

7. Универсальный язык моделирования

(Unified Modeling Language UML)

UML - это универсальный язык моделирования [5], разработанный в фирме Rational Software при участии других партнеров. Разработка методологии началась в октябре 1994 года под руководством сотрудников Rational Software Гради Буча (Grady Booch) и Джима Рамбаха (Jim Rumbaugh). Через год в октябре 1995 года появился проект языка UML 0.8. Далее к главным авторам присоединился Ивар Якобсон (Ivar Jacobson). В 1996 году появился проект UML 0.9. Далее очень многие организации проявили интерес к данной методологии. Был создан консорциум UML партнеров, в который вошли такие известные фирмы как DEC, HP, IBM, Oracle, Microsoft и другие. После создания такой группы появились спецификации UML 1.0 и UML 1.1. Ниже рассматривается незначительно сокращенный вариант графического синтаксиса языка UML 1.1.

7.1 Пакеты, как средство работы с большими проектами

Пакеты представляют собой универсальное средство для группирования элементов моделей. Пакеты могут вкладываться друг в друга и могут содержать пакеты или элементы моделей. Проект в целом может рассматриваться как один пакет верхнего уровня, в который вложены все остальные составляющие части проекта. Пакет может иметь два графических обозначения: полное и сокращенное. Сокращенное обозначение пакета предназначено для обозначения пакета, входящего в состав другого пакета:

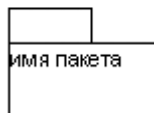


Рис. 7.1. Обозначение пакета.

Полное обозначение пакета предназначено для представления этого пакета как такового:

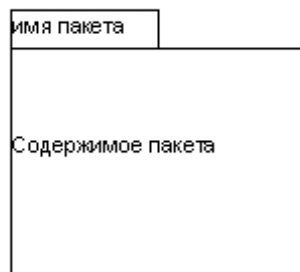


Рис. 7.2. Полное обозначение пакета.

В рамке “содержимое пакета” находится графическое изображение, представляющее другие пакеты или модели. Отношения между пакетами отображаются линиями и обозначают отношения между элементами пакета. Если разные элементы двух пакетов имеют разные отношения друг с другом, то нет четких рекомендаций какие отношения

показывать между самими пакетами. Таким образом наличие отношения X между двумя пакетами говорит только о том, что в пакетах присутствуют элементы связанные между собой этим отношением.

Пакеты обеспечивают более высокий уровень абстракции по сравнению с классами. Типичный большой проект содержит несколько иерархий наследования для классов. Возьмем в качестве примера многооконный графический редактор диаграмм. Набор пакетов для этой задачи можно было бы представить следующим образом:

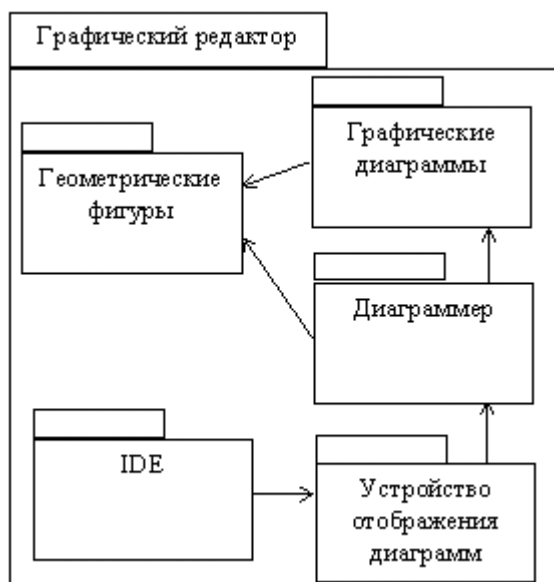


Рис. 7.3. Набор пакетов проекта "графический редактор"

Пакет "геометрические фигуры" содержит определение иерархии наследования для классов всех геометрических фигур. Эти классы должны быть независимы от контекста, в который они включаются (например, диаграмма), а так же от устройства, на котором они отображаются. Базовый класс этого пакета может выглядеть так как показано на рис. 7.5. (ниже)

Пакет "графические диаграммы" содержит определение всех классов для диаграмм. Например, все диаграммы, рассматриваемые здесь могут быть представлены в этом пакете. Диаграмма не должна зависеть от способа создания (например, она может быть создана не в диаграммере, а автоматически). Тем более диаграмма не должна зависеть от устройства отображения.

Пакет "диаграммер" отвечает за отображение диаграмм в некотором контексте и за ввод этих диаграмм с помощью некоторых виртуальных команд пользователя, независимых от контекста отображения.

Пакет "устройство отображения диаграмм" содержит классы, описывающие однооконный интерфейс конкретного диаграммера и способ создания виртуальных команд пользователя с помощью конкретных устройств ввода (например, мыши и клавиатуры).

Пакет "IDE" содержит средства, необходимые для включения одного диаграммера в интегрированную среду разработки с меню, карточками диалога, настройками параметров среды, многооконным интерфейсом.

Достоинства использования пакетов:

- декомпозиция задачи упрощает понимание каждой части в отдельности и задачи в целом,
- каждый пакет можно поручить отдельному разработчику за счет относительной независимости пакетов.

Все проектирование объектно-ориентированной системы должно начинаться именно с проектирования пакетов. Они позволят избежать лишних ошибок, проект станет более управляемым и обозримым. Можно сформулировать следующие рекомендации по составлению пакетов и классов в них:

- каждый пакет должен быть максимально независимым от других пакетов, все связи должны быть локализованы и сведены к минимуму, идеальный случай - связь через один класс, из которого используется один метод поведения;
- структура пакетов должна отражать структуру предметной области, это обеспечит возможность проектирования классов в четком соответствии с предметной областью, и в дальнейшем позволит легко модифицировать систему для других задач в рамках данной предметной области;
- каждый пакет должен содержать классы, однотипные с точки зрения предметной области;
- желательно, что бы иерархия наследования начиналась с одного базового объекта в каждом пакете, допустимо два, три, но не более;
- базовый объект в каждом пакете - это следующий принципиальный момент после составления самих пакетов; он должен отражать наиболее базовые свойства той части предметной области к которой относится пакет.

7.2 Диаграммы классов и объектов

Диаграмма классов представляет набор:

- классов,
- типов данных,
- интерфейсов и
- отношений между ними.

Диаграмма объектов представляет набор экземпляров классов и типов данных, наиболее типичным ее использованием является представление примеров структур данных. Поскольку диаграмма классов может включать в свой состав объекты, то отдельного вида диаграммы объектов не существует, это фактически подмножество диаграммы классов.

7.2.1 Классы

Графическое представление класса - это прямоугольник, который может быть разделен на три части:



Рис. 7.4. Пример изображения класса.

Верхняя часть прямоугольника содержит имя класса, средняя - атрибуты, нижняя - методы поведения (операции). Атрибуты или методы при изображении класса могут быть скрыты для того, что бы подчеркнуть другие аспекты диаграммы классов, например, состав классов и отношения между ними. В этом случае изображение класса принимает простейший вид прямоугольника с именем класса.

Каждый атрибут представляется в следующем виде:

видимость имя: тип = начальное значение

Перед именем может следовать знак обозначающий видимость атрибута для других классов:

+ общедоступный (public) атрибут

защищенный (protected) атрибут

-закрытый (private) атрибут

Каждый метод представляется в следующем виде:

видимость имя(список параметров): тип возвращаемого значения

Описатель видимости имеет те же значения, что и для атрибута.

Список параметров представляет собой перечень описателей параметров, разделенных запятой. Описатель каждого параметра имеет вид:

вид имя: тип = значение по умолчанию

вид параметра может быть следующим:

inвходной параметр

out выходной параметр

inout входной и выходной параметр

Текст реализации операции может быть сопоставлен в качестве примечания для каждого метода.

Пример изображения класса представлен на рис. 7.5.

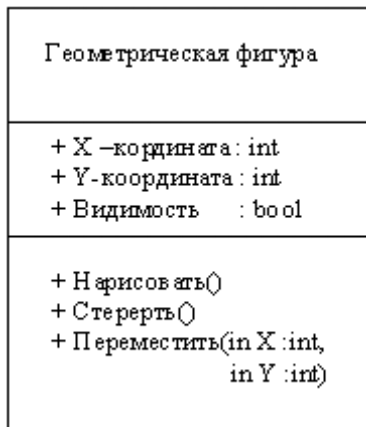


Рис. 7.5. Пример изображения класса "Геометрическая фигура".

7.2.2 Интерфейсы

Интерфейсы предназначены для спецификации внешнего вида операций для классов.

