26. Диаграмма компонентов (component diagram). Компоненты. Интерфейсы. Зависимости.

http://www.info-system.ru/designing/methodology/uml/theory/component_diagram_theory.html

http://book.uml3.ru/sec_3_4

http://khpi-iip.mipk.kharkiv.edu/library/case/leon/gl10/gl10.html

Большинство диаграмм UML отражают концептуальные аспекты построения модели системы и относятся к логическому уровню представления. Особенность логического представления заключается в том, что оно оперирует понятиями, которые не имеют самостоятельного материального воплощения. Другими словами, различные элементы логического представления, такие как классы, ассоциации, состояния, сообщения, не существуют материально или физически. Они лишь отражают наше понимание структуры физической системы или аспекты ее поведения. Основное назначение логического представления состоит в анализе структурных и функциональных отношений между элементами модели системы. Однако для создания конкретной физической системы необходимо некоторым образом реализовать все элементы логического представления в конкретные материальные сущности. Для описания таких реальных сущностей предназначен другой аспект модельного представления, а именно физическое представление модели.

В языке UML для физического представления моделей систем используются так называемые диаграммы реализации (implementation diagrams), которые включают в себя две отдельные канонические диаграммы: диаграмму компонентов и диаграмму развертывания.

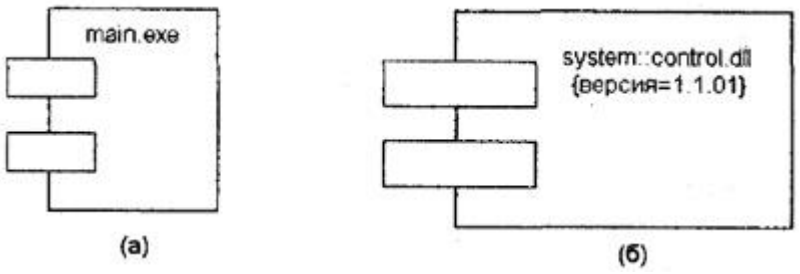
Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код.

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.

Компоненты

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. Внутри объемлющего прямоугольника записывается имя компонента и, возможно, некоторая дополнительная информация. Изображение этого символа может незначительно варьироваться в зависимости от характера ассоциируемой с компонентом информации.



(а) (б)

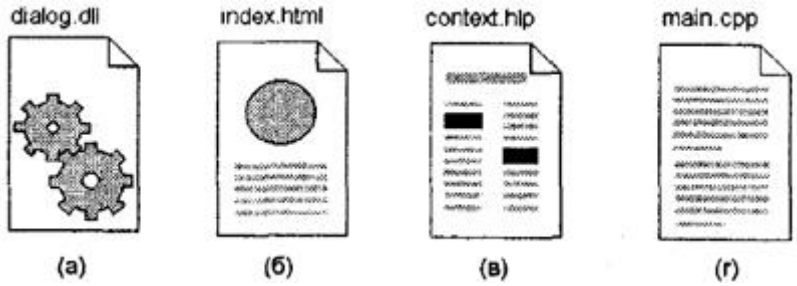
Компонент может иметь свои собственные свойства, такие как атрибуты и операции. Так, в первом случае (а) с компонентом уровня экземпляра связывается только его имя, а во втором (б) - дополнительно имя пакета и помеченное значение.

Виды компонентов

Поскольку компонент как элемент физической реализации модели представляет отдельный модуль кода, иногда его комментируют с указанием дополнительных графических символов, иллюстрирующих конкретные особенности его реализации. Строго говоря, эти дополнительные обозначения для примечаний не специфицированы в языке UML. Однако их применение упрощает понимание диаграммы компонентов, существенно повышая наглядность физического представления. Некоторые из таких общепринятых обозначений для компонентов изображены ниже.

В языке UML выделяют три вида компонентов.

