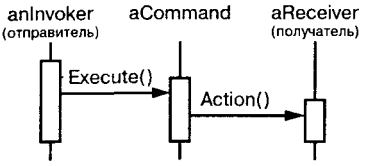
Разделение получателей и отправителей

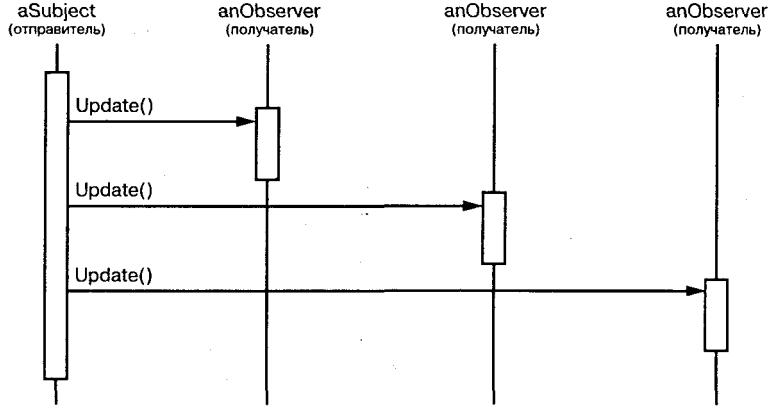
Когда взаимодействующие объекты напрямую ссылаются друг на друга, они становятся зависимыми, а это может отрицательно сказаться на повторном использовании системы и разбиении ее на уровни. Паттерны команда, наблюдатель, посредник и цепочка обязанностей указывают разные способы разделения получателей и отправителей запросов. Каждый способ имеет свои достоинства и недостатки.



Паттерн команда поддерживает разделение за счет объекта-команды, который определяет привязку отправителя к получателю.

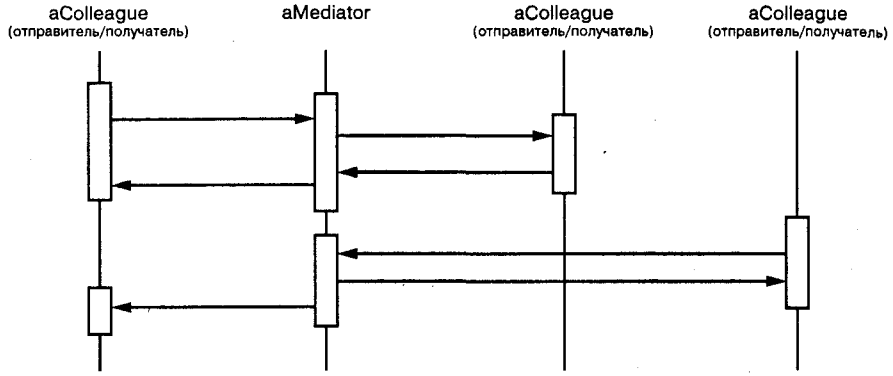
Паттерн команда предоставляет простой интерфейс для выдачи запроса (операцию `Execute`). Заключение связи между отправителем и получателем в самостоятельный объект позволяет отправителю работать с разными получателями. Он отделяет отправителя от получателей, облегчая тем самым повторное использование. Кроме того, объект-команду можно повторно использовать для параметризации получателя различными отправителями. Номинально паттерн команда требует определения подкласса для каждой связи отправитель-получатель, хотя имеются способы реализации, при которых удается избежать порождения подклассов.

Паттерн наблюдатель отделяет отправителей (субъектов) от получателей (наблюдателей) путем определения интерфейса для извещения о происшедших с субъектом изменениях. По сравнению с командой в наблюдателе связь между отправителем и получателем слабее, поскольку у субъекта может быть много наблюдателей и их число даже может меняться во время выполнения.



Интерфейсы субъекта и наблюдателя в паттерне наблюдатель предназначены для передачи информации об изменениях. Стало быть, этот паттерн лучше всего подходит для разделения объектов в случае, когда между ними есть зависимость по данным.