7.2.3 Отношения между классами

1. Двухместная связь (Binary Association)

это отношение между двумя классами, включая возможность рефлексивного отношения класса с самим собой. Изображается сплошной линией, соединяющей два класса. Линия может иметь один или несколько соединенных сегментов. Конец линии соединенный с классом называется ролью. Для связи может быть задано имя , которое представляет отношение в целом, оно не должно располагаться вблизи краев линии для того, что бы не возникало конфликтов с именованием ролей.

В следующем примере указано отношение между двумя классами “линия” и “точка”. Отношение имеет имя “Содержать”, роли для каждого из классов называются соответственно “Включает” и “Входит в”:

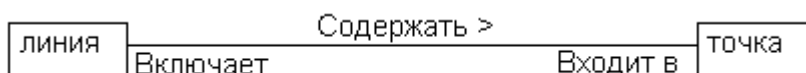


Рис. 7.7. Пример отношения.

Для каждой роли может быть определена дополнительная информация следующего вида:

- множественность,
- сортировка,
- квалификатор,
- имя роли,
- спецификация интерфейса,
- изменяемость,
- видимость.

множественность, определяет количество экземпляров классов, участвующих в отношении. Множественность определяется в виде одного или нескольких диапазонов:

нижняя граница .. верхняя граница

в качестве нижней или верхней границы может быть задан символ *, обозначающий произвольное количество. Может использоваться перечисление диапазонов через запятую. Пример допустимых вариантов:

1

1..*

*

0

сортировка, определяет тип упорядочения элементов для множественности больше чем 1. Возможные значения: упорядочено, неупорядочено.

символ агрегации, это ромб, находящийся между линией и классом. Символ агрегации обозначает, что класс, вблизи которого изображен данный знак является накопителем для класса, находящегося на другом конце связи. Если ромб заливается черным цветом, то это обозначает усиленный вариант агрегации, называемый композицией. Символ агрегации может находиться только на одном конце связи.

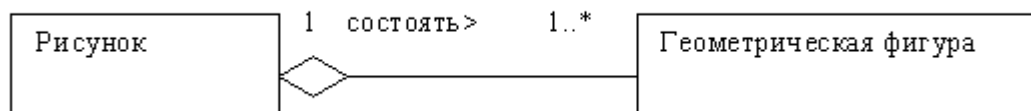


Рис. 7.8. Пример агрегации

квалификатор, это один или несколько атрибутов, значения которых обеспечивают идентификацию экземпляров класса по данной связи. Множественность, указанная для роли, при наличии квалификатора указывает на количество экземпляров класса, выбираемых одним значением квалификатора. Наиболее часто используются следующие значения для множественности: 0..1 - уникальный экземпляр может быть выбран, но не обязательно будет выбран; 1 - каждое значение квалификатора однозначно выбирает экземпляр класса; * - одному значению квалификатора соответствует множество экземпляров. Графически квалификатор обозначается как прямоугольник, в котором указаны имена атрибутов, являющиеся квалификатором. Прямоугольник располагается между линией связи и классом. Ниже представлен пример в котором классы линия и точка связаны отношением “входить в”. Квалификатором является координата точки. Множественность со стороны класса “точка” равна 0..1, что обозначает, что каждая точка линии должна иметь уникальные координаты. Множественность со стороны класса “линия” равна *, что обозначает, что разные линии могут иметь общие точки.

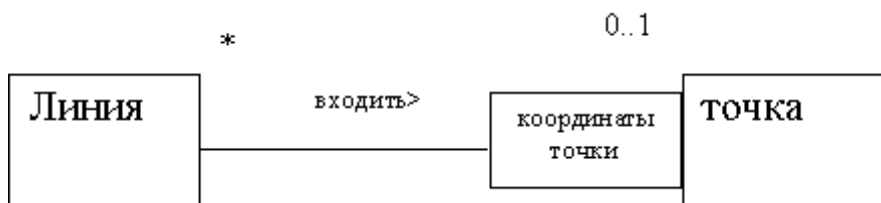


Рис. 7.9. Пример использования квалификатора

имя роли, произвольный идентификатор конкретизирующий роль связи для одного из классов, указывается на линии вблизи класса.

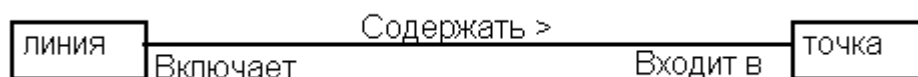


Рис. 7.10. Пример обозначения роли.

7.2.3.1 Класс, описывающий связь

(Association class). Связь между классами также может обладать свойствами, присущими некоему классу. Для определения таких свойств для связи может быть задан ассоциированный с ней класс. Связь и ассоциированный класс представляют одно целое.

Ассоциированный класс изображается обычным образом в виде прямоугольника и связывается с отношением, для которого он задан, штриховой линией. Имена связи (отношения) и описывающего его класса должны совпадать.

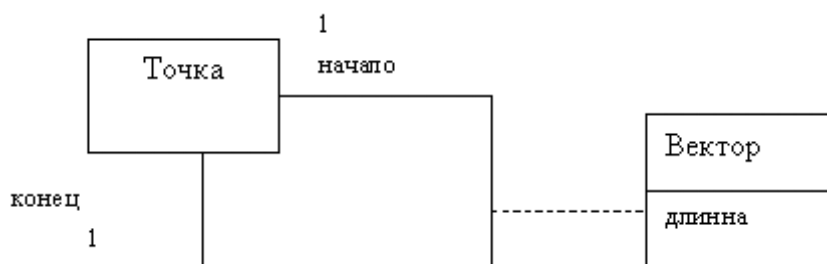


Рис. 7.11. Пример класса, описывающего связь

7.2.3.2 N - местная связь

(N-ry Association) Это связь между 3 и более классами. Эта связь графически обозначается ромбом, к которому подходят линии, соединяющие ромб с классами. Имя отношения указывается внутри или вблизи ромба. Класс, описывающий связь присоединяется к ромбу с помощью штрих - пунктирной линии. Пример такой связи представлен на рисунке:

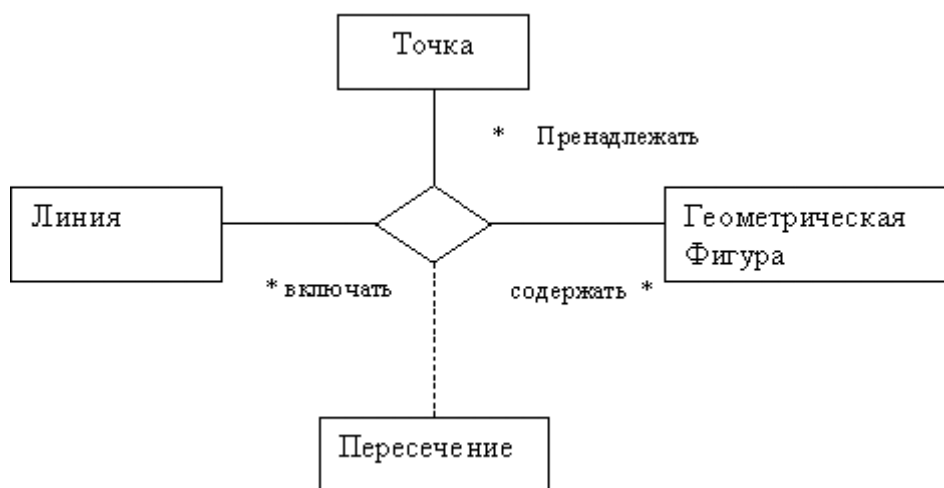


Рис. 7.12. Пример многоместной связи.

Для подобного вида связей не может быть применена агрегация.

7.2.3.3 Композиция, Сборка.

Композиция (Composition) описывает включение классов друг в друга, один из вариантов представления композиции определен при описании двуместной связи (с помощью закрашенных ромбов на конце линии). Альтернативная графическая форма для представления композиции - это размещение изображения включаемых классов внутри изображения включающего класса.

Композиция является усиленной формой агрегации и может быть рассмотрена только на основе двуместной связи. В случае композиции объект владелец и его составные части не могут существовать раздельно. Компоненты, участвующие в композиции, не могут одновременно иметь нескольких владельцев. Все компоненты изменяются вместе с владельцем. Графически, композиция может быть представлена в виде "дерева", корнем которого является составной объект, а листьями – его компоненты.

