UML Основы

Краткое руководство
по стандартному языку
объектного моделирования

Третье издание

Мартин Фаулер

Санкт-Петербург
2005

13

Составные структуры

Одной из наиболее значимых новых черт языка UML 2 является возможность превращать класс в иерархию внутренних структур. Это позволяет разбить сложный объект на составляющие.

На рис. 13.1 показан класс `TV Viewer` (Телевизор) вместе с интерфейсами, которые он предоставляет и которые требует (*стр. 96*). Я показал его двумя способами: с помощью шарово-гнездовой нотации и с помощью перечисления внутри объекта.

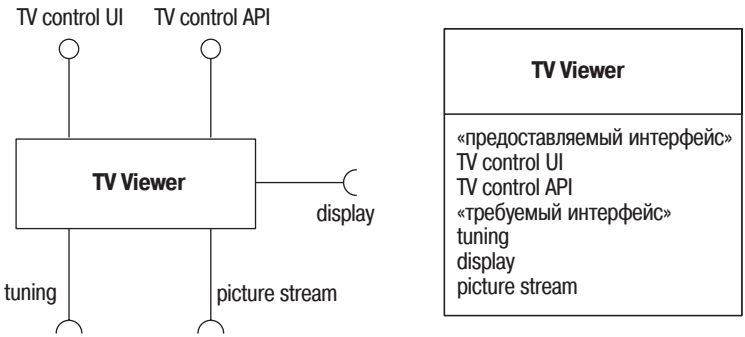

Рис. 13.1. *Два способа представления объекта TV Viewer и его интерфейсов*

На рис. 13.2 показан этот же класс, разбитый внутри на две части, которые предоставляют и требуют различные интерфейсы. Каждая часть имеет имя в виде `имя : класс`, в котором каждый из составляющих его элементов может отсутствовать. Составляющие части не являются описанием экземпляров, поэтому они выделены жирным шрифтом, а не подчеркнуты.

Можно также указать количество экземпляров конкретной части. На рис. 13.2 показано, что каждый объект `TV Viewer` содержит один генератор (`Generator`) и один блок управления (`controls`).

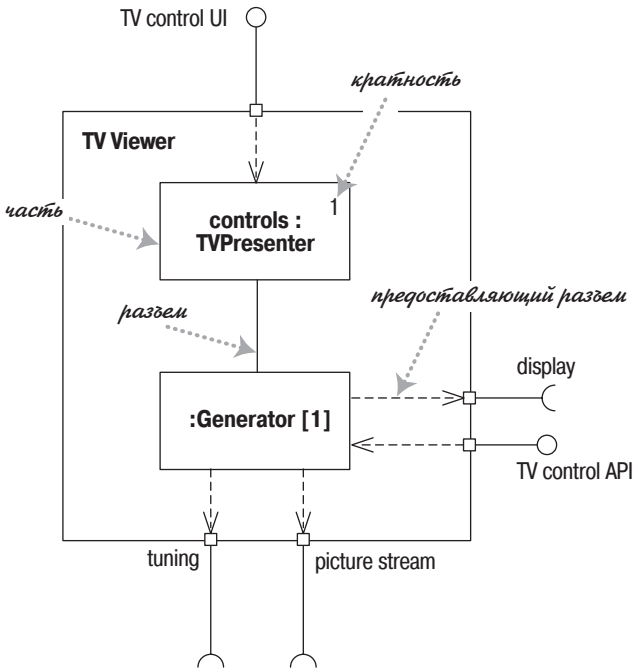

Рис. 13.2. *Внутренний вид компонента (пример, предложенный Джимом Рамбо)*

Чтобы обозначить часть, реализующую интерфейс, надо нарисовать разъем, предоставляемый этим интерфейсом. Аналогично, чтобы показать часть, нуждающуюся в интерфейсе, надо нарисовать разъем, предоставляемый этому интерфейсу. Кроме того, разъемы между частями можно показать или с помощью простой линии, как сделано в данном случае, или с помощью шарово-гнездовой нотации (*стр. 98*).