- Компоненты развертывания, которые обеспечивают непосредственное выполнение системой своих функций. Такими компонентами могут быть динамически подключаемые библиотеки с расширением dll (рис. 10.2, а), Web-страницы на языке разметки гипертекста с расширением html (рис. 10.2, б) и файлы справки с расширением hlp (рис. 10.2, в).
- Компоненты-рабочие продукты. Как правило - это файлы с исходными текстами программ, например, с расширениями h или cpp для языка C++ (рис. 10.2, г).
- Компоненты исполнения, представляющие исполнимые модули - файлы с расширением exe. Они обозначаются обычным образом.



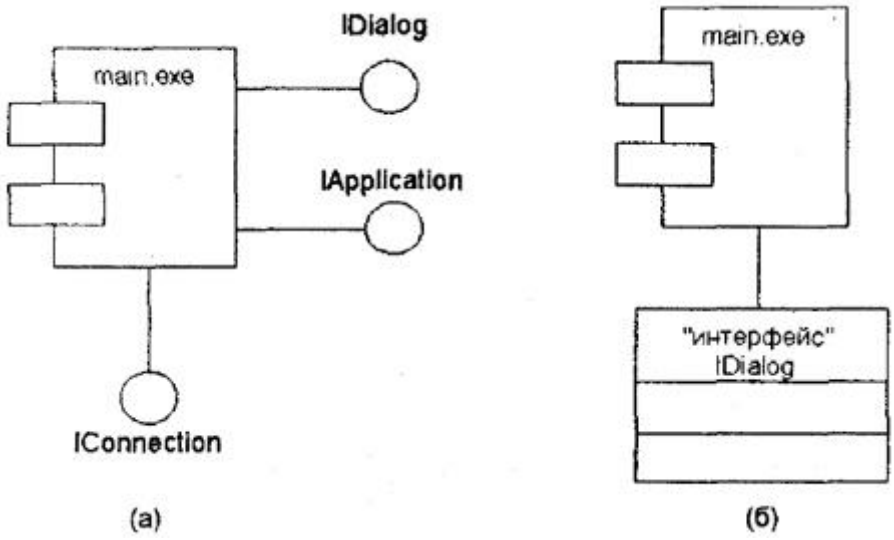
Другой способ спецификации различных видов компонентов - явное указание стереотипа компонента перед его именем. В языке UML для компонентов определены следующие стереотипы:

- Библиотека (library) - определяет первую разновидность компонента, который представляется в форме динамической или статической библиотеки.
- Таблица (table) - также определяет первую разновидность компонента, который представляется в форме таблицы базы данных.
- Файл (file) - определяет вторую разновидность компонента, который представляется в виде файлов с исходными текстами программ.
- Документ (document) - определяет вторую разновидность компонента, который представляется в форме документа.
- Исполнимый (executable) - определяет третий вид компонента, который может исполняться в узле.

Интерфейсы

В общем случае интерфейс графически изображается окружностью, которая соединяется с компонентом отрезком линии без стрелок (а). При этом имя интерфейса, которое обязательно должно начинаться с заглавной буквы "I", записывается рядом с окружностью. Семантически

линия означает реализацию интерфейса, а наличие интерфейсов у компонента означает, что данный компонент реализует соответствующий набор интерфейсов.



Графическое изображение интерфейсов на диаграмме компонентов

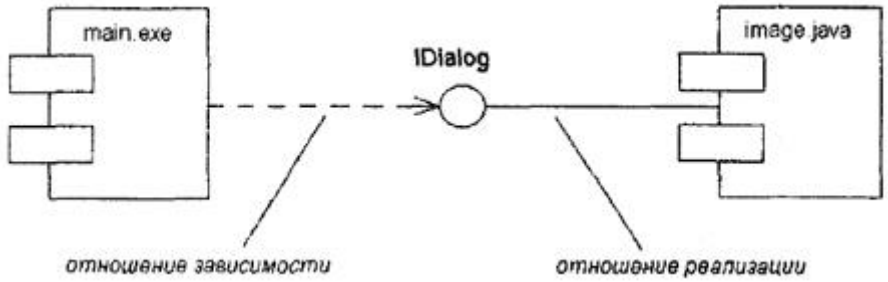
Другим способом представления интерфейса на диаграмме компонентов является его изображение в виде прямоугольника класса со стереотипом "интерфейс" и возможными секциями атрибутов и операций (б). Как правило, этот вариант обозначения используется для представления внутренней структуры интерфейса, которая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