Отличия сборки от композиции может быть продемонстрировано на следующем примере:

1. Сборка. Элементы входят в хэш-таблицу и образуют ее. Они могут существовать друг без друга. Таблица имеет собственную структуру и логику поведения. Она может не содержать ни одного элемента. В то же время элементы могут существовать изолированно от таблицы. Таким образом элементы собираются внутри таблицы и наполняют ее.
2. Композиция. Здание образуется набором внутренних помещений. Но они не могут существовать друг без друга и вне друг друга.

Соответственно такие отличия оказывают влияние на удаление объектов. Для композиции должно быть обеспечено удаление всех составных частей после удаления объемлющего объекта. Каждая часть также должна быть удалена в случае ее исключения из составного объекта.

7.2.3.4 Обобщение

Обобщение (Generalization) - это отношение между общим и уточняющим элементом. Обобщение показывается как сплошная линия с треугольником на конце. Треугольник располагается у общего элемента.

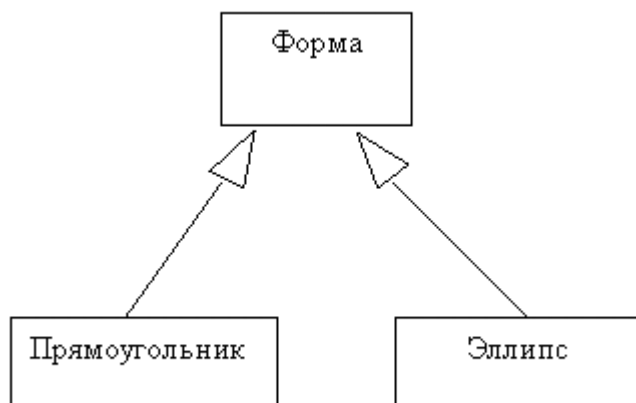


Рис. 7.13. Пример обобщения

7.2.3.5 Зависимость (Dependency)

Зависимость (Dependency) - это отношение обозначает любую зависимость между любыми элементами модели. Зависимость изображается штриховой линией с направлением. Элемент для которого линия является выходной зависим от элемента на другом конце линии. На линии может быть указано имя отношения.

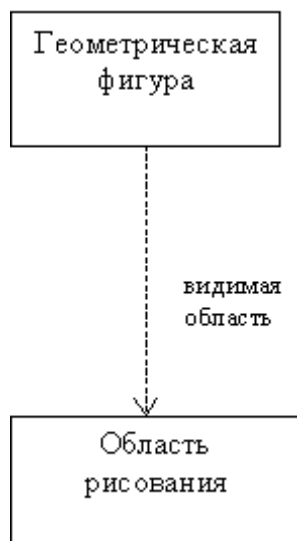
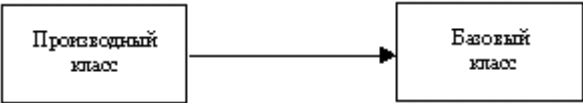
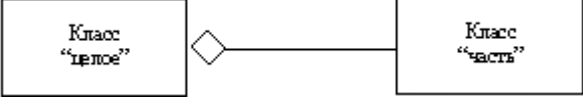
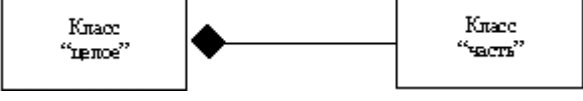
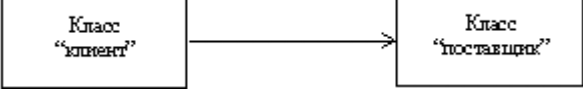
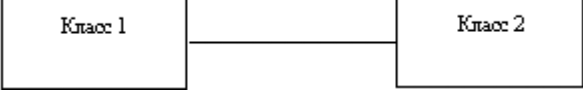
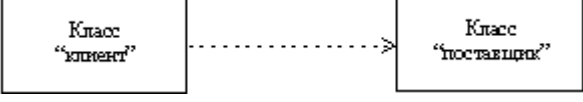
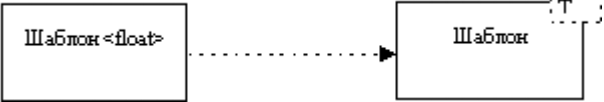


Рис. 7.14. Пример зависимости

Подобная связь никак не регламентирует вид отношений между объектами а указывает лишь на то, что зависимый класс использует какие-то особенности реализации иного класса .

Все отношения между классами сведены в следующую таблицу:

Таблица 7.1. Отношения между классами

Отношение	Изображение Класс Отношение Класс
1. Наследование (Inheritance)	
2. Сборка (Aggregation)	
3. Композиция (Composition)	
4. Однонаправленная ассоциация (Uni-directional Association)	
5. Двухнаправленная ассоциация (Bi-directional Association)	
6. Зависимость (Dependency)	
7. Шаблон (Template Instantiation)	

7.2.4 Пример диаграммы классов

Пример диаграммы классов представлен на рисунке ниже:

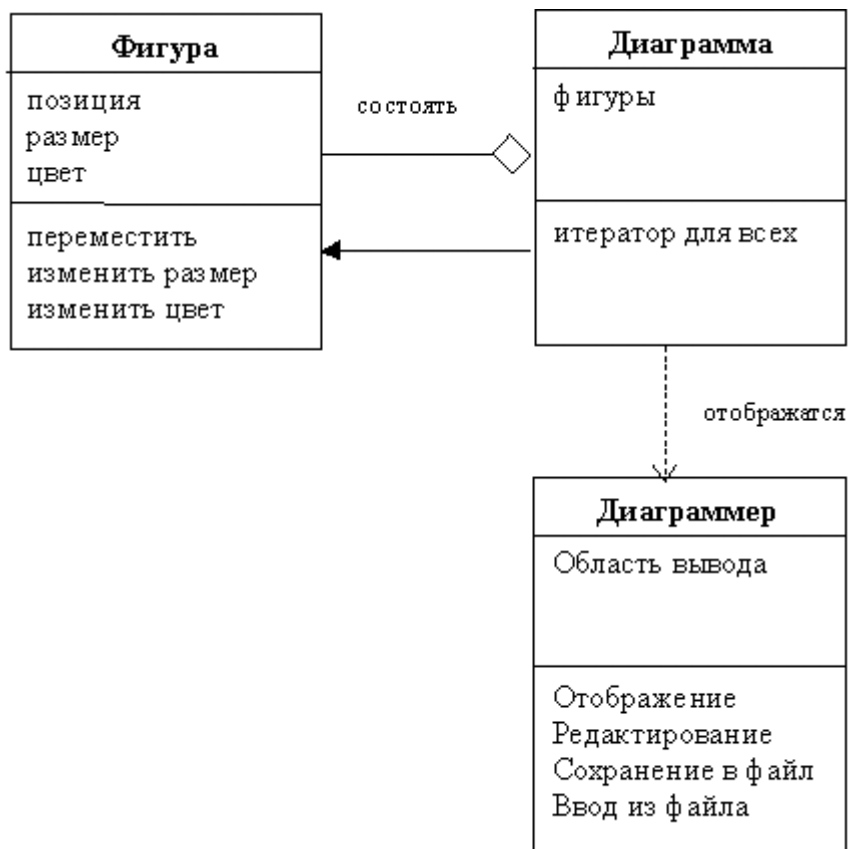


Рис. 7.15. Диаграмма классов для предметной области графического диаграммера.

7.3 Диаграммы использования

Диаграмма использования (use case diagram) предназначена для отображения внешнего функционирования проектируемой системы и ее взаимодействия с внешним миром пользователями. Эти диаграммы впервые были рассмотрены в книге [4] Иваром Якобсоном (Ivar Jacobson). Основой подхода являются так называемые блоки использования (use case), которые представляют собой некоторый набор функций системы, объединяемых в единое целое с точки зрения пользователя. Один блок использования не обязательно представляет собой одну часть системы или даже единую группу функций. Он представляет собой именно понимание пользователем поведения системы.

Диаграммы использования являются широко применяемыми в различных технологиях проектирования. Их главная задача - спецификация требований к системе на начальных этапах проектирования, когда решаются наиболее общие задачи предназначения разрабатываемой системы. Широкое распространение диаграмм использования привело в частности и к тому, что они имеют очень широкое и иногда различное толкование. Так в книге [4] автор указывает, что он обнаружил 18 различных определений, которые отличаются друг от друга по очень многим параметрам.

Диаграмма состоит из следующих элементов:

- внешние пользователи (actors), это такие воздействия, которые передают или получают информацию для системы, это могут быть физические объекты разной природы от людей и механизмов до программных систем, один

физический объект может описываться несколькими пользователями, если он взаимодействует с разными функциями,

- блоки использования (use case), это такие группы функций системы, которые объединяются в единое целое для внешнего пользователя,
- связи между блоками использования и связи между блоками использования и внешними пользователям.