К внешней структуре можно добавить порты (рис. 13.3). Порты позволяют группировать требуемые и предоставляемые интерфейсы в логи-

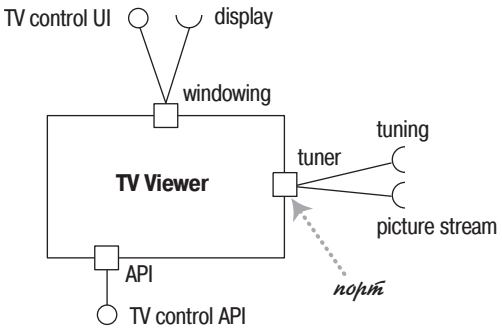

Рис. 13.3. *Компонент с несколькими портами*

ческие взаимодействия, которые компонент имеет с объектами внешнего мира.

Когда применяются составные структуры

Составные структуры являются нововведением в UML 2, хотя некоторые прежние методы реализовывали подобные идеи. Правильный подход к различию между пакетами и составными структурами заключается в том, что пакеты представляют группы времени компиляции, а составные структуры представляют группы времени выполнения. А раз так, то они лучше подходят для показа компонентов и способов их разбиения на части; следовательно, множество этих нотаций применяется в диаграммах компонентов.

Поскольку составные структуры – новый элемент языка UML 2, то слишком рано говорить об эффективности их практического применения; многие члены сообщества UML думают, что эти диаграммы станут весьма ценным дополнением.

14

Диаграммы компонентов

В объектно-ориентированном сообществе идут дебаты о том, в чем состоит различие между компонентом и обычным классом. Я не хочу обсуждать здесь этот спорный вопрос, но могу показать 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

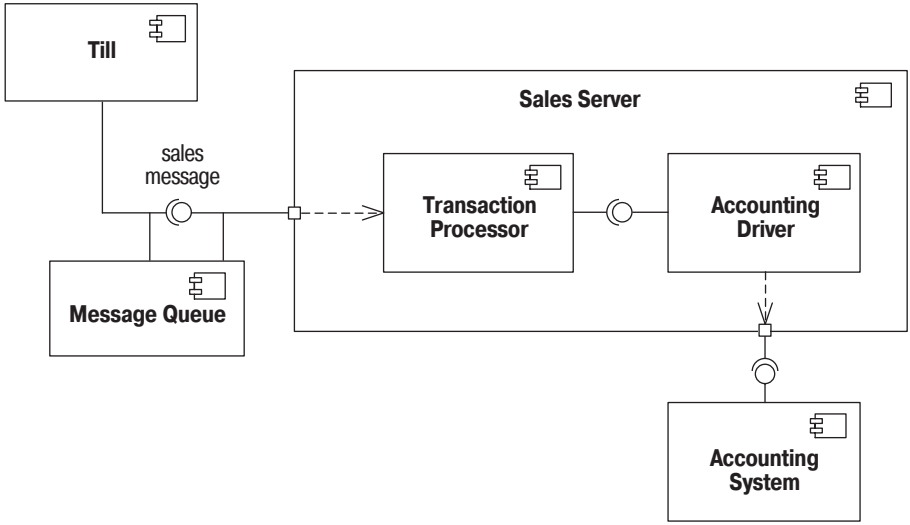

Рис. 14.2. *Пример диаграммы компонентов*

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).

Как я уже говорил, вопрос о сущности компонента является предметом бесконечных споров. Вот одно из наиболее продуманных суждений, обнаруженных мною:

> *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». 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) (*стр. 121*).

Когда применяются диаграммы компонентов

Диаграммы компонентов следует применять, когда система разделяется на компоненты и надо показать их взаимоотношения посредством интерфейсов или схему компонентов в низкоуровневой структуре системы.

15

Кооперации

В отличие от других глав этой книги, в данной не обсуждаются формальные диаграммы UML 2. В стандарте UML кооперации рассматриваются как часть составных структур, но диаграмма кооперации в действительности совершенно отличается от составных структур и применялась в UML 1 без всякой связи с составными структурами. Поэтому я счел, что лучше обсудить кооперации в отдельной главе.

