# Диаграмма компонентов (component diagram)

Все рассмотренные ранее диаграммы отражали концептуальные аспекты построения модели системы и относились 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.

Полный проект программной системы представляет собой совокупность моделей логического и физического представлений, которые должны быть согласованы между собой. 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, в отличие от ранее рассмотренных диаграмм, 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. Во многих средах разработки модуль или компонент соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости, аналогичные тем, которые имеют место при компиляции исходных текстов программ. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними.

Диаграмма компонентов разрабатывается для следующих целей:

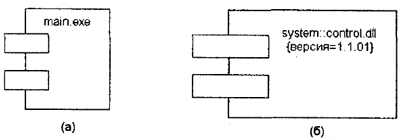
* Визуализации общей структуры исходного кода программной системы.
* Спецификации исполнимого варианта программной системы.
* Обеспечения многократного использования отдельных фрагментов программного кода.
* Представления концептуальной и физической схем баз данных.

В разработке диаграмм компонентов участвуют как системные аналитики и архитекторы, так и программисты. 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.

## 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 (рис. 1). 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.

В метамодели языка UML компонент является потомком классификатора. Он предоставляет организацию в рамках физического пакета ассоциированным с ним элементам модели. Как классификатор, компонент может иметь также свои собственные свойства, такие как атрибуты и операции.



**Рис. 1.** Графическое изображение компонента в языке UML

Так, в первом случае (рис. 1, а) с компонентом уровня экземпляра связывается только его имя, а во втором (рис. 1, б) - дополнительно имя пакета и помеченное значение. Необходимо заметить, что в этом случае номер версии записывается как помеченное значение в фигурных скобках. В других случаях символ компонента может быть разделен на секции, чтобы явно указать имена реализованных в нем интерфейсов. Такое обозначение компонента называется расширенным.

### Имя компонента

Имя компонента подчиняется общим правилам именования элементов модели в языке UML и может состоять из любого числа букв, цифр и некоторых знаков препинания. Отдельный компонент может быть представлен на уровне типа или на уровне экземпляра. Хотя его графическое изображение в обоих случаях одинаковое, правила записи имени компонента несколько отличаются. Если компонент представляется на уровне типа, то в качестве его имени записывается только имя типа с заглавной буквы.

Если же компонент представляется на уровне экземпляра, то в качестве его имени записывается <имя компонента ':' имя типаХ При этом вся строка имени подчеркивается.

В качестве простых имен принято использовать имена исполняемых файлов (с указанием расширения ехе после точки-разделителя), имена динамических библиотек (расширение dll), имена Web-страниц (расширение html), имена текстовых файлов (расширения txt или doc) или файлов справки (hip), имена файлов баз данных (DB) или имена файлов с исходными текстами программ (расширения h, cpp для языка C++, расширение Java для языка Java), скрипты (pi, asp) и др.

Поскольку конкретная реализация логического представления модели системы зависит от используемого программного инструментария, то и имена компонентов будут определяться особенностями синтаксиса соответствующего языка программирования.

### 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 (рис. 2).

В языке UML выделяют три вида компонентов.

* Во-первых, компоненты развертывания, которые обеспечивают непосредственное выполнение системой своих функций. Такими компонентами могут быть динамически подключаемые библиотеки с расширением dll (рис. 2, а), Web-страницы на языке разметки гипертекста с расширением html (рис. 2, б) и файлы справки с расширением htр (рис. 2, в).
* Во-вторых, компоненты-рабочие продукты. Как правило - это файлы с исходными текстами программ, например, с расширениями h или срр для языка C++ (рис. 2, г).
* В-третьих, компоненты исполнения, представляющие исполнимые модули - файлы с расширением ехе. Они обозначаются обычным образом.



