Раздел 3. Стандарты проектирования на унифицированном языке моделирования UML

Тема 6. Общие сведения об унифицированном языке моделирования UML

Особенности, цели создания и нотация языка UML.

Унифицированный язык моделирования (Unified Modeling Language (**UML**)) — это стандартный язык спецификации, визуализации, построения и документирования артефактов автоматизированной системы, применяется в бизнес-моделировании и других сферах, связанных с разработкой программного обеспечения. UML представляет собой ряд лучших инженерных нотаций, которые прошли успешную проверку в моделировании больших и комплексных систем. UML является одной из наиболее важных частей разработки объектно ориентированных программ и процесса разработки программного обеспечения. UML использует по большей части графические нотации, выражающие дизайн проекта автоматизированной системы. Использование UML помогает взаимодействовать проектной команде, рассматривать потенциальные схемы и проверять архитектурный план программного продукта.

Основными целями создания UML являются:

1) Обеспечение пользователей готовыми выражениями языка визуального моделирования, для того чтобы они могли создавать и изменять понятные модели.

2) Обеспечение расширяемого и специализированного механизма отражающего основную концепцию.

3) Независимость от особенностей языка программирования и процесса разработки.

4) Обеспечение формальной базы для понимания языка моделирования.

5) Поддержка развития рынка объпектно-ориентированных инструментов.

6) Поддержка высокоуровневой концепций разработки, таких как совместная работа, фреймворки, паттерны и компоненты.

7) Внедрение лучщих правил.

Унифицированный язык моделирования (UML) является стандартным инструментом для создания «чертежей» программного обеспечения. С помощью UML можно визуализировать, специфицировать, конструировать и документировать артефакты программных систем. UML пригоден для моделирования любых систем: от информационных систем масштаба предприятия до распределенных Web-приложений и даже встроенных систем реального времени. UML – это язык для визуализации, специфицирования, конструирования и документирования артефактов программных систем. Язык моделирования, подобный UML, является стандартным средством для составления «чертежей» программного обеспечения.

Для понимания любой нетривиальной системы приходится разрабатывать большое количество взаимосвязанных моделей. В применении к программным системам это означает, что необходим язык, с помощью которого можно с различных точек зрения описать представления архитектуры системы на протяжении цикла ее разработки. UML – это графический язык специфицирования, что означает построение точных и полных графических моделей, касающиеся анализа, проектирования и реализации, которые должны приниматься в процессе разработки и развертывания системы программного обеспечения. UML – это язык конструирования, и модели, созданные с его помощью, могут быть непосредственно переведены на различные языки программирования. Иными словами, UML-модель можно отобразить на такие языки, как Java, C++, Visual Basic, и даже на таблицы реляционной базы данных. Такое отображение модели на язык программирования позволяет осуществлять

прямое проектирование: генерацию кода из модели UML в какой-то конкретный язык. Можно решить и обратную задачу: реконструировать модель по имеющейся реализации.

UML позволяет решить проблему документирования системной архитектуры и всех ее деталей, предлагает язык для формулирования требований к системе и определения тестов и, наконец, предоставляет средства для моделирования работ на этапе планирования проекта и управления версиями. Использование UML эффективно в:

- информационных системах масштаба предприятия;
- банковских и финансовых услугах;
- телекоммуникациях;
- на транспорте;
- оборонной промышленности, авиации и космонавтике;
- розничной торговле;
- медицинской электронике;
- науке;
- распределенных Web-системах.

Сфера применения UML не ограничивается моделированием ПО. Он позволяет моделировать, скажем, документооборот в юридических системах, структуру и функционирование системы обслуживания пациентов в больницах, осуществлять проектирование аппаратных средств.

Конструктивное использование языка UML основывается на понимании общих принципов моделирования сложных систем и особенностей процесса объектно-ориентированного анализа и проектирования (ООАП). Выбор выразительных средств для построения моделей сложных систем основывается на нескольких принципах.

Первым является принцип **абстрагирования**, который предписывает включать в модель только те аспекты проектируемой системы, которые имеют непосредственное отношение к выполнению системой своих функций или своего целевого предназначения. При этом все второстепенные детали опускаются, чтобы чрезмерно не усложнять процесс анализа и исследования полученной модели.

Вторым принципом построения моделей сложных систем является принцип **многомодельности**. Это значит, что никакое единственное представление сложной системы не является достаточным для адекватного выражения всех ее особенностей.

Еще одним принципом прикладного системного анализа является принцип **иерархического построения** моделей сложных систем. Этот принцип предписывает рассматривать процесс построения модели на разных уровнях абстрагирования или детализации в рамках фиксированных представлений.

Таким образом, процесс ООАП можно представить как поуровневый спуск от наиболее общих моделей и представлений концептуального уровня к более частным и детальным представлениям логического и физического уровня. При этом на каждом из этапов ООАП данные модели последовательно дополняются все большим количеством деталей, что позволяет им более адекватно отражать различные аспекты конкретной реализации сложной системы.

Объектно-ориентированный анализ и проектирование системы предусматривает использование словаря языка UML, включающего три вида строительных блоков: **сущности, отношения, диаграммы.**

Основными объектно-ориентированными блоками являются сущности. В языке UML имеется четыре вида сущностей: структурные, поведенческие, группирующие, аннотационные.

Структурные сущности – это имена существительные в моделях на языке UML. Они представляют собой статические части модели, соответствующие концептуальным или физическим элементам системы. Существует пять разновидностей концептуальных и логических сущностей.

Класс (Class) – это описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой (рис. 3). Класс реализует один или несколько интерфейсов.

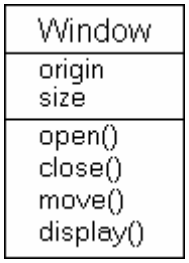


Рис. 3 Классы

Интерфейс (Interface) – это совокупность операций, которые определяют набор услуг, предоставляемый классом или компонентом. Интерфейс описывает видимое извне поведение элемента (рис. 4). Интерфейс редко существует сам по себе – обычно он присоединяется к реализующему его классу или компоненту.

Рис. 4 Интерфейс

Кооперация (Collaboration) определяет взаимодействие, и представляет совокупность ролей, работающих совместно, производят некоторый кооперативный эффект. Кооперация имеет как структурный, так и поведенческий аспект, – один и тот же класс может принимать участие в нескольких кооперациях (рис. 5).

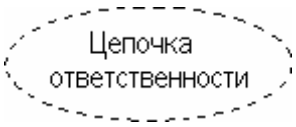


Рис. 5 Кооперация

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, и реализуются посредством кооперации (рис. 6).

Рис. 6 Прецедент

Активным классом (Active class) называется класс, объекты которого вовлечены в один или несколько **процессов** (Threads), и могут инициировать управляющее воздействие. Активный класс

отличается от обычного класса тем, что деятельность его объектов 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 (рис. 7)

Рис. 7 Активный класс

Компоненты и узлы соответствуют физическим сущностям системы.

Компонент (Component) – это физическая заменяемая часть системы, соответствующая некоторому набору интерфейсов и обеспечивает его реализацию (рис. 8). Компонент – это физическая упаковка логических элементов, (классов, интерфейсов и кооперации), например компоненты COM+ или Java Beans, а также файлы исходного кода.

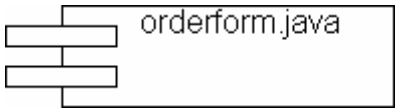


Рис. 8 Компонент

Узел (Node) – это элемент, представляющий вычислительный ресурс, обладающий памятью и способностью обработки. Совокупность компонентов может размещаться в узле, а также мигрировать с одного узла на другой (рис. 9).

Рис. 9 Узел

Существуют также разновидности этих семи 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. Существует всего два типа поведенческих сущностей.

Взаимодействие (Interaction) – это поведение, суть которого заключается в обмене сообщениями между объектами для достижения определенной цели. С помощью взаимодействия описывается как отдельная операция, так и поведение совокупности объектов. Взаимодействие предполагает ряд других элементов, таких как сообщение (рис. 10), последовательность действий (поведение, инициированное сообщением) и связь (между объектами).

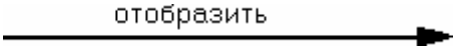
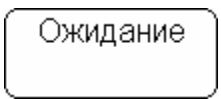


Рис. 10 Сообщение

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 (рис. 11). С помощью автомата можно описать поведение отдельного 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).

Рис. 11 Состояние

Взаимодействия и автоматы семантически связаны с различными структурными элементами,



такими как классы, кооперации и объекты.

Группирующие сущности являются организующими частями модели, это блоки, на которые можно разложить модель. Основной группирующей сущностью является пакет.

Пакет (Package) – это универсальный механизм организации элементов в группы (рис. 12). 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.

Рис. 12 Пакеты

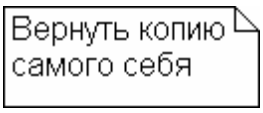
Существуют также вариации пакетов, например каркасы (Frameworks), модели и

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.

Примечание (Note) – это символ для изображения комментариев, присоединенных к элементу или группе элементов (рис. 13).

Рис. 13 Примечание



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.

Отношения

В языке UML определены четыре типа отношений: зависимость, ассоциация, обобщение, реализация. Эти отношения являются основными связующими строительными блоками в UML.

Зависимость (Dependency) – это семантическое отношение между двумя сущностями, при котором изменение одной из них, независимой, может повлиять на семантику другой, зависимой (рис. 14).

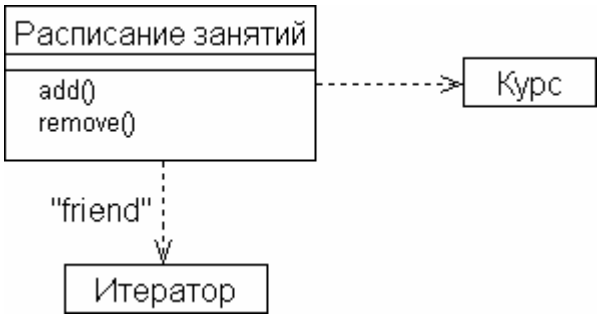


Рис. 14 Отношения зависимости

Ассоциация (Association) – отношение, описывающее совокупность связей между объектами. Разновидностью ассоциации является **агрегирование** (Aggregation) – структурное отношение между целым и его частями (рис. 15). Графическое изображение ассоциации может включать кратность и имена ролей (рис. 16).