Выделены следующие виды связей:

- взаимодействие; (только между пользователем и блоком использования)
- расширение - данный вид отношения от блока А к блоку В обозначает, что исполнение блока В может дополняется исполнением некоторых функций блока А;
- использование - данный вид отношения от блока А к блоку В обозначает, что исполнение А также включает исполнение блока В;

Графически блоки использования обозначаются эллипсами с указанием имени внутри эллипса или рядом с ним. Внешние пользователи графически обозначаются как прямоугольники с табулятором “Пользователь” или в виде схематичной фигурки человека, с именем под ней.

Графическое обозначение для связей следующее:

- взаимодействие - сплошная линия,
- расширение - линия со стрелкой от блока, предоставляющего расширение к базовому блоку, помеченная словом “extends”
- использование - линия со стрелкой от использующего блока к используемому блоку, помеченная словом “uses”.

Все блоки использования объединяются на рисунке в единый прямоугольник, очерчивающий рамки проектируемой системы.

Примером диаграммы может являться диаграмма, представленная на рис. , демонстрирующая структуру системы учета клиентов Internet для Internet Service Provider (ISP).

Внешними пользователями (actor) являются:

“клиент” - физическое или юридическое лицо, заказывающее услуги по подключению к сети Internet,

“оператор” - сотрудник ISP, осуществляющий работу с клиентом и системой учета,

“бухгалтер”- бухгалтер ISP,

“системный администратор” - сотрудник ISP, обеспечивающий системное администрирование системы учета клиентов.

Основными задачами “клиента” являются

1. заключение договора о предоставлении услуг сети Internet,

2. выбор перечня услуг для работы в сети (например, создание почтового ящика, создание web-сервера, регистрация и обслуживание домена),
3. оплата предоставленных услуг или авансовый платеж,
4. получение необходимых документов (договор, счет, счет-фактура, акт о выполненных работах),
5. получение статистических отчетов (состояние счета, сеансы работы, начисления за услуги),
6. подключение и работа в режиме on-line.

Основными задачами “бухгалтера” являются

1. получение отчетов о поступлении платежей

Основными задачами “системного администратора” являются

1. удаление сведений о клиенте
2. создание нового вида услуг

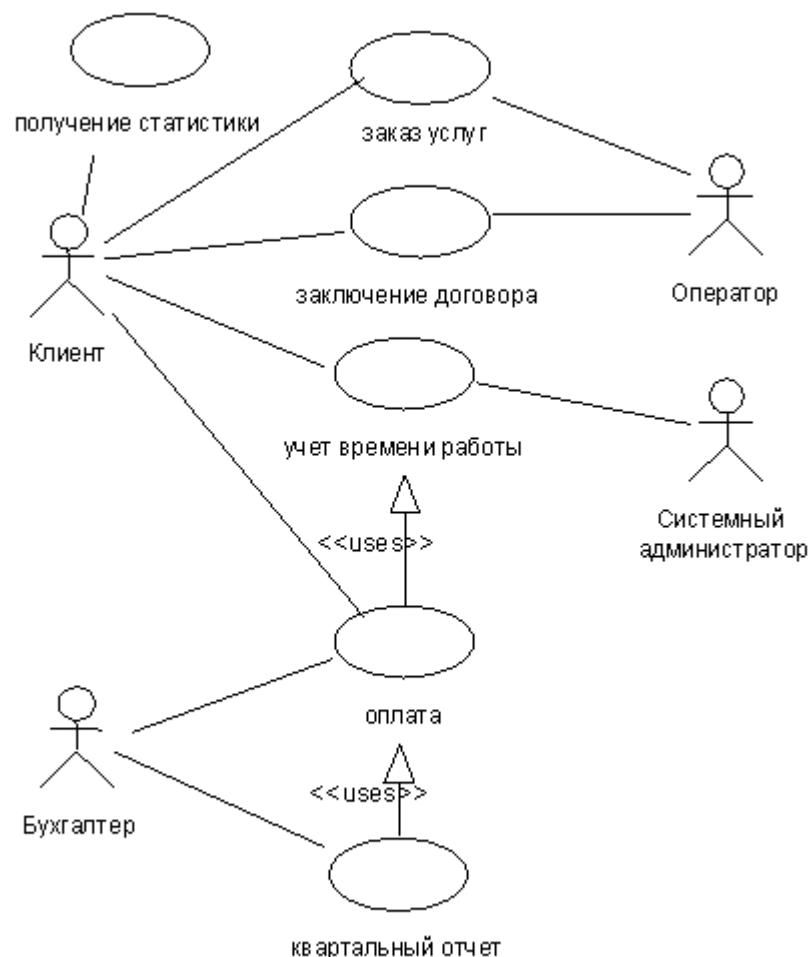


Рис. 7.16. Пример диаграммы использования.

Диаграммы использования являются весьма популярными и включаются в различные методики проектирования благодаря своим следующим достоинствам:

формулировка требований, ориентированная на пользователя, т.е. система заведомо проектируется так как ее будет видеть пользователь,

простота модели, ее ориентированность на неподготовленного пользователя,

возможность описания больших проектов за счет декомпозиции на крупные блоки использования, которых в любой системе не так много, как структурных единиц самой системы.

В соответствии с материалами [8] предлагается следующая последовательность формулирования требований к системе с помощью диаграмм использования:

- определить перечень главных пользователей (внешних входов-выходов) системы, это целесообразно сделать методом мозгового штурма с участием большого количества людей, начиная от руководителя разработки и кончая программистами;
- определить методику участия внешних пользователей в проекте (методы привлечения их к составлению требований);
- определить главные цели пользователей (что они хотят сделать с помощью системы) и задачи, которые нужно решить для достижения этих целей;
- составить диаграмму использования, в которой каждой задаче каждого пользователя сопоставить блок использования; соединить блоки использования и внешних пользователей, ответственных за данную задачу;
- проверить каждый блок использования какой-либо количественной метрикой, например, можно использовать оценку времени выполнения задачи или объема ресурсов, необходимого для ее решения; на этой стадии полезно составить прототип программного средства, для оценки реализуемости проекта (но не в коем случае не интерфейса пользователя);

Все перечисленные шаги не должны приводить к слишком сложной модели и занимать слишком много времени. Далее можно перечислить рекомендации по составлению диаграмм использования:

диаграммы должны быть простыми и понятными любому пользователю;

автор проекта и идеи системы должен участвовать в спецификации требований;

каждый участник проекта должен понимать конечную цель и предназначение разрабатываемой системы;

блоки использования должны быть основаны на их целях (а не, например, структуре);

не использовать слишком много отношений “extends” и “uses” - они усложняют понимание диаграммы;

блоки использования не должны отражать вопросов пользовательского интерфейса;

необходимо вовремя остановить внешних пользователей, т.к. они могут хотеть от системы слишком много и она будет нереализуема.

7.4 Диаграммы последовательностей

Диаграмма последовательностей (Sequence Diagram) предназначена для отображения временных зависимостей, возникающих в процессе общения между объектами. Диаграмма строится как график и имеет два измерения. По вертикали откладывается время, которое может быть схематичным или может иметь реальный масштаб. По горизонтали отображаются объекты. Она состоит из следующих элементов:

объект, обозначается прямоугольником с записанным в нем именем объекта;

линия жизни объекта, штрих - пунктирная линия, выходящая из объекта и расположенная вдоль оси времени, обозначает время жизни объекта,

активация, тонкий вертикальный прямоугольник, расположенный вдоль оси времени объекта, обозначающий период активной жизни объекта, либо выходит из объекта,

вызов метода поведения объекта (сообщение), обозначается стрелкой между активациями объектов с именем действия, направление стрелки задает направление передачи данных,

текстовые метки (отметки времени, описание действий и т.п.)

