Савва Юрий Болеславович

Проектирование распределенных информационных систем

Лекция 3.
Введение в язык UML

Значение моделирования

- Неудачные проекты заканчиваются крахом в силу самых разных причин, а вот успешные, как правило, имеют много общего. Хотя успех программного проекта обеспечивается множеством разных слагаемых, одним из общих является применение моделирования.
- Моделирование - это устоявшаяся и повсеместно принятая инженерная методика.

- Итак, что же такое модель? Попросту говоря, она является *упрощенным представлением реальности*.
- Модель - это чертеж системы: в нее может входить как детальный план, так и более абстрактное представление системы "с высоты птичьего полета".
- Хорошая модель всегда включает элементы, существенно влияющие на результат, и не включает те, которые малозначимы на данном уровне абстракции.
- Каждая система может быть описана с разных точек зрения, для чего используются различные модели, каждая из которых, следовательно, является семантически замкнутой абстракцией системы.
- Модель может быть структурной, подчеркивающей организацию системы, или поведенческой, то есть отражающей ее динамику.

Мы *строим модели для того, чтобы лучше понимать разрабатываемую систему*. Моделирование позволяет решить четыре различных задачи:

1. визуализировать систему в ее текущем или желательном для нас состоянии;
2. определить структуру или поведение системы;
3. получить шаблон, позволяющий затем сконструировать систему;
4. документировать принимаемые решения, используя полученные модели.

- Моделирование предназначено не только для создания больших систем. Даже программный эквивалент собачьей конуры выиграет от этого процесса. Чем больше и сложнее система, тем большее значение приобретает моделирование при ее разработке.
- Дело в том, что *моделировать сложную систему необходимо, поскольку иначе мы не можем воспринять ее как единое целое*.

- Восприятие человеком сложных сущностей ограничено. Моделируя, мы сужаем проблему, заостряя внимание в данный момент только на одном аспекте.
- В сущности, этот подход есть не что иное, как принцип "разделяй и властвуй", который Эдсгер Дейкстра провозгласил много лет назад: сложную задачу легче решить, если разделить ее на несколько меньших.
- Кроме того, моделирование усиливает возможности человеческого интеллекта, поскольку правильно выбранная модель дает возможность создавать проекты на более высоких уровнях абстракции.

- Сказать, что моделирование имеет смысл, еще не означает, что оно абсолютно необходимо.
- И действительно, многие исследования показывают, что в большинстве компаний, разрабатывающих программное обеспечение, моделирование применяют редко или вовсе не применяют. Чем проще проект, тем менее вероятно, что в нем будет использовано формальное моделирование.
- Ключевое слово здесь - "формальное".

Принципы моделирования

- Моделирование имеет богатую историю во всех инженерных дисциплинах. Длительный опыт его использования позволил сформулировать четыре основных принципа.
- Во-первых, *выбор модели оказывает определяющее влияние на подход к решению проблемы и на то, как будет выглядеть это решение*. Иначе говоря, подходите к выбору модели вдумчиво. Правильно выбранная модель высветит самые коварные проблемы разработки и позволит проникнуть в самую суть задачи, что при ином подходе было бы попросту невозможно. Неправильная модель заведет вас в тупик, поскольку внимание будет заостряться на несущественных вопросах.

- Второй принцип формулируется так: *каждая модель может быть воплощена с разной степенью абстракции*.

Иногда простая и быстро созданная модель пользовательского интерфейса - самый подходящий вариант. В других случаях приходится работать на уровне битов, например когда вы специфицируете межсистемные интерфейсы или боретесь с узкими местами в сети. В любом случае лучшей моделью будет та, которая позволяет выбрать уровень детализации в зависимости от того, кто и с какой целью на нее смотрит. Для аналитика или конечного пользователя наибольший интерес представляет вопрос "что", а для разработчика - вопрос "как".

В обоих случаях необходима возможность рассматривать систему на разных уровнях детализации в разное время.

- Третий принцип: *лучшие модели - те, что ближе к реальности*.

Говоря о программном обеспечении, можно сказать, что "ахиллесова пята" структурного анализа - несоответствие принятой в нем модели и модели системного проекта. Если этот разрыв не будет устранен, то поведение созданной системы с течением времени начнет все больше отличаться от задуманного.

При объектно-ориентированном подходе можно объединить все почти независимые представления системы в единое семантическое целое.

- Четвертый принцип заключается в том, что *нельзя ограничиваться созданием только одной модели.Наилучший подход при разработке любой нетривиальной системы - использовать совокупность нескольких моделей, почти независимых друг от друга*.

