**ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ СРЕДНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ МАРИЙ ЭЛ**

"**ЙОШКАР**-**ОЛИНСКИЙ** **ТЕХНОЛОГИЧЕСКИЙ** **КОЛЛЕДЖ**"

**ОТЧЁТ**

**О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

**«Диаграмма состояний по теме (космос)»**

Студента 3 курса, А-31 группы

Специальность 09.02.07 – «Информационные системы и программирования»

Квалификация «Администратор БД»

Руководитель: Пинешкин Ю.С.

Разработал:Тимофеев М.А.

г. Йошкар-Ола, 2022

Диаграмма деятельности полезна для описания алгоритма действий, но она не дает представления о поведении определенного объекта в рамках отдельного варианта использования или системы в целом, что необходимо при объектно-ориентированном программировании.

На сегодняшний день при проектировании сложной Системы принято делить ее на части, каждую из которых затем рассматривать отдельно. Таким образом, при объектной декомпозиции Система разбивается на объекты или компоненты, которые взаимодействуют друг с другом, обмениваясь сообщениями. Сообщения  описывают или представляют собой некоторые события. Получение объектом сообщения активизирует его и побуждает выполнять предписанные его программным кодом действия.

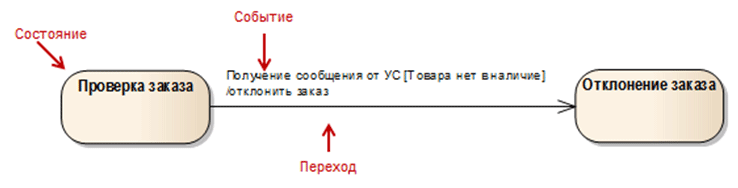
При данном подходе Система становится событийно управляемой, поэтому разработчикам  зачастую важно знать, как должен реагировать тот или иной объект на определенные события. Инициаторами событий могут быть как объекты самой Системы, так и её внешнее окружение.

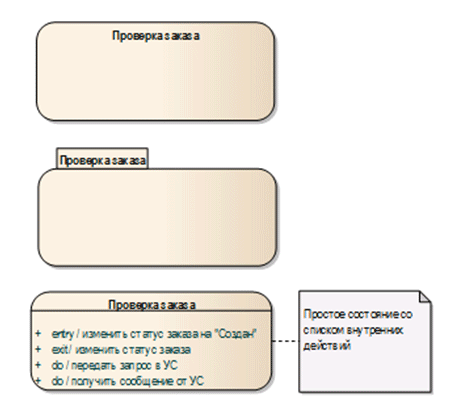
Описать поведение отдельно взятого объекта помогает диаграмма состояний.

Также зачастую диаграмма состояний используется аналитиками для описания последовательности переходов объекта из одного состояния в другое.

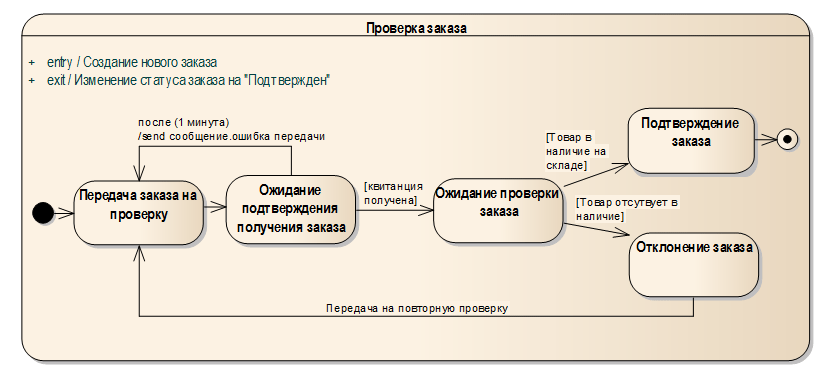
Диаграмма состояний покажет нам все возможные состояния, в которых может находиться объект, а также процесс смены состояний в результате внешнего влияния.

Основными элементами диаграммы состояний являются «Состояние» и «Переход». Диаграмма состояний имеет схожую семантику с диаграммой деятельности, только деятельность здесь заменена состоянием, переходы символизируют действия. Таким образом, если для диаграммы деятельности отличие между понятиями «Деятельность» и «Действие» заключается в возможности дальнейшей декомпозиции, то на диаграмме состояний деятельность символизирует состояние, в котором объект находится продолжительное количество времени, в то время как действие моментально.



Переход может быть инициирован событием, которое также отражается на диаграмме состояний.  
  
