Лекция 3 часть 6:

Элементы графической нотации диаграммы компонентов

Аннотация: Назначение диаграммы компонентов, ее основные элементы. Особенности физического представления программных систем. Компоненты программных систем, их разновидности. Интерфейсы, их реализация компонентами. Использование диаграммы компонентов для проектирования зависимостей между компонентами. Рекомендации по построению диаграммы компонентов.

Ключевые слова: представление, UML, аппаратное обеспечение, физическая система, physical system, концептуальная схема базы, разбиение, каноническая диаграмма, диаграмма развертывания, диаграмма, компонент, исполняемый код, модуль, место,интерфейс, помеченное значение, специальный символ, прямоугольник, информация, инкапсуляция данных, запись, значение,рабочий продукт, очередь, interface, экспорт интерфейса, импорт интерфейса, отношение, зависимость компонент, диаграмма компонентов, control, связь, library, home, файл, Search, исполнение, производительность, базы данных

Диаграмма компонентов и особенности ее построения

Все рассмотренные ранее диаграммы отражали концептуальные и логические аспекты построения модели системы. Особенность логического представления заключается в том, что оно оперирует понятиями, которые не имеют материального воплощения. Другими словами, различные элементы логического представления, такие как классы, ассоциации, состояния, сообщения, не существуют материально или физически. Они лишь отражают понимание статической структуры той или иной системы или динамические аспекты ее поведения.

Для создания конкретной физической системы необходимо 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 контексте языка *UML* это означает совокупность связанных физических сущностей, включая программное и *аппаратное обеспечение*, а также персонал, которые организованы для выполнения специальных задач.

Физическая система (*physical system*) — реально существующий прототип модели системы.

С тем чтобы пояснить отличие логического и физического представлений, необходимо в общих чертах рассмотреть процесс разработки программной системы. Ее исходным логическим представлением могут служить структурные схемы алгоритмов и процедур, описания интерфейсов и *концептуальные схемы баз* данных. Однако для реализации этой системы необходимо разработать исходный текст программы на языке программирования. При этом уже в тексте программы предполагается организация программного кода, определяемая синтаксисом языка программирования и предполагающая *разбиение* исходного кода на отдельные модули.

Однако исходные тексты программы еще не являются окончательной реализацией проекта, хотя и служат фрагментом его физического представления. Программная система может считаться реализованной в том случае, когда она будет способна выполнять функции своего целевого предназначения. А это возможно, только если программный код системы будет реализован в

форме исполняемых модулей, библиотек классов и процедур, стандартных графических интерфейсов, файлов баз данных. Именно эти компоненты являются базовыми элементами физического представления системы в нотации языка *UML*.

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, 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.

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 отношений между ними.

Компоненты

Для представления физических сущностей в языке *UML* применяется специальный термин – *компонент*.

Компонент (component) — физически существующая часть системы, которая обеспечивает реализацию классов и отношений, а также функционального поведения моделируемой программной системы.

Компонент предназначен для представления физической организации ассоциированных с ним элементов модели. Дополнительно *компонент* может иметь текстовый стереотип и *помеченные значения*, а некоторые *компоненты* – собственное графическое *представление*. *Компонентом* может быть *исполняемый код* отдельного *модуля*, командные файлы или файлы, содержащие интерпретируемые скрипты.

Компонент служит для общего обозначения элементов физического представления модели и может реализовывать некоторый набор *интерфейсов*. Для графического представления *компонента* используется *специальный символ – прямоугольник* со вставленными слева двумя более мелкими прямоугольниками (рис. 3-6.1). Внутри объемлющего прямоугольника записывается имя *компонента* и, возможно, дополнительная *информация*. Этот символ является базовым обозначением *компонента* в языке *UML*.

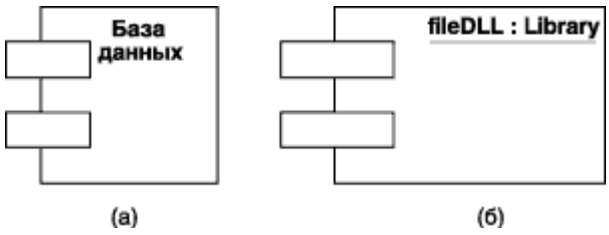

Рис. 3-6.1. Графическое изображение компонента

Графическое изображение *компонента* ведет свое происхождение от обозначения *модуля* программы, применявшегося некоторое время для отображения особенностей *инкапсуляции данных* и процедур.