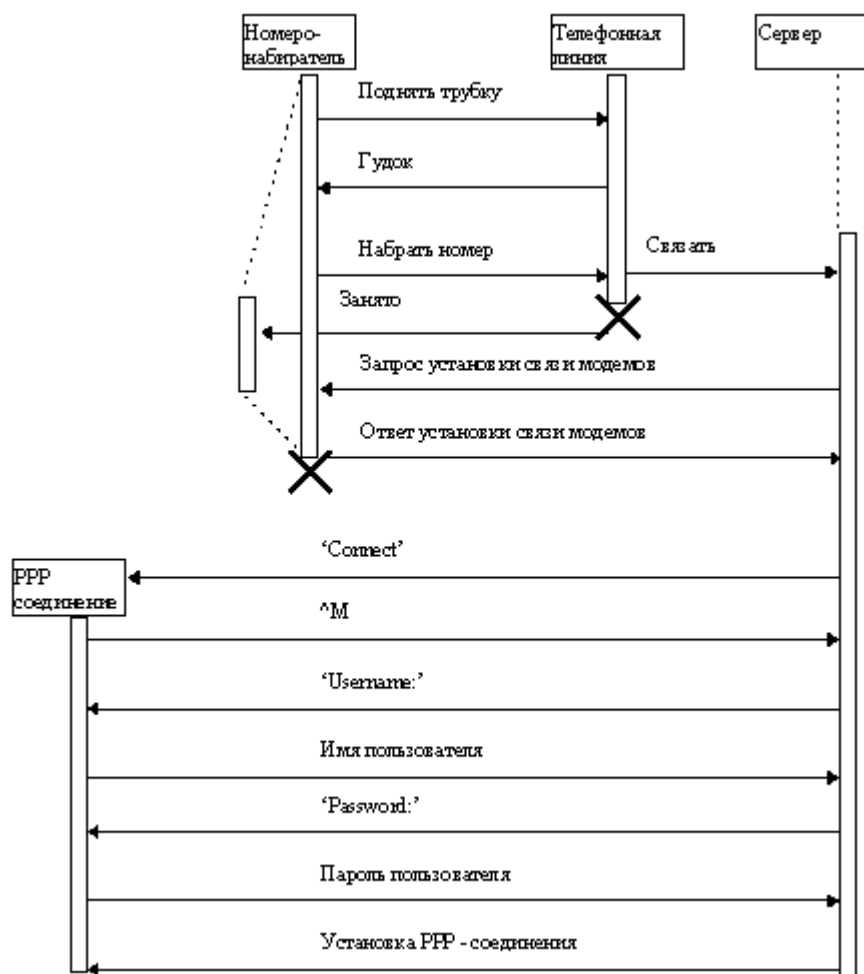


Рис. 7.17. Пример диаграммы последовательностей.

На данном рисунке рассмотрена диаграмма для установки PPP - соединения через модем между сервером и клиентом. Такая задача выполняется, например, при

подключении персонального компьютера к Internet через модем. На рисунке изображены четыре объекта: “PPP - соединение”, “Номеронабиратель”, “Телефонная линия”, “Сервер”. В рамках данной задачи объекты “Номеронабиратель” и “Телефонная линия” начинают свою жизнь сразу с активации, тогда как другие объекты имеют неактивную линию жизни (см. штрих - пунктир). Активация объектов “PPP - соединение” и “Сервер” начинается только с получения соответствующего сообщения. Объект “PPP - соединение” создается только после получения соответствующего сообщения. В этом случае стрелка с сообщением соединяется не с активацией, а непосредственно с объектом. Черный крест в конце активации обозначает, что объект перестает существовать в рамках данной задачи.

Линии жизни объектов могут разветвляться для обозначение альтернативных вариантов поведения. На альтернативных линиях жизни могут располагаться различные активации. Альтернативная линия жизни показана для объекта “Номеронабиратель”, она начинается с получения сигнала “занято” от телефонной линии.

Линии, обозначающие передачу сообщений (вызовы методов) помечаются именем выполняемого действия или передаваемым сообщением. Могут быть отображены фактические параметры передаваемые в вызове, или результат возвращаемый после вызова.

7.5 Диаграммы сотрудничества

Диаграмма сотрудничества (Collaboration diagram) предназначена для описания методов взаимодействия между объектами. Для пояснения смысла и назначения диаграммы необходимо ввести такое понятие как “сотрудничество”.

Сотрудничество представляет собой набор объектов, которые взаимодействуют друг с другом (вызывают методы поведения друг друга) для достижения конкретной группы целей. В данном случае в процессе проектирования необходимо сосредоточиться только на тех объектах и их методах поведения, которые необходимы для достижения определенной цели или единой группы целей. Сотрудничество может быть сопоставлено операции, блоку использования (из диаграммы использования) или классу для описания его статической структуры. Важно то, что сотрудничество не предназначено для описания поведения объектов, для этого могут быть использованы диаграммы последовательностей или диаграммы действий. Поведение некоторой части проекта может быть рассмотрена в двух аспектах: статическая структура того, что определяет поведение и динамические аспекты реализации этого поведения. Диаграмма сотрудничества описывает именно статическую структуру объектов, участвующих в реализации поведения.

Сотрудничество может быть параметрическим, и в этом случае оно представляет собой шаблон, который может использоваться в различных частях проекта. Параметрами могут являться участники сотрудничества.

Диаграмма сотрудничества включает в себя объекты и отношения между ними, заключающееся в вызове методов друг друга. Некоторые объекты появляются только в рамках реализации сотрудничества, они помечаются специальным словом “new” (новый). Те объекты, которые уничтожаются во время реализации сотрудничества помечаются специальным словом “destroy” (уничтожить).

На диаграмме могут быть показаны связи между объектами представляющие:

параметры процедур,

локальные переменные,

self ссылки (ссылки на сам объект).

В случае вызова метода одного объекта другим объектом, рядом со связью указывается имя метода и задается направление взаимодействия (чей метод вызывается). Так как диаграммы сотрудничества очень часто используются для построения процедурных спецификаций, допускается указывать последовательности вызовов методов путем их нумерации.

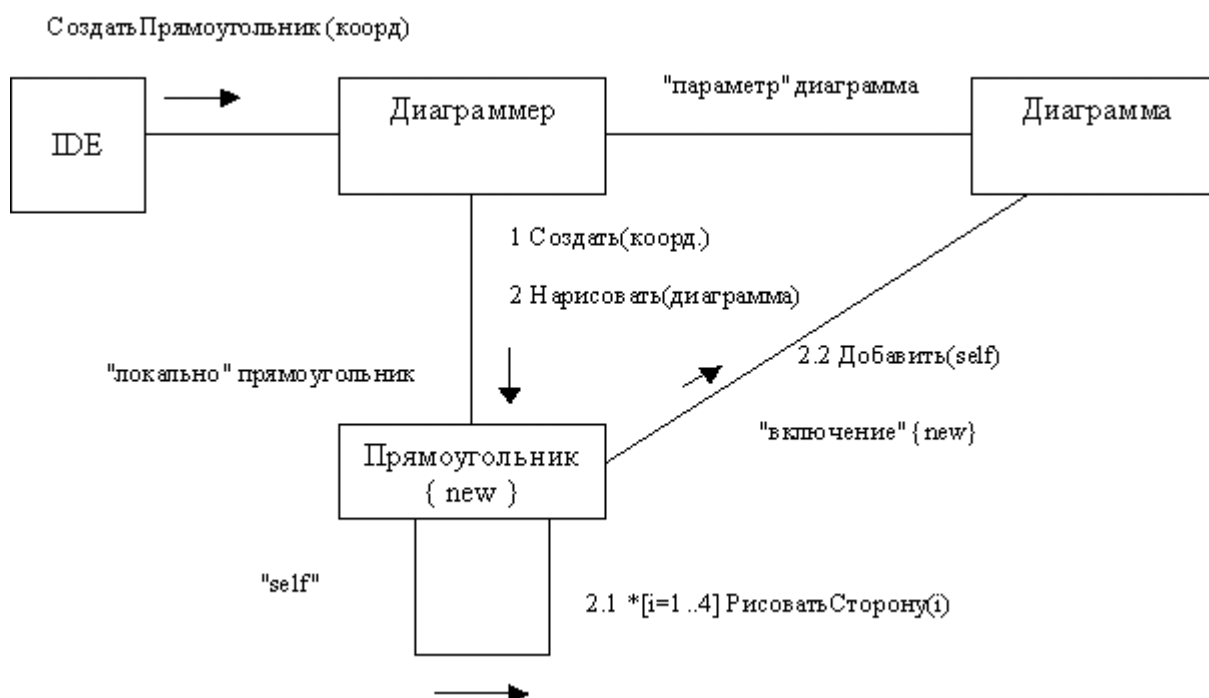


Рис. 7.17. Пример диаграммы сотрудничества

7.6 Диаграммы состояний

Диаграмма состояний (State chart diagram) представляет собой конечный автомат и показывает последовательность состояний объекта, через которые он проходит во время своего существования под воздействием внешних событий. Диаграмма представляет собой набор состояний и переходов между ними. Диаграмма состояний назначается классу или методу поведения.

