**Диаграммы состояний**

6 Диаграммы состояний (statechart diagram)

Главное предназначение этой диаграммы – описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Чаще всего диаграммы состояний используются для описания поведения отдельных экземпляров классов (объектов), но они также могут быть применены для спецификации функциональности других компонентов моделей, таких как варианты использования, актеры, подсистемы, операции и методы.

На диаграмме состояний отображают жизненный цикл одного объекта, начиная с момента его создания и кончая разрушением.

*Диаграмма состояний (stateshar tdiagram)* - показывает положение одиночного объекта, события или сообщения, которые вызвают переход из одного состояния в другое, и действия, являющиеся результатом смены состояния.

Диаграмма состояний по существу является графом специального вида, который представляет некоторый автомат. Вершинами этого графа являются состояния и некоторые другие типы элементов автомата (псевдосостояния), которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние. Для понимания семантики конкретной диаграммы состояний необходимо представлять не только особенности поведения моделируемой сущности, но и знать общие сведения по теории автоматов.  
  
***Автоматы***

Автомат (state machine) в языке UML представляет собой некоторый формализм для моделирования поведения элементов модели и системы в целом. Автомат описывает поведение отдельного объекта в форме последовательности состояний, которые охватывают все этапы его жизненного цикла, начиная от создания объекта и заканчивая его уничтожением. Каждая диаграмма состояний представляет некоторый автомат.

Простейшим примером визуального представления состояний и переходов на основе формализма автоматов может служить ситуация с исправностью технического устройства, такого как компьютер. В этом случае вводятся в рассмотрение два самых общих состояния: "исправен" и "неисправен" и два перехода: "выход из строя" и "ремонт". Графически эта информация может быть представлена в виде изображенной ниже диаграммы состояний компьютера (рис. 1)

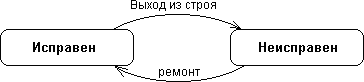


Рис. 1 Простейший пример диаграммы состояний для технического устройства типа компьютер

Основными понятиями, входящими в формализм автомата, являются состояние и переход. Главное различие между ними заключается в том, что длительность нахождения системы в отдельном состоянии существенно превышает время, которое затрачивается на переход из одного состояния в другое. Предполагается, что в пределе время перехода из одного состояния в другое равно нулю (если дополнительно ничего не сказано). Другими словами, переход объекта из состояния в состояние происходит мгновенно.

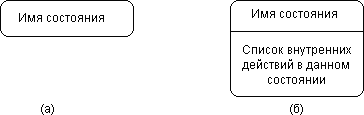
В языке UML под состоянием понимается абстрактный метакласс, используемый для моделирования отдельной ситуации, в течение которой имеет место выполнение некоторого условия. Состояние может быть задано в виде набора конкретных значений атрибутов класса или объекта, при этом изменение их отдельных значений будет отражать изменение состояния моделируемого класса или объекта.  
  


Рис. 2 Графическое изображение состояний на диаграмме состояний  
  
Состояние на диаграмме изображается прямоугольником со скругленными вершинами (рис. 2). Этот прямоугольник, в свою очередь, может быть разделен на две секции горизонтальной линией. Если указана лишь одна секция, то в ней записывается только имя состояния (рис.2, а). В противном случае в первой из них записывается имя состояния, а во второй – список некоторых внутренних действий или переходов в данном состоянии (рис. 2, б).

Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий. В этом состоянии находится объект по умолчанию в начальный момент времени. Графически начальное состояние в языке UML обозначается в виде закрашенного кружка (рис. 3, а), из которого может только выходить стрелка, соответствующая переходу.

Описание: Рис53

Рис. 3 Графическое изображение начального и конечного состояний на диаграмме состояний

Конечное (финальное) состояние представляет собой частный случай состояния, которое также не содержит никаких внутренних действий. В этом состоянии будет находиться объект по умолчанию после завершения работы автомата в конечный момент времени. Графически конечное состояние в языке UML обозначается в виде закрашенного кружка, помещенного в окружность (рис. 3, б), в которую может только входить стрелка, соответствующая переходу.

Простой переход (simple transition) представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим. Пребывание моделируемого объекта в первом состоянии может сопровождаться выполнением некоторых действий, а переход во второе состояние будет возможен после завершения этих действий, а также после удовлетворения некоторых дополнительных условий. На диаграмме состояний переход изображается сплошной линией со стрелкой, которая направлена в целевое состояние (рис. 456).

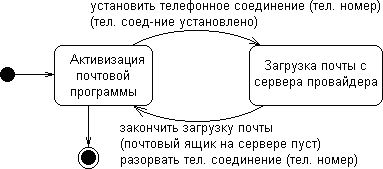


Рис. 4 Диаграмма состояний для моделирования почтовой программы-клиента  
  
Пример диаграммы состояний Рассмотрим пример диаграммы состояний для моделирования поведения банкомата (рис. 5).