Ключевым определением здесь является ***"почти независимые".*** В данном контексте оно означает, что модели могут создаваться и изучаться по отдельности, но вместе с тем остаются взаимосвязанными. Например, можно изучать только схемы электропроводки проектируемого здания, но при этом наложить их на поэтажный план пола и даже рассмотреть совместно с прокладкой труб на схеме водоснабжения.

Такой подход верен и в отношении объектно-ориентированных программных систем. Для понимания архитектуры подобной системы требуется несколько взаимодополняющих видов:

- Вид с точки зрения прецедентов, или вариантов использования (чтобы выявить требования к системе),
- Вид с точки зрения проектирования (чтобы построить словарь предметной области и области решения),
- Вид с точки зрения процессов (чтобы смоделировать распределение процессов и потоков в системе),
- Вид с точки зрения реализации, позволяющий рассмотреть физическую реализацию системы, и вид с точки зрения развертывания, помогающий сосредоточиться на вопросах системного проектирования.

Каждый из перечисленных видов имеет множество структурных и поведенческих аспектов, которые в своей совокупности составляют детальный чертеж программной системы.

В зависимости от природы системы некоторые модели могут быть важнее других.

Так, при создании систем для обработки больших объемов данных более важны модели, обращающиеся к точке зрения статического проектирования.

В приложениях, ориентированных на интерактивную работу пользователя, на первый план выходят представления с точки зрения статических и динамических прецедентов.

В системах реального времени наиболее существенными будут представления с точки зрения динамических процессов.

Наконец, в распределенных системах, таких как Web-приложения, основное внимание нужно уделять моделям реализации и развертывания.

Объектное моделирование

- Инженеры-строители создают огромное количество моделей. Чаще всего это ***структурные*** модели, позволяющие визуализировать и специфицировать части системы и то, как они соотносятся друг с другом. Иногда, в особо критичных случаях, создаются также ***динамические*** модели - например, если надо изучить поведение конструкции при землетрясении. Эти два типа различаются по организации и по тому, на что в первую очередь обращается внимание при проектировании.
- При разработке программного обеспечения тоже существует несколько подходов к моделированию. Важнейшие из них - **алгоритмический** и **объектно-ориентированный.**

- ***Алгоритмический*** метод представляет традиционный подход к созданию программного обеспечения. Основным строительным блоком является процедура или функция, а внимание уделяется прежде всего вопросам передачи управления и декомпозиции больших алгоритмов на меньшие. Ничего плохого в этом нет, если не считать того, что системы не слишком легко адаптируются. При изменении требований или увеличении размера приложения (что происходит нередко) сопровождать их становится сложнее.

- Наиболее современным подходом к разработке программного обеспечения является ***объектно-ориентированный***. Здесь в качестве основного строительного блока выступает *объект* или *класс*. В самом общем смысле ***объект - это сущность, обычно извлекаемая из словаря предметной области или решения***, а ***класс является описанием множества однотипных объектов.***
- Каждый объект обладает идентичностью (его можно поименовать или как-то по-другому отличить от прочих объектов), состоянием (обычно с объектом бывают связаны некоторые данные) и поведением (с ним можно что-то делать или он сам может что-то делать с другими объектами).

Объектно-ориентированный подход к разработке программного обеспечения является сейчас преобладающим просто потому, что он продемонстрировал свою полезность при построении систем в самых разных областях любого размера и сложности. Кроме того, большинство современных языков программирования, инструментальных средств и операционных систем являются в той или иной мере объектно-ориентированными, а это дает веские основания судить о мире в терминах объектов. Объектно-ориентированные методы разработки легли в основу идеологии сборки систем из отдельных компонентов; в качестве примера можно назвать такие технологии, как JavaBeans и CQM+.

Визуализация, специфицирование, конструирование и документирование объектно-ориентированных систем - это и есть назначение языка UML.

Концептуальная модель UML

Для понимания UML необходимо усвоить его концептуальную модель, которая включает в себя три составные части:

1. основные строительные блоки языка,
2. правила их сочетания;
3. некоторые общие для всего языка механизмы.

Строительные блоки UML

- Словарь языка UML включает три вида строительных блоков:

1. сущности
2. отношения;
3. диаграммы.

- Сущности - это абстракции, являющиеся основными элементами модели.
- Отношения связывают различные сущности.
- Диаграммы группируют представляющие интерес совокупности сущностей.