Модуль (module) — часть программной системы, требующая памяти для своего хранения и процессора для исполнения.

В этом случае верхний маленький *прямоугольник* концептуально ассоциировался с данными, которые реализует этот *компонент*(иногда он изображается в форме овала). Нижний маленький *прямоугольник* ассоциировался с операциями или методами, реализуемыми *компонентом*. В простых случаях имена данных и методов записывались явно в маленьких прямоугольниках, однако в языке *UML* они не указываются.

Имя *компонента* подчиняется общим правилам именования элементов модели в языке *UML* и может состоять из любого числа букв, цифр и знаков препинания. Отдельный *компонент* может быть представлен на уровне типа или экземпляра. И хотя его графическое изображение в обоих случаях одинаково, правила записи имени *компонента* несколько отличаются.

Если *компонент* представляется на уровне типа, то записывается только имя типа с заглавной буквы в форме: <Имя типа>. Если же *компонент* представляется на уровне экземпляра, то его имя записывается в форме: <имя *компонента* ‘:' Имя типа>. При этом вся строка имени подчеркивается. Так, в первом случае (рис. 3-6.1, а) для *компонента* уровня типов указывается имя типа, а во втором (рис. 3-6.1, б) для *компонента* уровня экземпляра – собственное имя *компонента* и имя типа.

Правила именования объектов в языке *UML* требуют подчеркивания имени отдельных экземпляров, но применительно к *компонентам* подчеркивание их имени часто опускают. В этом случае *запись* имени *компонента* со строчной буквы характеризует *компонент* уровня примеров.

В качестве собственных имен *компонентов* принято использовать имена исполняемых файлов, динамических библиотек, Web-страниц, текстовых файлов или файлов справки, файлов баз данных или файлов с исходными текстами программ, файлов скриптов и другие.

В отдельных случаях к простому имени *компонента* может быть добавлена *информация* об имени объемлющего пакета и о конкретной версии реализации данного *компонента*. 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 классов или *интерфейсов*. Такое обозначение *компонента* называется **расширенным**.

Поскольку *компонент* как элемент модели может иметь различную физическую реализацию, иногда его изображают в форме специального графического символа, иллюстрирующего конкретные особенности реализации. Строго говоря, эти дополнительные обозначения не

специфицированы в нотации языка *UML*. Однако, удовлетворяя общим механизмам расширения языка *UML*, они упрощают понимание диаграммы *компонентов*, существенно повышая наглядность графического представления.

Для более наглядного изображения *компонентов* были предложены и стали общепринятыми следующие графические стереотипы:

- Во-первых, стереотипы для *компонентов* развертывания, которые обеспечивают непосредственное выполнение системой своих функций. Такими *компонентами* могут быть динамически подключаемые библиотеки (рис. 3-6.2, а), Web-страницы на языке разметки гипертекста (рис. 3-6.2, б) и файлы справки (рис. 3-6.2, в).
- Во-вторых, стереотипы для *компонентов* в форме *рабочих продуктов*. Как правило – это файлы с исходными текстами программ (рис. 3-6.2, г).

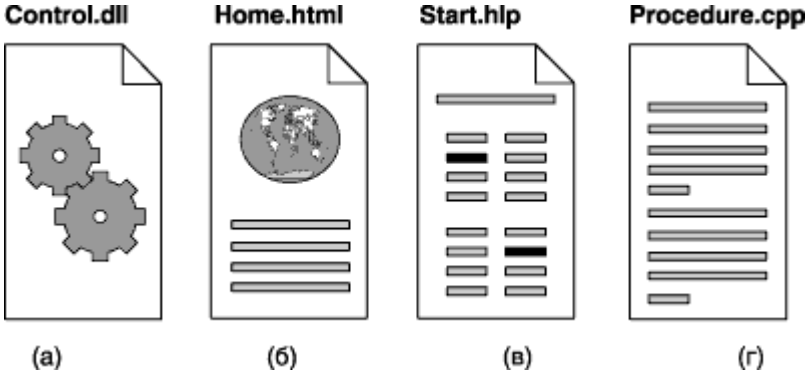

Рис. 3-6.2. Варианты графического изображения компонентов на диаграмме компонентов.

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 артефактов.

