

ТЕМА 6

План лекции:

1. Диаграммы Statechart и Activity в Rational Rose.
2. Диаграмма Statechart и Activity в Rational XDE.

6.1 Построение диаграмм Statechart и Activity в Rational Rose

Диаграмма состояний *Statechart* предназначена для изучения состояний объектов и условий переходов между ними. Модель состояний позволяет представить поведение объекта при получении им сообщений и взаимодействии с другими объектами. Кроме ранее предложенных способов создания диаграмм, *Statechart* можно построить из контекстного меню рассматриваемого класса. После активизации диаграммы, станет доступным набор ее инструментов.

Рассмотрим диаграмму *Statechart*, приведенную на рис. 6.1. Первым шагом на пути построения является создание точки начала работы с помощью инструмента *Start State*. Обычно следующим состоянием системы после начала ее работы является ожидание наступления событий. После чего в нашем случае создается состояние (*State*), которое соединяется стрелкой *State Transition* с начальной точкой.

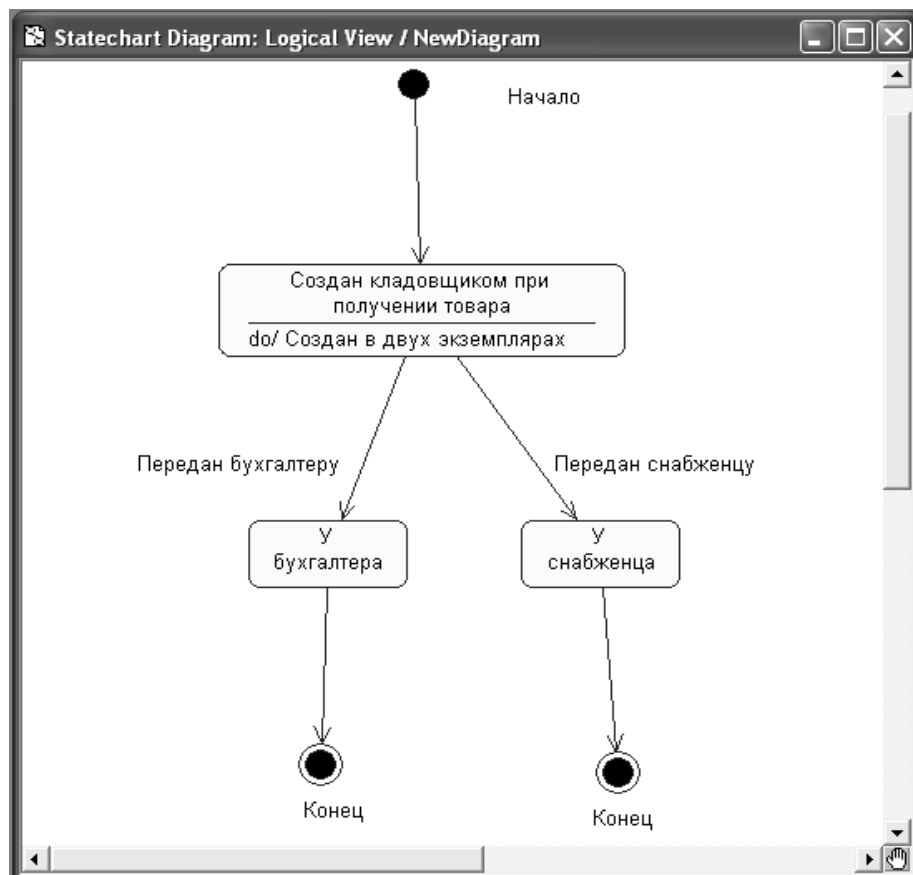









Рисунок 6.1 – Диаграмма Statechart в Rational Rose




Для повышения информативности состояниям и событиям, переводящим объекты из одних состояний в другие, присваиваются имена. С состоянием объекта могут быть связаны события и действия. Разница между ними заключается в том, что действие осуществляется самим классом, для которого строится диаграмма *Statechart*, то есть вызывается метод данного класса, а посылка сообщения направлена на объект другого класса, чей метод вызывается при помощи сообщения. Для того чтобы получить доступ к действиям и событиям, нужно через контекстное меню объекта перейти в окно его спецификации, где выбрать вкладку *Actions*, а в ней – кнопку *Insert*. На рис. 6.1 состояние «Создан кладовщиком при получении товара» связано с действием «Создан в двух экземплярах». Построение диаграммы завершается добавлением на нее значка  – End State, который отражает окончание работы. Направление перехода может быть установлено только в End State. Однако нет ограничений на количество самих элементов и переходов в них.

