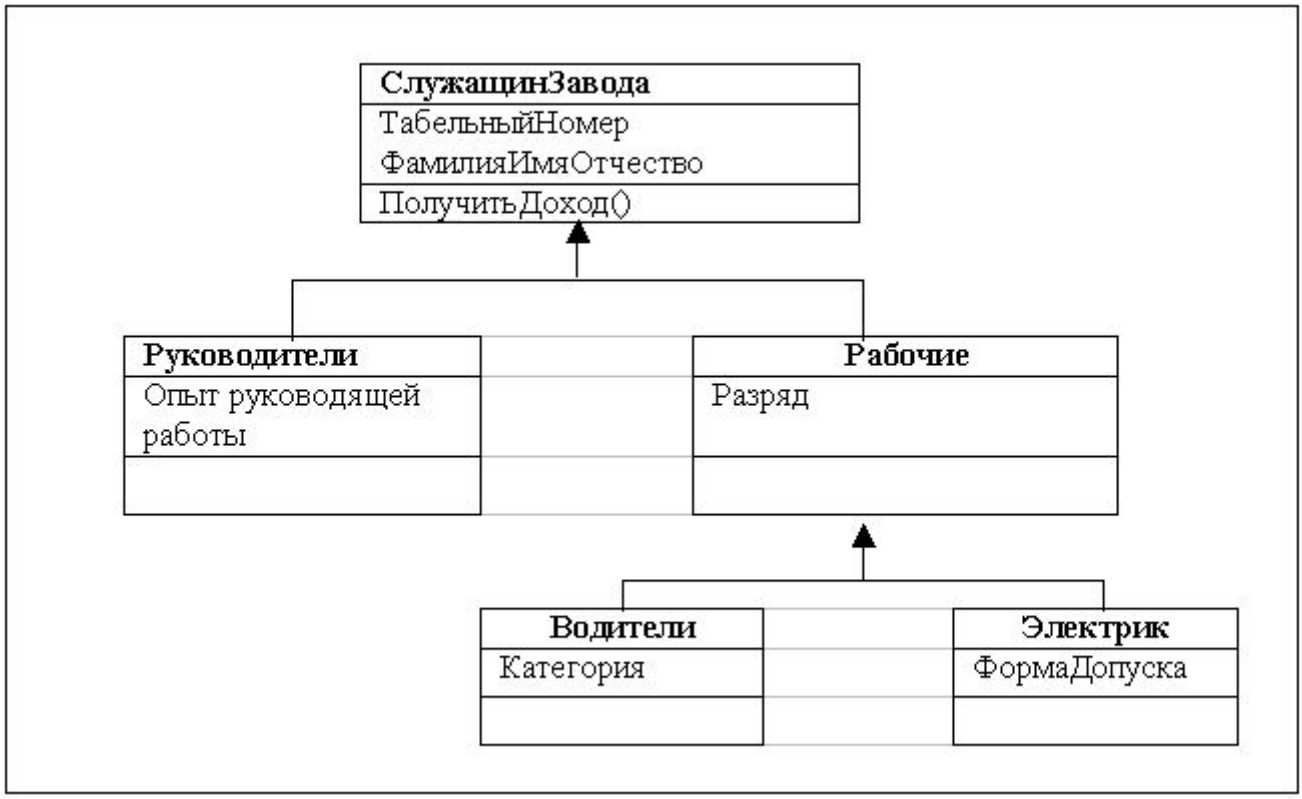
Операцией класса называется именованная услуга, которую можно запросить у любого объекта этого класса. Операция - это абстракция того, что можно делать с объектом. Класс может содержать любое число операций (в частности, не содержать ни одной операции). Набор операций класса является общим для всех объектов данного класса.