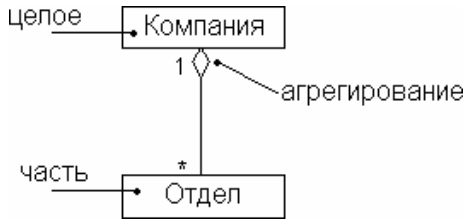


Рис. 15 Агрегирование

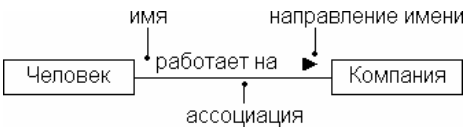


Рис. 16 Имена ассоциаций

Обобщение (Generalization) – это отношение "специализация/обобщение" (рис. 17), при котором объект специализированного элемента (потомок) может быть подставлен вместо объекта обобщенного элемента (родителя или предка). Таким образом, потомок (Child) наследует структуру и поведение своего родителя (Parent).

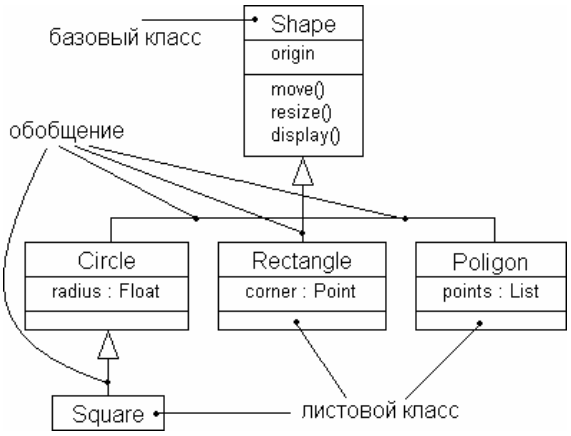


Рис. 17 Обобщение

Наконец, **реализация** (Realization) – это отношение между классификаторами, при котором один классификатор определяет "контракт", а другой гарантирует его выполнение (рис. 18).

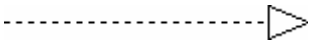


Рис. 18 Реализация

Отношения реализации встречаются в двух случаях: во-первых, между интерфейсами и реализующими их классами или компонентами, а во-вторых, между прецедентами и реализующими их кооперациями.

Диаграммы

Диаграмма в UML – это графическое представление набора элементов, изображаемое в виде связанного графа с вершинами (сущностями) и ребрами (отношениями), используемое для визуализации системы с разных точек зрения. В UML выделяют 8 типов диаграмм (рис. 19).

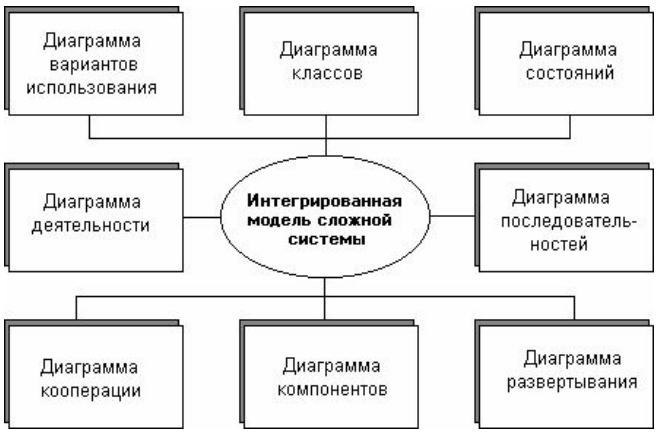


Рис. 19 Интегрированная модель сложной системы в нотации UML

На диаграмме **классов** (Class diagram) изображаются классы, интерфейсы, объекты и кооперации, а также их отношения. Используется при моделировании объектно-ориентированных систем.

На диаграмме **вариантов использования** (Use case diagram) представлены прецеденты и актеры (частный случай классов), а также отношения между ними. Они используются при моделировании поведения системы.

Диаграммы **последовательностей** (Sequence diagram) и **кооперации** (Collaboration diagram) являются частными случаями диаграмм взаимодействия. На диаграммах взаимодействия представлены связи между объектами (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 могут быть преобразованы друг в друга.

На диаграммах **состояний** (Statechart diagrams) представлен автомат, включающий состояния, переходы, события и виды действий. Диаграммы состояний используются при моделировании поведения интерфейса, класса или кооперации, зависящем от последовательности событий.

Диаграмма **деятельности** (Activity diagram) представляют переходы потока управления между объектами от одной деятельности к другой внутри системы.

Диаграмма **компонентов** (Component diagram) представляет зависимости между компонентами. Диаграммы компонентов отображаются на один или несколько классов, интерфейсов или кооперацией.

На диаграмме **развертывания** (Deployment diagram) представлена конфигурация обрабатывающих узлов системы и размещенных в них компонентов.

Тема 7. Создание проекта на UML

Назначение, синтаксис, инструменты ***диаграммы вариантов использования*** *на языке UML.*

На диаграммах вариантов использования отображается взаимодействие между вариантами использования, представляющими функции системы, и действующими лицами, представляющими людей или системы, получающие или передающие информацию в данную систему. Из диаграмм вариантов использования можно получить довольно много информации о системе. Этот тип диаграмм описывает общую функциональность системы. Пользователи, менеджеры проектов, аналитики, разработчики, специалисты по контролю качества и все, кого интересует система в целом, могут, изучая диаграммы вариантов использования, понять, что система должна делать.

К базовым элементам рассматриваемой диаграммы относятся **вариант использования**, **актер** и **интерфейс**.

Вариант использования (рис. 20) применяется для спецификации общих особенностей поведения системы или другой сущности без рассмотрения ее внутренней структуры (например, оформление заказа на покупку товара, получение информации о кредитоспособности клиента, отображение графической формы на экране монитора).

Рис. 20 Графическое обозначение варианта использования

Актер – это внешняя по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для решения

определенных задач (рис. 21). При этом актеры служат для обозначения согласованного множества ролей, которые могут играть пользователи в процессе взаимодействия с проектируемой системой.

Рис. 21 Графическое обозначение актера

Имя актера должно быть достаточно информативным с точки зрения семантики, например клиент банка, продавец магазина, пассажир авиарейса, водитель автомобиля, сотовый телефон.

Так как в общем случае актер всегда находится вне системы, его внутренняя структура никак не определяется. Для актера имеет значение только его внешнее представление, т.е. то, как он воспринимается со стороны системы. Актеры взаимодействуют с системой посредством передачи и приема сообщений от вариантов использования. Сообщение представляет собой запрос актером сервиса от системы и получение этого сервиса. Это взаимодействие может быть выражено посредством ассоциаций между отдельными актерами и вариантами использования или классами. Кроме этого, с актерами могут быть связаны интерфейсы, которые определяют, каким образом другие элементы модели взаимодействуют с этими актерами.

Интерфейс служит для спецификации параметров модели, которые видимы извне без указания их внутренней структуры (рис. 22). В диаграммах вариантов использования интерфейсы определяют совокупность операций, обеспечивающих необходимый набор сервисов или функциональности для актеров. Интерфейсы не могут содержать ни атрибутов, ни состояний, ни направленных ассоциаций. Они содержат только операции без указания особенностей их реализации. Формально интерфейс эквивалентен абстрактному классу без атрибутов и методов с наличием только абстрактных операций.

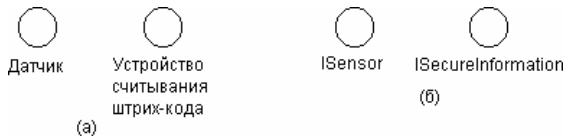


Рис. 22 Графическое изображение интерфейсов на диаграммах вариантов использования

Интерфейс соединяется с вариантом использования сплошной линией, если он реализует все операции, необходимые для данного интерфейса (рис. 23, а). Если же вариант использования определяет только тот сервис, который необходим для реализации данного интерфейса, используется пунктирная стрелка (рис. 23, б).

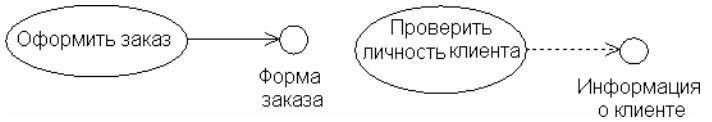


Рис. 23 Графическое изображение взаимосвязей интерфейсов с вариантами использования

а) реализация всех операций, б) реализация только необходимого сервиса

Важность интерфейсов заключается в том, что они определяют стыковочные узлы в проектируемой системе, что совершенно необходимо для организации коллективной работы над проектом. Более того, спецификация интерфейсов способствует “безболезненной” модификации уже существующей системы при переходе на новые технологические решения. В этом случае изменению подвергается только реализация операций, но никак не функциональность самой системы. А это обеспечивает совместимость последующих версий программ с первоначальными при спиральной технологии разработки программных систем.

Примечания в языке UML предназначены для включения в модель произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемого проекта (рис. 24). В качестве такой информации могут быть комментарии разработчика (например, дата и версия разработки диаграммы или ее отдельных компонентов), ограничения (например, на значения отдельных связей или экземпляры сущностей) и помеченные значения.

Рис. 24 Примеры примечаний в языке UML

Применительно к диаграммам вариантов использования примечание может носить самую общую информацию, относящуюся к общему контексту системы.

Отношения на диаграмме вариантов использования

Для выражения **отношений** между актерами и вариантами использования применяются стандартные виды отношений, описанные в предыдущей теме.

Отношение **ассоциации** применительно к диаграммам вариантов использования служит для обозначения специфической роли актера в отдельном варианте использования (рис. 25). Другими словами, ассоциация определяет семантические особенности взаимодействия актеров и вариантов использования в графической модели системы. Таким образом, это отношение устанавливает, какую конкретную роль играет актер при взаимодействии с экземпляром варианта использования. Графическое обозначение отношения ассоциации может включать дополнительные условные обозначения (имя и кратность).

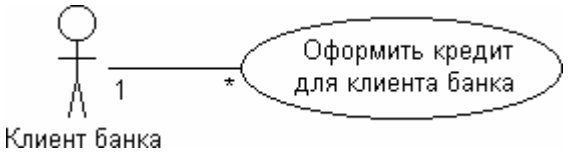


Рис. 25 Отношение ассоциации между актером и вариантом использования

Отношение **расширения** между вариантами использования обозначается пунктирной линией со стрелкой (вариант отношения зависимости), направленной от того варианта использования, который является расширением для исходного варианта использования. Данная линия со стрелкой помечается ключевым словом “extend” (“расширяет”), как показано на рис. 26.