 Состояние может содержать только имя или имя и дополнительно список внутренних действий. Список внутренних действий содержит перечень действий или деятельностей, которые выполняются во время нахождения объекта в данном состоянии. Данный список фиксированный. Список основных действий включает следующие значения:

* entry - действие, которое выполняется в момент входа в данное состояние (входное действие);
* exit - действие, которое выполняется в момент выхода из данного состояния (выходное действие);
* do - выполняющаяся деятельность ("do activity") в течение всего времени, пока объект находится в данном состоянии
* defer - событие, обработка которого предписывается в другом состоянии, но после того, как все операции в текущем будут завершены.



Факт смены одного состояния другим изображается с помощью **перехода**.   Переход осуществляется при наступлении некоторого события: окончания выполнения деятельности (do activity), получении объектом сообщения или приемом сигнала (подробнее события будут рассмотрены позднее). Переход может быть **тригерным** и **нетригерным**. Если переход срабатывает, когда все операции исходного состояния завершены, он называется нетригерным или переходом по завершении. Если переход инициируется каким-либо событием, он считается тригерным. Для тригерного перехода характерно наличие имени,  которое может быть записано в следующем формате:

<имя события>'('<список параметров, разделенных запятыми>')'['<сторожевое условие>']' <выражение действия>.

Обязательным параметром является только имя события.

В качестве **события** могут выступать сигналы, вызовы, окончание фиксированных промежутков времени или моменты окончания выполнения определенных действий. После имени события могут следовать круглые скобки для явного задания параметров соответствующего события-триггера (например, пользователь инициирующий действие).

Различаются следующие виды событий:

* *Событие вызова (call event)*  - событие, возникающее при вызове метода класса. При срабатывании данного вида события объектом ассинхронно создается Сигнал, который принимается другим объектом. Для указания на то, что некоторая операция посылает сигнал, можно воспользоваться зависимостью со стереотипом send.
* *Событие сигнала (signal event)*- событие, возникающее при посылке сигнала. Если событие сигнала представляет возбуждение сигнала, то событие вызова предназначено для описания выполнения операции. Т.е. переход осуществляется при получении сигнала от другого объекта. В то время как сигнал является событием асинхронным, событие вызова обычно синхронно. Сигнал также может быть представлен на диаграмме в виде объекта со стереотипом «signal».
* *Событие таймера (time event)*- возникает, когда истек заданный интервал времени с момента попадания автомата в данное состояние. В UML событие времени моделируется с помощью ключевого слова after(после), за которым следует выражение, вычисляющее некоторый промежуток времени.
* *Событие изменения (change event)*- событие, которое возникает, когда некоторое логическое условие становится истинным, будучи до этого ложным. Данное событие моделируется с помощью ключевого слова when

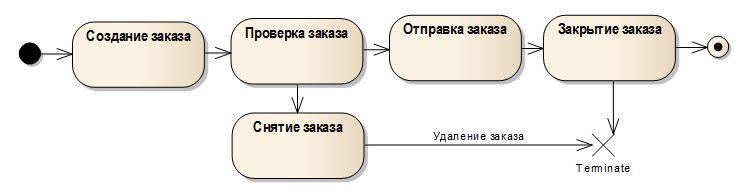
Если при срабатывании перехода возможно ветвление, в имени перехода используется сторожевое условие. **Сторожевое условие (guard condition)** всегда записывается в прямых скобках после события-триггера и представляет собой некоторое булевское выражение. В общем, случае из одного состояния может быть несколько переходов с одним и тем же событием-триггером, при этом целевое состояние будет зависеть от того какое из сторожевых условий примет значение «истина».

Также имя перехода может содержать**выражение действия (action expression)**.  В данном случае указанное действие выполняется сразу при срабатывании перехода и до начала каких бы то ни было действий в целевом состоянии. В общем случае выражение действия может содержать целый список отдельных действий, разделенных символом «;».

При создании диаграммы состояний для отдельных компонентов Системы выражение действия записывается на одном из языков программирования, который предполагается использовать для реализации модели.

Помимо основных узлов, на диаграмме состояний могут использоваться, так называемые, псевдосостояния – вершины которые не обладают поведением, и объект не находится в ней, а «мгновенно» ее проходит.

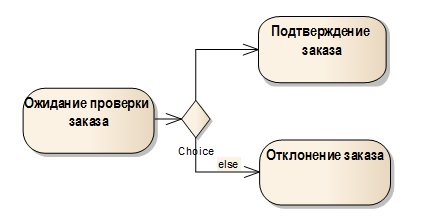
Под псевдосостояниями на диаграмме состояний понимаются, знакомые уже нам начальное и конечное состояние. **Начальное состояние** обычно не содержит никаких внутренних действий и определяет точку, в которой находится объект по умолчанию в начальный момент времени. **Конечное состояние**  также не содержит никаких внутренних действий и служит для указания на диаграмме области, в которой завершается процесс изменения состояний в контексте конечного автомата.