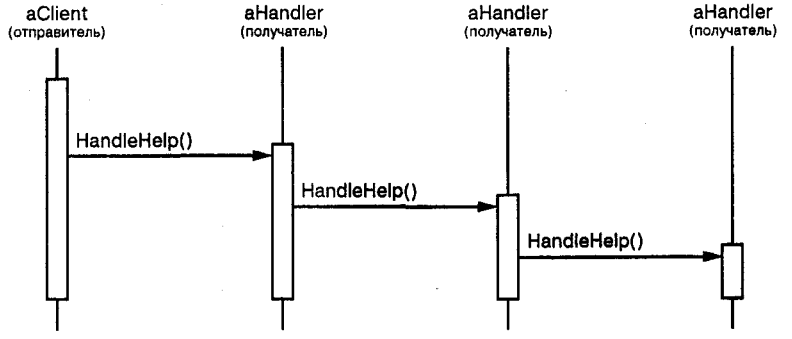
Паттерн посредник разделяет объекты, заставляя их ссылаться друг на друга косвенно, через объект-посредник.



Объект-посредник распределяет запросы между объектами-коллегами и централизует обмен информацией между ними. Таким образом, коллеги могут «общаться» между собой только с помощью интерфейса посредника. Поскольку этот интерфейс фиксирован, посредник может реализовать собственную схему диспетчеризации для большей гибкости. Разрешается кодировать запросы и упаковывать аргументы так, что коллеги смогут запрашивать выполнение операций из заранее неизвестного множества.

Паттерн посредник часто способствует уменьшению числа подклассов в системе, поскольку централизует весь обмен информацией в одном классе, вместо того чтобы распределять его по подклассам.

Наконец, паттерн цепочка обязанностей отделяет отправителя от получателя за счет передачи запроса по цепочке потенциальных получателей.



Поскольку интерфейс между отправителями и получателями фиксирован, то цепочка обязанностей также может нуждаться в специализированной схеме диспетчеризации. Поэтому она обладает теми же недостатками с точки зрения безопасности типов, что и посредник. Цепочка обязанностей – это хороший способ

разделить отправителя и получателя в случае, если она уже является частью структуры системы, а один объект из группы может принять на себя обязанность обработать запрос. Данный паттерн повышает гибкость и за счет того, что цепочку можно легко изменить или расширить.

Резюме

Как правило, паттерны поведения дополняют и усиливают друг друга. Например, класс в цепочке обязанностей, скорее всего, будет содержать хотя бы один шаблонный метод. Он может пользоваться примитивными операциями, чтобы определить, должен ли объект обработать запрос сам, а также в случае необходимости выбрать объект, которому следует переадресовать запрос. Цепочка может применять паттерн команда для представления запросов в виде объектов. Зачастую интерпретатор пользуется паттерном состояние для определения контекстов синтаксического разбора. Иногда Итератор обходит агрегат, а посетитель выполняет операцию для каждого его элемента.

Паттерны поведения хорошо сочетаются и с другими паттернами. Например, система, в которой применяется паттерн компоновщик, время от времени использует посетитель для выполнения операций над компонентами, а также задействует цепочку обязанностей, чтобы обеспечить компонентам доступ к глобальным свойствам через их родителя. Бывает, что в системе применяется и паттерн декоратор для переопределения некоторых свойств частей композиции. А паттерн наблюдатель может связать структуры разных объектов, тогда как паттерн состояние позволит компонентам варьировать свое поведение при изменении состояния. Сама композиция может быть создана с применением строителя и рассматриваться как прототип какой-то другой частью системы.

Хорошо продуманные объектно-ориентированные системы внешне похожи на собрание многочисленных паттернов, но вовсе не потому, что их проектировщики мыслили именно такими категориями. Композиция на уровне паттернов, а не классов или объектов, позволяет добиться той же синергии, но с меньшими усилиями.