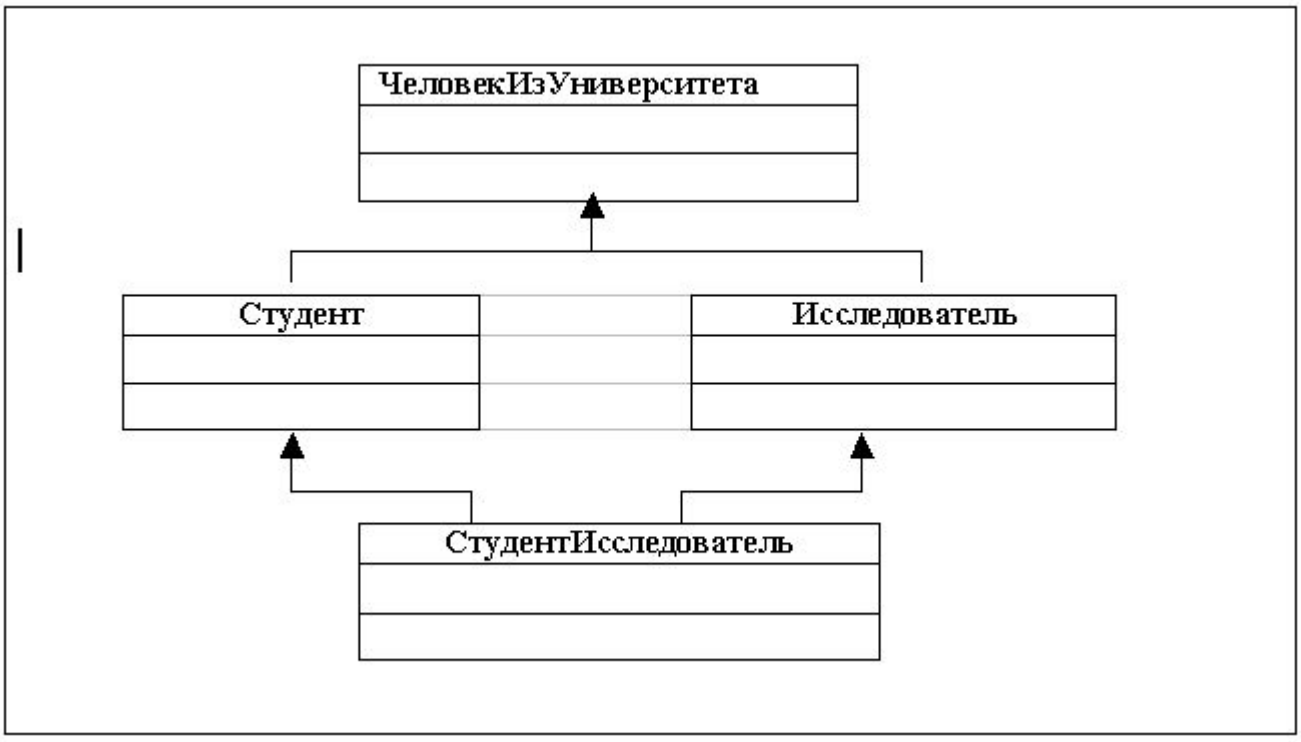
В диаграмме классов могут участвовать связи трех разных категорий: зависимость (dependency), обобщение (generalization) и ассоциация (association).

Зависимостью называют связь по применению, когда изменение в спецификации одного класса может повлиять на поведение другого класса, использующего первый класс. Если интерфейс второго класса изменяется, это влияет на поведение объектов первого класса. Зависимость показывается прерывистой линией со стрелкой, направленной к классу, от которого имеется зависимость.

Связью-обобщением называется связь между общей сущностью, называемой *суперклассом*, или родителем, и более специализированной разновидностью этой сущности, называемой *подклассом*, или потомком. Обобщения иногда называют связями *"is a"*, имея в виду, что класс-потомок является частным случаем класса-предка. Класс-потомок наследует все атрибуты и операции класса-предка, но в нем могут быть определены дополнительные атрибуты и операции.



Одиночное наследование, когда у каждого подкласса имеется только один суперкласс) является достаточным в большинстве случаев применения связи-обобщения. Однако в UML допускается и множественное наследование, когда один подкласс определяется на основе нескольких суперклассов.



