**Лекция. Унифицированный язык объектно-ориентированного моделирования UML**

Унифицированный язык моделирования UML (Unified Modeling Language) представляет собой язык для определения, представления, проектирования и документирования программных систем, организационно-экономических систем, технических систем других систем различной природы.

UML содержит стандартный набор диаграмм для моделирования:

* диаграммы вариантов использования (use case diagrams);
* диаграммы классов (class diagrams);
* диаграммы поведения системы (behavior diagrams):
  + диаграммы взаимодействия (interaction diagrams);

диаграммы последовательности (sequence diagrams)

кооперативные диаграммы (collaboration diagrams);

* + диаграммы состояний (statechart diagrams);
  + диаграммы деятельностей (activity diagrams);
* диаграммы реализации (implementation diagrams):
  + диаграммы компонентов (component diagrams);
  + диаграммы размещения (deployment diagrams).

**1.2.1 Диаграммы вариантов использования**

Визуальное моделирование с использованием нотации UML можно представить как процесс поуровневого спуска от наиболее общей и абстрактной концептуальной модели исходной бизнес-системы к логической, а затем и к физической модели соответствующей программной системы. Для достижения этих целей вначале строится модель в форме так называемой диаграммы вариантов использования (use case diagram), которая описывает функциональное назначение системы или, другими словами, то, что бизнес-система должна делать в процессе своего функционирования.

Диаграмма вариантов использования (use case diagram) – диаграмма, на которой изображаются отношения между актерами и вариантами использования.

Диаграмма вариантов использования - это исходное концептуальное представление или концептуальная модель системы в процессе ее проектирования и разработки. Создание диаграммы вариантов использования имеет следующие цели:

- определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы;

* сформулировать общие требования к функциональному поведению проектируемой системы;
* разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;
* подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Назначение данной диаграммы состоит в следующем: проектируемая программная система представляется в форме так называемых вариантов использования, с которыми взаимодействуют внешние сущности или актеры. При этом актером или действующим лицом называется любой объект, субъект или система, взаимодействующая с моделируемой бизнес-системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая служит источником воздействия на моделируемую систему так, как определит разработчик. Вариант использования служит для описания сервисов, которые система предоставляет актеру. Другими словами каждый вариант использования определяет набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится

* том, каким образом будет реализовано взаимодействие актеров с системой и собственно выполнение вариантов использования.

Базовыми элементами диаграммы вариантов использования являются вариант использования и актер.

**Вариант использования** (use case) –внешняя спецификация последовательности действий, которые система или другая сущность могут выполнять в процессе взаимодействия с актерами. Вариант использования описывает типичное взаимодействие между пользователем и системой. В простейшем случае вариант использования определяется в процессе обсуждения с пользователем тех функций, которые он хотел бы реализовать

**Актер** (actor) -представляют собой роли,а не конкретных людей или наименованияработ. Актер может также быть внешней системой, которой необходима информация от данной системы. Показывать на диаграмме действующих лиц следует только в том случае, когда им действительно необходимы некоторые варианты использования.

* + языке UML имеется несколько стандартных видов отношений между актерами и вариантами использования:
* ассоциации (association relationship) – **Рис. 1**;
* включения (include relationship) – **Рис. 2**;
* расширения (extend relationship) – **Рис. 3**
* обобщения (generalization relationship) – **Рис. 4**.

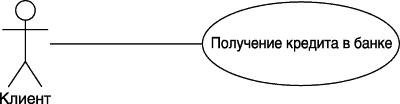


Рис. 1. Пример графического представления отношения ассоциации

между актером и вариантом использования

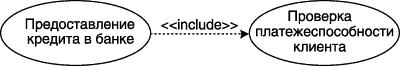


Рис. 2. Пример графического изображения отношения

включения между вариантами использования

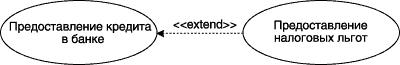


Рис. 3. Пример графического изображения отношения

расширения между вариантами использования

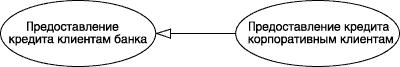


Рис. 4. Пример графического изображения отношения

обобщения между вариантами использования

**1.2.2 Диаграммы классов**

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. Диаграмма классов состоит из множества элементов, которые в совокупности отражают декларативные знания о предметной области. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами. На данной диаграмме не указывается информация о временных аспектах функционирования системы.