7.6.1 Состояния

Состояния автомата соответствуют состояниям объектов в которых объект удовлетворяет некоторому условию, выполняет некоторое действие или ожидает некоторого события. Объект может находиться в каждом состоянии в течение конечного времени. Каждому состоянию может соответствовать вложенный автомат. Состояние изображается как прямоугольник со скругленными краями. Каждое состояние имеет две части. В верхней части отображается имя состояния. Имя

состояния может быть пустым, это так называемое анонимное состояние. Все анонимные состояния отличаются друг от друга. Нижняя часть состояния предназначена для отображения внутренних действий, выполняемых в ответ на определенные события, возникающие когда автомат объект находится в данном состоянии, без смены текущего состояния. Описание внутренних действий имеет следующий формат:

имя - события список - параметров [условие] / действие

Каждое имя события должно быть уникальным в пределах одного состояния. Определены зарезервированные имена событий:

entry / действие - Действие, выполняемое при входе в состояние.

exit / действие - Действие, выполняемое при выходе из состояния.

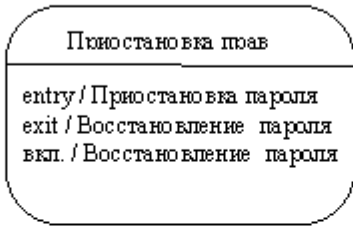


Эти два predetermined события не имеют списка параметров и условия возникновения, т.к. и то и другое predetermined.

Введено специальное ключевое слово do, обозначающее вызов вложенного автомата:

do / имя - автомата (список - параметров)

Имя автомата задает конкретный автомат, которому при инициализации может быть передан список параметров, который должен быть совместимым со списком параметров этого автомата. Графические обозначения различных классов состояний автомата приведены в таблице ниже:

Таблица 7.3. Обозначения элементов диаграммы состояний

Наименование	Графическое обозначение	Смысл
состояние		В верхней части - имя, в нижней - внутренние действия, выполняемые автоматом при возникновении определенных событий, когда он находится в данном состоянии
начальное состояние		С него начинается выполнение автомата
конечное состояние		На нем заканчивается выполнение автомата

7.6.2 События

События это любое действие, имеющее значение с точки зрения смены состояний автомата. Определены следующие виды событий (одно событие может относиться одновременно к нескольким видам):

- условие становится истинным,
- прием внешнего сигнала от одного объекта к другому,
- запрос на выполнение метода объекта,
- истечение заданного периода времени.

Сигналы определяются в следующем формате:

имя - события (список - параметров)

каждый параметр имеет формат:

имя - параметра : тип

Сигналы могут образовывать иерархию наследования и, следовательно, изображаться с помощью диаграммы классов. Если на конечном автомате событие X приводит к переходу из состояния в состояние, то любое производное событие от X так же приводит к смене этому переходу. На диаграмме классов состояния изображаются с ключевым словом “signal”. Они не имеют методов поведения, могут иметь только атрибуты.

7.6.3 Простые переходы между состояниями

Простой переход из состояния 1 в состояние 2 показывает, что объект, находящийся в состоянии 1 перейдет в состояние 2 и выполнит определенные действия когда произойдет предопределенное для перехода событие и будут истинными специфицированные условия. Такая смена состояний называется срабатыванием перехода. Событие, помечающее переход, может иметь параметры, которые доступны внутри действий, сопоставленных переходу или тому состоянию, в который ведет переход. События, определяющие выполнение переходов, обрабатываются по одному в один момент времени. Если событие не приводит к выполнению ни одного перехода, то оно игнорируется. Переходы изображаются линией с указанием направления , и дополнительной текстовой информацией, представленной с следующим виде:

имя - события (параметры) [условие] / выражение ^ посылка

Условие - это логическое выражение с участием параметров события, имен состояний автомата и атрибутов объекта или класса, которым назначен данный автомат. Выражение - это процедурное выражение, которое выполняется при выполнении перехода. Оно может включать в себя параметры события , методы поведения и атрибуты того объекта или класса, которым назначен данный автомат. Посылка описывает ту операцию или сигнал, которые могут быть посланы другим объектам. Она имеет следующий формат:

назначение . операция (параметры)

Назначение - это имя объекта для которого выполняется операция, или посылается сообщение или сигнал. Операция - это имя вызываемого метода, посылаемого сигнала или сообщения. Примером текста, который может сопоставляться переходу является следующий:

нажата левая кнопка мыши (позиция)

[позиция внутри окна]

/ выбор = выбрать (позиция)

^ выбор . выделить.

Здесь “нажата левая кнопка мыши” - это имя события, “позиция” - это параметр данного события, показывающий в какой позиции на экране произошло нажатие. “позиция внутри окна” - это условие которое должно быть выполнено для того, что бы переход мог выполняться, т.е. будут обрабатываться не все нажатия кнопки мыши, а только те которые произошли внутри заданного окна. Выражение “выбор = выбрать (позиция)” представляет собой присвоение переменной “выбор” значения функции “выбрать” с параметром “позиция”. И, наконец “выбор . выделить” - это обращение к методу “выделить” объекта “выбор”.

7.6.4 Составные переходы между состояниями

Составные переходы предназначены для отражения операций распараллеливания и синхронизации. Составной переход может иметь несколько входных и несколько выходных состояний. Составной переход может выполняться только тогда, когда все его входные состояния активны. После выполнения составного перехода все его выходные состояния становятся активными, а все входные перестают быть активными. Таким образом автомат может одновременно находиться в нескольких состояниях. Составной переход изображается как прямоугольник, входные и выходные линии определяют входные и выходные состояния:

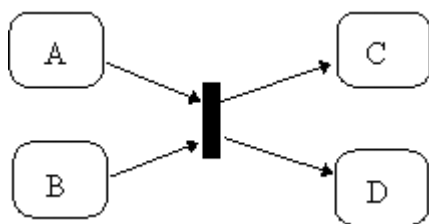


Рис. 7.18. Обозначение составного перехода.

На данном рисунке изображен один составной переход, имеющий два входных состояния (A, B) и два выходных состояния (C, D).

7.6.5 Вложенные автоматы

Если состоянию соответствует вложенный автомат, то после выполнения действий entry для этого состояния, начинает работать вложенный автомат, начиная со своего начального состояния. Когда достигается одно из конечных состояний вложенного автомата, то состояние считается законченным, выполняются действия по выходу (exit), и автомат готов к смене состояния под воздействием внешних воздействий.

Одному состоянию может соответствовать либо один вложенный автомат, либо несколько конкурирующих (взаимно исключающих) вложенных автоматов. Графически это обозначается следующим образом:

- один вложенный автомат изображается внутри прямоугольника для состояния,
- несколько вложенных автоматов так же изображаются внутри прямоугольника для состояния, но они разделяются пунктирным линиями.

Пример вложенного автомата приведен ниже:

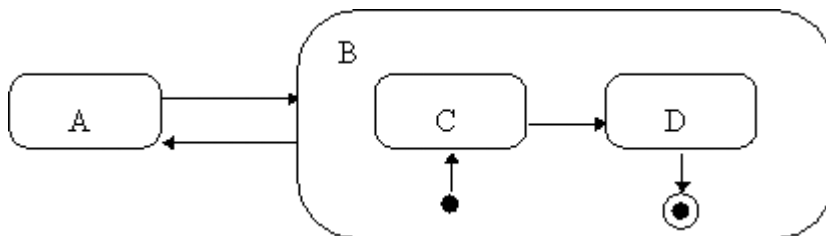


Рис. 7.19. Пример изображения вложенного автомата

На данном рисунке состояние В является составным и содержит внутренний автомат, который состоит из начального, конечного состояний и состояний С и D.

Пример конкурирующих вложенных автоматов представлен на рисунке ниже:

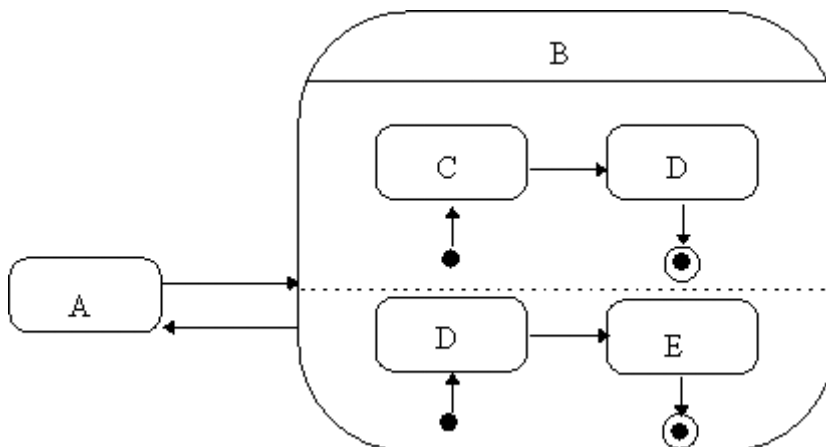


Рис. 7.20. Пример изображения конкурирующих вложенных автоматов.