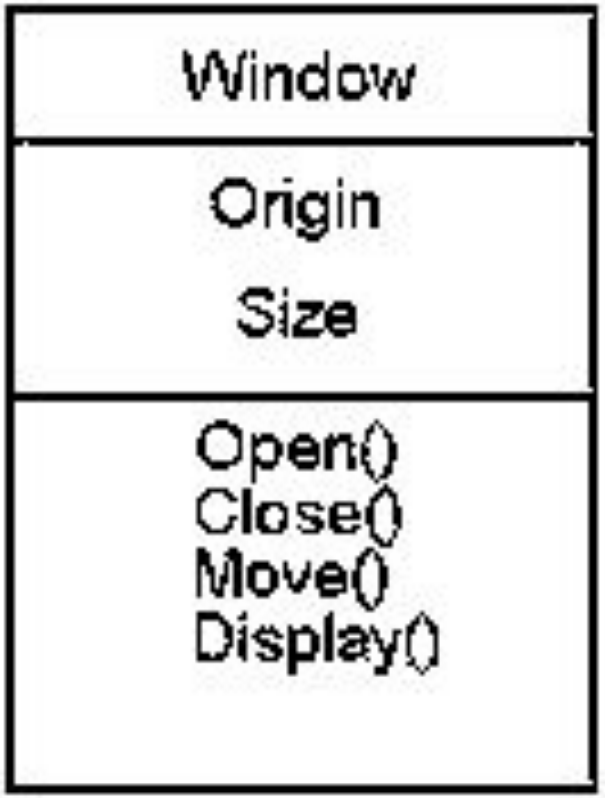
- В UML имеется четыре типа сущностей:
- структурные;
- поведенческие;
- группирующие;
- аннотационные.

Сущности являются основными объектно-ориентированными блоками языка. С их помощью можно создавать корректные модели.

- Структурные сущности - это имена существительные в моделях на языке UML.
- Как правило, они представляют собой статические части модели, соответствующие концептуальным или физическим элементам системы. Существует семь разновидностей структурных сущностей.

Класс (Class) - это описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой.

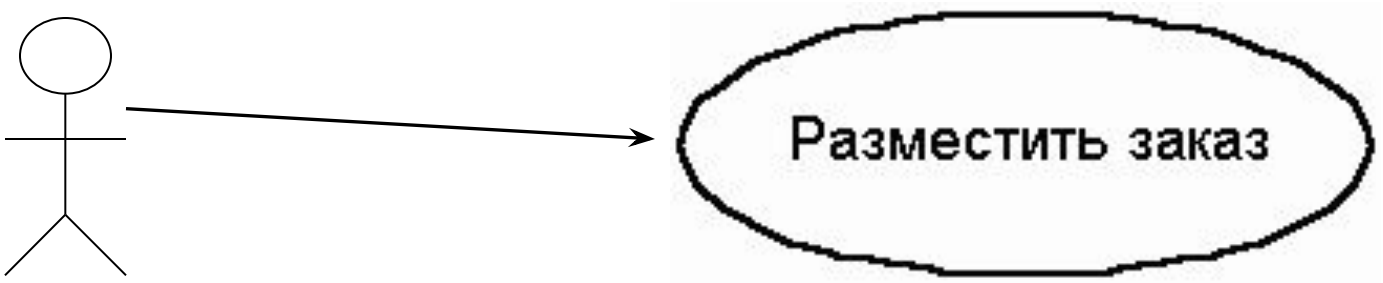
Класс реализует один или несколько интерфейсов. Графически класс изображается в виде прямоугольника, в котором обычно записаны его имя, атрибуты и операции, как показано на рисунке.



- Интерфейс (Interface) - это совокупность операций, которые определяют сервис (набор услуг), предоставляемый классом или компонентом.
- Таким образом, интерфейс описывает видимое извне поведение элемента.
- Интерфейс может представлять поведение класса или компонента полностью или частично; он определяет только спецификации операций (сигнатуры), но никогда - их реализации.
- Графически интерфейс изображается в виде круга, под которым пишется его имя, как показано на рисунке.
- Интерфейс редко существует сам по себе - обычно он присоединяется к реализующему его классу или компоненту.

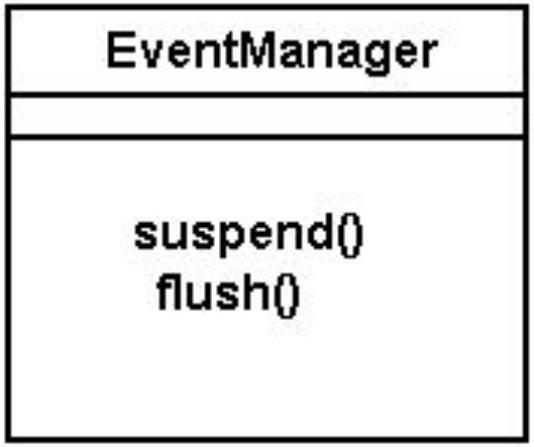
- Кооперация (Collaboration) определяет взаимодействие;
- она представляет собой совокупность ролей и других элементов, которые, работая совместно, производят некоторый кооперативный эффект, не сводящийся к простой сумме слагаемых.
- Кооперация, следовательно, имеет как структурный, так и поведенческий аспект.
- Один и тот же класс может принимать участие в нескольких кооперациях; таким образом, они являются реализацией образцов поведения, формирующих систему.
- Графически кооперация изображается в виде эллипса, ограниченного пунктирной линией, в который обычно заключено только имя, как показано на рисунке:

- Прецедент (Use case) - это описание последовательности выполняемых системой действий, которая производит наблюдаемый результат, значимый для какого-то определенного актера (Actor).
- Прецедент применяется для структурирования поведенческих сущностей модели.
- Прецеденты реализуются посредством кооперации.
- Графически прецедент изображается в виде ограниченного непрерывной линией эллипса, обычно содержащего только его имя как показано на рисунке.



- Три другие сущности - активные классы, компоненты и узлы - подобны классам: они описывают совокупности объектов с общими атрибутами, операциями, отношениями и семантикой.
- Тем не менее они в достаточной степени отличаются друг от друга и от классов и, учитывая их важность при моделировании определенных аспектов объектно-ориентированных систем, заслуживают специального рассмотрения.

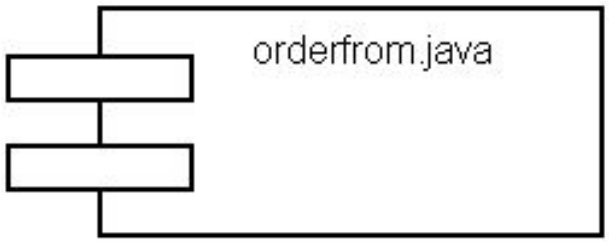
- Активным классом (Active class) называется класс, объекты которого вовлечены в один или несколько процессов, или нитей (Threads), и поэтому могут инициировать управляющее воздействие.
- Активный класс во всем подобен обычному классу, за исключением того, что его объекты представляют собой элементы, деятельность которых осуществляется одновременно с деятельностью других элементов.
- Графически активный класс изображается так же, как простой класс, но ограничивающий прямоугольник рисуется жирной линией и обычно включает имя, атрибуты и операции, как показано на рисунке.



Два оставшихся элемента - компоненты и узлы - также имеют свои особенности. Они соответствуют физическим сущностям системы, в то время как пять предыдущих - концептуальным и логическим сущностям.

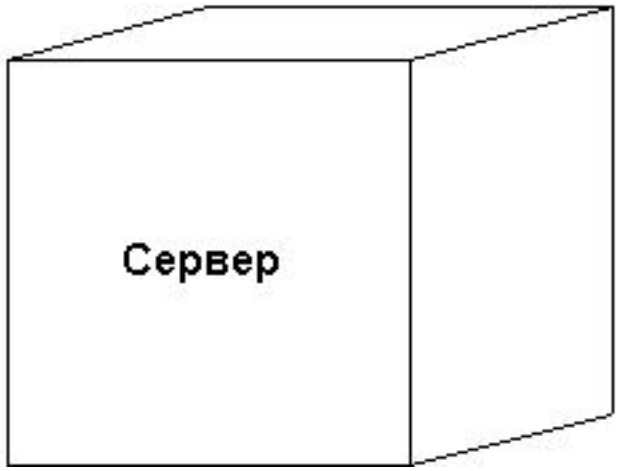
- Компонент (Component) - это физическая заменяемая часть системы, которая соответствует некоторому набору интерфейсов и обеспечивает его реализацию.
- В системе можно встретить различные виды устанавливаемых компонентов, такие как COM+ или Java Beans, а также компоненты, являющиеся артефактами процесса разработки, например файлы исходного кода.
- Компонент, как правило, представляет собой физическую упаковку логических элементов, таких как классы, интерфейсы и кооперации.

•Графически компонент изображается в виде прямоугольника с вкладками, содержащего обычно только имя, как показано на рисунке.



- Узел (Node) - это элемент реальной (физической) системы, который существует во время функционирования программного комплекса и представляет собой вычислительный ресурс, обычно обладающий как минимум некоторым объемом памяти, а часто еще и способностью обработки.
- Совокупность компонентов может размещаться в узле, а также мигрировать с одного узла на другой.

•Графически узел изображается в виде куба, обычно содержащего только имя, как показано на рисунке.



- Эти семь базовых элементов - классы, интерфейсы, кооперации, прецеденты, активные классы, компоненты узлы - являются основными структурными сущностями, которые могут быть включены в модель UML.
- Существуют также разновидности этих сущностей: актеры, сигналы, утилиты (виды классов), процессы и нити (виды активных классов), приложения, документы, файлы, библиотеки, страницы и таблицы (виды компонентов).