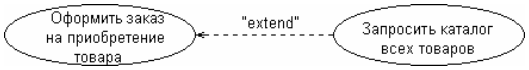
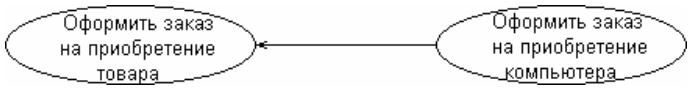


Рис. 26 Отношение расширения между вариантами использования

Отношение расширения отмечает тот факт, что один из вариантов использования может присоединять к своему поведению некоторое дополнительное поведение, определенное для другого варианта использования.

Отношение **обобщения** графически обозначается сплошной линией со стрелкой, которая указывает на родительский вариант использования (рис. 27).

Рис. 27 Отношение обобщения между вариантами использования



Отношение обобщения между вариантами использования применяется в том случае, когда необходимо отметить, что дочерние варианты использования обладают всеми атрибутами и особенностями поведения родительских вариантов. При этом дочерние варианты использования участвуют во всех отношениях родительских вариантов. В свою очередь, дочерние варианты могут наделяться новыми свойствами поведения, которые отсутствуют у родительских вариантов использования, а также уточнять или модифицировать наследуемые от них свойства поведения.

Отношение **включения** между двумя вариантами использования указывает, что поведение одного варианта использования включается в качестве составного компонента в последовательность поведения другого варианта использования. Графически данное отношение обозначается пунктирной линией со стрелкой (вариант отношения зависимости), направленной от базового варианта использования к включаемому. При этом данная линия со стрелкой помечается ключевым словом “include” (“включает”), как показано на рис. 28.

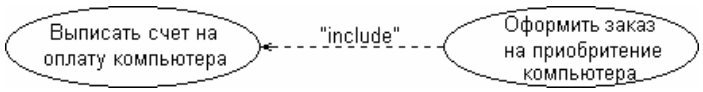


Рис. 28 Отношение включения между вариантами использования

При моделировании возникает необходимость в указании количества объектов, связанных посредством одного экземпляра ассоциации. Это число называется **кратностью** (Multiplicity) роли ассоциации и записывается либо как выражение, значением которого является диапазон значений, либо в явном виде (рис. 29). Кратность указывает на то, столько объектов должно соответствовать каждому объекту на противоположном конце. Кратность можно задать равной единице (1), указать диапазон: “ноль или единица” (0..1), “много” (0..*), “единица или больше” (1..*). Разрешается также указывать определенное число.

Рис. 29 Кратность

Пример диаграммы вариантов использования

Рассмотрим диаграмму вариантов использования отражающую систему работы банковского автомата (Automated Teller Machine, ATM) (рис. 30).

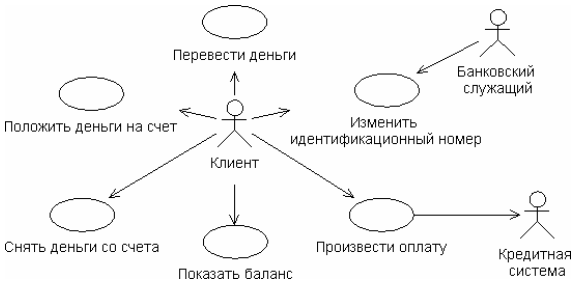


Рис. 30 Диаграмма вариантов использования для ATM

Клиент банка инициирует различные варианты использования: снять деньги со счета, перевести деньги, положить деньги на счет, Показать баланс, изменить идентификационный номер, произвести оплату. Банковский служащий может инициировать вариант использования "Изменить идентификационный номер". Действующими лицами могут быть и внешние системы, в данном случае кредитная система показана именно как действующее лицо – она является внешней для системы ATM. Стрелка, направленная от варианта использования к действующему лицу, показывает, что вариант использования предоставляет некоторую информацию действующему лицу. В данном случае вариант использования "Произвести оплату" предоставляет кредитной системе информацию об оплате по кредитной карточке.

Тема 8. Создание проекта на UML

Назначение, синтаксис, инструменты ***диаграмм состояний и деятельности*** *на языке UML.*

Диаграммы состояний (statechart diagram)

Главное предназначение этой диаграммы – описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Чаще всего диаграммы состояний используются для описания поведения отдельных экземпляров классов (объектов), но они также могут быть применены для спецификации функциональности других компонентов моделей, таких как варианты использования, актеры, подсистемы, операции и методы.

Диаграмма состояний по существу является графом специального вида, который представляет некоторый автомат. Вершинами этого графа являются состояния и некоторые другие типы элементов автомата (псевдосостояния), которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние. Для понимания семантики конкретной диаграммы состояний необходимо представлять не только особенности поведения моделируемой сущности, но и знать общие сведения по теории автоматов.

Автоматы

Автомат (state machine) в языке UML представляет собой некоторый формализм для моделирования поведения элементов модели и системы в целом. Автомат описывает поведение отдельного объекта в форме последовательности состояний, которые охватывают все этапы его жизненного цикла, начиная от создания объекта и заканчивая его уничтожением. Каждая диаграмма состояний представляет некоторый автомат.

Простейшим примером визуального представления состояний и переходов на основе формализма автоматов может служить ситуация с исправностью технического устройства, такого как компьютер. В этом случае вводятся в рассмотрение два самых общих состояния: "исправен" и "неисправен" и два перехода: "выход из строя" и "ремонт". Графически эта информация может быть представлена в виде изображенной ниже диаграммы состояний компьютера (рис. 53).

Рис. 53 Простейший пример диаграммы состояний для технического устройства типа компьютер

Основными понятиями, входящими в формализм автомата, являются состояние и переход. Главное различие между ними заключается в том, что длительность нахождения системы в отдельном состоянии существенно превышает время, которое затрачивается на переход из одного состояния в другое. Предполагается, что в пределе время перехода из одного состояния в другое равно нулю (если дополнительно ничего не сказано). Другими словами, переход объекта из состояния в состояние происходит мгновенно.

В языке UML под *состоянием* понимается абстрактный метакласс, используемый для моделирования отдельной ситуации, в течение которой имеет место выполнение некоторого условия. Состояние может быть задано в виде набора конкретных значений атрибутов класса или объекта, при этом изменение их отдельных значений будет отражать изменение состояния моделируемого класса или объекта.

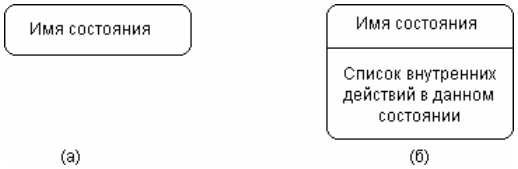


Рис. 54 Графическое изображение состояний на диаграмме состояний

Состояние на диаграмме изображается прямоугольником со скругленными вершинами (рис. 54). Этот прямоугольник, в свою очередь, может быть разделен на две секции горизонтальной линией. Если указана лишь одна секция, то в ней записывается только имя состояния (рис. 54, а). В противном случае в первой из них записывается имя состояния, а во второй – список некоторых внутренних действий или переходов в данном состоянии (рис. 54, б).

Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий. В этом состоянии находится объект по умолчанию в начальный момент времени. Графически начальное состояние в языке UML обозначается в виде закрашенного кружка (рис. 55, а), из которого может только выходить стрелка, соответствующая переходу.

Рис. 55 Графическое изображение начального и конечного состояний на диаграмме состояний

Конечное (финальное) состояние представляет собой частный случай состояния, которое также не содержит никаких внутренних действий . В этом состоянии будет находиться объект по умолчанию после завершения работы автомата в конечный момент времени. Графически

конечное состояние в языке UML обозначается в виде закрашенного кружка, помещенного в окружность (рис. 55, б), в которую может только входить стрелка, соответствующая переходу.

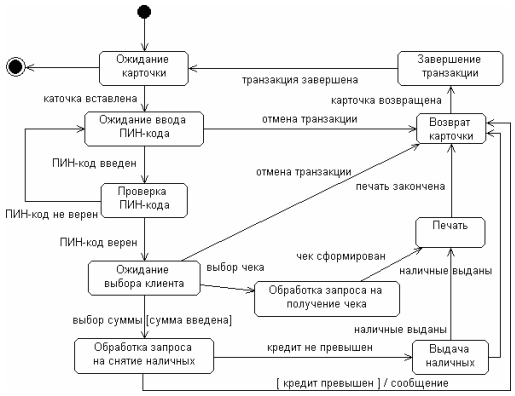
Простой переход (simple transition) представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим. Пребывание моделируемого объекта в первом состоянии может сопровождаться выполнением некоторых действий, а переход во второе состояние будет возможен после завершения этих действий, а также после удовлетворения некоторых дополнительных условий. На диаграмме состояний переход изображается сплошной линией со стрелкой, которая направлена в целевое состояние (рис. 56).

Рис. 56 Диаграмма состояний для моделирования почтовой программы-клиента

Пример диаграммы состояний

Рассмотрим пример диаграммы состояний для моделирования поведения банкомата (рис. 57).

Следует заметить, что в разрабатываемой модели диаграмма состояний является единственной и описывает поведение системы управления банкоматом в целом. Главное достоинство данной диаграммы состояний – возможность моделировать условный характер



реализации всех вариантов использования в форме

Рис. 57 Диаграмма состояний для моделирования поведения банкомата

изменения отдельных состояний разрабатываемой системы. Иногда разработку диаграммы состояний, особенно в условиях дефицита времени, отпущенного на выполнение проекта, опускают, т.к. часто происходит дублированиеинформации, представленно на диаграммах кооперации и последовательности.

Диаграммы деятельности (activity diagram)

При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций.

Для моделирования процесса выполнения операций в языке UML используются так называемые *диаграммы деятельности*. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов. Отличие заключается в семантике состояний, которые используются для представления не деятельностей, а действий, и в отсутствии на переходах сигнатуры событий. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой, операции в предыдущем состоянии. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами – переходы от одного состояния действия к другому.

В контексте языка UML деятельность (activity) представляет собой некоторую совокупность отдельных вычислений, выполняемых автоматом. При этом отдельные элементарные вычисления могут приводить к некоторому результату или действию (action). На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Сам же результат может привести к изменению состояния системы или возвращению некоторого значения.

Основные элементы диаграммы деятельности

Состояние действия (action state) является специальным случаем состояния с некоторым входным действием и по крайней мере одним выходящим из состояния переходом. Графически состояние действия изображается фигурой, напоминающей прямоугольник, боковые стороны которого заменены выпуклыми дугами (рис. 58). Внутри этой фигуры записывается выражение действия (action-expression), которое должно быть уникальным в пределах одной диаграммы деятельности.

(а) простое действие (б) выражение

Рис. 58 Графическое изображение состояния действия

Действие может быть записано на естественном языке, некотором псевдокоде или языке программирования. Никаких дополнительных или неявных ограничений при записи действий не накладывается.