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

- <<file>> (файл) – определяет наиболее общую разновидность *компонента*, который представляется в виде произвольного физического файла.
- <<executable>> (исполнимый) – определяет разновидность компонента-файла, который является исполнимым файлом и может выполняться на компьютерной платформе.
- <<document>> (документ) – определяет разновидность компонента-файла, который представляется в форме документа произвольного содержания, не являющегося исполнимым файлом или файлом с исходным текстом программы.
- <<library>> (библиотека) – определяет разновидность компонента-файла, который представляется в форме динамической или статической библиотеки.
- <<source>> (источник) – определяет разновидность компонента-файла, представляющего собой файл с исходным текстом программы, который после компиляции может быть преобразован в исполнимый файл.
- <<table>> (таблица) – определяет разновидность *компонента*, который представляется в форме таблицы базы данных.

Отдельными разработчиками предлагались собственные графические стереотипы для изображения тех или иных типов *компонентов*, однако, за небольшим исключением они не нашли широкого применения. В свою *очередь* ряд инструментальных CASE-средств также содержат дополнительный набор графических стереотипов для обозначения *компонентов*.

Интерфейсы

Следующим графическим элементом диаграммы *компонентов* являются *интерфейсы*. В общем случае *интерфейс* графически изображается окружностью, которая соединяется с *компонентом* отрезком линии без стрелок (рис. 3-6.3, а). При этом имя *интерфейса*, которое рекомендуется начинать 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*.

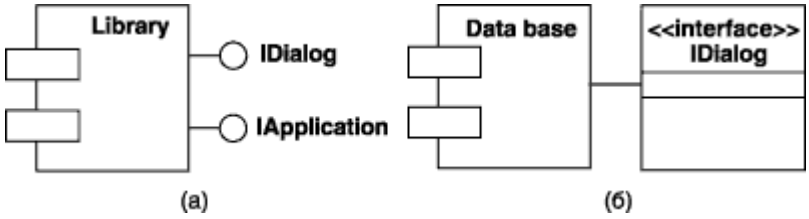

Рис. 3-6.3. Графическое изображение интерфейсов на диаграмме компонентов.

Кроме того, *интерфейс* на диаграмме *компонентов* может быть изображен в виде прямоугольника класса со стереотипом <<*interface*>> и секцией поддерживаемых операций (рис. 3-6.3, б). Как правило, этот вариант обозначения используется для представления внутренней структуры *интерфейса*.

При разработке программных систем *интерфейсы* обеспечивают не только совместимость различных версий, но и возможность вносить существенные изменения в одни части программы, не изменяя другие . Характер применения *интерфейсов* отдельными *компонентами* может отличаться.

Различают два способа связи *интерфейса* и *компонента*. Если *компонент* реализует некоторый *интерфейс*, то такой *интерфейс* называют *экспортируемым* или **поддерживаемым**, 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 между компонентами

В общем случае *отношение* зависимости также было рассмотрено ранее. *Отношение* зависимости служит для представления факта наличия специальной формы связи между двумя элементами модели, 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 или зависимого элемента к источнику или независимому элементу модели.

Зависимости могут отражать связи отдельных файлов программной системы на этапе компиляции и генерации объектного кода. В других случаях зависимость может указывать на 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 этом случае рисуют стрелку от компонента-клиента к *импортируемому интерфейсу* (рис. 3-6.4). Наличие такой стрелки означает, что *компонент* не реализует соответствующий *интерфейс*, а использует его в процессе своего выполнения. При этом на этой же диаграмме может присутствовать и другой *компонент*, который реализует этот *интерфейс*. *Отношение* реализации *интерфейса* обозначается на диаграмме *компонентов* обычной линией без стрелки.

Так, например, изображенный ниже фрагмент *диаграммы компонентов* представляет информацию о том, что *компонент* с именем*Control* зависит от *импортируемого интерфейса* IDialog, который, в свою *очередь*, реализуется *компонентом* с именем DataBase . При этом для второго *компонента* этот *интерфейс* является *экспортируемым*. Изобразить *связь* второго *компонента* DataBase с этим *интерфейсом* в форме зависимости нельзя , поскольку этот *компонент* реализует указанный *интерфейс*.