Здесь состояние “В” декомпозируется с помощью двух конкурирующих вложенных автоматов.

Пример диаграммы состояний представлен на рис. Здесь представлен конечный автомат, описывающий функционирование класса "Диаграммер" графического редактора диаграмм.

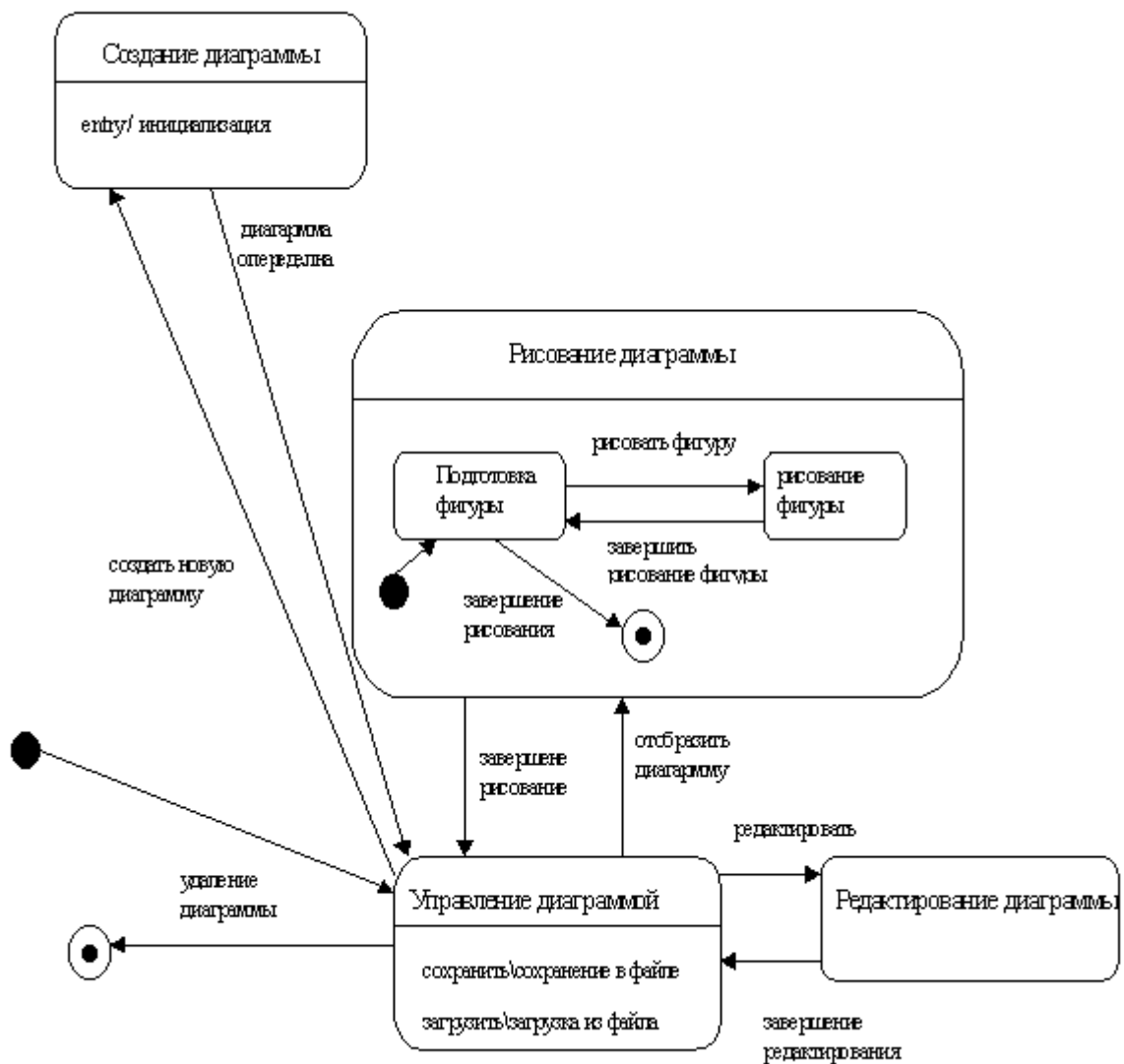


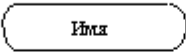
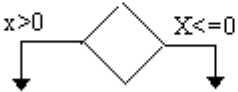




Рис. 7.21. Диаграмма состояний для класса "Диаграммер"

7.7 Диаграммы действий

Диаграммы действий (activity diagrams) показывают выполнение операций. Они являются разновидностью автомата. Предназначение данной диаграммы - показать поток управления, внутренний для операции, в противоположность показу реакции на внешние события (как это делается в диаграмме состояний).

Диаграмма действий состоит из следующих элементов:

Таблица 7.4. Обозначение компонентов диаграммы действий.

Наименование	Графическое обозначение	Смысл
действие		Неделимая операция
условие		Выбор одного из вариантов в зависимости от условия
переход		передача управления от одного действия к другому, с возможностью распараллеливания
передача сигнала		Операция отправки сигнала при выполнении перехода
прием сигнала		ожидание прихода сигнала для выполнения перехода
Объект		Объекты, используемые при выполнении действий

7.7.1 Действия

Действия показывают выполнение некоторой неделимой операции. Каждое действие имеет имя, определяющее смысл этого действия. Имя может представлять собой текст на естественном языке, псевдокод операции или фрагмент текста на некотором языке программирования. Внутри описания могут использоваться атрибуты того объекта, за которым закреплена диаграмма действий.

7.7.2 Условия

Условия предназначены для обозначения возможности условной передачи управления в соответствии со значением некоторого логического выражения. Условие может иметь один или более входов и два или более выходов. Каждый выход должен быть помечен условием, истинность которого обеспечивает переход по данной дуге.

7.7.3 Переходы

Переходы имеют тот же смысл, что и в автоматной модели диаграммы состояний. Но здесь они не помечаются никаким событием и имеют условие только для специальных состояний - "условие", т.е. они просто передают управление от одного действия к другому. Окончание входных действий непосредственно приводит к выполнению перехода. Возможность распараллеливания и синхронизации остается.

7.7.4 Полосы выполнения

Диаграмма действий может быть разделена на полосы (swim lanes), которые включают в себя определенный набор действий и переходов. Каждая полоса имеет

собственное имя и тем самым позволяет группировать действия в единое целое. Его можно сравнить с пакетом для классов. Графически каждая полоса представляет собой вертикальное разделение диаграммы действий с помощью сплошной линии. Каждое действие может находиться только в одной полосе, тогда как переходы могут пересекать полосы.

Ниже представлен пример, на котором используются практически все возможные элементы диаграммы действий. Здесь обсуждается процедура обслуживания клиента узла Internet, подающего заявку на обслуживания. Заявка может быть двух типов: заявка на регистрацию и заявка на предоставление некоторой услуги. Диаграмма содержит две полосы “клиент” и “отдел обслуживания”. После получения запроса на обслуживание клиент может подготовить платеж, а отдел обслуживания займется одновременно с этим выполнением заказа. В зависимости от типа запроса будет выполнено либо действие “создать услугу” либо действие “заполнить карточку регистрации”. Далее будет выполнена подготовка документов и после выполнения платежа, он будет учтен и обслуживание закончится.