При построении диаграммы деятельности используются только те переходы, которые переводят деятельность в последующее состояние сразу, как только закончится действие в предыдущем состоянии (нетриггерные). На диаграмме такой переход изображается сплошной линией со стрелкой.

Ветвление на диаграмме деятельности обозначается небольшим ромбом, внутри которого нет никакого текста (рис. 59).

В качестве примера рассмотрим фрагмент алгоритма нахождения корней квадратного уравнения. В общем случае после приведения уравнения второй степени к каноническому виду: $a*x*x + b*x + c = 0$ необходимо вычислить его дискриминант. Причем, в случае отрицательного дискриминанта уравнение не имеет решения на множестве действительных чисел, и дальнейшие вычисления должны быть прекращены. При неотрицательном дискриминанте уравнение имеет решение, корни которого могут быть получены на основе конкретной расчетной формулы.

Процедуру вычисления корней квадратного уравнения можно представить в виде диаграммы деятельности с тремя состояниями действия и ветвлением (рис. 59). Каждый из переходов, выходящих из состояния "Вычислить дискриминант", имеет сторожевое условие, определяющее единственную ветвь, по которой может быть продолжен процесс вычисления корней в зависимости от знака дискриминанта.

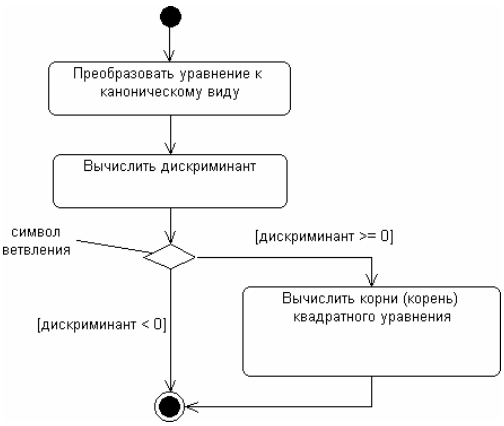
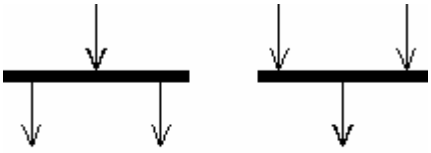


Рис. 59 Фрагмент диаграммы деятельности для алгоритма нахождения корней квадратного уравнения

В языке UML для распараллеливания вычислений используется специальный символ для



разделения (рис. 60, а) и слияния (рис. 60, б) параллельных вычислений или потоков управления.

Рис. 60 Разделения и слияния параллельных потоков управления

Диаграммы деятельности могут быть использованы не только для спецификации алгоритмов вычислений или потоков управления в программных системах. Не менее важная область их применения связана с моделированием бизнес-процессов. Для моделирования этих особенностей в языке UML используется специальная конструкция, получившее название дорожки (swimlanes). Имеется в виду визуальная аналогия с плавательными дорожками в бассейне, если смотреть на соответствующую диаграмму. При этом все состояния действия на диаграмме деятельности делятся на отдельные группы, которые отделяются друг от друга вертикальными линиями. Две соседние линии и образуют дорожку, а группа состояний между этими линиями выполняется отдельным подразделением (отделом, группой, отделением, филиалом) компании (рис. 61).

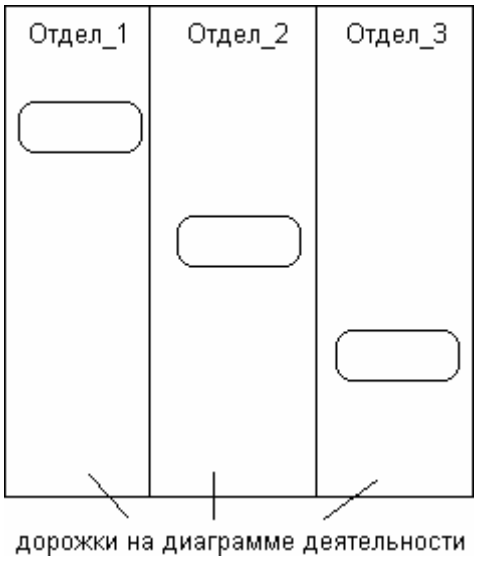


Рис. 61 Вариант диаграммы деятельности с дорожками

В общем случае действия на диаграмме деятельности выполняются над теми или иными объектами. Эти объекты либо инициируют выполнение действий, либо определяют некоторый

результат этих действий. При этом действия специфицируют вызовы, которые передаются от одного объекта графа деятельности к другому. Поскольку в таком ракурсе объекты играют определенную роль в понимании процесса деятельности, иногда возникает необходимость явно указать их на диаграмме деятельности.

Для графического представления объектов используются прямоугольник класса, с тем отличием, что имя объекта подчеркивается. Далее после имени может указываться характеристика состояния объекта в прямых скобках. Такие прямоугольники объектов присоединяются к состояниям действия отношением зависимости пунктирной линией со стрелкой. Соответствующая зависимость определяет состояние конкретного объекта после выполнения предшествующего действия.

Пример диаграммы деятельности

В качестве примера рассмотрим фрагмент диаграммы деятельности торговой компании, обслуживающей клиентов по телефону. Подразделениями компании являются отдел приема и оформления заказов, отдел продаж и склад.

Этим подразделениям будут соответствовать три дорожки на диаграмме деятельности, каждая из которых специфицирует зону ответственности подразделения (рис. 62).

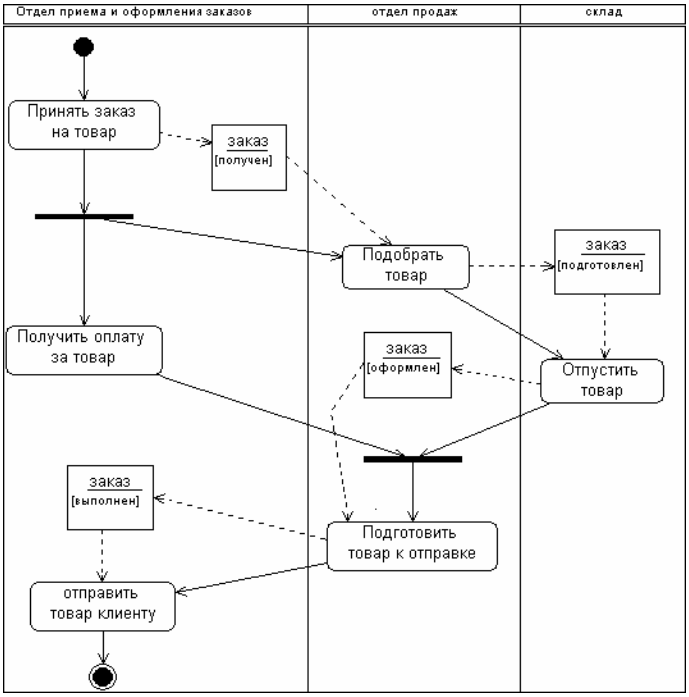


Рис. 62 Диаграмма деятельности торговой компании с объектом-заказом

В данном случае диаграмма деятельности заключает в себе не только информацию о последовательности выполнения рабочих действий, но и о том, какое из подразделений торговой компании должно выполнять то или иное действие. Кроме того центральным объектом процесса продажи является заказ или вернее состояние его выполнения. Вначале до звонка от клиента заказ как объект отсутствует и возникает лишь после такого звонка. Однако этот заказ еще не заполнен до конца, поскольку требуется еще подобрать конкретный товар в отделе продаж. После его подготовки он передается на склад, где вместе с отпуском товара заказ окончательно дооформляется. Наконец, после получения подтверждения об оплате товара эта информация заносится в заказ, и он считается выполненным и закрытым.

Тема 9. Создание проекта на UML

Назначение, синтаксис, инструменты ***диаграмм взаимодействия и сотрудничества*** *на языке UML.*

Диаграммы последовательности (sequence diagram)

Для моделирования взаимодействия объектов в языке UML используются соответствующие диаграммы взаимодействия. При этом учитываются два аспекта: 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.

Объекты диаграммы последовательности

Диаграммы последовательности отражают поток событий, происходящих в рамках варианта использования. На этих диаграммах изображаются только те объекты, которые непосредственно участвуют во взаимодействии т.к. ключевым моментом является именно динамика взаимодействия объектов во времени и не используются возможные статические ассоциации с другими объектами. При этом диаграмма последовательности имеет два измерения (рис. 31). Одно – слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии. Второе измерение – вертикальная временная ось, направленная сверху вниз. При этом взаимодействия объектов реализуются посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения и также образуют порядок по времени своего возникновения. Другими словами, сообщения, расположенные на диаграмме последовательности выше, инициируются раньше тех, которые расположены ниже.

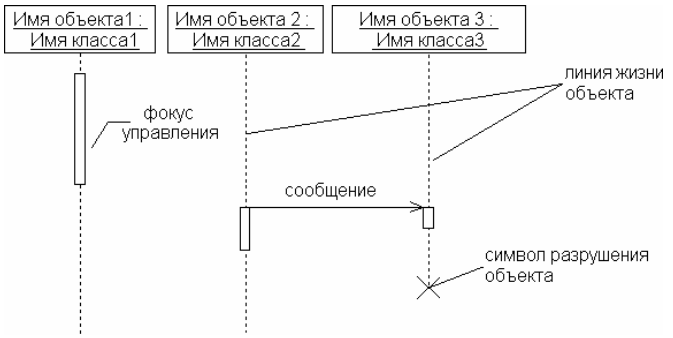


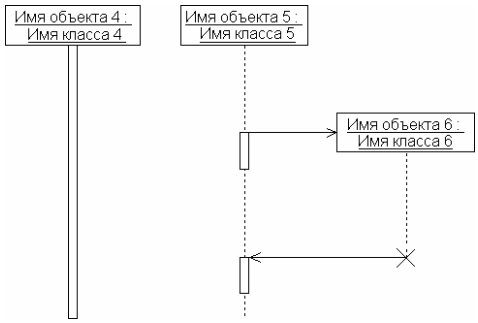
Рис. 31 Графические примитивы диаграммы последовательности

Линия жизни объекта (object lifeline) изображается пунктирной вертикальной линией, ассоциированной с единственным объектом на диаграмме последовательности. Линия жизни служит для обозначения периода времени, в течение которого объект существует в системе и, следовательно, может потенциально участвовать во всех ее взаимодействиях. Если объект существует в системе постоянно, то его линия жизни должна начинаться в верхней части диаграммы и заканчиваться в нижней части (объекты 1 и 2 на рис. 31). Отдельные объекты, выполнив свою роль в системе, могут быть уничтожены, чтобы освободить занимаемые ими ресурсы. Для обозначения момента уничтожения объекта в языке UML используется специальный символ в форме латинской буквы “X” (объект 3 на рис. 31). Ниже этого символа пунктирная линия не изображается, поскольку соответствующего объекта в системе уже нет, и этот объект должен быть исключен из всех последующих взаимодействий.