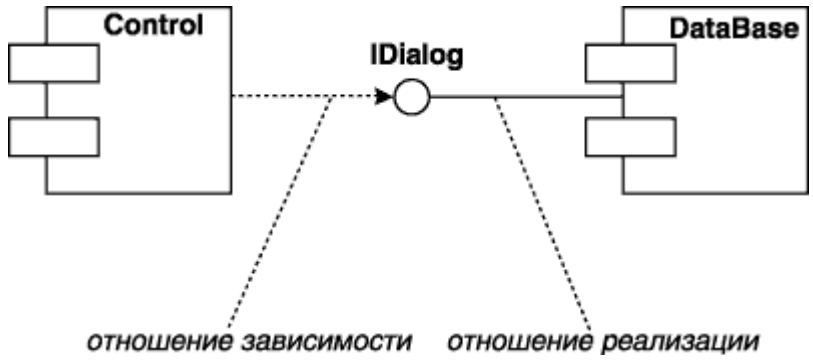

Рис. 3-6.4. Фрагмент диаграммы компонентов с отношениями зависимости и реализации

Другим случаем отношения зависимости на диаграмме компонентов является *отношение* программного вызова и компиляции между различными видами *компонентов*. Для рассмотренного фрагмента *диаграммы компонентов* (рис. 3-6.5) наличие подобной зависимости означает, что исполнимый *компонент Control* .exe использует или импортирует некоторую функциональность *компонента Library* .dll, вызывает страницу гипертекста *Home* .html и *файл* помощи *Search* .hlp, а исходный текст этого исполнимого *компонента* хранится в файле *Control* .cpp. При этом характер отдельных видов зависимостей может быть отмечен дополнительно с помощью текстовых стереотипов.

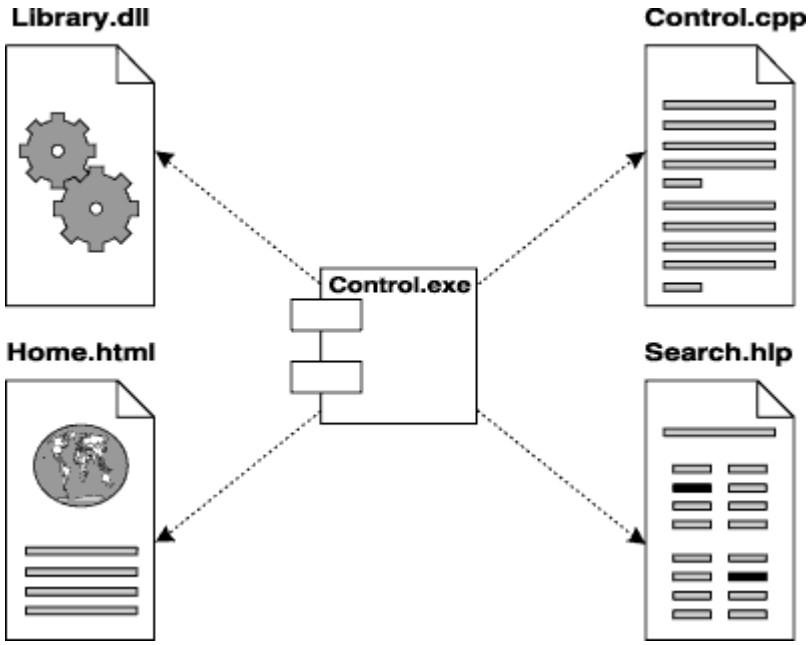

Рис. 3-6.5. Графическое изображение отношения зависимости между компонентами.

На диаграмме компонентов могут быть также представлены отношения зависимости между *компонентами* и реализованными в них классами. Эта *информация* имеет *значение* для обеспечения согласования логического и физического представлений модели системы. Разумеется, изменения в структуре описаний классов могут привести к изменению этой зависимости. Ниже приводится фрагмент зависимости подобного рода, когда исполнимый *компонент Control* .exe зависит от соответствующих классов (рис. 3-6.6).

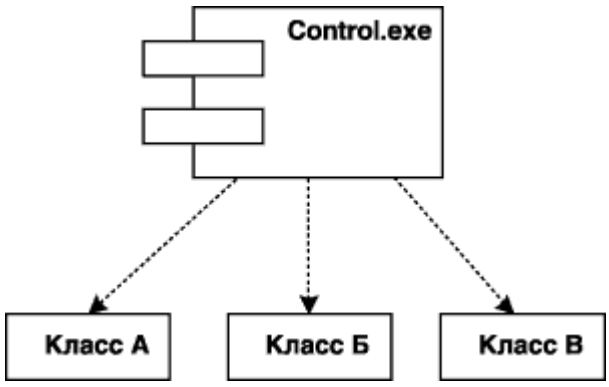

Рис. 3-6.6. Графическое изображение зависимости между компонентом и классами.