- Поведенческие сущности (Behavioral things) являются динамическими составляющими модели UML. Это глаголы языка: они описывают поведение модели во времени и пространстве. Существует всего два основных типа поведенческих сущностей - взаимодействия и автоматы.
- Семантически они часто бывают связаны с различными структурными элементами, в первую очередь - классами, кооперациями и объектами.

- Взаимодействие (Interaction) - это поведение, суть которого заключается в обмене сообщениями (Messages) между объектами в рамках конкретного контекста для достижения определенной цели.
- С помощью взаимодействия южно описать как отдельную операцию, так и поведение совокупности объектов. взаимодействие предполагает ряд других элементов, таких как сообщения, последовательности действий (поведение, инициированное сообщением) и связи (между объектами).
- Графически сообщения изображаются в виде стрелки, над которой почти всегда пишется имя соответствующей операции, как показано на рисунке.

- Автомат (State machine) - это алгоритм поведения, определяющий последовательность состояний, через которые объект или взаимодействие проходят на протяжении своего жизненного цикла в ответ на различные события, а также реакции на эти события.
- С помощью автомата можно описать поведение сдельного класса или кооперации классов.
- С автоматом связан ряд других элементов: состояния, переходы (из одного состояния в другое), события (сущности, инициирующие переходы) и виды действий (реакция на переход).

- Графически состояние изображается в виде прямоугольника с закругленными углами, содержащего имя и, возможно, подсостояния.

Группирующие сущности являются организующими частями модели UML. Это блоки, на которые можно разложить модель. Есть только одна первичная группирующая сущность, а именно пакет.

- Пакеты (Packages) представляют собой универсальный механизм организации элементов в группы.
- В пакет можно поместить структурные, поведенческие и даже другие группирующие сущности.
- В отличие от компонентов, существующих во время работы программы, пакеты носят чисто концептуальный характер, то есть существуют только во время разработки.

Изображается пакет в виде папки с закладкой, содержащей, как правило, только имя и иногда - содержимое

Пакеты - это основные группирующие сущности, с помощью которых можно организовать модель UML. Существуют также вариации пакетов, например каркасы (Frameworks), модели и подсистемы.

- Аннотационные сущности - пояснительные части модели UML.
- Это комментарии для дополнительного описания, разъяснения или замечания к любому элементу модели.
- Имеется только один базовый тип аннотационных элементов - примечание (Note). Примечание - это просто символ для изображения комментариев или ограничений, присоединенных к элементу или группе элементов.
- Графически примечание изображается в виде прямоугольника с загнутым краем, содержащим текстовый или графический комментарий, как показано на рисунке.

Этот элемент является основной аннотационной сущностью, которую можно включать в модель UML.

Чаще всего примечания используются, чтобы снабдить диаграммы комментариями или ограничениями, которые можно выразить в виде неформального или формального текста.

Существуют вариации этого элемента, например требования, где описывают некое желательное поведение с точки зрения внешней по отношению к модели.

Диаграммы UML

- Диаграмма в UML - это графическое представление набора элементов, изображаемое чаще всего в виде связанного графа с вершинами (сущностями) и ребрами (отношениями). Диаграммы рисуют для визуализации системы с разных точек зрения.
- Диаграмма - в некотором смысле одна из проекций системы. Как правило, за исключением наиболее тривиальных случаев, диаграммы дают свернутое представление элементов, из которых составлена система. Один и тот же элемент может присутствовать во всех диаграммах, или только в нескольких (самый распространенный вариант), или не присутствовать ни в одной (очень редко).
- Теоретически диаграммы могут содержать любые комбинации сущностей и отношений.

- На практике, однако, применяется сравнительно небольшое количество типовых комбинаций, соответствующих пяти наиболее употребительным видам, которые составляют архитектуру программной системы.
- Таким образом, в UML выделяют девять типов диаграмм:

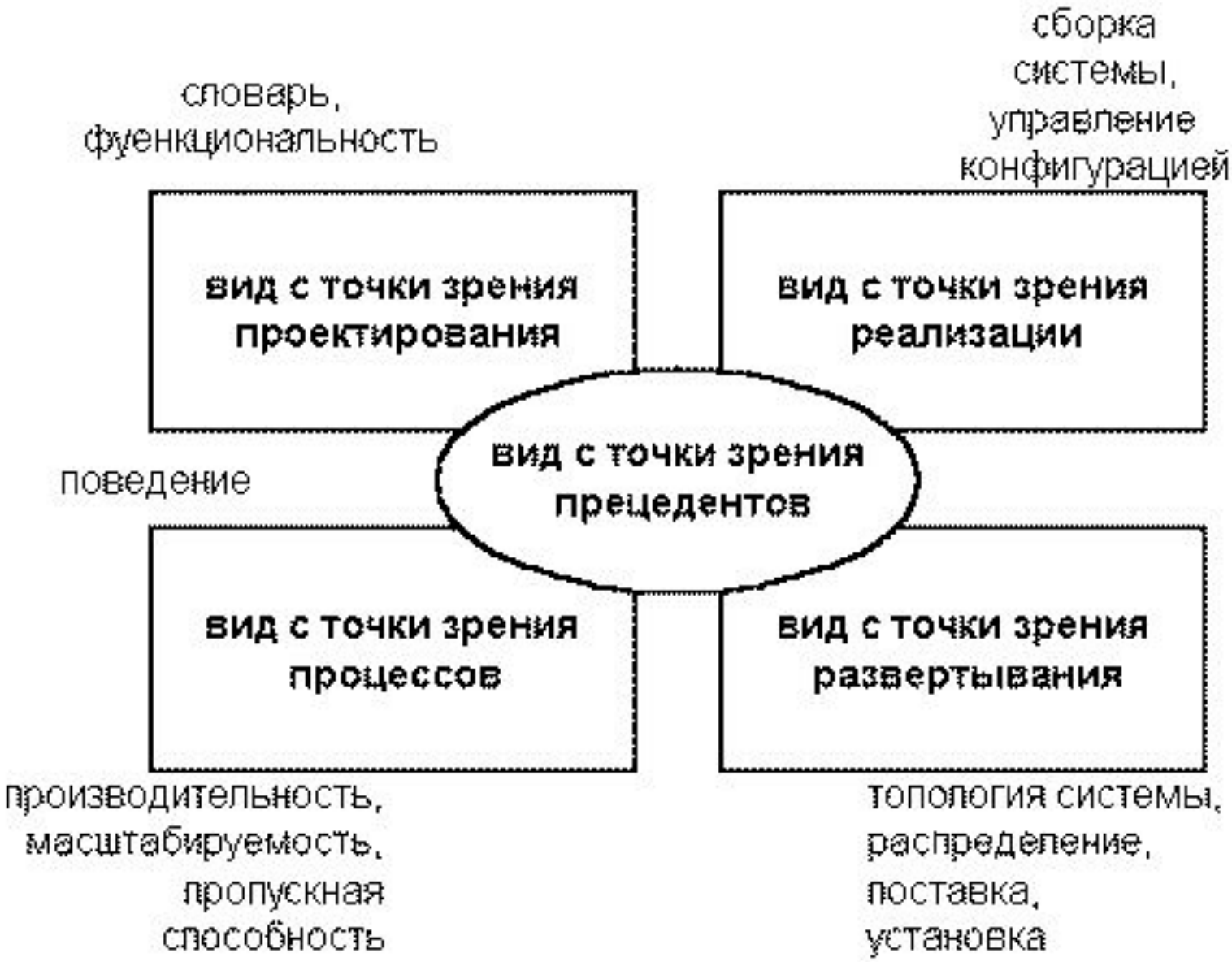
- диаграммы классов;
- диаграммы объектов;
- диаграммы прецедентов;
- диаграммы последовательностей;
- диаграммы кооперации;
- диаграммы состояний;
- диаграммы действий;
- диаграммы компонентов;
- диаграммы развертывания.