Отдельные объекты в системе могут создаваться по мере необходимости, существенно экономя ресурсы системы и повышая ее производительность (объект 6 на рис. 32).

Рис. 32 Варианты линий жизни и фокусов управления объектов

Комментарии или примечания уже рассматривались ранее при изучениидругих видов диаграмм. Они могут включаться и в диаграммы последовательности, ассоциируясь с



отдельными объектами или сообщениями.

Как уже отмечалось выше, взаимодействия объектов реализуются с помощью сообщений. У каждого сообщения должно быть имя, соответствующее его цели. Существует несколько видов сообщений: простое, синхронное, с отказом становиться в очередь и др. (рис. 33).

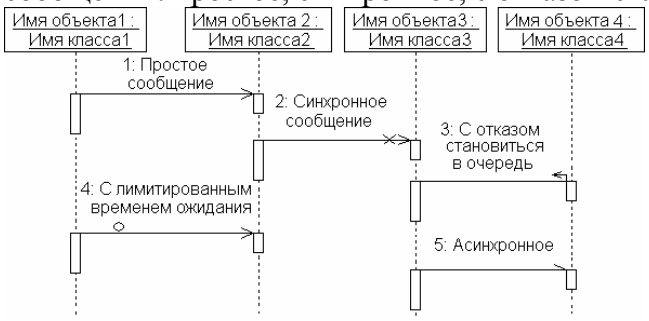


Рис. 33 Примеры сообщений

Простое сообщение используется по умолчанию. Означает, что все сообщения выполняются в одном потоке управления (рис. 33, 1).

Синхронное (synchronous) применяется, когда клиент посылает сообщение и ждет ответа пользователя (рис. 33, 2).

Сообщение с отказом становиться в очередь (balking): клиент посылает сообщение серверу и, если сервер не может немедленно принять сообщение, оно отменяется (рис. 33, 3).

Сообщение с лимитированным временем ожидания (timeout): клиент посылает сообщение серверу, а затем ждет указанное время; если в течение этого времени сервер не принимает сообщение, оно отменяется (рис. 33, 4).

Асинхронное сообщение (asynchronous): клиент посылает сообщение серверу и продолжает свою работу, не ожидая подтверждения о получении (рис. 33, 5).

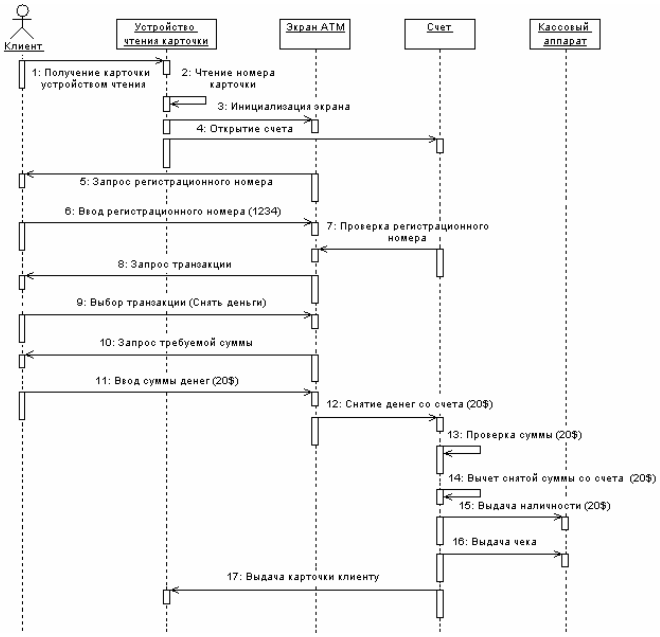
Пример диаграммы последовательности

Пример сценария снятия 20$ со счета (при отсутствии таких проблем, как неправильный идентификационный номер или недостаток денег на счету) показан на рис. 34.

Эта диаграмма последовательности отображает поток событий в рамках варианта использования “Снять деньги”. В верхней части диаграммы показаны все действующие лица и объекты, требуемые системе для выполнения варианта использования “Снять деньги”. Стрелки соответствуют сообщениям, передаваемым между действующим лицом и объектом или между объектами для выполнения требуемых функций. Следует отметить также, что на диаграмме Последовательности показаны именно объекты, а не классы. Классы представляют собой типы объектов. Объекты конкретны; вместо класса Клиент на диаграмме Последовательности представлен конкретный клиент Джо.

Вариант использования начинается, когда клиент вставляет свою карточку в устройство для чтения – этот объект показан в прямоугольнике в верхней части диаграммы. Он считывает номер

карточки, открывает объект “счет” (account) и инициализирует экран ATM. Экран запрашивает у клиента его регистрационный номер. Клиент вводит число 1234. Экран проверяет номер у объекта “счет” и обнаруживает, что он правильный. Затем экран предоставляет клиенту меню для выбора, и тот выбирает пункт *“Снять деньги”*. Экран запрашивает, сколько он хочет снять, и клиент указывает 20$. Экран снимает деньги со счета. При этом он инициирует серию процессов, выполняемых объектом “счет”. Во-первых, осуществляется проверка, что на этом счету лежат, по крайней мере, 20$. Во- вторых, из счета вычитается требуемая сумма. Затем кассовый аппарат получает инструкцию выдать чек и $20 наличными. Наконец все тот же объект



“счет ” дает устройству для чтения карточек инструкцию вернуть карточку.

Рис. 34 Диаграмма последовательности для снятия клиентом 20$

Таким образом, диаграмма последовательности иллюстрирует последовательность действий, реализующих вариант использования “Снять деньги со счета” на примере снятия клиентом 20$. Глядя на эту диаграмму, пользователи знакомятся со спецификой своей работы. Аналитики видят последовательность (поток) действий, разработчики – объекты, которые надо создать, и их операции. Специалисты по контролю качества поймут детали процесса и смогут разработать тесты для их проверки. Таким образом, диаграммы последовательности полезны всем участникам проекта.

Диаграммы кооперации (collaboration diagram)

Подобно диаграммам последовательности, *диаграммы кооперации* отображают поток событий в конкретном сценарии варианта использования. Главная особенность диаграммы кооперации заключается в возможности графически представить не только последовательность взаимодействия, но и все структурные отношения между объектами, участвующими в этом взаимодействии.

Прежде всего, на диаграмме кооперации в виде прямоугольников изображаются участвующие во взаимодействии объекты, содержащие имя объекта, его класс и, возможно, значения атрибутов. Далее, как и на диаграмме классов, указываются ассоциации между объектами в виде различных соединительных линий. При этом можно явно указать имена ассоциации и ролей, которые играют объекты в данной ассоциации. Дополнительно могут быть изображены динамические связи – потоки сообщений. Они представляются также в виде соединительных линий между объектами, над которыми располагается стрелка с указанием направления, имени сообщения и порядкового номера в общей последовательности инициализации сообщений.

В отличие от диаграммы последовательности, на диаграмме кооперации изображаются только отношения между объектами, играющими определенные роли во взаимодействии, а

последовательность взаимодействий и параллельных потоков определяется с помощью порядковых номеров.

Объекты диаграммы кооперации

Отдельные аспекты спецификации объектов как элементов диаграмм уже рассматривались ранее при описании диаграмм последовательности. Эти же объекты являются основными элементами из которых строится диаграмма кооперации. Для графического изображения объектов используется такой же символ прямоугольника, что и для классов.

Объект является отдельным экземпляром класса, который создается на этапе выполнения программы. Он может иметь свое собственное имя и конкретные значения атрибутов. Для обозначения роли классификатора небходимо указать либо имя класса (вместе с двоеточием), либо имя роли (вместе с наклонной чертой). В противном случае прямоугольник будет соответствовать обычному классу. Если роль, которую должен играть объект, наследуется от нескольких классов, то все они должны быть указаны явно и разделяться запятой и двоеточием.

Отдельные примеры изображения объектов и классов на диаграмме кооперации приводятся на рис. 35. В первом случае (рис. 35, а) обозначен объект с именем “клиент”, играющий роль “инициатор запроса”. Далее (рис. 35, б) следует обозначение анонимного объекта, который играет роль инициатора запроса. В обоих случаях не указан класс, на основе которого будут созданы эти объекты. Обозначение класса присутствует в следующем варианте записи (рис. 35, в), причем объект также анонимный.

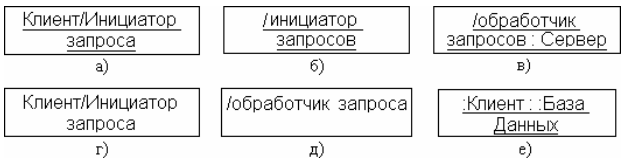


Рис. 35 Варианты записи имен объектов, ролей и классов на диаграммах кооперации

Применительно к уровню спецификации на диаграммах кооперации могут присутствовать именованные классы с указанием роли класса в кооперации (рис. 35, г) или анонимные классы, когда указывается только его роль (рис. 35, д). Последний случай характерен для ситуации, когда в модели могут присутствовать несколько классов с именем “Клиент”, поэтому требуется явно указать имя соответствующего пакета База данных (рис. 35, е).

Мультиобъект (multiobject) представляет собой целое множество объектов на одном из концов ассоциации (рис. 36, а). На диаграмме кооперации мультиобъект используется чтобы показать операции и сигналы, адресованные всему множеству объектов, а не только одному. При этом стрелка сообщения относится ко всему множеству объектов, которые обозначают данный мультиобъект. На диаграмме кооперации может быть явно указано отношение композиции между мультиобъектом и отдельным объектом из его множества (рис. 36, б).

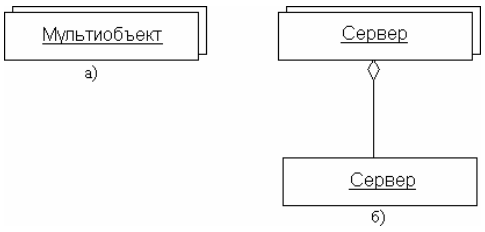


Рис. 36 Графическое изображение мультиобъектов на диаграмме кооперации

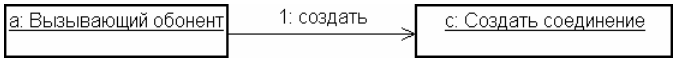
В контексте языка UML все объекты делятся на две категории: *пассивные* и *активные*. Пассивный объект оперирует только данными и не может инициировать деятельность по управлению другими объектами. В тоже время пассивные объекты могут посылать сигналы в процессе выполнения запросов, которые они получают.