Рассмотрим систему обозначений аукциона. В любом аукционе (`Auction`) могут участвовать продавец (`seller`), покупатели (`buyer`), множество вещей (`lot`) и какие-либо предложения о покупке (`offer`). Мы можем описать эти элементы в терминах диаграммы классов (рис. 15.1) и, возможно, посредством нескольких диаграмм взаимодействий (рис. 15.2).

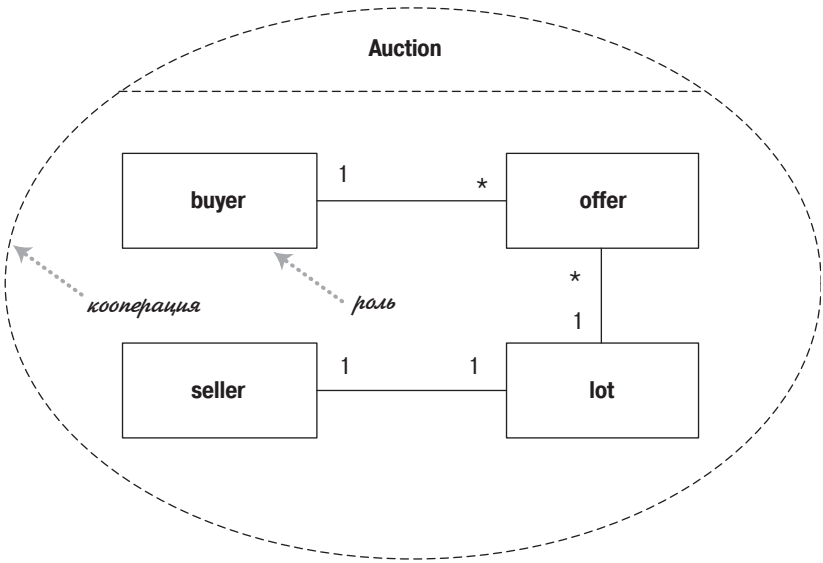

Рис. 15.1. *Кооперация вместе с ее классами и ролями*

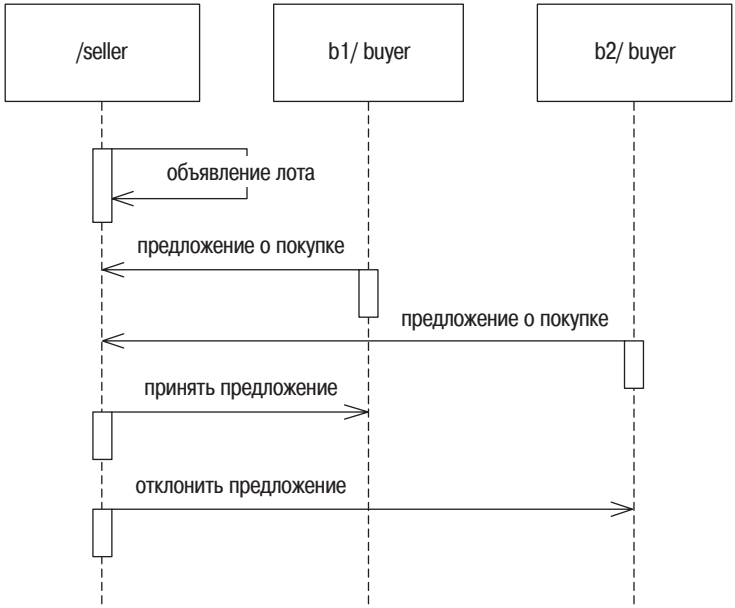

Рис. 15.2. *Диаграмма последовательности для аукционной кооперации*

На рис. 15.1 представлена не совсем обычная диаграмма классов. Во-первых, она обведена пунктирным эллипсом, который представляет аукционную кооперацию. Во-вторых, так называемые классы в кооперации – это не классы, а **роли** (roles), которые реализовываются в процессе выполнения кооперации, поэтому их имена начинаются с маленькой буквы. Сопоставление фактических интерфейсов или классов ролям кооперации не является чем-то необычным, но тем не менее вы не обязаны это делать.

Как видите, на диаграмме взаимодействий участники именуются немного необычно. В кооперации схема именования выглядит следующим образом: `имя-участника / имя-роли : имя-класса`. Как всегда, все эти элементы необязательны.