Рис. 5. Варианты графического изображения класса на диаграмме классов

Одним из несомненных достоинств языка UML является наличие механизмов расширения, которые позволяют ввести в рассмотрение дополнительные графические обозначения, ориентированные для решения задач из определенной предметной области. Язык UML содержит два специальных расширения: профиль для процесса разработки программного обеспечения (The UML Profile for Software Development Processes) и профиль для бизнес-моделирования (The UML Profile for Business Modeling).

* рамках первого из них предложено три специальных графических примитива, которые могут быть использованы для уточнения семантики отдельных классов при построении различных диаграмм (**Рис. 6**):

- Управляющий класс (control class) – класс, отвечающий за координацию действий других классов. На каждой диаграмме классов должен быть хотя бы один управляющий класс, причем количество посылаемых объектам управляющего класса сообщений мало, по сравнению с числом рассылаемых ими. Управляющий класс от-вечает за координацию действий других классов. У каждой диаграммы классов должен быть хотя бы один управляющий класс, контролирующий последовательность выполнения действий этого варианта использования. Как правило, данный класс является активным и инициирует рассылку множества сообщений другим классам модели. Кроме специального обозначения управляющий класс может быть изображен в форме прямоугольника класса со стереотипом <<control>>.

- Класс-сущность (entity class) – пассивный класс, информация о котором должна храниться постоянно и не уничтожаться с выключением системы. Класс-сущность содержит информацию, которая должна храниться постоянно и не уничтожается с уничтожением объектов данного класса или прекращением работы моделируемой системы, связанные с выключением системы или завершением программы. Как правило, этот класс соответствует отдельной таблице базы данных. В этом случае его атрибуты являются полями таблицы, а операции – присоединенными или хранимыми процедурами. Этот класс пассивный и лишь принимает сообщения от других классов модели. Класс-сущность может быть изображен также стандартным образом в форме прямоугольника класса со стереотипом <<entity>> .

- Граничный класс (boundary class) – класс, который располагается на границе системы с внешней средой и непосредственно взаимодействует с актерами, но является составной частью системы. Граничный класс может быть изображен также стандартным образом в форме прямоугольника класса со стереотипом <<boundary>>.

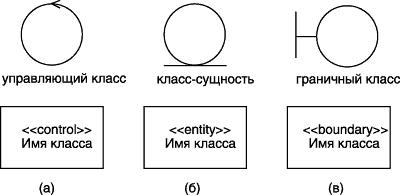


Рис. 6. Графическое изображение классов для моделирования программного обеспечения

* рамках второго профиля также предложено три специальных графических примитива, которые могут быть использованы для уточнения семантики отдельных классов при построении моделей бизнес-систем:

- Сотрудник (business worker) – класс, служащий на диаграмме классов для представления любого сотрудника, который является элементом бизнес-системы и взаимо-действует с другими сотрудниками при реализации бизнес-процесса. Этот класс также может быть изображен в форме прямоугольника класса со стереотипом

<<worker>> или <<internalWorker>>.

- Сотрудник для связи с окружением (caseworker) – класс, служащий для представления в бизнес-системе такого сотрудника, который, являясь элементом бизнес-системы, непосредственно взаимодействует с актерами (бизнес-актерами) при реализации бизнес-процесса. Этот класс также может быть изображен в форме прямоугольника класса со стереотипом <<caseWorker>>.

* Бизнес-сущность (business entity) – специальный случай класса-сущности, который также не инициирует никаких сообщений. Этот класс служит для сохранения информации о результатах выполнения бизнес-процесса в моделируемой бизнес-системе или организации. Этот класс также может быть изображен в форме прямоугольника класса со стереотипом <<business entity>>.

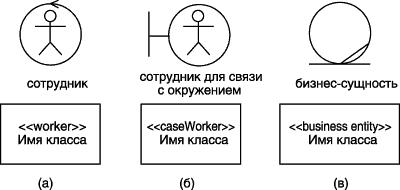


Рис. 7. Графическое изображение классов для моделирования бизнес-систем

Базовые отношения, изображаемые на диаграммах классов:

* Отношение ассоциации (association relationship) – **Рис. 8**;
* Отношение обобщения (generalization relationship) – **Рис. 9**;
* Отношение агрегации (aggregation relationship) – **Рис. 10**;
* Отношение композиции (composition relationship) – **Рис. 11**.