Диаграмм *Activity* является разновидностью диаграмм состояний. Главное различие между *Activity* и *Statechart* заключается в том, что первая характеризует действия, а вторая – статичные состояния. При этом *Activity* больше подходит для моделирования последовательности действия, а *Statechart* – для моделирования дискретных состояний объекта.

Для построения *Activity* доступны те же способы, что и для *Statechart*. При создании новой диаграммы состояний будет предложено выбрать из двух возможных вариантов: *Statechart Diagram* и *Activity Diagram*. После активизации *Activity* станет доступным набор ее инструментов. Как и *Statechart diagram*, *Activity* начинается значком *Start State* и завершается значком *End State*. Одноименный с диаграммой значок  обозначает выполнение определенных действий в течение жизни объекта. В отличие от , обычно обозначающего ожидание какого-либо события,  показывает непосредственное действие. Деятельности соединяются на диаграмме значком  – *State Transition* (переход состояния). Кроме этого,  показывает получение и обработку сообщения объектом. Переход состояния может происходить между *Action-Action*, *State-State*, *State-Action*, *Action-State*. Возможна установка нескольких переходов между двумя состояниями или действиями. Каждый такой переход уникален и показывает реакцию объекта на определенное сообщение. Поэтому нельзя создать несколько переходов между двумя состояниями с указанием одного и того же сообщения.

Уникальным инструментом диаграммы *Activity* является значок  – *Swimlanes* (плавательные дорожки), который позволяет моделировать последовательность действий различных объектов и связи между ними. С его помощью можно строить бизнес-процессы организации, отражая на диаграмме различные подразделения и объекты. *Swimlanes* помогают показать роли каждого участника бизнес-процесса. Для этого необходимо

переместить соответствующие значки активности или состояний в определенную часть диаграммы, отделенную от остальных *Swimlanes*.

Для построения диаграмм *Activity* имеется еще несколько важных инструментов. Значок  - *Decision* (решение), позволяет показать зависимость дальнейшей работы от внешних условий и решений. Этот инструмент аналогичен командам языка программирования *if* или *case* и может иметь больше двух выходов, но обычно используют выбор из двух переходов, определенных булевыми выражением. Значки  и  – *Synchronizations* (синхронизация) позволяют определить независимо выполняемые действия. При этом действия разделяются на несколько выполняемых независимо, и только по завершении всех действий объект продолжает работу.

6.2 Построение диаграмм *Statechart* и *Activity* в *Rational XDE*

Согласно теории конечных автоматов, любую сложную машину можно разложить на простые автоматы, имеющие определенные состояния, поэтому в объектно-ориентированных программных системах этот подход действительно оправдан. Кроме моделирования поведения самих объектов, диаграмма состояний может применяться для конкретизации прецедентов, что отражает взгляд на поведение объектов со стороны. Будем использовать эту диаграмму для описания состояний приложения в целом. Согласно концепции создания приложения, в *.NET* любая программа должна иметь главный объект, следовательно, выбранный подход не противоречит правилам и стилю разработки интернет приложений при помощи *.NET Framework*.

Хотя назначение диаграммы *Statechart* и принципы ее построения не изменились в *Rational XDE* по сравнению с *Rational Rose*, диаграмма состояний в *Rational XDE* дополнилась новыми возможностями.

Для создания диаграммы необходимо из контекстного меню модели выбрать пункт *Add=> Statechart*. *Toolbox* для нее показан на рис. 6.2.



Рисунок 6.2 – Toolbox для *Statechart*

Поскольку этот тип диаграммы отображает состояние объектов, то на ней должно явно отображаться место, где происходит инициализация или создание объекта и начало его работы. Значок *Initial State* позволяет обозначить событие, которое переводит объект в первое состояние на диаграмме. Это будет обозначением начала работы виртуального магазина. Для *Web*-приложения обычным началом работы является активизация по запросу пользователя начальной страницы. Таким образом, магазин доступен через интернет в любое время, а система начинает работать только по запросу пользователя, и после того, как последний пользователь покидает виртуальный магазин, работа заканчивается. После запроса браузером пользователя начальной страницы система переходит в первое состояние: ожидание дальнейших команд пользователя. После начала работы система переходит в состояние ожидания выбора пользователя. Поэтому с помощью элемента *State* создается одноименное состояние и связывается стрелкой *Transition* с начальным состоянием. Значок *State* позволяет отразить на диаграмме состояние или ситуацию в течение времени жизни программного объекта, которая отвечает некоторому положению объекта или ожиданию им внешнего события.