Характер использования интерфейсов отдельными компонентами может отличаться. Поэтому различают два способа связи интерфейса и компонента. Если компонент реализует некоторый интерфейс, то такой интерфейс называют экспортируемым, поскольку этот компонент предоставляет его в качестве сервиса другим компонентам. Если же компонент использует некоторый интерфейс, который реализуется другим компонентом, то такой интерфейс для первого компонента называется импортируемым. Особенность импортируемого интерфейса состоит в том, что на диаграмме компонентов это отношение изображается с помощью зависимости.

Зависимости

Зависимость не является ассоциацией, а служит для представления только факта наличия такой связи, когда изменение одного элемента модели оказывает влияние или приводит к изменению другого элемента модели. Отношение зависимости на диаграмме компонентов изображается пунктирной линией со стрелкой, направленной от клиента (зависимого элемента) к источнику (независимому элементу).
Зависимости могут отражать связи модулей программы на этапе компиляции и генерации объектного кода. В другом случае зависимость может отражать наличие в независимом компоненте описаний классов, которые используются в зависимом компоненте для создания соответствующих объектов. 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. ниже). 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 I Dialog, который, в свою

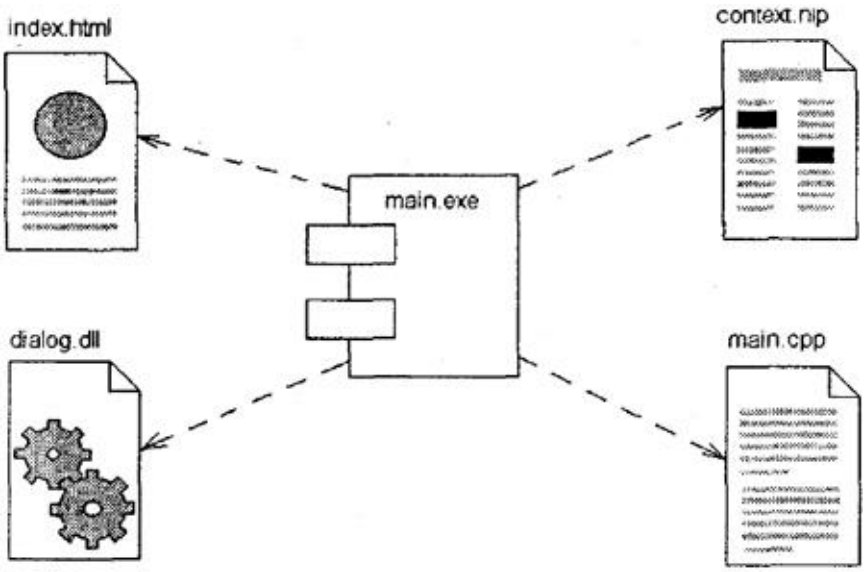
очередь, реализуется компонентом с именем "image.java". Для второго компонента этот же интерфейс является экспортируемым.



Фрагмент диаграммы компонентов с отношением зависимости

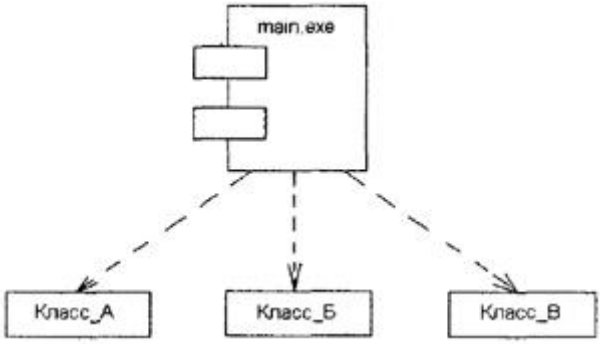
Заметим, что изобразить второй компонент с именем "image.java" в форме варианта примечания нельзя именно в силу того факта, что этот компонент реализует интерфейс.