На этой диаграмме классы **Студент** и **Исследователь** порождены из одного суперкласса **ЧеловекИзУниверситета**. К классу **Студент** относятся те объекты класса **ЧеловекИзУниверситета**, которые соответствуют студентам, а к классу**Исследователь** - объекты класса **ЧеловекИзУниверситета**, соответствующие исследователям. Часто случается, многие студенты уже в студенческие годы начинают участвовать в исследовательских проектах, так что могут существовать такие два объекта классов **Студент** и **Исследователь**, которым соответствует один объект класса **ЧеловекИзУниверситета**. Таким образом среди объектов класса **Студент** могут быть исследователи, а некоторые исследователи могут быть студентами. Тогда можно определить класс **СтудентИсследователь** путем множественного наследования от суперклассов **Студент** и **Исследователь**. Объект класса **СтудентИсследователь** обладает всеми свойствами и операциями классов **Студент** и **Исследователь** и может быть использован везде, где могут применяться объекты этих классов. Так что полиморфизм по включению продолжает работать. Однако это влечет за собой ряд проблем. Например, при образовании подклассов **Студент** и **Исследователь** в них обоих может быть определен атрибут с именем **Ip_адресКомпьютера**. Для объектов класса **Студент** значениями этого атрибута будут ip-адрес компьютера терминального класса, а для объектов класса **Исследователь** - ip-адрес компьютера исследовательской лаборатории. При этом для объекта класса **СтудентИсследователь** могут быть существенны оба одноименных атрибута (у студента-исследователя могут иметься и ip-адрес компьютера терминального класса и ip-адрес компьютера исследовательской лаборатории). На практике применяется одно из следующих решений:

- запретить образование подкласса **СтудентИсследователь**, пока в одном из суперклассов не будет произведено переименование атрибута **Ip_адресКомпьютера**;
- наследовать это свойство только от одного из суперклассов, так что, например, значением атрибута **Ip_адресКомпьютера** у объектов класса **СтудентИсследователь** всегда будут ip-адресом компьютера исследовательской лаборатории;
- унаследовать в подклассе оба свойства, но автоматически переименовать оба атрибута, чтобы прояснить их смысл; назвать их, например, **Ip_адресКомпьютераСтудента** и **Ip_адресКомпьютераИсследователя**.

Каждое из указанных решений имеет недостатки, поэтому при использовании UML для проектирования реляционных БД нужно очень осторожно использовать наследование классов вообще и стараться избегать множественного наследования.

Ассоциацией называется структурная связь, показывающая, что объекты одного класса некоторым образом связаны с объектами другого или того же самого класса. Допускается, чтобы оба конца ассоциации относились к одному классу. В ассоциации могут связываться два класса, и тогда она называется бинарной. Допускается создание ассоциаций, связывающих сразу n классов (они называются