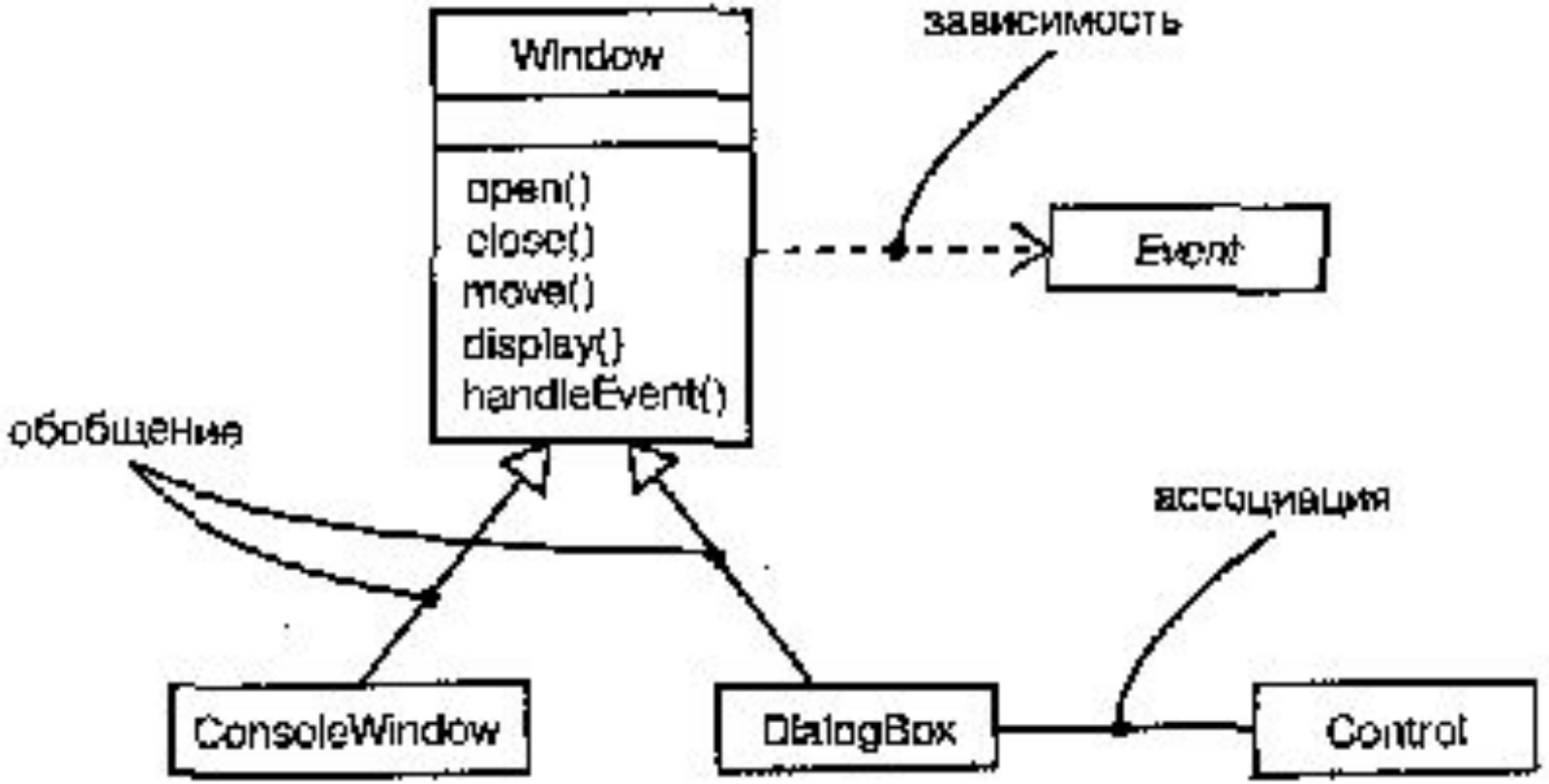
- На диаграмме классов показывают классы, интерфейсы, объекты и кооперации, а также их отношения. При моделировании объектно-ориентированных систем этот тип диаграмм используют чаще всего. Диаграммы классов соответствуют статическому виду системы с точки зрения проектирования. Диаграммы классов, которые включают активные классы, соответствуют статическому виду системы с точки зрения процессов.
- На диаграмме объектов представлены объекты и отношения между ними. Они являются статическими "фотографиями" экземпляров сущностей, показанных на диаграммах классов. Диаграммы объектов, как и диаграммы классов, относятся к статическому виду системы с точки зрения проектирования или процессов, но с расчетом на настоящую или макетную реализацию.

- На диаграмме прецедентов представлены прецеденты и актеры (частный случай классов), а также отношения между ними. Диаграммы прецедентов относятся к статическому виду системы с точки зрения прецедентов использования. Они особенно важны при организации и моделировании поведения системы.
- Диаграммы последовательностей и кооперации являются частными случаями диаграмм взаимодействия. На диаграммах взаимодействия представлены связи между объектами; показаны, в частности, сообщения, которыми объекты могут обмениваться. Диаграммы взаимодействия относятся к динамическому виду системы. При этом диаграммы последовательности отражают временную упорядоченность сообщений, а диаграммы кооперации - структурную организацию обменивающихся сообщениями объектов. Эти диаграммы являются изоморфными, то есть могут быть преобразованы друг в друга.

