Вопрос № 3.22

Моделирование программного обеспечения. Model-driven architecture. Языки UML и Executable UML, основные диаграммы.

Моделирование программного обеспечения

Для быстрой и эффективной разработки программного продукта с минимальным браком требуется привлечь рабочую силу, выбрать правильные инструменты и определить верное направление работы. Чтобы справиться с поставленной задачей, принимая во внимание затраты на жизненный цикл системы, необходимо, чтобы процесс разработки приложения был тщательно продуман и мог быть адаптирован к изменяющимся потребностям вашего бизнеса и технологии.

Центральным элементом деятельности, ведущей к созданию первоклассного программного обеспечения, является моделирование. Модели позволяют нам наглядно продемонстрировать желаемую структуру и поведение системы. Они также необходимы для визуализации и управления ее архитектурой. Модели помогают добиться лучшего понимания создаваемой нами системы, что зачастую приводит к ее упрощению и обеспечивает возможность повторного использования. Наконец, модели нужны для минимизации риска.

Значение моделирования

Если вы хотите соорудить собачью будку, то можете приступить к работе, имея в наличии лишь кучу досок, горсть гвоздей, молоток, плоскогубцы и рулетку.

Если вам надо построить дом для своей семьи: качественное, и не нарушающее строительных норм и правил, то придется сделать чертежи с учетом назначения каждой комнаты и таких деталей, как освещение, отопление и водопровод.

Задавшись целью построить небоскреб для офиса, было бы совсем неразумно браться за дело, имея в распоряжении груду досок и молоток. Поскольку в этом случае, скорее всего, будут привлекаться многочисленные капиталовложения, инвесторы потребуют, чтобы вы учли их пожелания относительно размера, стиля и формы строения. Так как любой просчет вам обойдется слишком дорого, значение планирования многократно возрастает. Вам потребуется выполнять множество разнообразных чертежей и моделей здания.

Хотя это и кажется комичным, многие компании, разрабатывающие программное обеспечение, хотят создать небоскреб, в то время как их подход к делу очень напоминает строительство собачьей будки.

Если вы действительно хотите создать программный продукт, по масштабности замысла сопоставимый с жилым домом или небоскребом, то ваша задача не сводится к написанию большого объема кода. На самом деле проблема в том, чтобы написать *правильный код минимального* объема. При таком подходе разработка качественного программного обеспечения сводится к вопросам выбора архитектуры, подходящего инструментария и средств управления процессом.

Кстати, нужно иметь в виду, что многие проекты, задуманные «по принципу будки», быстро вырастают до размеров небоскреба, становясь жертвой собственного успеха. Если такой рост не был учтен в архитектуре приложения, технологическом процессе или при выборе инструментария, то неизбежно наступает время, когда выросшая до размеров огромного дома будка обрушится под тяжестью собственного веса.

Неудачные проекты терпят крах в силу самых разных причин, а вот успешные, как правило, имеют много общего. Хотя успех программного проекта обеспечивается множеством разных составляющих, одной из главных является применение моделирования.

Моделирование - это устоявшаяся и повсеместно принятая инженерная методика.

Итак, что же такое модель? Попросту говоря, модель - это упрощенное представление 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, которые существенно влияют на результат, и не включает те, которые малозначимы на данном уровне абстракции. Каждая система может быть описана с разных точек зрения, для чего используются разные модели, каждая из которых, следовательно, является семантически замкнутой абстракцией системы. Модель может быть структурной, подчеркивающей организацию системы, или же поведенческой, отражающей ее динамику.

Мы строим модель для того, чтобы лучше понимать разрабатываемую систему.

Моделирование позволяет решить четыре различные задачи:

Визуализировать систему в ее текущем или желательном для нас состоянии;

Описать структуру или поведение системы;

Получить шаблон, позволяющий сконструировать систему;

Документировать принимаемые решения, используя полученные модели.

Моделирование предназначено не только для создания больших систем. Даже программный эквивалент собачьей будки выиграет от его применения. Чем больше и сложнее система, тем большее значение приобретает моделирование при ее разработке. Дело в том, что *мы строим модели сложных систем, поскольку иначе такие системы невозможно воспринять как единое целое.*

Однако многие исследования показывают, что в большинстве компаний, разрабатывающих программное обеспечение, моделирование применяется редко или же не применяется вообще. Чем проще проект, тем менее вероятно, что в нем будет использовано формальное моделирование. Ключевое слово здесь - «формальное». На практике даже при реализации простейшего проекта разработчики в той или иной мере применяют моделирование, хотя бы неформально. Для визуализации части системы ее проектировщик может нарисовать что-то на доске или клочке бумаги. И хотя ничего плохого в таких моделях нет (если они работают, то их существование оправдано), но такие неформальные модели часто создаются для однократного применения и не обеспечивают общего языка, который был бы понятен другим участникам проекта.