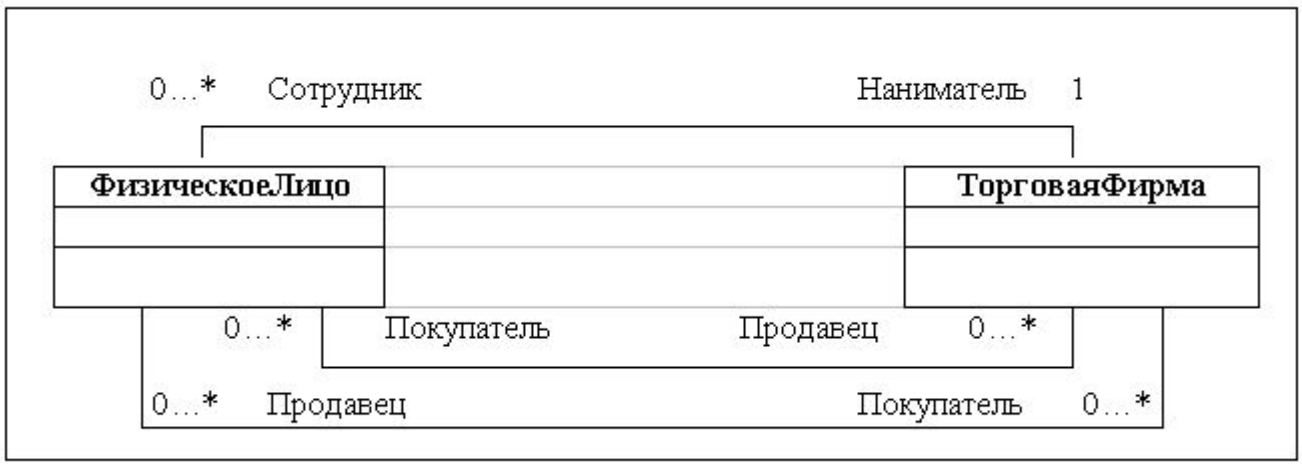
n-арными ассоциациями).1) Графически ассоциация изображается в виде линии, соединяющей класс сам с собой или с другими классами.

С понятием ассоциации связаны четыре важных дополнительных понятия: имя, роль, кратность и агрегация. Ассоциации может быть присвоено имя, характеризующее природу связи. Другим способом именования ассоциации является указание роли каждого класса, участвующего в этой ассоциации. Роль класса, как и имя конца связи в ER-модели, задается именем, помещаемым под линией ассоциации ближе к данному классу.

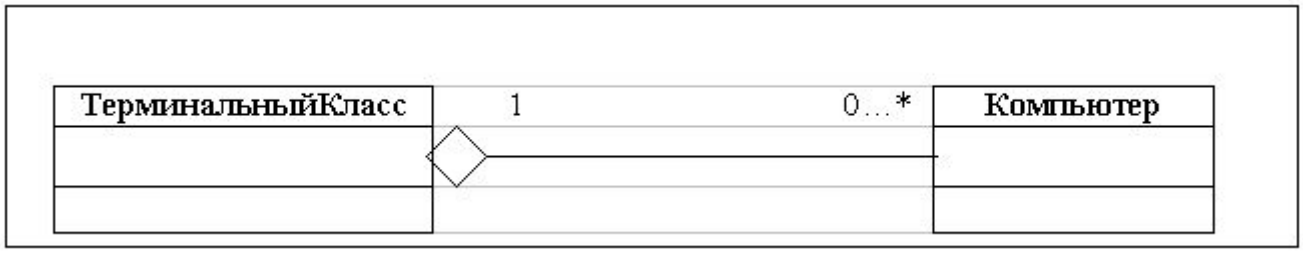
В общем случае, для ассоциации могут задаваться и ее собственное имя, и имена ролей классов. Связано это с тем, что класс может играть одну и ту же роль в разных ассоциациях, так что в общем случае пара имен ролей классов не идентифицирует ассоциацию. В простых случаях, когда между двумя классами определяется только одна ассоциация, можно вообще не связывать с ней дополнительные имена.

Кратностью (multiplicity) роли ассоциации называется характеристика, указывающая, сколько объектов класса с данной ролью может или должно участвовать в каждом экземпляре ассоциации.

Наиболее распространенным способом задания кратности роли ассоциации является указание конкретного числа или диапазона. Указание "1" говорит о том, что каждый объект класса с данной ролью должен участвовать в некотором экземпляре данной ассоциации, причем в каждом экземпляре ассоциации может участвовать только один объект класса с данной ролью. Указание диапазона "0..1" говорит о том, что не все объекты класса с данной ролью обязаны участвовать в каком-либо экземпляре данной ассоциации, но в каждом экземпляре ассоциации может участвовать только один объект. Аналогично, указание диапазона "1..*" говорит о том, что все объекты класса с данной ролью должны участвовать в некотором экземпляре данной ассоциации, и в каждом экземпляре ассоциации должен участвовать хотя бы один объект (верхняя граница не задана). В более сложных случаях определения кратности можно использовать списки диапазонов.

