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.

https://www.youtube.com/watch?v=KQUGFFN4M90#t=60

[перевод с “UML: distilled” Фаулера]

В объектно-ориентированном сообществе идут дебаты о том, в чем состоит различие между компонентом и обычным классом. Мы не станем обсуждать здесь этот спорный вопрос, но покажем нотацию языка UML, используемую, чтобы отличить их друг от друга.

В UML 1 был отдельный символ для компонента (рис. 14.1). В UML 2 этого значка нет, но можно обозначить прямоугольник класса похожим значком. Или можно воспользоваться ключевым словом «component» (компонент).

Рис. 14.1. Нотация для компонентов

Кроме этого значка компоненты не принесли с собой никаких новых обозначений. Компоненты связываются между собой с помощью предоставляемых или требуемых интерфейсов, при этом шарово-гнездовая нотация (стр. 98) обычно применяется только на диаграммах классов. Можно также разбивать компоненты на части с помощью диаграмм составных структур.

На рис. 14.2 показан пример простой диаграммы компонентов. В этом примере компонент Till (Касса) может взаимодействовать с компонентом Sales Server (Сервер продаж) с помощью интерфейса sales message (Сообщение о продажах). Поскольку сеть ненадежна, то компонент Message Queue (Очередь сообщений) установлен так, чтобы касса могла общаться с сервером, когда сеть работает, и разговаривать с очередью сообщений, когда сеть отключена. Тогда очередь сообщений сможет поговорить с сервером, когда сеть снова станет доступной. В результате очередь сообщений предоставляет интерфейс для разговора с кассой, и требует такой же интерфейс для разговора с сервером. Сервер разделен на два основных компонента: Transaction Processor (Процессор транзакций) реализует интерфейс сообщений, а Accounting Driver (Драйвер счетов) общается с Accounting System (Система ведения счетов).