На диаграмме создан элемент *Final State* (завершающее состояние) и соединим с элементом состояния. Завершение работы означает, что все внутренние процессы, входящие в состояние, должны быть завершены. Предположим, что переход в финальное состояние происходит по

следующему условию: если пользователь в течение десяти минут не предпринимал никаких действий, работа приложения завершается. Чтобы отразить это на диаграмме, необходимо выполнить следующие действия:

1. Из контекстного меню элемента *Ожидание выбора пользователя* выбрать пункт *Collections*.
2. Перейти во вкладку *Outgoing Transition*.
3. Найти строку перехода, которая отмечена значком *Final State*.
4. Ввести в поле *Guard Condition* строку *StateTime > 10 min*.
5. Нажать кнопку *Close* для сохранения изменений.

Рассмотрим посылку сообщений на диаграмме *Statechart*. Для этого добавлено состояние *Регистрация нового пользователя*, которое соединено стрелкой перехода с состоянием *Ожидание выбора пользователя*. Затем из контекстного меню перехода активизирован пункт *Collection*, из него выполнен переход во вкладку *Triggers*, и в поле *Name* введено название сообщения *OnUserRegister*. Затем создано состояние *Просмотр каталога* и соединено с состоянием *Ожидание выбора пользователя*. Для состояния *Просмотр каталога* выбран элемент *SelfTransition* (переход на себя), затем в окне *Collection* для элемента *SelfTransition* задано имя сообщения *OnChangeView*. На рис. 6.3 приведен фрагмент диаграммы *Statechart*.

Перечислим назначение остальных инструментов диаграммы *Statechart*.

Shallow History (неглубокая история) позволяет создавать значок, показывающий, что данное состояние должно отслеживать историю переходов.

Deep History (неглубокая история) похож на предыдущий, однако показывает, что необходимо восстановить последнее состояние любого уровня вложенности, а не последнего, как *Shallow History*.

Submachine State (вложенное состояние) создает элемент, показывающий вложенные состояния.

Stub State (состояние-заглушка) создает элемент, отражающий наличие скрытых вложенных состояний, в которые направлен переход.

Concurrent State (параллельные состояния) создает элемент, отражающий параллельные состояния. По умолчанию в элементе создаются две области, в которых можно задавать состояния.

Junction Point (точка объединения) обозначает на диаграмме точку, в которой соединяются несколько переходов из различных состояний.

Choice Point (точка выбора) показывает на диаграмме точку, из которой могут возникнуть несколько переходов в различные состояния.

Synchronizaiton (синхронизация) показывает на диаграмме точку синхронизации, в которой соединяются несколько переходов, и дальнейшего перехода не происходит, пока не завершатся все входящие состояния.

Synch State (состояние синхронизации) показывает на диаграмме точку синхронизации между двумя параллельными процессами.