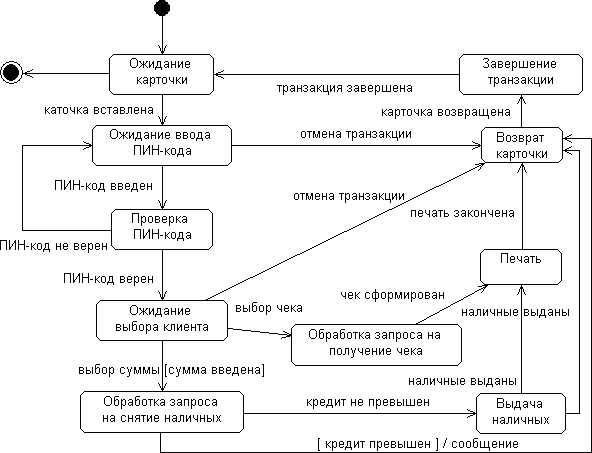


Рис.5 Диаграмма состояний для моделирования поведения банкомата Следует заметить, что в разрабатываемой модели диаграмма состояний является единственной и описывает поведение системы управления банкоматом в целом. Главное достоинство данной диаграммы состояний – возможность моделировать условный характер реализации всех вариантов использования в форме изменения отдельных состояний разрабатываемой системы. Иногда разработку диаграммы состояний, особенно в условиях дефицита времени, отпущенного на выполнение проекта, опускают, т.к. часто происходит дублирование информации, представленной на диаграммах кооперации и последовательности.

Диаграммы деятельности можно использовать для моделирования динамических аспектов системы. Как правило, они применяются , чтобы промоделировать последовательные шаги вычислительного процесса. Если в диаграммах взаимодействий акцент делается на переходах потока управления от одного объекта к другому, то диаграммы деятельности описывют переходы от одной деятельности к другой.

**Диаграмма деятельности (Activitydiagram)-**показывает поток перехода от одной деятельности к другой.

**Деятельность** –это продолжающийся во времени неатомарный шаг вычислений в автомате.

Деятельности, в конечном счете, приводят к выполнению некоего действия, составленного из выполняемых атомарных вычисленийкадое из которых либо изменяет состояние системы, либо возвращет какое-то значение.

Диаграмма деятельности состоит из:

Состояний деятельности и состояний действия

Переходов

Объектов

Состояния действия могут быть подвергнуты декомпозиции, они атомарны. Выполняемая в состоянии действия работа не может быть прервана, при этом предполагается, что длительность одного состояния действия занимает неощутимо малое время.

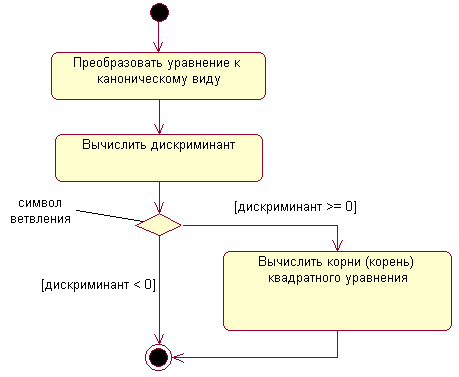
Состояния деятельности могут быть подвергнуты дальнейшей декомпозиции, они не являются атомарными и могут быть прерваны. Предполагается, что для их завершения требуется заметное время.

Когда действие или деятельность в некотором состоянии завершается, поток управления сразу переходит в следующее состояние деятельности или действия. Для описания потока управления используются **переходы (transaction)**  - показывающие путь от одного состояния действия /деятельности в другое. Такие переходы называются нетриггерными, поскольку управление по завершении работы в исходном состоянии немедленно передается дальше.

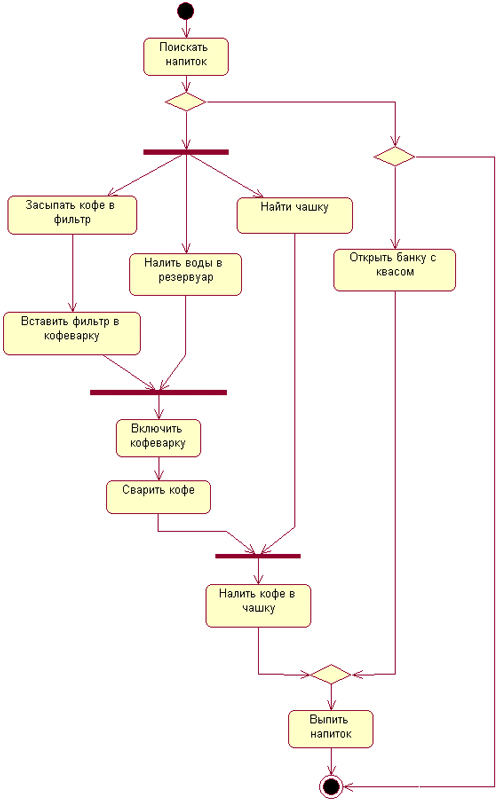
Поток управления где-то должен начинаться и где-то заканчиваться. Можно задать начало и конец потока.