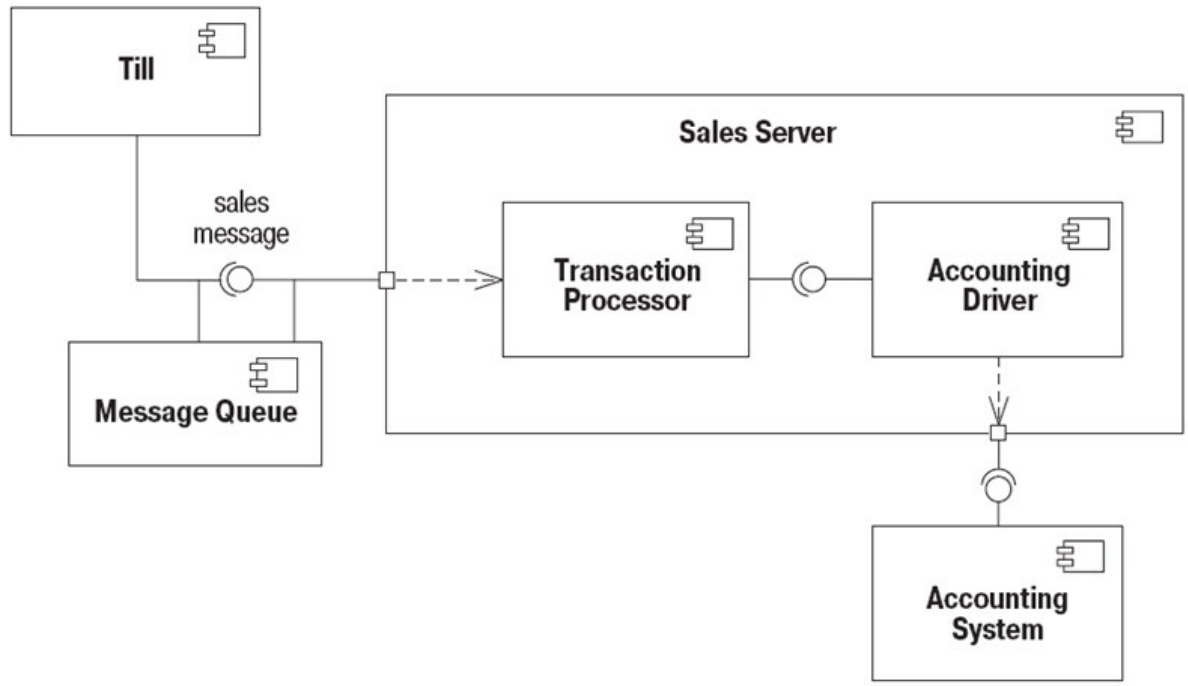
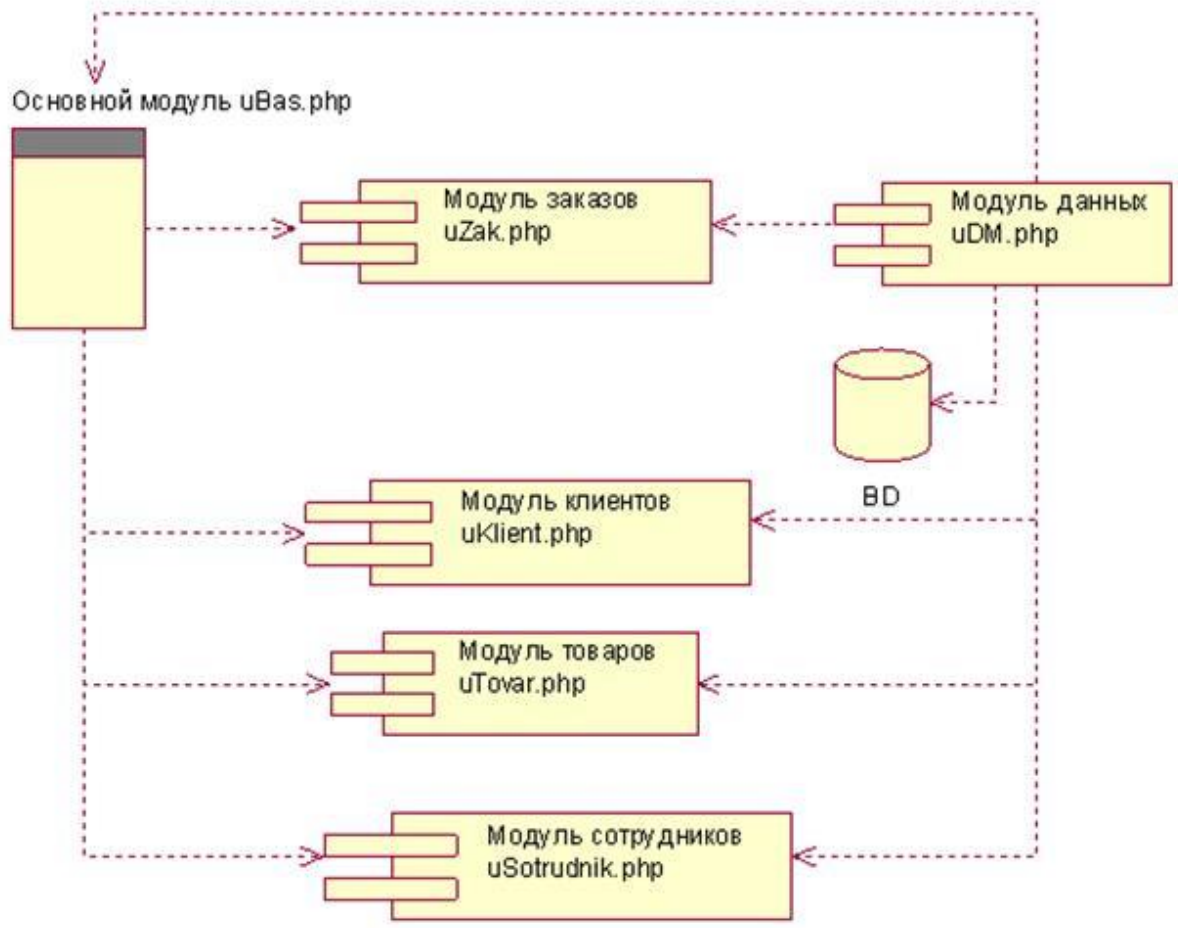