Обратно, чем сложнее проект, тем более вероятно, что из-за отсутствия моделирования он свернется раньше времени - или будет создано не то, что нужно. Все полезные и интересные системы с течением времени обычно усложняются. Пренебрегая моделированием в самом начале создания системы, вы, возможно, серьезно пожалеете об этом, когда будет уже слишком поздно.

Принципы моделирования

Можно сформулировать четыре основных принципа.

Во-первых, выбор модели оказывает определяющее влияние на подход к решению проблемы и на то, как будет выглядеть это решение. Правильно выбранная модель высветит самые коварные проблемы разработки и позволит проникнуть в самую суть задачи, что при ином подходе было бы просто невозможно. Неправильная модель заведет вас в тупик, поскольку внимание будет заостряться на несущественных нюансах.

Второй принцип формулируется так: каждая модель может быть представлена с различной степенью точности. Иногда простая и быстро созданная модель программного 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. 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 будет все больше и 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. При этом надо понимать, что модели могут создаваться и изучаться по отдельности, но вместе с тем остаются взаимосвязанными.

Для понимания архитектуры системы требуется несколько взаимодополняющих представлений:

1. с точки зрения вариантов использования (чтобы выявить требования к системе).
2. с точки зрения проектирования (чтобы построить словарь предметной области и области решения)
3. с точки зрения взаимодействий (чтобы смоделировать взаимодействия между частями системы, системой *в* целом и средой ее функционирования)
4. с точки зрения реализации (позволяющее рассмотреть физическую реализацию системы) и с точки зрения размещения (помогающее сосредоточиться на вопросах системного проектирования).

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

Объектное моделирование

При разработке программного обеспечения 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 вопросам декомпозиции больших алгоритмов на меньшие.

Наиболее современный подход к разработке программного обеспечения – *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* - описание множества однотипных объектов.

Если вы приняли объектно-ориентированный взгляд на мир, вам придется ответить на ряд вопросов. Какая структура должна быть у хорошей объектно-ориентированной архитектуры? Какие артефакты должны быть созданы в процессе работы над проектом? Кто должен создавать их? И, наконец, как оценить результат?

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

Model Driven Architecture

Model Driven Architecture (MDA) — создаваемая консорциумом OMG концепция модельно ориентированного подхода к разработке программного обеспечения. Его суть состоит в построении абстрактной метамодели управления и обмена метаданными (моделями) и задании способов ее трансформации в поддерживаемые технологии программирования (Java, CORBA, XML и др.). Создание метамодели определяется технологией моделирования MOF (Meta Object Facility), являющейся частью концепции MDA.

Языки UML и Executable UML, основные диаграммы.

UML (Unified Modeling Language) — унифицированный язык моделирования. Авторами, в основном, являются Грэди Буч, Джеймс Рамбо, Айвар Якобсон.

UML – это стандартный инструмент для разработки «чертежей» программного обеспечения. Его можно использовать для визуализации, спецификации, конструирования и документирования артефактов программных систем.

В данном контексте *специфицирование* - это построение точных, недвусмысленных и полных моделей. В частности, UML позволяет специфицировать все важные решения, касающиеся анализа, дизайна и реализации, принимаемые в процессе разработки и внедрения программных систем.

UML не является визуальным языком программирования, но его модели могут быть непосредственно ассоциированы с различными языками программирования. А это значит, что существует возможность отобразить UML-модель на такой язык как Java, C++ или Visual Basic, а при необходимости даже на таблицы реляционной базы данных.

Отображение модели на язык программирования позволяет осуществить *прямое проектирование* (forward engineering) – генерацию кода на языке программирования из модели UML. Обратное также возможно: вы можете восстановить модель UML на основе существующей реализации (обратное проектирование – reverse engineering).

UML также предназначен для документирования архитектуры системы и всех ее деталей. Кроме того, это язык для выражения требований к системе и описания тестов. И, наконец, он подходит для моделирования работ на этапе проектирования и управления версиями.

- требования;
- архитектура;
- проектные решения (дизайн);
- проектные планы;
- тесты;
- прототипы;
- релизы (версии).