Активный объект имеет свою собственную нить управления и может инициировать деятельность по управлению другими объектами. При этом под нитью понимается поток управления, который

может выполняться параллельно с другими вычислительными нитями или нитями управления в пределах одного вычислительного процесса.

В приведенном фрагменте диаграммы кооперации (рис. 37) активный объект “a: Вызывающий абонент” является инициатором процесса установления соединения для обмена информацией с другим абонентом.

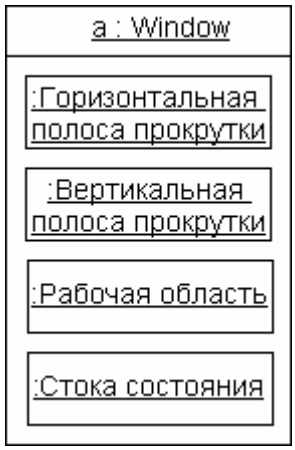
Рис. 37 Активный объект (слева) на диаграмме кооперации



Составной объект (composite object) или объект-контейнер предназначен для представления объекта, имеющего собственную структуру и внутренние потоки (нити) управления. Составной объект является экземпляром составного класса (класса-контейнера), который связан отношением агрегации или композиции со своими частями. Аналогичные отношения связывают между собой и соответствующие объекты.

На диаграммах кооперации составной объект состоит из двух секций: верхней и нижней. В верхней секции записывается имя составного объекта, а в нижней – его элементы (рис. 38), которые могут быть составными объектами.

Рис. 38 Составной объект на диаграмме кооперации



Связь (link) является экземпляром или примером произвольной ассоциации. Связь как элемент языка UML может иметь место между двумя и более объектами. Связь на диаграмме кооперации изображается отрезком прямой линии, соединяющей два прямоугольника объектов. На каждом из концов этой линии могут быть явно указаны имена ролей данной ассоциации.

Рядом с линией в ее средней части может записываться имя соответствующей ассоциации. Связи не имеют собственных имен, поскольку полностью идентичны как экземпляры ассоциации. Для связей не указывается также и кратность.

Применительно к диаграммам кооперации сообщения имеют некоторые дополнительные семантические особенности. Они определяют коммуникацию между двумя объектами, один из которых передает другому некоторую информацию. При этом первый объект ожидает, что после получения сообщения вторым объектом последует выполнение некоторого действия. Таким образом, именно сообщение является причиной или стимулом для начала выполнения операций, отправки сигналов, создания и уничтожения отдельных объектов. Связь обеспечивает канал для направленной передачи сообщений между объектами от объекта-источника к объекту-получателю.

Пример диаграммы кооперации

На рис. 39 приведена кооперативная диаграмма, описывающая, как клиент снимает со счёта 20$.

Рис. 39 Диаграмма кооперации для снятия клиентом 20$

Из Кооперативной диаграммы легче понять поток событий и отношения между объектами, однако труднее уяснить последовательность событий, поэтому для сценария создают диаграммы обоих типов.

Тема 10. Создание проекта на UML

*Назначение, синтаксис, **инструменты диаграмм компонентов и классов** на языке UML.*

Диаграммы классов(class diagram)

Диаграммы классов при моделировании объектно-ориентированных систем встречаются чаще других. На таких диаграммах отображается множество классов, интерфейсов, кооперации и отношений между ними. Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Кроме того, диаграммы классов составляют основу еще двух диаграмм – компонентов и развертывания.

Диаграмма классов может отражать различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы.

Компоненты диаграммы классов

Классом называется описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. Например, класс “Стена” описывает объекты с общими свойствами: высотой, длиной, толщиной, и т.д. При этом конкретные стены будут рассматриваться как отдельные экземпляры класса «стена». У каждого класса есть имя, (простое или составное, к которому спереди добавлено имя пакета, в который входит класс). Имя класса в пакете должно быть уникальным. Класс реализует один или несколько интерфейсов.

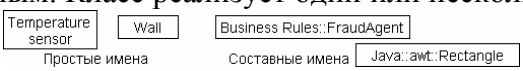


Рис. 40 Простые и составные имена

Атрибут – это именованное свойство класса, включающее описание множества значений, которые могут принимать экземпляры этого класса. Класс может иметь любое число атрибутов или не иметь их вовсе. В языках высокого уровня, таких, как C++, Java, атрибуты соответствуют переменным, объявленным в классе. Например, у любой стены есть высота, ширина и толщина. Атрибуты представлены в разделе, расположенном под именем класса; при этом указываются их имена, и иногда начальное значение (рис. 41).

Рис. 41 Атрибуты и их класс

Операция – это некоторый сервис, который предоставляет экземпляр или объект класса по требованию своих клиентов (других объектов, в том числе и экземпляров данного класса). Класс может содержать любое число операций или не содержать их вовсе. В языках высокого уровня, таких, как C++, Java, операции соответствуют функциям, объявленным в классе. Операцию можно описатьболее подробно, указав имена и типы параметров, их значения, принятые по умолчанию, а также тип возвращаемого значения (рис. 42).

Рис. 42 Операции

При изображении класса необязательно сразу показывать все его атрибуты и операции. Их может быть много, однако для данного представления системы лишь небольшое подмножество атрибутов и операций имеет значение. В это случае класс сворачивают, и изображают только некоторые из атрибутов и операций, а дополнительные атрибуты или операции обозначают многоточием. Для лучшей организации списков атрибутов и операций

можно снабдить каждую группу дополнительным описанием (стереотипами).

Обязанности класса – это своего рода контракт, которому он должен подчиняться. Атрибуты и операции являются свойствами, посредством которых выполняются обязанности класса. Например, класс FraudAgent (агент по предотвращению мошенничества), который встречается в приложениях по обработке кредитных карточек, отвечает за оценку платежных требований – законные, подозрительные или подложные (рис. 43). Моделирование классов лучше всего начинать с определения обязанностей сущностей, которые входят в словарь системы. Число обязанностей класса может быть произвольным, но на практике хорошо структурированный класс имеет по меньшей мере одну обязанность; с другой стороны, их не должно быть и слишком много. При уточнении модели обязанности класса преобразуются в совокупность атрибутов и операций, которые должны наилучшим образом обеспечить их выполнение.

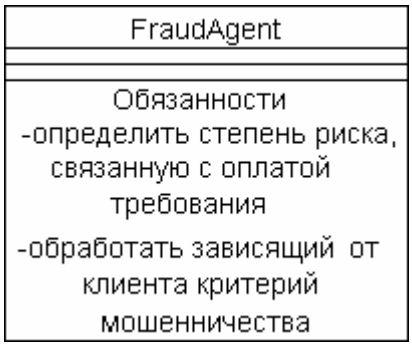


Рис. 43 Обязанности

Классы редко существуют автономно, они взаимодействуют между собой. Это значит, что, при моделировании системы, необходимо идентифицировать не только сущности, составляющие ее словарь, но и описать, как они соотносятся друг с другом. Существует основные три вида отношений между классами: *зависимости, обобщения, ассоциации* (рис. 44).

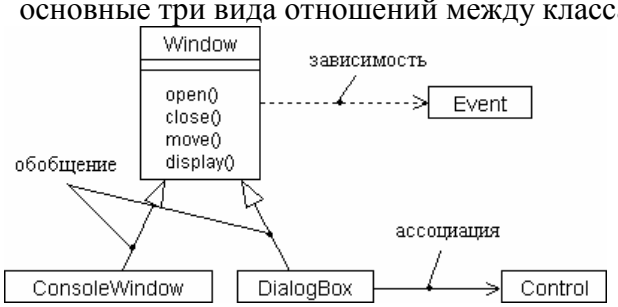


Рис. 44 Отношения

Кроме перечисленных отношений на диаграммах классов также применяются отношения агрегации и композиции.

Отношение агрегации применяется для представления системных взаимосвязей типа "часть-целое". Например, деление персонального компьютера на составные части: системный блок, монитор, клавиатуру и мышь (рис. 45).

Рис. 45 Отношения агрегации

Отношение композиции является частным случаем агрегации. Оно служит для выделения специальной формы отношения "часть-целое", при которой составляющие части в некотором смысле находятся внутри целого. Специфика взаимосвязи между ними заключается в том, что части не могут выступать в отрыве от целого, т.е. с уничтожением целого уничтожаются и все его составные части. Например – окно интерфейса программы, состоящее из строки заголовка, кнопок управления размером, полос прокрутки, главного меню, рабочей области и строки состояния (рис. 46).

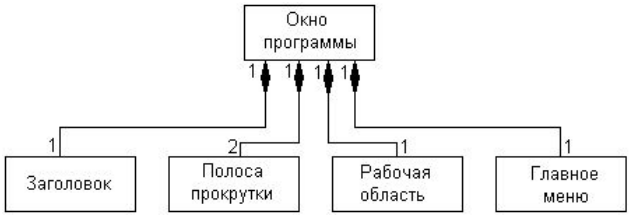


Рис. 46 Отношения композиции

Кроме описанных выше элементов, на диаграмме классов также могут присутствовать примечания, дополнения, стереотипы, помеченные значения, ограничения.

Стереотипом называют расширение словаря UML, позволяющее создавать новые виды строительных блоков, аналогичные существующим, но специфичные для данной задачи. Стереотип представлен в виде имени, заключенного в кавычки и расположенного над именем другого элемента. С помощью стереотипа создаётся новый строительный блок, который напоминает существующий, но обладает новыми свойствами (рис. 47).

Рис. 47 Стереотипы

Помеченное значение позволяет включать новую информацию в спецификацию элемента. Например, при создании нескольких версий ПО необходимо отслеживать версию и автора какой-нибудь важной абстракции. Ни версия, ни автор не являются первичными концепциями UML, но их можно добавить к любому блоку, например, к классу, задавая для него новые помеченные значения (рис. 48).

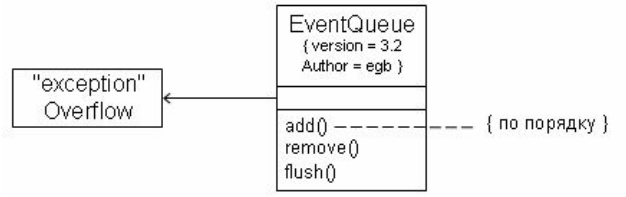


Рис. 48 Помеченное значение

Ограничение – это расширение семантики элемента UML, позволяющее создавать новые или изменять существующие правила (рис. 49).