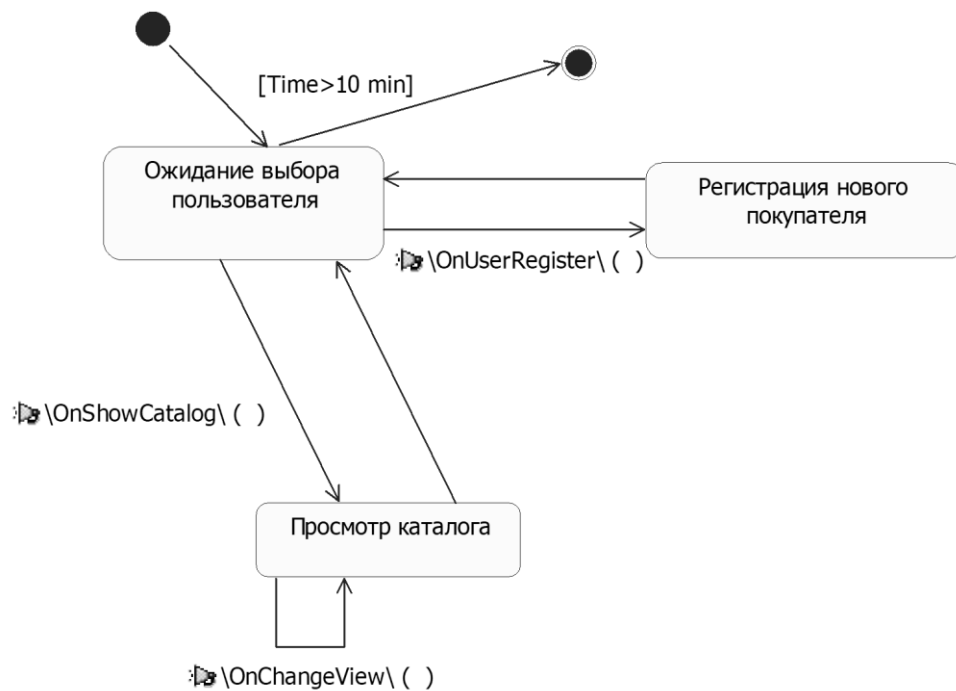


Рисунок 6.3 – Фрагмент диаграммы Statechart

Здесь диаграмму *Activity* также можно считать разновидностью диаграммы состояний. Однако в отличие от диаграммы *Statechart*, описывающей состояния объекта и переходы между ними, стандартным применением диаграммы деятельности является моделирование шагов какой-либо процедуры.

Диаграммы деятельности могут создаваться на всех стадиях разработки ПО. Перед началом работ с их помощью моделируются важные рабочие процессы предметной области с целью определения структуры и динамики бизнеса. На этапе обсуждения требований диаграммы деятельности используются для описания процессов и событий выявленных прецедентов. В течение фазы анализа и проектирования *Activity* помогает моделировать процесс операций объектов.

Для создания диаграммы из контекстного меню модели необходимо выбрать пункт *Add Diagram => Activity*. В модель проекта добавится *Activity Graph*, так как фактически диаграмма деятельности представляет собой граф. На рис. 6.4 приведен *Toolbox* диаграммы *Activity*.

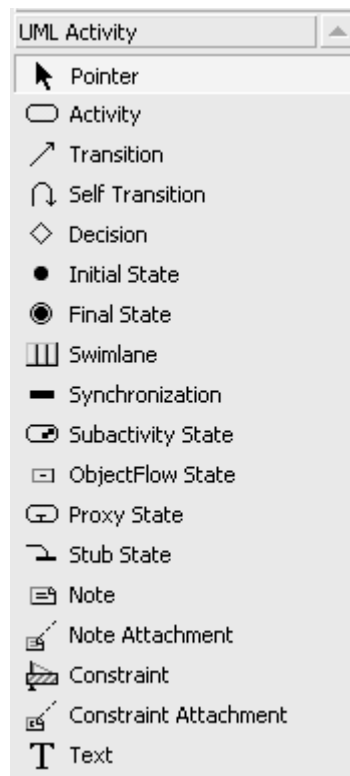


Рисунок 6.4 – Toolbox диаграммы Activity

Построение диаграммы начинается с элемента *Initial State*, который соединяется с элементом *Activity* – Запрос *бланка заказа*.

Для наглядности отображения элементов на диаграмме используется инструмент *Swimlane*. Он позволяет моделировать области ответственности исполнителей при описании процессов. На диаграмме выделены две плавательные дорожки: *Покупатель* и *Система*. В каждой из них отражаются объекты системы, выполняющие определенные действия в системе.

Значок *Synchronization* позволяет отображать момент разделения процесса. При этом действия разделяются на несколько, выполняемых независимо, и только по завершении всех действий объект продолжает работу. Для иллюстрации использования этого инструмента в модель добавлено несколько действий. После запроса покупателя система генерирует форму заказа, которая предлагается для заполнения покупателю. Он заполняет форму и передает ее системе, которая требует от покупателя подтверждения заказа по e-mail. В этот момент происходит разветвление процесса, и система ожидает подтверждения заказа, причем ожидание длится не более трех суток. Для отражения этого действию *Ожидание подтверждения* во вкладке *OutGoingTransition* в окне *Collections* установлено условие перехода *GuardCondition*. Получив подтверждение от пользователя, система изменяет статус заказа на *Подтверждено*. Здесь разделение потоков деятельности выполняется без дополнительных условий.

Для отражения разделения потоков деятельности по определенным условиям введен инструмент *Decision*. Условия перехода из него указываются в окне *Collection*. Для демонстрации использования значка

Decision на диаграмму добавлено еще несколько действий. Окончательный вариант диаграммы *Activity* приведен на рис. 6.5.