Рис. 8. Графическое изображение направленной бинарной ассоциации между классами



Рис. 9. Графическое изображение отношения обобщения



Рис. 10. Графическое изображение отношения агрегации



Рис. 11. Графическое изображение отношения композиции

1.2.3 Диаграммы состояний

Диаграммы состояний определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате наступления некоторых событий. Таким образом, диаграмма состояний используется для моделирования поведения объектов системы при переходе из одного состояния в другое.

Состояние может быть задано в виде набора конкретных значений атрибутов объекта некоторого класса, при этом изменение отдельных значений этих атрибутов будет отражать изменение состояния моделируемого объекта или системы в целом. Однако не каждый атрибут класса может характеризовать состояние его объектов. Как правило, имеют значение только те свойства элементов системы, которые отражают динамический или функциональный аспект ее поведения. В этом случае состояние будет характеризоваться некоторым инвариантным условием, включающим в себя только принципиальные для поведения объекта или системы атрибуты классов и их значения.

Такое условие может соответствовать ситуации, когда моделируемый объект находится состоянии ожидания возникновения внешнего события. В то же время нахождение объекта некотором состоянии может быть связано с выполнением определенных действий. В последнем случае соответствующая деятельность начинается в момент перехода моделируемого элемента в рассматриваемое состояние, а после и элемент может покинуть данное состоя-ние в момент завершения этой деятельности.



Рис. 12. Графическое изображение состояний на диаграмме состояний

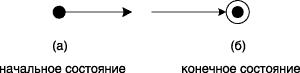


Рис. 13. Графическое изображение начального и конечного состояний

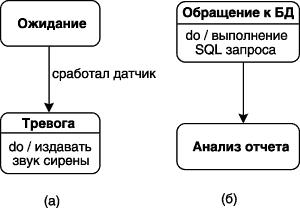


Рис. 14. Графическое изображение триггерного (а) и нетриггерного (б)

переходов на диаграмме состояний

**1.2.4 Диаграммы деятельностей**

При моделировании поведения проектируемой или анализируемой программной системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и процедурной реализации выполняемых системой операций. Для этой цели, как правило, используются блок-схемы или структурные схемы алгоритмов. Каждая такая схема акцентирует внимание на последовательности выполнения определенных процедур или элементарных операций, которые в совокупности приводят к получению желаемого результата**.**

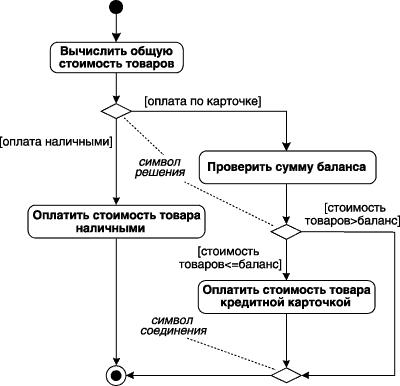
****

Рис. 15. Различные варианты ветвлений на диаграмме деятельности

1.2.5 Диаграммы взаимодействия

Диаграммы взаимодействия описывают поведение взаимодействующих групп объектов. Как правило, диаграмма взаимодействия охватывает поведение объектов в рамках только одного варианта использования. На такой диаграмме отображаются ряд объектов и те сообщения, которыми они обмениваются между собой.

Для моделирования взаимодействия объектов в языке UML используются соответствующие диаграммы взаимодействия. Говоря об этих диаграммах, имеют в виду два аспекта взаимодействия. *Во-первых*, взаимодействия объектов можно рассматривать во времени, и тогда для представления временных особенностей передачи и приема сообщений между объектами используется диаграмма последовательности.

*Во-вторых*,можно рассматривать структурные особенности взаимодействия объектов.Для представления структурных особенностей передачи и приема сообщений между объектами используется диаграмма кооперации.

**Диаграммы последовательности**

Диаграмма последовательности (sequence diagram) - диаграмма, на которой показаны взаимодействия объектов, упорядоченные по времени их проявления.

На диаграмме последовательности неявно присутствует ось времени, что позволяет визуализировать временные отношения между передаваемыми сообщениями. С помощью диаграммы последовательности можно представить взаимодействие элементов модели как своеобразный временной график "жизни" всей совокупности объектов, связанных между собой для реализации варианта использования программной системы, достижения бизнес-цели или выполнения какой-либо задачи.