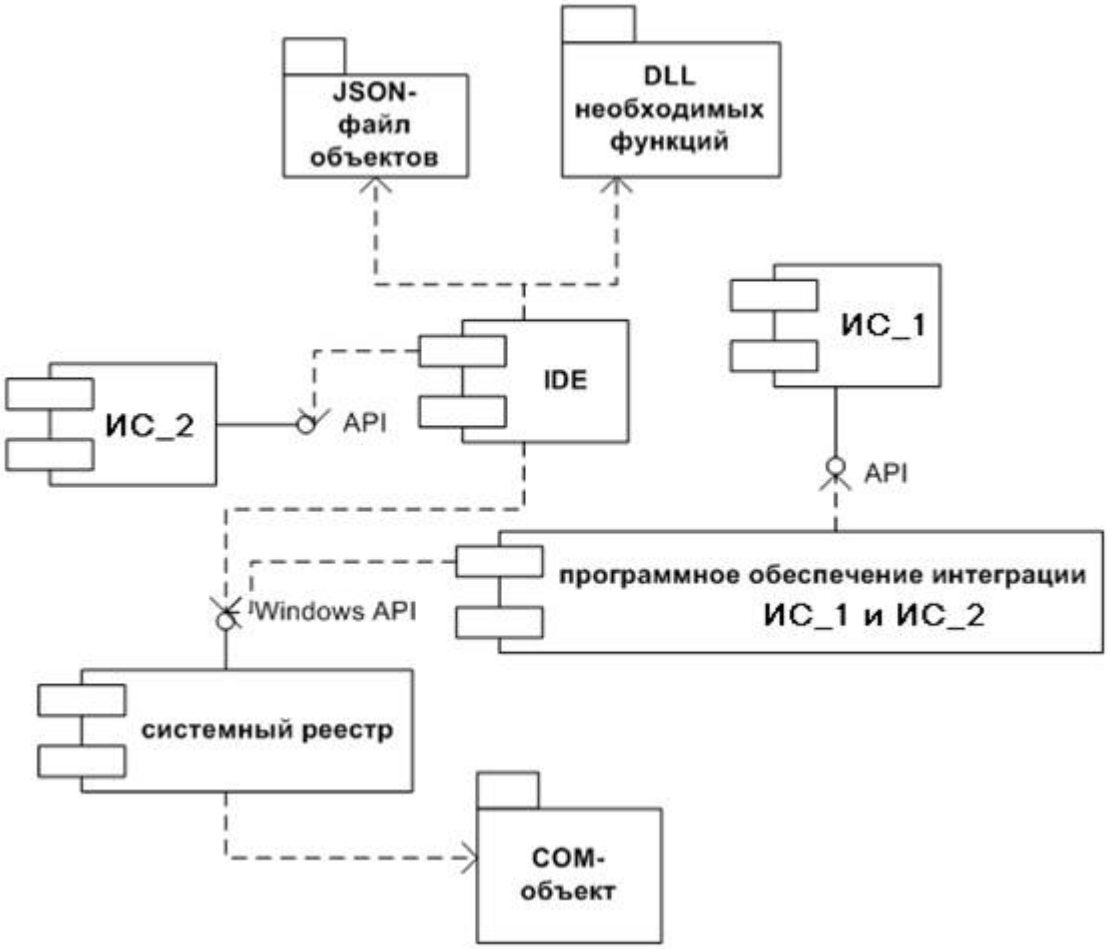
***Рис. 14.2.** Пример диаграммы компонентов*

Вопрос о сущности компонента является предметом бесконечных споров. Вот одно из наиболее продуманных суждений, обнаруженных нами:

> *Компоненты – это не технология. Технические специалисты считают их трудными для понимания. Компоненты – это скорее стиль отношения клиентов к программному обеспечению. Они хотят иметь возможность покупать необходимое им программное обеспечение частями, а также иметь возможность обновлять его, как они обновляют свою стереосистему. Они хотят, чтобы новые компоненты работали так же, как и прежние, и обновлять их согласно своим планам, а не по указанию производителей. Они хотят, чтобы системы различных производителей могли работать вместе и были взаимозаменяемыми. Это очень разумные требования. Одна загвоздка: их трудно выполнить.*
> *Ральф Джонсон (Ralph Johnson),*
> *http://www.c2.com/cgi/wiki?DoComponentsExist*

Важно то, что компоненты представляют элементы, которые можно независимо друг от друга купить и обновить. В результате разделение системы на компоненты является в большей мере маркетинговым решением, чем техническим. Прекрасное руководство по данному вопросу представляет книга Хохмана [23]. Она также напоминает о том, что следует остерегаться разделения системы на слишком мелкие компоненты, поскольку очень большим количеством компонентов трудно управлять, особенно когда производство версий поднимает свою уродливую голову; отсюда пошло выражение «ад DLL» или "dll hell". В ранних версиях языка UML компоненты применялись для представления физических
структур, таких как DLL. Теперь это не актуально; в настоящее время эта задача решается при помощи артефактов (artifacts).




JSON-
файл
объектов
DLL
необходимых
функций
ИС_1
IDE
ИС_2
API
API
программное обеспечение интеграции
ИС_1 и ИС_2
Windows API
системный реестр
COM-
объект