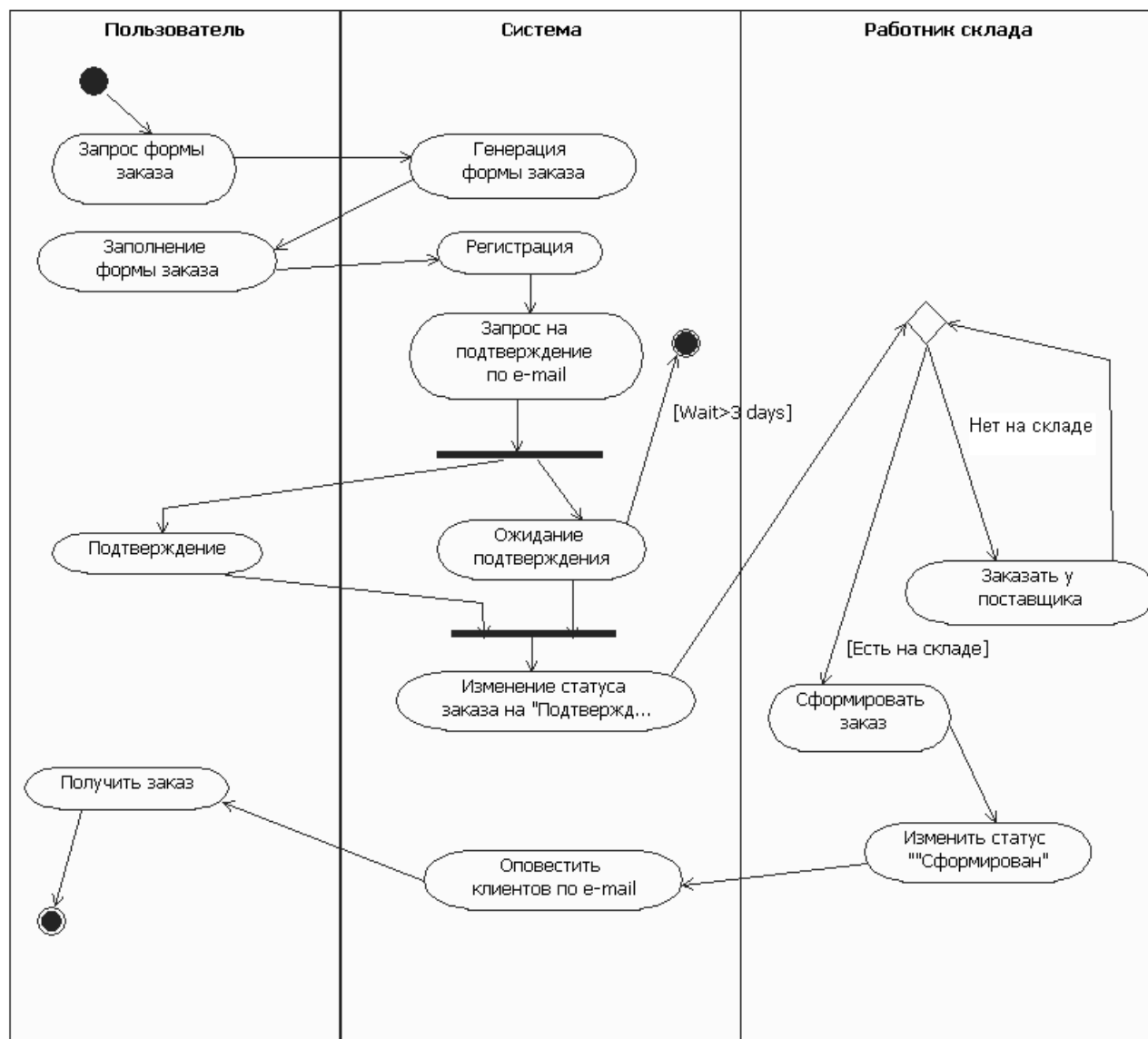


Рисунок 6.5 – Диаграмма Activity в Rational XDE

Рассмотрим инструменты *Activity*, не использованные в примере.

Элемент *Object Flow States* позволяет показать состояние объекта на диаграмме деятельности и отражает промежуточное состояние между входящим и исходящим видом деятельности. Элемент помогает моделировать процесс перевода объекта из одного состояния в другое.

Элемент *Sub Activity State* отражает на диаграмме вложенные состояния.

Инструмент *Sub State* позволяет создавать элемент, который показывает, что в данном состоянии есть скрытые вложенные состояния, в которые направлен переход.

Элемент *Proxy State* предназначен для создания элемента, который не подходит под стандарт UML. *Proxy State* необходим для отражения ссылки на элемент другой модели и помогает не копировать элементы в текущую модель в случае необходимости ссылки.