Концептуальная модель UML

Три основные элемента: строительные блоки языка, правила, определяющие их сочетания, и некоторые общие для всего языка механизмы.

Строительные блоки UML

Словарь UML включает три вида строительных блоков:

- **Сущности.**
- **Связи.**
- **Диаграммы.**

Сущности (things) - это абстракции, которые являются основными элементами модели, *связи* (relationships) соединяют их между собой, а диаграммы (diagrams) группируют представляющие интерес наборы сущностей.

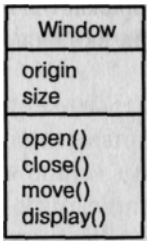
Есть четыре вида сущностей UML:

- **Структурные.**
- **Поведенческие.**
- **Группирующие.**
- **Аннотирующие**.

Все они представляют собой базовые объектно-ориентированные строительные блоки моделей UML. Вы используете их для описания хорошо согласованных моделей.

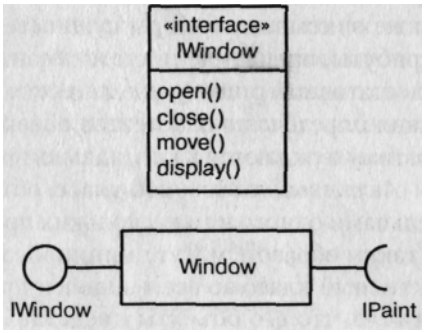
Структурные сущности – «имена существительные» в моделях UML. Это в основном статические части модели, представляющие либо концептуальные, либо физические элементы. В совокупности структурные сущности называются классификаторами (classifiers).

Класс (class) - это описание набора объектов с одинаковыми атрибутами, операциями, связями и семантикой. Класс реализует один или несколько интерфейсов.



Классы

Интерфейс (interface) - это набор операций, который специфицирует сервис (набор услуг) класса или компонента. Таким образом, интерфейс описывает видимое извне поведение элемента.



Интерфейсы

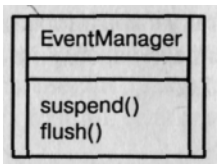
Кооперация (collaboration) определяет взаимодействие и представляет собой совокупность ролей и других элементов, которые функционируют вместе, обеспечивая некоторое совместное поведение, представляющее нечто большее, чем сумма поведений отдельных элементов.

Кооперации

Вариант использования (use case) - это описание последовательности действий, выполняемых системой и приносящих значимый результат конкретному *действующему лицу* (actor). Варианты использования применяются для структурирования поведенческих сущностей модели. Реализуются посредством кооперация.

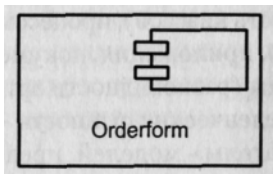
Варианты использования

Активный класс - это класс, объекты которого являются владельцами одного или нескольких процессов или потоков (threads) и, таким образом, могут инициировать управляющие воздействия.



Активные классы

Компонент - это модульная часть системы, которая скрывает свою реализацию за набором внешних интерфейсов. Компоненты системы, разделяющие общие интерфейсы, могут замещать друг друга, сохраняя при этом одинаковое логическое поведение. Реализация компонента может быть выражена объединением частей и коннекторов; при этом части могут включать в себя более мелкие компоненты.

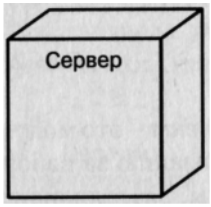


Компоненты

Артефакт (artifact) - это физическая и замещаемая часть системы, несущая физическую информацию («биты»). В системе вы можете встретить разные виды артефактов, таких как файлы исходного кода, исполняемые программы и скрипты.

Артефакты

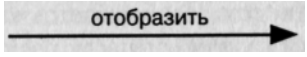
Узел (node) - это физический элемент, который существует во время исполнения и представляет вычислительный ресурс, обычно имеющий по меньшей мере некоторую память и часто - вычислительные возможности. Набор компонентов может находиться на узле, а также мигрировать с одного узла на другой.



Узлы

Поведенческие сущности - динамические части моделей UML. Это «глаголы» моделей, представляющие поведение во времени и пространстве. Всего существует три основных вида поведенческих сущностей.