В модель можно включить *ветвление*, которое описывает различные пути выполнения в зависимости от некоторого условия.В точку ветвления может входить один переход, а выходжит –два и более. Для каждого исходящего перехода задается будево выражение, которое вычисляется один раз при входе в точку перехода.

Простые и ветвлящиеся потоки в диаграммах деятельности встречаются чаще всего. Но можно встретитьи параллельно выполняемые потоки, что характерно для моделирования бизнес-процессов. В UML для организации таких потоков используется **синхронизация.** Точка слияния представляет собой механизм синхронизации нескольких параллельных потоков выполнения.- каждый из них ждет, пока все остальные достигнут этой точки, после чего выполнение продолжается в рамках одного процесса. В эту точку входят один или несколько переходов, а выходит один.



Фрагмент диаграммы деятельности для алгоритма нахождения корней квадратного уравнения

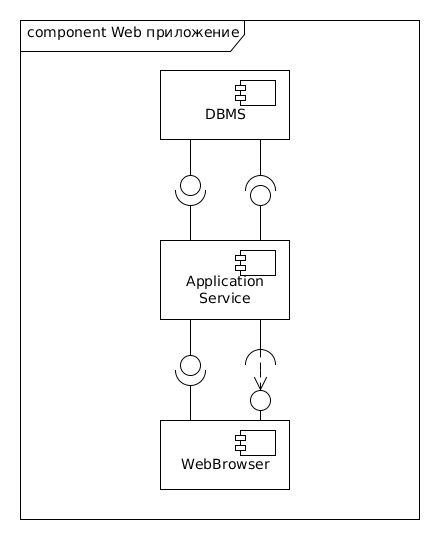


**Рис. 7.6.** Диаграмма деятельности для примера с приготовлением напитка

Таким образом, диаграмма деятельности есть не что иное, как специальный случай диаграммы состояний, в которой все или большинство состояний являются действиями или состояниями под-деятельности. А все или большинство переходов являются нетригтерными переходами, которые срабатывают по завершении действий или под-деятельностей в состояниях-источниках.

**Диаграмма компонентов**

**Диаграмма компонентов** (component diagram) — показывает взаимосвязи между модулями (логическими или физическими), из которых состоит моделируемая система.

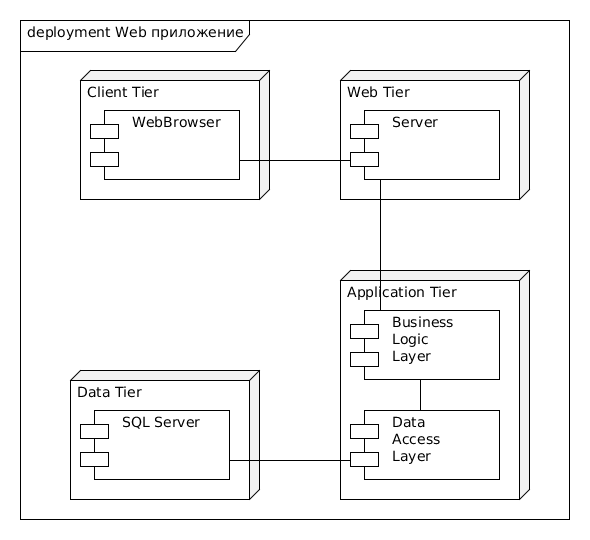


Основной тип сущностей на диаграмме компонентов — это сами компоненты, а также интерфейсы, посредством которых указывается взаимосвязь между компонентами. На диаграмме компонентов применяются следующие отношения:

* реализации между компонентами и интерфейсами (компонент реализует интерфейс);
* зависимости между компонентами и интерфейсами (компонент использует интерфейс);

**Диаграмма размещения**

**Диаграмма размещения** (deployment diagram) наряду с отображением состава и связей элементов системы показывает, как они физически размещены на вычислительных ресурсах во время выполнения.



На диаграмме размещения, по сравнению с диаграммой компонентов, добавляется два типа сущностей: артефакт, который является реализацией компонента и узел (может быть как классификатор, описывающий тип узла, так и конкретный экземпляр), а также отношение ассоциации между узлами, показывающее, что узлы физически связаны во время выполнения.

**ГЕНЕРАЦИЯ ПРОГРАММНОГО КОДА В С++**

Выполните команду для проверки модели: Tools🡪CheckModel

Выполните команду: Report🡪ShowAccessViolations для проверки нарушений правил доступа.

Установить свойства генерации программного кода, выполнив команду: Tools🡪Options🡪C++

Выбрать класс или компонент для генерации на диаграмме классов или компонентов. Можно выбрать несколько, используя клавишу ALT.

Выполнить команду: Tools🡪C++🡪Code Generation

Для просмотра файла заголовка (. h) выполнить команду: Tools🡪C++🡪Browse Header

Для просмотра файла реализации (.срр) выполнить команду: Tools🡪C++🡪Browse Body