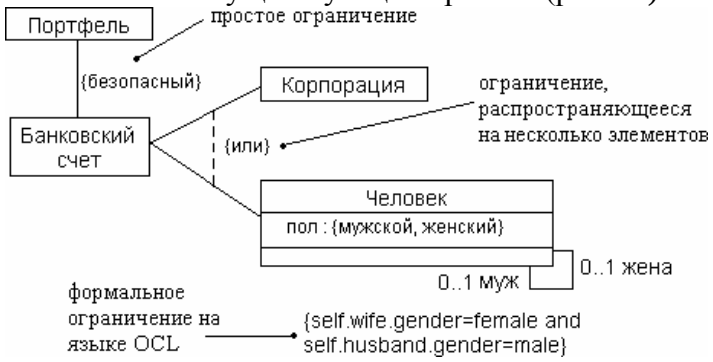


Рис. 49 Ограничения

Каждый элемент языка UML имеет свою семантику. С помощью ограничений можно создавать новую семантику или изменять существующие семантические правила. Например, на рис. 49 показано, что информация, передаваемая между классами “Портфель” и “Банковский

счёт", зашифрована. При моделировании систем реального времени часто используются временные и пространственные ограничения.

Прямое и обратное проектирование

Основным результатом деятельности группы разработчиков являются не диаграммы, а программное обеспечение, поэтому модели и основанные на них реализации должны соответствовать друг другу с минимальными затратами по поддержанию синхронизации между ними. Чаще всего разработанные модели преобразуются в программный код. Хотя UML не определяет конкретного способа отображения на какой-либо объектно-ориентированный язык, он проектировался с учетом этого требования. В наибольшей степени это относится к диаграммам классов, содержание которых без труда отображается на такие известные объектно-ориентированные языки программирования, как Java, C++, ObjectPascal, Visual Basic и др.

Прямым проектированием (Forward engineering) называется процесс преобразования модели в код путем отображения на некоторый язык реализации. Процесс прямого проектирования приводит к потере информации, поскольку написанные на языке UML модели семантически богаче любого из существующих объектно-ориентированных языков. Фактически именно это различие и является основной причиной необходимости моделей. Некоторые структурные свойства системы, такие как кооперации, или ее поведенческие особенности, например взаимодействия, могут быть легко визуализированы в UML, но в чистом коде наглядность теряется.

При прямом проектировании диаграммы классов следует учитывать, что так как модели зависят от семантики выбранного языка программирования, вероятно, придется отказаться от использования некоторых возможностей UML. Например, язык UML допускает множественное наследование, а язык программирования Smalltalk – только одиночное. В связи с этим можно запретить авторам моделей пользоваться множественным наследованием. Для специфицирования языка программирования применяются помеченные значения как на уровне индивидуальных классов (если нужна тонкая настройка), так и на более высоком уровне, например для пакетов или кооперации.

На рис. 50 изображена диаграмма классов, на которой проиллюстрирована реализация образца цепочки обязанностей. В данном примере представлены три класса: Client (Клиент), EventHandler (ОбработчикСобытий) и GUIEventHandler (ОбработчикСобытийГИП). Первые два из них являются абстрактными, а последний – конкретным. Класс EventHandler содержит обычную для данного образца операцию handleRequest (ОбработатьЗапрос), хотя в рассматриваемом случае было добавлено два скрытых атрибута.

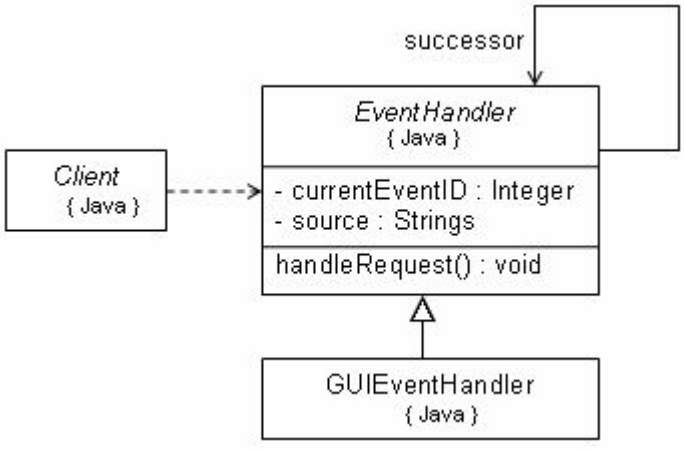


Рис. 50 Прямое проектирование

Определенное для каждого класса помеченное значение показывает, что они будут преобразованы в код на языке Java. Прямое проектирование в данном случае легко осуществимо с помощью специального инструмента. Так, для класса EventHandler будет сгенерирован следующий код:

```
public abstract class EventHandler { EventHandler successor;
private Integer currentEventID; private String source;
```

```
EventHandler() {}
  public void handleRequest ()  {}
}
```

Обратным проектированием (Reverse engineering) называется процесс преобразования в модель кода, записанного на каком-либо языке программирования. В результате этого процесса вы получаете огромный объем информации, часть которой находится на более низком уровне детализации, чем необходимо для построения полезных моделей. В то же время обратное проектирование никогда не бывает полным. Как уже упоминалось, прямое проектирование ведет к потере информации, так что полностью восстановить модель на основе кода не удастся, если только инструментальные средства не включали в комментариях к исходному тексту информацию, выходящую за пределы семантики языка реализации.

Обратное проектирование диаграммы классов осуществляется следующим образом:

1. Идентифицируются правила для преобразования из выбранного языка реализации. Это можно сделать на уровне проекта или организации в целом.

2. С помощью инструментального средства указывается код, который будет подвергнут обратному проектированию.

3. Используя инструментальные средства, создаётся диаграмма классов путем опроса полученной модели. Следует начать, например, с одного или нескольких классов, а затем расширить диаграмму, следуя вдоль некоторых отношений или добавив соседние классы. При этом можно раскрыть или спрятать детали содержания диаграммы в зависимости от ваших намерений.

Примеры диаграмм классов

На рис. 51 показана совокупность классов, взятых из информационной системы вуза. Этот рисунок содержит достаточное количество деталей для конструирования физической базы данных. В нижней части диаграммы расположены классы Студент, Курс и Преподаватель. Между классами Студент и Курс есть ассоциация, показывающая, что студенты могут посещать курсы. Более того, каждый студент может посещать любое количество курсов и на каждый курс может записаться любое число студентов.

Два класса (Вуз и Факультет) содержат несколько операций, позволяющих манипулировать их частями. Эти операции включены из-за их важности для поддержания целостности данных (например, добавление или удаление Факультета затрагивает целый ряд других объектов).

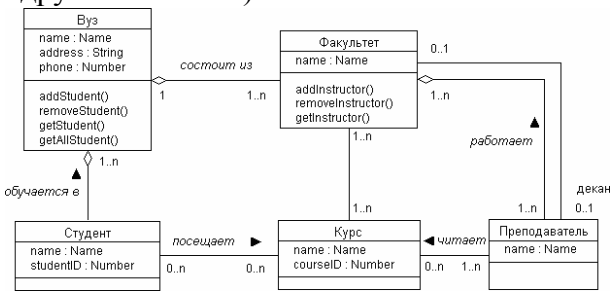


Рис. 51 Диаграмма классов для информационной системы ВУЗа

Существует много других операций, которые стоило бы рассмотреть при проектировании этих и иных классов, например запрос о необходимых предварительных знаниях перед зачислением студента на курс. Но это, скорее, бизнес-правила, а не операции по поддержанию целостности данных, поэтому лучше располагать их на более высоком уровне абстракции.

На рис. 52 представлен пример диаграммы классов структуры компании.

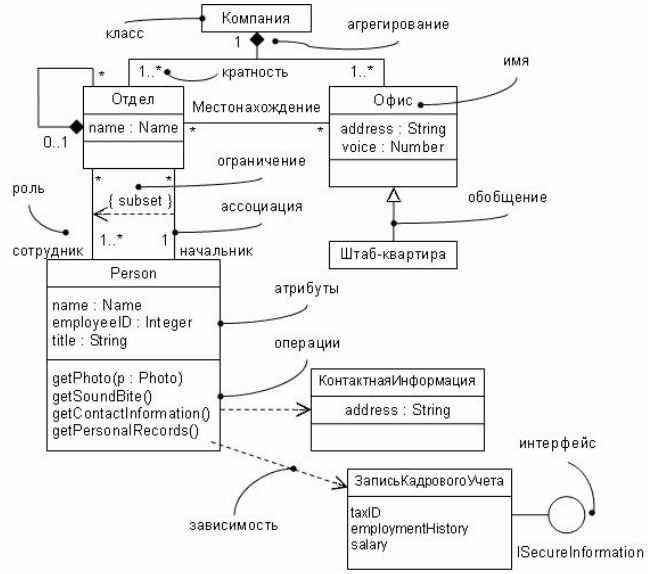


Рис. 52 Диаграмма классов

Диаграммы компонентов (component diagram)

Все рассмотренные ранее диаграммы отражали концептуальные аспекты построения модели системы и относились к логическому уровню представления. *Диаграмма компонентов* описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними.

Диаграмма компонентов разрабатывается для следующих целей:

- визуализации общей структуры исходного кода программной системы;
- спецификации исполнимого варианта программной системы;
- обеспечения многократного использования отдельных фрагментов программного кода;
- представления концептуальной и физической схем баз данных.

Диаграмма компонентов обеспечивает согласованный переход от логического представления к конкретной реализации проекта в форме программного кода. Одни компоненты могут существовать только на этапе компиляции программного кода, другие – на этапе его исполнения. Диаграмма компонентов отражает общие зависимости между компонентами, рассматривая последние в качестве классификаторов.

8.1 Основные графические элементы диаграммы компонентов

Для представления физических сущностей в языке UML применяется специальный термин – *компонент* (component). Компонент реализует некоторый набор интерфейсов и служит для общего обозначения элементов физического представления модели. Для графического представления компонента может использоваться специальный символ – прямоугольник со вставленными слева двумя более мелкими прямоугольниками (рис. 63). Внутри объемлющего прямоугольника записывается имя компонента и, возможно, некоторая дополнительная информация.

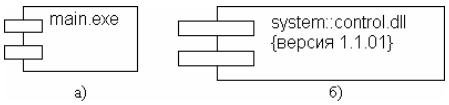


Рис. 63 Графическое изображение компонента в языке UML

В первом случае (рис. 63, а) с компонентом уровня экземпляра связывается только его имя, а во втором (рис. 63, б) – дополнительно имя пакета и помеченное значение.