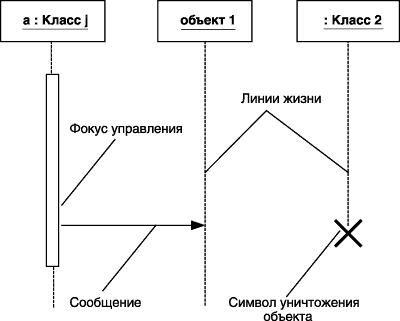


Рис. 16. Графические элементы диаграммы последовательности

**Кооперативные диаграммы**

Диаграмма кооперации предназначена для описания поведения системы на уровне отдельных объектов, которые обмениваются между собой сообщениями, чтобы достичь нужной цели или реализовать некоторый вариант использования. С точки зрения аналитика или архитектора системы в проекте важно представить структурные связи отдельных объектов между собой. Такое представление структуры модели как совокупности взаимодействующих объектов и обеспечивает диаграмма кооперации.

Кооперация (collaboration) – спецификация множества объектов отдельных классов, совместно взаимодействующих с целью реализации отдельных вариантов использования в общем контексте моделируемой системы.

Понятие кооперации – одно из фундаментальных в языке UML. Цель самой кооперации состоит в том, чтобы специфицировать особенности реализации отдельных вариантов использования или наиболее значимых операций в системе. Кооперация определяет структуру поведения системы в терминах взаимодействия участников этой кооперации.

На диаграмме кооперации размещаются объекты, представляющие собой экземпляры классов, связи между ними, которые в свою очередь являются экземплярами ассоциаций и сообщения. Связи дополняются стрелками сообщений, при этом показываются только те объекты, которые участвуют в реализации моделируемой кооперации. Далее, как и на диаграмме классов, показываются структурные отношения между объектами в виде различных соединительных линий. Связи могут дополняться именами ролей, которые играют объекты в данной взаимосвязи. И, наконец, изображаются динамические взаимосвязи — потоки сооб-щений в форме стрелок с указанием направления рядом с соединительными линиями между объектами, при этом задаются имена сообщений и их порядковые номера в общей последо-вательности сообщений.

Одна и та же совокупность объектов может участвовать в реализации различных коо-пераций. В зависимости от рассматриваемой кооперации, могут изменяться как связи между отдельными объектами, так и поток сообщений между ними. Именно это отличает диаграм-му кооперации от диаграммы классов, на которой должны быть указаны все без исключения классы, их атрибуты и операции, а также все ассоциации и другие структурные отношения между элементами модели.

**1.2.6 Диаграммы компонентов**

Диаграммы компонентов показывают, как выглядит модель на физическом уровне. На них изображены компоненты программного обеспечения и связи между ними.

Физическая система (physical system) – реально существующий прототип модели системы. Для представления физических сущностей в языке UML применяется специальный термин – компонент.

**Компонент** (component) –физически существующая часть системы,которая обеспечивает реализацию классов и отношений, а также функционального поведения моделируемой программной системы. Компонент предназначен для представления физической организации ассоциированных с ним элементов модели. Дополнительно компонент может иметь текстовый стереотип и помеченные значения, а некоторые компоненты – собственное графическое представление. Компонентом может быть исполняемый код отдельного модуля, командные файлы или файлы, содержащие интерпретируемые скрипты.

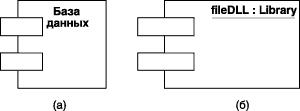


Рис. 17. Графическое изображение компонента

**1.2.7 Диаграммы размещения**

Диаграмма размещения (deployment diagram) - диаграмма, на которой представлены узлы выполнения программных компонентов реального времени, а также процессов и объектов.

Диаграмма развертывания применяется для представления общей конфигурации и топологии распределенной программной системы и содержит изображение размещения компонентов по отдельным узлам системы. Кроме того, диаграмма развертывания показывает наличие физических соединений - маршрутов передачи информации между аппаратными устройствами, задействованными в реализации системы.

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих только на этапе ее исполнения (run-time). При этом представляются только те компоненты программы, которые являются исполнимыми файлами или динамическими библиотеками. Компоненты, не используемые на этапе исполнения, на диаграмме развертывания не показываются. Так, компоненты с исходными текстами программ могут присутствовать только на диаграмме компонентов. На диаграмме развертывания они не указываются.

Диаграмма развертывания содержит графические изображения процессоров, устройств, процессов и связей между ними. В отличие от диаграмм логического представления, диа-грамма развертывания является единственной для системы в целом, поскольку должна отражать все особенности ее реализации. Эта диаграмма, по сути, завершает процесс ООАП для конкретной программной системы и ее разработка, как правило, последний этап специфика-ции модели.



Рис. 18. Варианты изображения графических стереотипов узлов