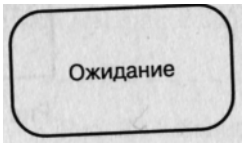
Первый из них - *взаимодействие* (interaction) – представляет собой поведение, которое заключается в обмене сообщениями между наборами объектов или ролей в определенном контексте для достижения некоторой цели. Поведение совокупности объектов или индивидуальная операция могут быть выражены взаимодействием. Взаимодействие включает множество других элементов – таких как *сообщения*, *действия* (actions) и *коннекторы* (соединения между объектами).



Сообщения

Вторая из поведенческих сущностей - *автомат* (state machine) – представляет собой поведение, характеризуемое последовательностью состояний объекта, в которых он оказывается на протяжении своего жизненного цикла в ответ на события, вместе с его реакцией на эти события.

Автомат включает в себя множество других элементов: состояния, переходы (из одного состояния в другое), события (сущности, которые инициируют переходы), а также действия (реакции на переходы).



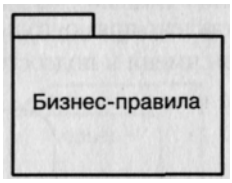
Состояния

Третья из поведенческих сущностей - *деятельность* (activity) – специфицирует последовательность шагов процесса вычислений. Во взаимодействии внимание сосредоточено на наборе взаимодействующих объектов, в автомате - на жизненном цикле одного объекта; для деятельности же в центре внимания - последовательность шагов безотносительно к объектам, выполняющим каждый шаг. Отдельный шаг деятельности называется *действием* (action).

Эти три элемента - взаимодействия, автоматы и деятельности – представляют собой базовые поведенческие сущности, которые вы можете включить в UML-модель. Семантически эти элементы обычно связаны с различными структурными элементами – в первую очередь, классами, кооперациями и объектами.

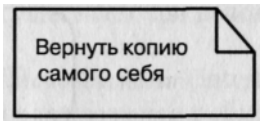
Группирующие сущности – организационная часть моделей UML. Это «ящики», по которым можно разложить модель.

Главная из группирующих сущностей - пакет. *Пакет* (package) - это механизм общего назначения для организации проектных решений, который упорядочивает конструкции реализации.



Пакеты

Аннотирующие сущности - это поясняющие части 1ШЬ-моде-лей, иными словами, комментарии, которые вы можете применить для описания, выделения и пояснения любого элемента модели. Главная из аннотирующих сущностей - *примечание* (note).



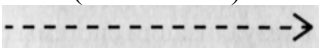
Примечания

Существует четыре типа связей в UML:

- Зависимость.
- Ассоциация.
- Обобщение.
- Реализация.

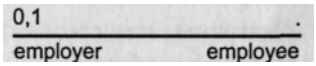
Эти связи представляют собой базовые строительные блоки для описания отношений в UML используемые для разработки хорошо согласованных моделей.

Зависимость (dependency) - семантически представляет собой связь между двумя элементами модели, в которой изменение одного элемента (независимого) может привести к изменению семантики другого элемента (зависимого).



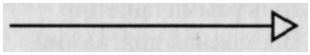
Зависимости

Ассоциация (association), - это структурная связь между классами, которая описывает набор связей, существующих между объектами - экземплярами классов. *Агрегация* (aggregation) - особая разновидность ассоциации, представляющая структурную связь целого с его частями.



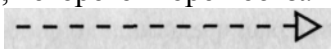
Ассоциации

Третья связь - *обобщение* (generaliztion) - выражает специализацию или обобщение, в котором специализированный элемент (потомок) строится по спецификациям обобщенного элемента (родителя).



Обобщения

Реализация (realization) - это семантическая связь между классификаторами, когда один из них специфицирует соглашение, которого второй обязан придерживаться.



Реализации

Эти четыре элемента представляют основные сущности отношений, которые могут быть включены в UML-модели. Есть также различные их вариации: уточнение (refinements), след (trace), включение (include) и расширение (extend).

Диаграммы UML.

Диаграмма - это графическое представление набора элементов, чаще всего изображенного в виде связного графа вершин (сущностей) и путей (связей). Вы рисуете диаграммы для визуализации системы с различных точек зрения, поэтому отдельная диаграмма - это проекция системы. Для всех систем, кроме самых тривиальных, диаграмма представляет собой ограниченный взгляд на элементы, составляющие систему. Один и тот же элемент может появляться либо во всех диаграммах, либо в некоторых (наиболее частый случай), либо вообще ни в одной (очень редкий случай). Теоретически диаграмма может включать в себя любую комбинацию сущностей и связей. На практике, однако, используется лишь небольшое число общих комбинаций, состоящих из пяти наиболее часто применяемых представлений архитектуры программных систем. По этой причине UML включает 13 видов диаграмм:

1. Диаграмма классов.
2. Диаграмма объектов.
3. Диаграмма компонентов.
4. Диаграмма составной структуры.
5. Диаграмма вариантов использования.
6. Диаграмма последовательности.
7. Диаграмма коммуникации.
8. Диаграмма состояний.
9. Диаграмма деятельности.
10. Диаграмма размещения.
11. Диаграмма пакетов.
12. Временная диаграмма.
13. Диаграмма обзора взаимодействий.