Если необходимо отразить уничтожение объекта используется *узел завершения (terminate node),* псевдосостояние, вход в который означает завершение выполнения поведения конечного автомата в контексте его объекта.

Узлы ветвления и объединения аналогичны узлам на диаграмме деятельности. Основная цель данных подсостояний показать параллельную работу подавтоматов. На диаграмме состояний обычно данные подсостяония используются распараллеливания переходов в композитных состояниях, о которых речь пойдет позже. После срабатывания перехода моделируемый объект одновременно будет находиться во всех целевых состояниях этого перехода.

Варианты принятия решений на диаграмме состояний могут быть показаны также как и на диаграмме деятельности с помощью узла выбора. При этом переход в состояние выбора должен быть триггерным и содержать имя события. Переходы из псевдосостояния выбора в целевые состояния должны содержать сторожевые условия. Переход, который должен срабатывать, если ни одно из условий не примет значение «истина» должен содержать метку «else».

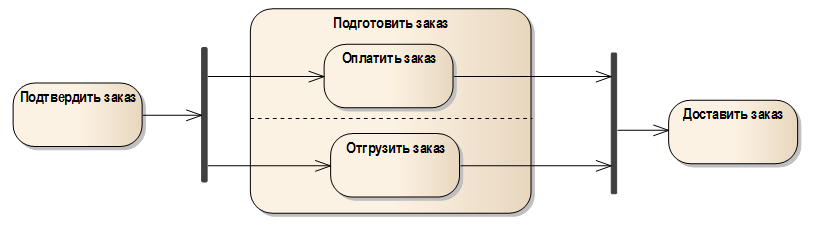


В отличие от диаграммы деятельности, при отображении возможных вариантов перехода на диаграмме состояний узел выбора использовать не обязательно. Диаграмма состояний должна показывать возможное изменение состояния объекта, и не имеет своей целью выстраивать четкую последовательность переходов. Таким образом, из одного состояния могут выходить несколько переходов, конечной целью которых будут различные целевые состояния. Для отображения возможности выбора в данном случае достаточно в имени всех переходов добавить триггер и сторожевое условие.

Также узел выбора на диаграмме состояний может быть заменен узлом соединения.  Данное псевдосостояние имеет достаточно свободную семантику и предназначено для соединения нескольких переходов. В отличие от псевдосостояния выбора узел соединения может иметь несколько входящих переходов и несколько исходящих переходов, т.е. он может принимать значение логического выражения «или».

На диаграмме могут быть представлены как простые состояния, так и сложные состояния. Сложные или составные состояния (composite state)  включают в себя вложенные подсостояния.  Декомпозиция сложного состояния может осуществляться как на основной диаграмме, так и отдельно, при этом на основной диаграмме следует использовать элемент с пиктограммой декомпозиции.

Составное состояние может содержать два или более параллельных подавтомата или несколько последовательных подсостояний.



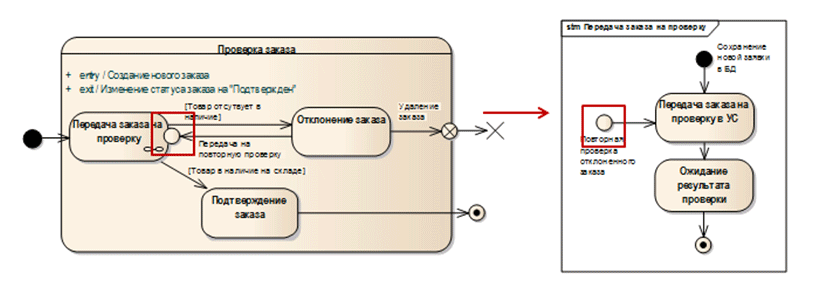
**Последовательные подсостояния** (sequential substates) используются для моделирования такого поведения объекта, во время которого в каждый момент времени объект может находиться в одном и только одном подсостояний. Поведение объекта в этом случае представляет собой последовательную смену подсостояний, начиная от начального и заканчивая конечным подсостояниями.

**Параллельные подсостояния** (concurrent substates) позволяют специфицировать два и более подавтомата, которые могут выполняться параллельно внутри составного события. Каждый из подавтоматов занимает некоторую область (регион) внутри составного состояния.

Переходы могут осуществляться как в само композитное состояние, так и в одно из его подсостояний. Таким образом, переход, стрелка которого соединена с границей некоторого составного состояния, обозначает переход в составное состояние. Он эквивалентен переходу в начальное состояние каждого из подавтоматов. Переход, выходящий из составного состояния относится к каждому из вложенных подсостояний. Это означает, что объект может покинуть составное суперсостояние, находясь в любом из его подсостояний. Если необходимо указать конкретное подсостояние из которого может осуществиться выход из композитного состояния, достаточно добавить переход от подсостояния в целевое состояние.