В языке UML выделяют три вида компонентов:

- *компоненты развертывания*, которые обеспечивают непосредственное выполнение системой своих функций: динамически подключаемые библиотеки с расширением dll, Web-страницы на языке разметки гипертекста с расширением html и файлы справки с расширением hlp.
- *компоненты-рабочие продукты*: файлы с исходными текстами программ, например, с расширениями h или cpp для языка C++.
- *компоненты исполнения*, представляющие исполнимые модули – файлы с расширением exe.

Следующим элементом диаграммы компонентов являются *интерфейс*. Этот элемент уже рассматривался ранее, поэтому отметим только его особенности, которые характерны для представления на диаграммах компонентов. В общем случае интерфейс графически изображается окружностью, которая соединяется с компонентом отрезком линии без стрелок (рис. 64, а). Семантически линия означает реализацию интерфейса, а наличие интерфейсов у

компонента означает, что данный компонент реализует соответствующий набор интерфейсов.

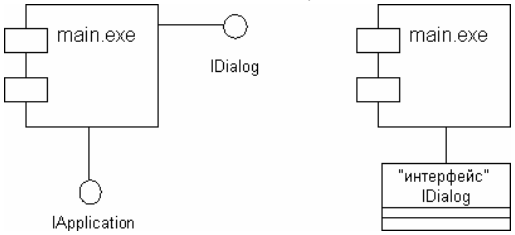


Рис. 64 Графическое изображение интерфейсов на диаграмме компонентов Другим способом

представления интерфейса на диаграмме компонентов
является его изображение в виде прямоугольника класса со стереотипом "интерфейс" и возможными секциями атрибутов и операций (рис. 64, б). Как правило, этот вариант обозначения используется для представления внутренней структуры интерфейса, которая может быть важна для реализации.

Применительно к диаграмме компонентов зависимости могут связывать компоненты и импортируемые этим компонентом интерфейсы, а также различные виды компонентов между собой.

В первом случае рисуют стрелку от компонента-клиента к импортируемому интерфейсу (рис. 65). Наличие такой стрелки означает, что компонент не реализует соответствующий интерфейс, а использует его в процессе своего выполнения. Причем на этой же диаграмме может присутствовать и другой компонент, который реализует этот интерфейс. Так, например, изображенный ниже фрагмент диаграммы компонентов представляет информацию о том, что компонент с именем "main.exe" зависит от импортируемого интерфейса IDialog, который, в свою очередь, реализуется компонентом с именем "image.java". Для второго компонента этот же интерфейс является экспортируемым.

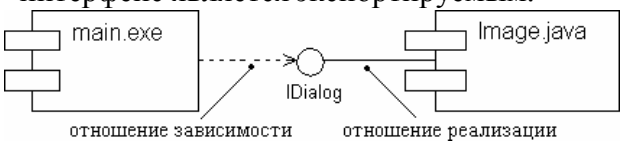


Рис. 65. Фрагмент диаграммы компонентов с отношением зависимости

На диаграмме компонентов также могут быть представлены отношения зависимости между компонентами и реализованными в них классами (рис. 66). Эта информация имеет важное значение для обеспечения согласования логического и физического представлений модели системы.

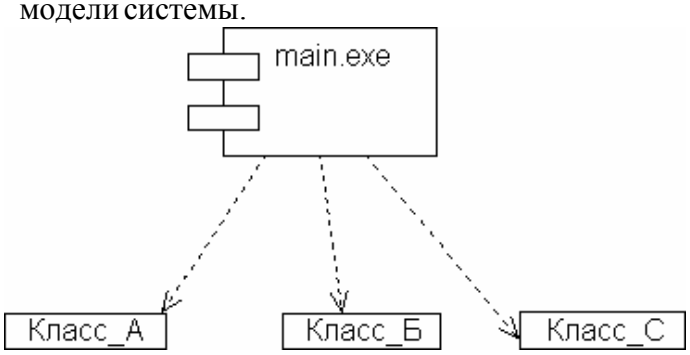


Рис. 66 Графическое изображение зависимости между компонентом и классами

Диаграммы развертывания (deployment diagram)

Физическое представление программной системы не может быть полным, если отсутствует информация о том, на какой платформе и на каких вычислительных средствах она реализована. *Диаграмма развертывания* предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения (runtime). При этом представляются только компоненты-экземпляры программы, являющиеся исполнимыми файлами или динамическими библиотеками. Те компоненты, которые не используются 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, диаграмма развертывания является единой для системы в целом, поскольку должна всецело отражать особенности ее реализации. Эта диаграмма, по сути, завершает процесс ООАП для конкретной программной системы и ее разработка, как правило, является последним этапом спецификации модели.

Элементы диаграммы компонентов

К основным элементам диаграммы развертывания относятся узлы и соединения.

Узел (node) представляет собой некоторый физически существующий элемент системы, обладающий некоторым вычислительным ресурсом. В качестве вычислительного ресурса узла может рассматриваться наличие по меньшей мере некоторого объема электронной или магнитооптической памяти и/или процессора. Понятие узла также может включать в себя и другие механические или электронные устройства, такие как датчики, принтеры, модемы, цифровые камеры, сканеры и манипуляторы.

Графически на диаграмме развертывания узел изображается в форме трехмерного куба. Узел имеет собственное имя, которое указывается внутри этого графического символа. Сами узлы могут представляться как в качестве типов (рис. 67, а), так и в качестве экземпляров (рис. 67, б).

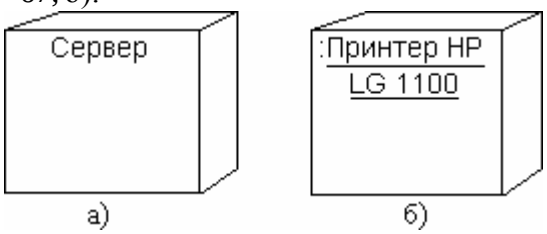


Рис. 67 Графическое изображение узла на диаграмме развертывания

Помеченное значение – это расширение свойств элемента UML, позволяющее вводить новую информацию в его спецификацию. У каждой сущности в UML есть фиксированный набор свойств: классы имеют имена, атрибуты и операции; ассоциации-имена и концевые точки (каждая со своими свойствами) и т.д. Помеченные значения позволяют добавлять новые свойства.

Например, как показано на рис. 68, в диаграмме развертывания можно указать число процессоров, установленных на узле каждого вида, или потребовать, чтобы каждому компоненту был приписан стереотип библиотеки, если его предполагается развернуть на клиенте или сервере.

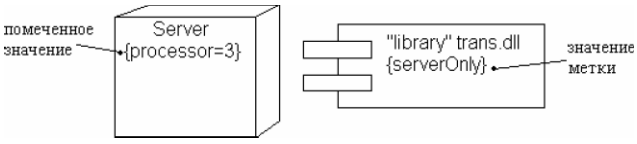


Рис. 68 Помеченные значения

Так же, как и на диаграмме компонентов, изображения узлов могут расширяться, чтобы включить некоторую дополнительную информацию о спецификации узла. Если дополнительная информация относится к имени узла, то она записывается под этим именем в форме помеченного значения (рис. 69).

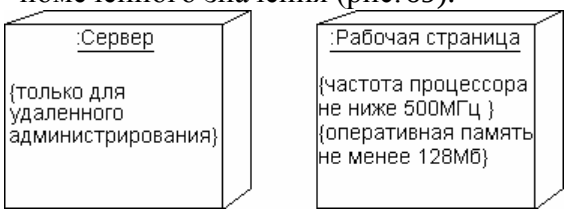


Рис. 69 Графическое изображение узла-экземпляра с дополнительной информацией в форме помеченного значения

Соединения указывают отношения между узлами и являются разновидностью ассоциации. Изображаются отрезками линий без стрелок. Наличие такой линии указывает на необходимость организации физического канала для обмена информацией между соответствующими узлами. Характер соединения может быть дополнительно специфицирован примечанием, помеченным значением или ограничением (рис. 70). В рассмотренном примере явно определены не только требования к скорости передачи данных в локальной сети с помощью помеченного значения, но и рекомендации по технологии физической реализации соединений в форме примечания.

Рис. 70 Фрагмент диаграммы развертывания с соединениями между узлами

Кроме соединений на диаграмме развертывания могут присутствовать отношения зависимости между узлом и развернутыми на нем компонентами. Подобный способ является альтернативой вложенному изображению компонентов внутри символа узла, что не всегда удобно, поскольку делает этот символ излишне объемным (рис. 71).

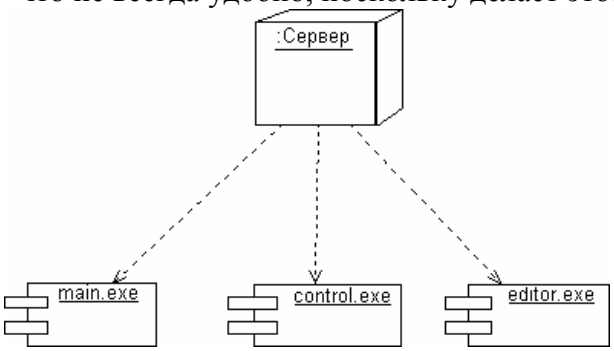


Рис. 71 Диаграмма развертывания с отношением зависимости между узлом и развернутыми на нем компонентами

Пример диаграммы развертывания

Рассмотрим фрагмент физического представления системы удаленного обслуживания клиентов банка (рис. 72).

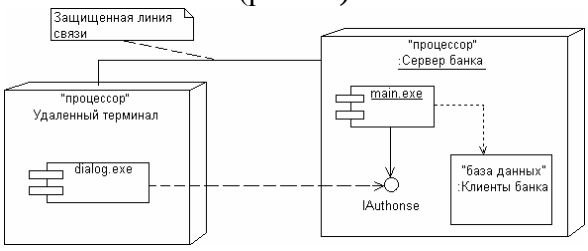


Диаграмма развертывания для системы удаленного обслуживания клиентов банка

На диаграмме развертывания узлами системы являются удаленный терминал (узел-тип) и сервер банка (узел-экземпляр). Указана зависимость компонента реализации диалога "dialog.exe" на удаленном терминале от интерфейса lAuthorise, реализованного компонентом "main.exe", который, в свою очередь, развернут на анонимном узле-экземпляре "Сервер банка". Последний зависит от компонента базы данных "Клиенты банка", который развернут на этом же узле. Примечание указывает на необходимость использования защищенной линии связи для обмена данными в данной системе. Другой вариант записи этой информации заключается в дополнении диаграммы узлом со стереотипом "закрытая сеть".