В этом случае из *диаграммы компонентов* не следует, что классы реализованы данным *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* и, возможно, дополнительной информации, а нижняя секция – для указания реализуемых данным *компонентом* классов (рис. 3-6.7).

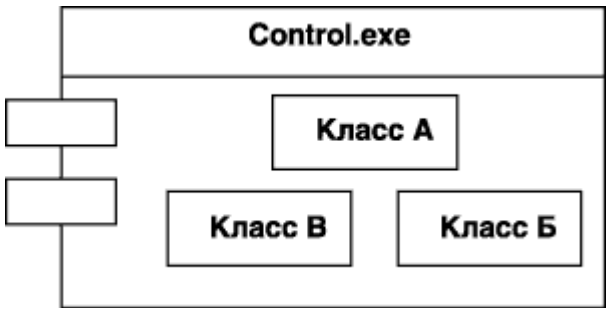

Рис. 3-6.7. Графическое изображение компонента с информацией о реализуемых им классах.

В случае если *компонент* является экземпляром и реализует три отдельных объекта, он изображается в форме *компонента* уровня экземпляров (рис. 3-6.8). Объекты, которые находятся в отдельном компоненте-экземпляре, изображаются вложенными в символ данного *компонента*. Подобная вложенность означает, что выполнение *компонента* влечет за собой выполнение операций соответствующих объектов. При этом существование *компонента* в течение времени исполнения программы обеспечивает функциональность всех вложенных в него объектов. Что касается доступа к этим объектам, то он может быть дополнительно специфицирован с помощью видимости, подобно видимости пакетов.

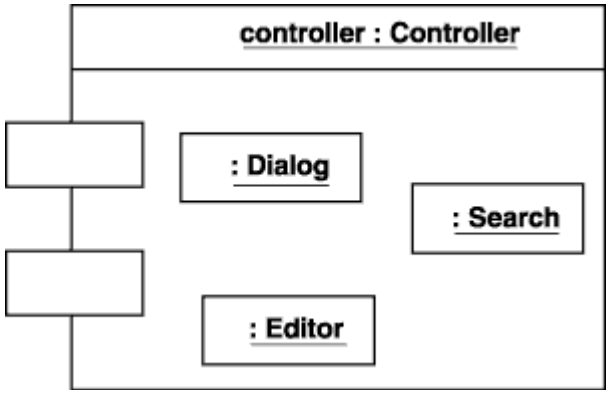

Рис. 3-6.8. Графическое изображение компонента-экземпляра, реализующего отдельные объекты.

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.

Рекомендации по построению диаграммы компонентов

Разработка *диаграммы компонентов* предполагает использование информации не только о логическом представлении модели системы, но и об особенностях ее физической реализации. В первую *очередь*, необходимо решить, из каких физических частей или файлов будет состоять программная система. На этом этапе следует обратить внимание на такую реализацию системы, которая обеспечивала бы возможность повторного использования кода за счет рациональной декомпозиции *компонентов*, а также создание объектов только при их необходимости.

Общая *производительность* программной системы существенно зависит от рационального использования 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 различных частей программной системы. Включение в модель схемы *базы данных* предполагает спецификацию отдельных таблиц и установление информационных связей между ними.

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. Для этой цели можно использовать различные графические стереотипы *компонентов*.

При разработке диаграммы *компонентов* следует придерживаться общих принципов создания моделей на языке *UML*. В частности, в первую *очередь* необходимо использовать уже имеющиеся в языке *UML* и общепринятые графические и текстовые стереотипы. В большинстве типовых проектов этого набора достаточно для представления *компонентов* и зависимостей между ними.

Если же проект содержит физические элементы, описание которых отсутствует в языке *UML*, то следует воспользоваться механизмом расширения. В частности, можно применить дополнительные стереотипы для отдельных нетиповых *компонентов* или *помеченные значения* для уточнения отдельных характеристик *компонентов*.

Наконец, следует обратить внимание на то, что *диаграмма компонентов*, как правило, разрабатывается совместно с диаграммой развертывания, на которой представляется *информация* о физическом размещении *компонентов* программной системы по ее отдельным узлам. Особенности построения диаграммы развертывания будут рассмотрены в следующей части лекции.