Диаграмма классов (class diagram) показывает набор классов, интерфейсов и кооперации, а также их связи. Диаграммы этого вида чаще всего используются для моделирования объектно-ориентированных систем. Предназначены для статического представления системы. Диаграммы классов, включающие активные классы,

представляют статическое представление процессов системы. Диаграммы компонентов - это разновидность диаграмм классов.

Диаграмма объектов (object diagram) показывает набор объектов и их связи. Диаграммы объектов представляют статические копии состояний экземпляров сущностей, описанных в диаграмме классов. Также представляют статическое представление дизайна или статическое представление процессов системы (как и диаграммы классов, но с точки зрения реальных или прототипных ситуаций).

Диаграмма компонентов (component diagram) демонстрирует инкапсулированные классы и их интерфейсы, порты и внутренние структуры, состоящие из вложенных компонентов и коннекторов. Диаграммы компонентов описывают статическое представление дизайна системы. Они важны при построении больших систем из мелких частей (UML отличает *диаграмму составной структуры* (composite structure diagram), применимую к любому классу, от компонентной диаграммы, но мы рассматриваем их вместе, потому что различие между ними весьма тонкое.)

Диаграмма вариантов использования (use case diagram) демонстрирует набор вариантов использования и действующих лиц (которые являются специальным видом классов), а также их связи. Диаграммы этого типа описывают статическое представление вариантов использования системы. Особенно важны для организации и моделирования поведения системы.

И диаграммы последовательностей, и диаграммы коммуникаций являются видами диаграмм взаимодействия. *Диаграмма взаимодействия* (interaction diagram) показывает взаимодействие, состоящее из набора объектов и ролей, включая сообщения, которые могут передаваться между ними. Диаграммы взаимодействия предназначены для динамического представления системы. *Диаграмма последовательности* (sequence diagram) - это разновидность диаграммы взаимодействия, показывающая временную последовательность сообщений. *Диаграмма коммуникаций* (communication diagram) - разновидность диаграммы взаимодействия, показывающая структурную организацию объектов или ролей, отправляющих и принимающих сообщения. И диаграммы последовательности, и диаграммы коммуникации представляют похожие базовые концепции, но с разных точек зрения. Диаграммы последовательности описывают временную последовательность, а коммуникационные диаграммы - структуры данных, через которые проходит поток сообщений.

Диаграмма состояний (state diagram) показывает автомат (state machine), включающий в себя состояния, переходы, события и деятельности. Диаграммы состояний описывают динамическое представление объекта. Они особенно важны для моделирования поведения интерфейсов, классов или кооперацийи подчеркивают событийно-зависимое поведение объекта, что особенно удобно для моделирования реактивных систем.

Диаграмма деятельности (activity diagram) показывает структуру процесса или других вычислений как пошаговый поток управления и данных. Диаграммы деятельности описываютдинамическое представление системы. Они особенно важны при моделировании функций системы и выделяют поток управления между объектами.

Диаграмма размещения (deployment diagram) показывает конфигурацию узлов-процессоров, а также размещаемые на них компоненты. Диаграммы размещения дают статическое представление размещения архитектуры. Узлы, как правило, содержат один или несколько артефактов.

Диаграмма артефактов (artifact diagram) показывает физический состав компьютерной системы. Артефакты представляют собой файлы, базы данных и подобные им физические наборы битов. Диаграммы данного типа часто применяются в сочетании с диаграммами размещения. Также показывают классы и компоненты, реализованные ими. UML трактует диаграммы артефактов как разновидность диаграмм размещения, но мы рассматриваем их отдельно.

Диаграмма пакетов (package diagram) показывает декомпозицию самой модели на организационные единицы и их зависимости.

Временная диаграмма (timing diagram) - это диаграмма взаимодействий, показывающая реальное время жизни различных объектов или ролей, в противовес простой последовательности со-

общений. *Диаграмма обзора взаимодействий* (interaction overview diagram) - это гибрид диаграммы деятельности и диаграммы последовательности.