полиморфной операции Accept, принадлежащей посещаемому объекту. Посетитель никогда не рассматривается как часть посещаемых объектов, хотя традиционным альтернативным вариантом этому паттерну служит распределение кода посетителя между классами объектов, входящих в структуру.

Другие паттерны определяют объекты, выступающие в роли волшебных палочек, которые передаются от одного владельца к другому и активизируются в будущем. К этой категории относятся команда и хранитель. В паттерне команда такой «палочкой» является запрос, а в хранителе она представляет внутреннее состояние объекта в определенный момент. И там, и там «палочка» может иметь сложную внутреннюю структуру, но клиент об этом ничего не «знает». Но даже здесь есть различия. В паттерне команда важную роль играет полиморфизм, поскольку выполнение объекта-команды – полиморфная операция. Напротив, интерфейс хранителя настолько «узок», что его можно передавать лишь как значение. Поэтому вполне вероятно, что хранитель не предоставляет полиморфных операций своим клиентам.

Должен ли обмен информацией быть инкапсулированным или распределенным

Паттерны посредник и наблюдатель конкурируют между собой. Различие между ними в том, что наблюдатель распределяет обмен информацией за счет объектов наблюдатель и субъект, а посредник, наоборот, инкапсулирует взаимодействие между другими объектами.

В паттерне наблюдатель участники наблюдатель и субъект должны кооперироваться, чтобы поддержать ограничение. Паттерны обмена информацией определяются тем, как связаны между собой наблюдатели и субъекты; у одного субъекта обычно бывает много наблюдателей, а иногда наблюдатель субъекта сам является субъектом наблюдения со стороны другого объекта. В паттерне посредник ответственность за поддержание ограничения возлагается исключительно на посредника.

Нам кажется, что повторно использовать наблюдатели и субъекты проще, чем посредники. Паттерн наблюдатель способствует разделению и ослаблению связей между наблюдателем и субъектом, что приводит к появлению сравнительно мелких классов, более приспособленных для повторного использования.

С другой стороны, потоки информации в посреднике проще для понимания, нежели в наблюдателе. Наблюдатели и субъекты обычно связываются вскоре после создания, и понять, каким же образом организована их связь, в последующих частях программы довольно трудно. Если вы знаете паттерн наблюдатель, то понимаете важность того, как именно связаны наблюдатели и субъекты, и представляете, какие связи надо искать. Однако из-за присущей наблюдателю косвенности разобраться в системе все же нелегко.

В языке Smalltalk наблюдатели можно параметризовать сообщениями, применяемыми для доступа к состоянию субъекта, поэтому степень их повторного использования даже выше, чем в C++. Вот почему в Smalltalk паттерн наблюдатель более привлекателен, чем в C++. Следовательно, программист, пишущий на Smalltalk, нередко использует наблюдатель там, где программист на C++ применил бы посредник.