**Рис. 2.** Варианты графического изображения компонентов на диаграмме компонентов

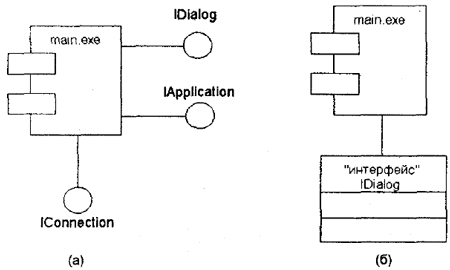
Эти элементы иногда называют артефактами, подчеркивая при этом их законченное информационное содержание, зависящее от конкретной технологии реализации соответствующих компонентов. Более того, разработчики могут для этой цели использовать самостоятельные обозначения, поскольку в языке UML нет строгой нотации для графического представления примечаний.

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

Следующим элементом диаграммы компонентов являются интерфейсы. Напомним, что в общем случае интерфейс графически изображается окружностью, которая соединяется с компонентом отрезком линии без стрелок (рис. 3, 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.



**Рис. 3.** 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. 3, б). Как правило, этот вариант обозначения используется для представления внутренней структуры интерфейса, которая может быть важна для реализации.

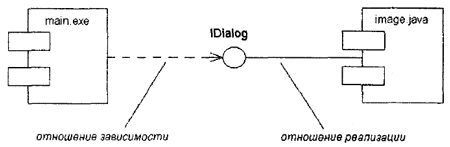
При разработке программных систем интерфейсы обеспечивают не только совместимость различных версий, но и возможность вносить существенные изменения в одни части программы, не изменяя другие ее части. Таким образом, назначение интерфейсов существенно шире, чем спецификация взаимодействия с пользователями системы (актерами).

## Зависимости

Напомним, что 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. 4). 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.



**Рис. 4.** Фрагмент диаграммы компонентов с отношением зависимости

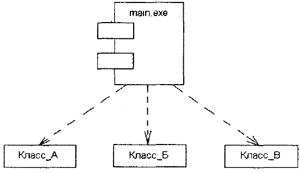
Заметим, что изобразить второй компонент с именем "image.java" в форме варианта примечания нельзя именно в силу того факта, что этот компонент реализует интерфейс.

Другим случаем отношения зависимости на диаграмме компонентов является отношение между различными видами компонентов (рис. 5). 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.



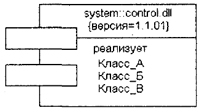
**Рис. 5.** Графическое изображение отношения зависимости между компонентами

Наконец, на диаграмме компонентов могут быть представлены отношения зависимости между компонентами и реализованными в них классами(рис. 6). 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. Ниже приводится фрагмент зависимости подобного рода, когда некоторый компонент зависит от соответствующих классов.



**Рис. 6.** Графическое изображение зависимости между компонентом и классами

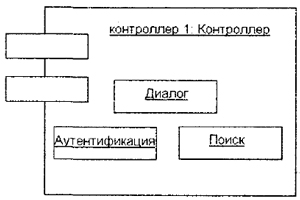
Следует заметить, что в данном случае из диаграммы компонентов не следует, что классы реализованы этим компонентом. Если требуется подчеркнуть, что некоторый компонент реализует отдельные классы, то для обозначения компонента используется расширенный символ прямоугольника. При этом прямоугольник компонента делится на две секции горизонтальной линией. Верхняя секция служит для записи имени компонента, а нижняя секция - для указания дополнительной информации (рис. 7).



**Рис. 7.** 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 (рис. 8) является экземпляром и реализует три отдельных объекта.



**Рис. 8.** Графическое изображение компонента уровня экземпляра, реализующего отдельные объекты

Объекты, которые находятся в отдельном компоненте-экземпляре, изображаются вложенными в символ данного компонента. Подобная вложенность означает, что выполнение компонента влечет выполнение соответствующих объектов. Другими словами, существование компонента в течение времени исполнения программы обеспечивает существование, а возможно, и доступ всех вложенных в него объектов. Что касается доступа к этим объектам, то он может быть дополнительно специфицирован с помощью квантификаторов видимости, подобно видимости пакетов. Так, для компонентов с исходным текстом программы видимость может означать возможность внесения изменений в соответствующие тексты программ с их последующей перекомпиляцией. Для компонентов с исполняемым кодом программы видимость может характеризовать возможность запуска на исполнение соответствующего компонента или вызова реализованных в нем операций или методов.