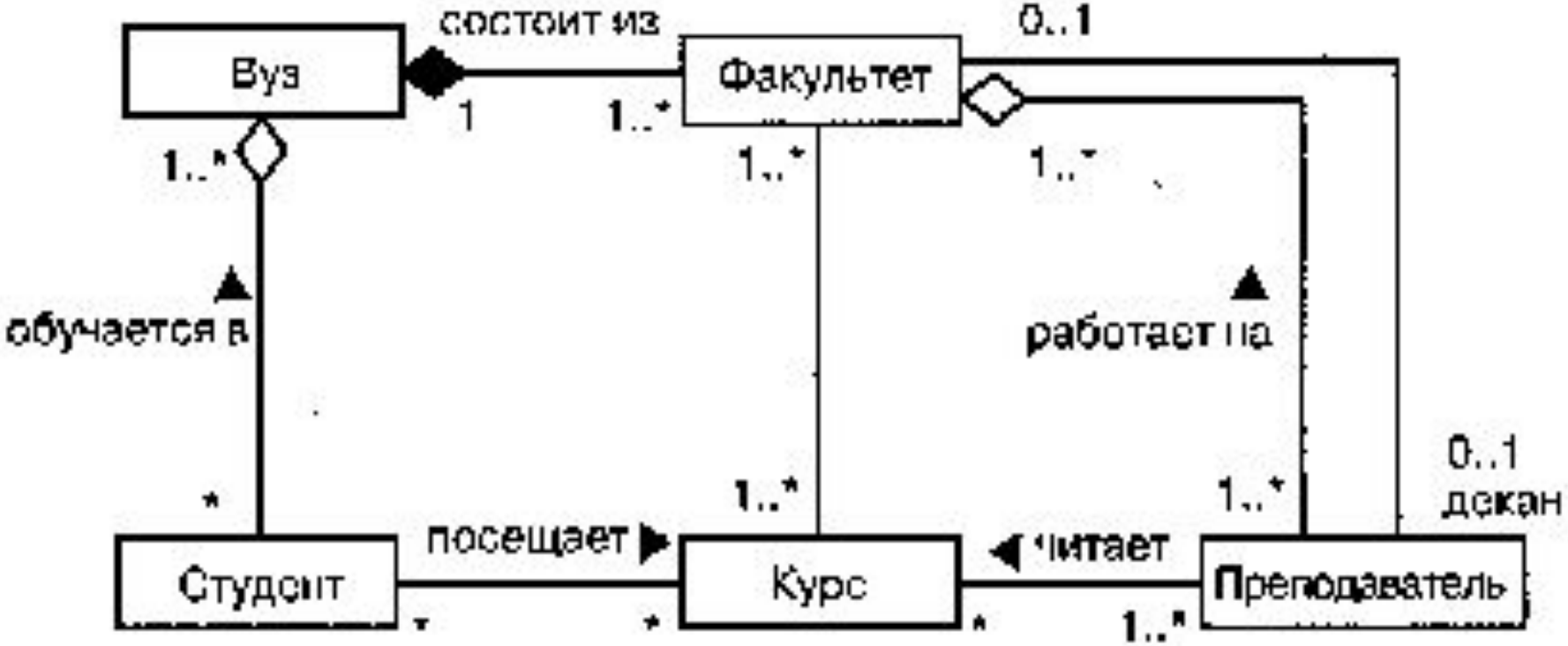
- На диаграммах состояний (Statechart diagrams) представлен автомат, включающий в себя состояния, переходы, события и виды действий. Диаграммы состояний относятся к динамическому виду системы; особенно они важны при моделировании поведения интерфейса, класса или кооперации. Они акцентируют внимание на поведении объекта, зависящем от последовательности событий, что очень полезно для моделирования реактивных систем.
- Диаграмма деятельности - это частный случай диаграммы состояний; на ней представлены переходы потока управления от одной деятельности к другой внутри системы. Диаграммы деятельности относятся к динамическому виду системы; они наиболее важны при моделировании ее функционирования и отражают поток управления между объектами.

- На диаграмме компонентов представлена организация совокупности компонентов и существующие между ними зависимости. Диаграммы компонентов относятся к статическому виду системы с точки зрения реализации. Они могут быть соотнесены с диаграммами классов, так как компонент обычно отображается на один или несколько классов, интерфейсов или кооперацией.
- На диаграмме развертывания представлена конфигурация обрабатывающих узлов системы и размещенных в них компонентов. Диаграммы развертывания относятся к статическому виду архитектуры системы с точки зрения развертывания. Они связаны с диаграммами компонентов, поскольку в узле обычно размещаются один или несколько компонентов.
- Здесь приведен неполный список диаграмм, применяемых в UML. Инструментальные средства позволяют генерировать и другие диаграммы, но девять перечисленных встречаются на практике чаще всего.


словарь,
функциональность
сборка
системы,
управление
конфигурацией
вид с точки зрения
проектирования
вид с точки зрения
реализации
поведение
вид с точки зрения
прецедентов
вид с точки зрения
процессов
вид с точки зрения
развертывания
производительность,
масштабируемость,
пропускная
способность
топология системы,
распределение,
поставка,
установка


Window
open()
close()
move()
display()
handleEvent()
зависимость
Event
обобщение
ассоциация
ConsoleWindow
DialogBox
Control

Отношения



Структурные отношения