Другим случаем отношения зависимости на диаграмме компонентов является отношение между различными видами компонентов. Наличие подобной зависимости означает, что внесение изменений в исходные тексты программ или динамические библиотеки приводит к изменениям самого компонента. При этом характер изменений может быть отмечен дополнительно.



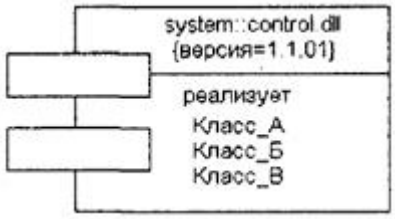
Графическое изображение отношения зависимости между компонентами

Наконец, на диаграмме компонентов могут быть представлены отношения зависимости между компонентами и реализованными в них классами. Эта информация имеет важное значение для обеспечения согласования логического и физического представлений модели системы. Разумеется, изменения в структуре описаний классов могут привести к изменению компонента.



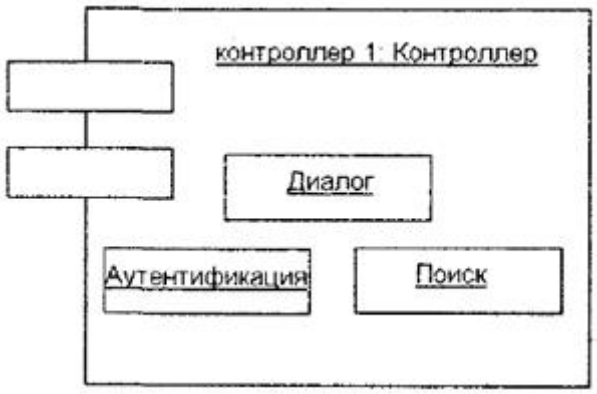
Графическое изображение зависимости между компонентом и классами

Следует заметить, что в данном случае из диаграммы компонентов не следует, что классы реализованы этим компонентом. Если требуется подчеркнуть, что некоторый компонент реализует отдельные классы, то для обозначения компонента используется расширенный символ прямоугольника. При этом прямоугольник компонента делится на две секции горизонтальной линией. Верхняя секция служит для записи имени компонента, а нижняя секция - для указания дополнительной информации.



Графическое изображение компонента с дополнительной информацией о реализуемых им классах

Внутри символа компонента могут изображаться другие элементы графической нотации, такие как классы (компонент уровня типа) или объекты (компонент уровня экземпляра). В этом случае символ компонента изображается таким образом, чтобы вместить эти дополнительные символы. Так, например, изображенный ниже компонент является экземпляром и реализует три отдельных объекта.



Графическое изображение компонента уровня экземпляра, реализующего отдельные объекты

Рекомендации по построению диаграммы компонентов

До начала разработки необходимо принять решения о выборе вычислительных платформ и операционных систем, на которых предполагается реализовывать систему, а также о выборе конкретных баз данных и языков программирования.

После этого можно приступать к общей структуризации диаграммы компонентов. В первую очередь, необходимо решить, из каких физических частей (файлов) будет состоять программная система. На этом этапе следует обратить внимание на такую реализацию системы, которая обеспечивала бы не только возможность повторного использования кода за счет рациональной декомпозиции компонентов, но и создание объектов только при их необходимости.

Речь идет о том, что общая производительность программной системы существенно зависит от рационального использования ею вычислительных ресурсов. Для этой цели необходимо большую часть описаний классов, их операций и методов вынести в динамические библиотеки, оставив в исполняемых компонентах только самые необходимые для инициализации программы фрагменты программного кода.

После общей структуризации физического представления системы необходимо дополнить модель интерфейсами и схемами базы данных. При разработке интерфейсов следует обращать внимание на согласование (стыковку) различных частей программной системы. Включение в модель схемы базы данных предполагает спецификацию отдельных таблиц и установление информационных связей между таблицами.

Наконец, завершающий этап построения диаграммы компонентов связан с установлением и нанесением на диаграмму взаимосвязей между компонентами, а также отношений реализации. Эти отношения должны иллюстрировать все важнейшие аспекты физической реализации системы, начиная с особенностей компиляции исходных текстов программ и заканчивая исполнением отдельных частей программы на этапе ее выполнения.