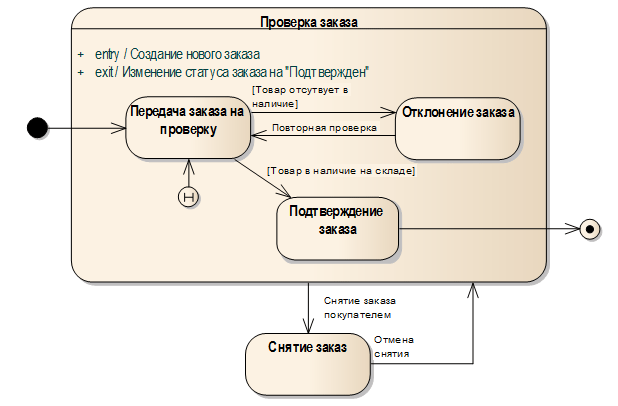
В случае перехода в сложное состояние для каждого из начальных подсостояний выполняются необходимые входные ("entry") действия. При выходе из сложного состояния для каждого из конечных подсостояний выполняются необходимые выходные ("exit") действия.

Иногда возникает ситуация, когда необходимо показать переход из одного состояния в подсостояние композитного состояния, декомпозиция которого производится на отдельной диаграмме. В UML 1.0 для подобных случаем использовались элементы «заглушка» и «ссылочное состояние». В UML 2.0 данные элементы были замены «точкой входа» и «точкой выхода».



Точка входа – псевдосостояние, моделирующее вход  в композитное состояние. При этом данная точка вход должна представлять альтернативный вход в композитное состояние, т.е. целевое подсостояние должно отличаться от начального подсостояния данного суперсостояния.

Точка выхода также символизирует альтернативный выход из композитного состояния. Данная семантика также может применяться при отображении повторяющихся действий.

Также с понятием композитного состояния тесно связано понятие исторического состояния.  


Историческое состояние используется для запоминания того из последовательных подсостояний, которое было текущим в момент выхода из составного состояния. Существует две разновидности исторического состояния: недавнее и давнее.

Недавнее историческое состояние (shallow history state) является первым подсостоянием в составном состоянии, и переход извне в это составное состояние должен вести непосредственно в это историческое состояние. При первом попадании в недавнее историческое состояние оно не хранит никакой истории (история пуста), то есть заменяет собой начальное состояние подавтомата. Далее следует последовательное изменение вложенных подсостояний.

Если в некоторый момент происходит выход из вложенного состояния (например, в случае некоторого внешнего события), то это историческое состояние запоминает то из подсостояний, которое являлось текущим на момент выхода. При следующем входе в это же составное состояние историческое подсостояние уже имеет непустую историю и сразу отправляет подавтомат в запомненное подсостояние, минуя все предшествующие ему подсостояния. В момент перехода в конечное состояние подавтомата, историческое состояние теряет свою историю.

Недавнее историческое состояние запоминает историю только того подавтомата, к которому он относится. Если запомненное состояние, в свою очередь, также являться композитным, для запоминания его подсостояния необходимо использовать давнее историческое состояние (deep history state). Давнее историческое состояние служит для запоминания всех подсостояний любого уровня вложенности для текущего подавтомата.

Автоматы состояний можно использовать при моделировании поведения графического интерфейса, как реакции на действия пользователя, различные приложения с множеством разных режимов работы в которых система ведет себя по-разному, моделирование объектов. Диаграмма автоматов зачастую используется в системах реального времени, где требуется высокая вычислительная скорость, поскольку за счет статического анализа, изменения состояний и переходы осуществляются очень быстро, как правило, это сводится к присваиванию полю класса, нескольких вызовов и запуска событий. При использовании диаграммы состояний для классов можно на ее основе сразу сгенерировать код (прямое проектирование).

Если система ожидает наступления каких-либо событий и выполняет определенные действия в ответ, конечный автомат может быть легко использован для спецификации требуемого поведения в определенном варианте использования.

При использовании диаграммы состояний важно следовать следующим правилам:

* Диаграмма состояний должна создаваться только для объектов, обладающих реактивным поведением. Не следует делать диаграмму автоматов для всех классов или объектов, достаточно выбрать только основные классы или объекты, обладающие сложным поведением.
* Диаграмма состояний должна быть сосредоточена на описании только одного аспекта поведения объекта. Следует создавать диаграмму автомата, моделирующую поведение только одного объекта. Если необходимо показать поведение нескольких, взаимосвязанных объектов, допустимо создавать для них диаграмму состояний в рамках определенного варианта использования (диаграмма состояний для варианта использования).
* На диаграмме состояний целесообразно использовать только те элементы, которые существенны для понимания описываемого аспекта.