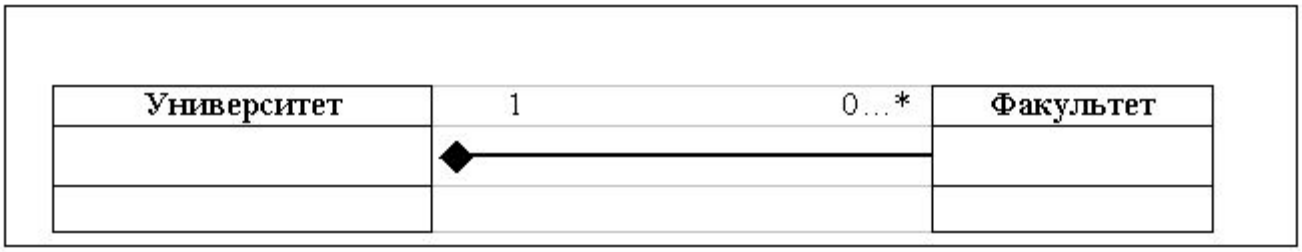


Обычная ассоциация между двумя классами характеризует связь между равноправными сущностями: оба класса находятся на одном концептуальном уровне. Но иногда в диаграмме классов требуется отразить тот факт, что ассоциация между двумя классами имеет специальный вид "часть-целое". В этом случае класс "целое" имеет более высокий концептуальный уровень, чем класс "часть". Ассоциация такого рода называется *агрегатной*.



В каждом терминальном классе должны быть установлены компьютеры. Поэтому класс **Компьютер** является "частью" класса **ТерминальныйКласс**. Роль класса **Компьютер** необязательна, поскольку принципиально могут существовать терминальные классы без компьютеров. При этом, хотя терминальные классы, не оснащенные компьютерами, не выполняют своей функции, объекты классов **ТерминальныйКласс** и **Компьютер** существуют независимо.

Бывают случаи, когда связь "части" и "целого" настолько сильна, что уничтожение "целого" приводит к уничтожению всех его "частей". Агрегатные ассоциации, обладающие таким свойством, называются *композитными*, или просто композициями.



Любой факультет является частью одного университета, и ликвидация университета приводит к ликвидации всех существующих в нем факультетов, хотя во время существования университета отдельные факультеты могут ликвидироваться и создаваться.

В диаграммах классов могут указываться ограничения целостности, которые должны поддерживаться в проектируемой БД. В UML допускаются два способа определения ограничений: на естественном языке и на языке OCL (Object Constraints Language).

С точки зрения определения ограничений целостности баз данных более важны средства определения инвариантов классов.

Под *инвариантом* класса в OCL понимается условие, которому должны удовлетворять все объекты данного класса. Если говорить более точно, инвариант класса - это логическое выражение, вычисление которого должно давать истину при создании любого объекта данного класса и сохранять истинное значение в течение всего времени существования этого объекта. При определении инварианта требуется указать имя класса и выражение, определяющее инвариант указанного класса. Синтаксически это выглядит следующим образом:

```
context <class_name> inv:
<OCL-выражение>
```

Ниже приведены примеры инвариантов для задания ограничений на языке OCL.

1. Выразить на языке OCL ограничение, в соответствии с которым в отделах с номерами больше 5 должны работать сотрудники старше 30 лет

```
context Отдел inv:
self.номер  5 or
self.служащий  select (возраст 30)  size () = 0
```

2. Определить ограничение, в соответствии с которым у каждого отдела должен быть менеджер и любой отдел должен быть основан не раньше соответствующей компании

```
context Отдел inv:
self.служащий  exists (должность = “manager”) and
self.компания.годОснования  self.годОснования
```

3. Условие четвертого инварианта ограничивает максимально возможное количество сотрудников компании числом 1000

```
context Компания inv:
self.отдел  collect (служащие)  size ( ) < 1000
```