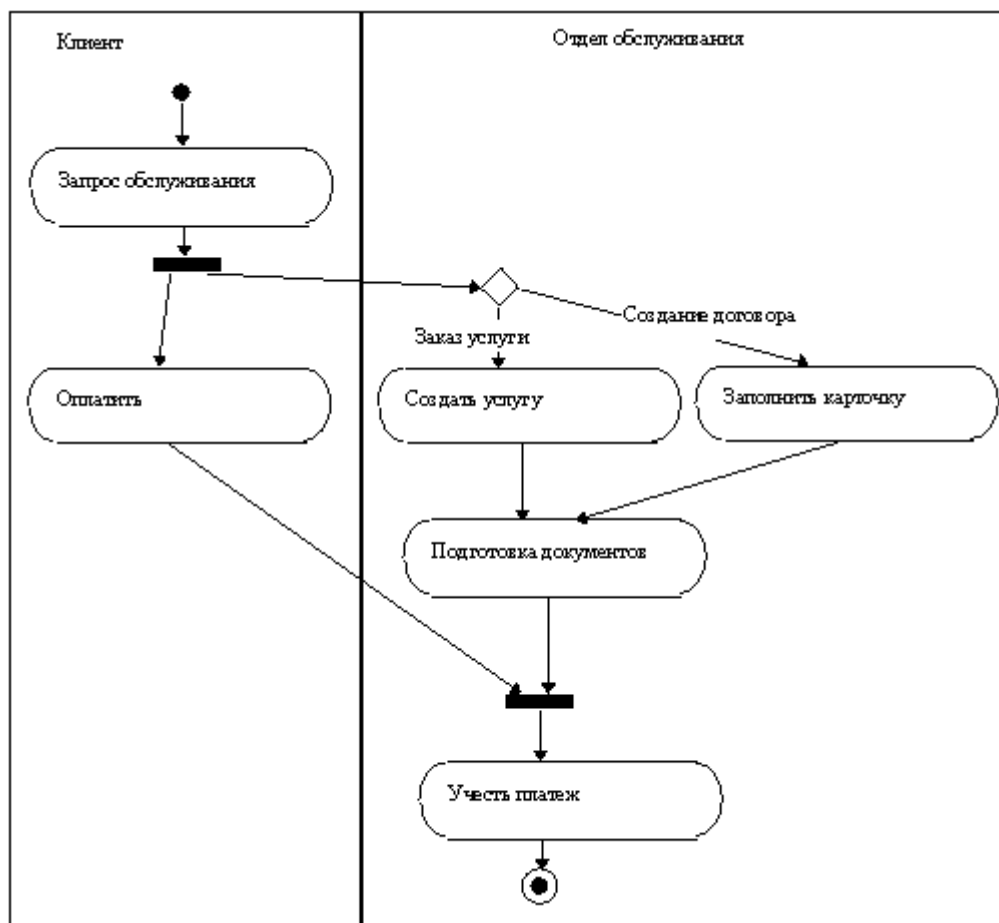


Рис. 7.22. Пример диаграммы действий.

7.7.5 Отношения между действиями и объектами

Все действия выполняются над объектами. Целесообразно показать на диаграмме действий отношения между объектами и операциями. Различаются два вида отношений:

- объект отвечает за выполнение операции,

- атрибуты объекта используются для выполнения операции.

Эти виды отношений показываются соответственно как:

- действие вызывает метод поведения объекта,
- объект является входным или выходным для действия.

Объекты на диаграмме действий показываются как обычно в виде прямоугольников с записанными внутри именами. Если объект является выходным для действия, то от действия к объекту идет штриховая линия, если же объект является входным для действия, то от объекту к действию идет штриховая линия. Вызов метода показывается сплошной линией с указанием имени и параметров операции, а так же может быть указано направление передачи или получения данных (см. диаграмму последовательностей).

7.7.6 Специальные символы

Специальные символы могут не использоваться, т.к. их использование не обязательно, они являются дополнительным средством выражения посылки и получения сигналов. Это можно обозначить указанием имени сигналов непосредственно на переходе. А можно использовать специальные символы, которые должны соединяться с линией перехода.

“Получение сигнала” (см. таблицу выше) - это символ, предназначенный для обозначения получения сигнала для выполнения перехода, т.е. передачи управления.

“Посылка сигнала” (см. таблицу выше) - это символ, предназначенный для обозначения посылки сигнала в момент выполнения перехода.

7.8 Диаграммы реализации

Диаграммы реализации предназначены для отображения состава компилируемых и выполняемых модулей системы, а так же связей между ними. Диаграммы реализаций разделяются на два конкретных вида: диаграммы компонентов (component diagrams) и диаграммы развертывания (deployment diagrams).

7.8.1 Диаграммы компонентов

Диаграмма компонентов отражает зависимости составных частей программного обеспечения, в которые включаются файлы исходных текстов, двоичные файлы библиотек объектных модулей и исполняемые файлы. Она состоит из компонентов и отношений между ними. Используются отношения двух типов:

- зависимость - это зависимость любого типа (использование, совместная компиляция),
- композиция - это включение одних компонентов в состав других.

Компонент изображается в виде прямоугольника с двумя маленькими прямоугольниками у левого края, внутри прямоугольника записывается имя компонента.

Зависимость изображается штриховой линией от использующего компонента к используемому. Композиция (или включение) изображается размещением включаемого компонента внутри включающего. Компоненты могут иметь интерфейсы, через которые выражаются зависимости. Интерфейсами могут являться, например, имена вызываемых подпрограмм. Интерфейсы изображаются окружностями, соединенными с компонентой линией без направления, рядом записывается имя интерфейса.

Ниже (рис. 7.23) представлен пример диаграммы компонентов, состоящий из компонентов “графический редактор” и “оконная система”. Оконная система зависит от интерфейса “нарисовать”, имеющегося у компонента “графический редактор”.



Рис. 7.23. Пример диаграммы компонентов

7.8.2 Диаграммы развертывания

Диаграммы развертывания показывают конфигурацию исполняемой программной системы, состоящей из программных компонентов, процессов, объектов. Она состоит из узлов и отношений взаимодействия между узлами и компонентами. Узлы могут включать компоненты и объекты.

Узлы представляют собой физические элементы времени выполнения, обозначающие вычислительный ресурс, обладающий как минимум запоминающим устройством и возможно вычислительным устройством. Узлы могут обозначать компьютеры, человеческие ресурсы или механические устройства. Внутри узлов могут содержаться компоненты и объекты, что обозначает, что данный компонент или объект существует в рамках данного узла. Узлы изображаются как проекция трехмерного куба. Узел может представлять собой тип узла или конкретный экземпляр узла. В зависимости от этого происходит именованная узла. В случае узла - типа его имя выглядит так:

имя типа,

в случае узла - экземпляра имя выглядит так:

имя узла : имя типа.

На диаграмме развертывания компоненты могут представлять не только типы, но и конкретные экземпляры, поэтому их имя может быть дополнено именем типа через двоеточие.

Отношение взаимодействия между узлами, компонентами или объектами обозначается штриховой линией, направленной от использующего элемента к используемому.

Ниже на рисунке (рис.7.24) представлен пример, состоящий из трех вычислительных систем, представляющих типичный случай системы доступа к базам данным на основе Internet/Intranet технологии, которая предполагает наличие двух

компьютеров, один из которых выполняет функции web - сервера с возможностью исполнения ASP или CGI, который готовит данные для отображения на компьютере клиента. Для подготовки отображаемых данных web - сервер, в лице компонента “исполнение ASP программ”, выполняет запросы к SQL - серверу, который хранит данные и обрабатывает их. Все это делается по запросу с клиентского компьютера и на него же посылаются результаты запросов в подготовленном для отображения виде (HTML).

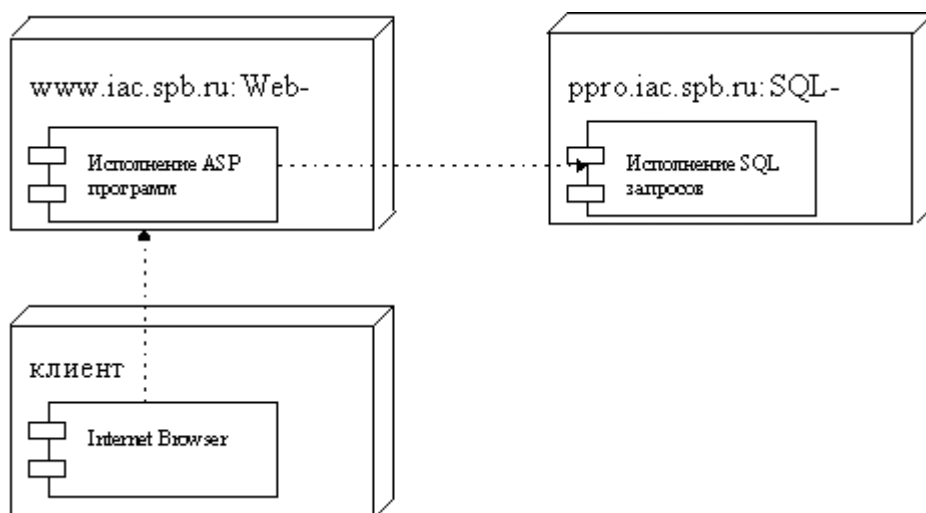


Рис. 7.24. Пример диаграммы развертывания