Применение кооперации можно обозначить, отмечая ее наличие на диаграмме классов, как показано на рис. 15.3, где представлены некоторые классы приложения. Связи, идущие от кооперации к этим классам, показывают, как классы играют различные роли, определенные в кооперации.

В языке UML предполагается, что можно показать применение паттернов, но вряд ли автор каких-либо паттернов будет это делать. Эрих Гамма (Erich Gamma) разработал прекрасную альтернативную нотацию (рис. 15.4). Элементы диаграммы обозначаются либо именем паттерна, либо комбинацией `паттерн:роль`.

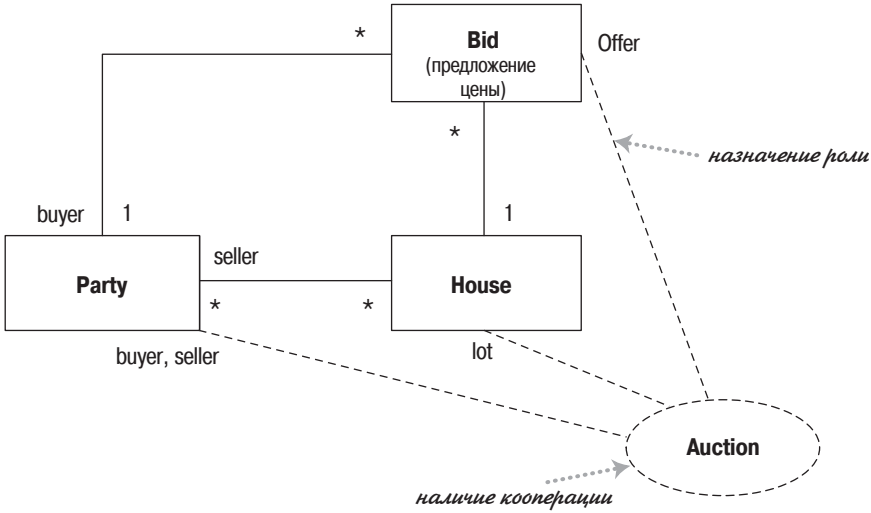

Рис. 15.3. *Наличие кооперации*

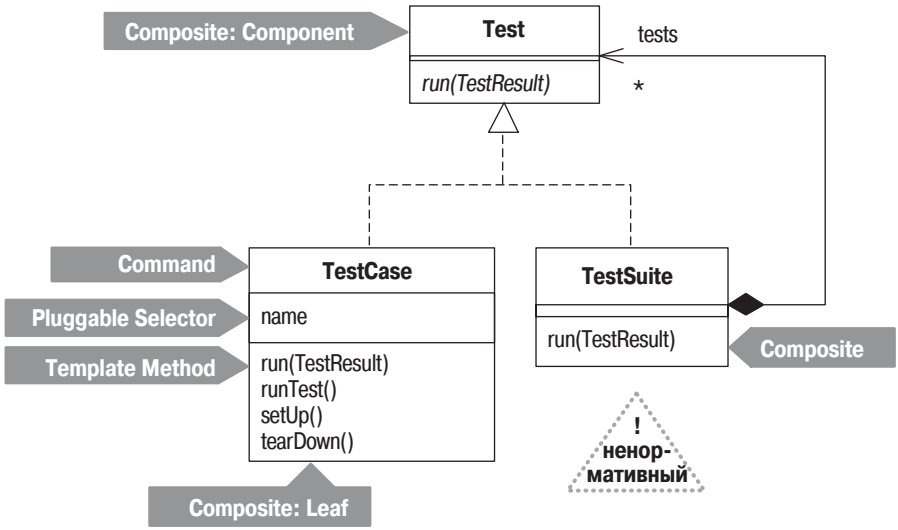

Рис. 15.4. *Необычный способ показа применения паттерна в JUnit (junit.org)*

Когда применяются кооперации

Кооперации существуют со времен UML 1, но я признаю, что вряд ли применял их даже для разработки паттернов. Кооперации предоставляют способ группирования элементов взаимодействия, когда роли исполняются различными классами. Однако на практике я не встречал, чтобы этот тип диаграмм кого-то покорил.