ПРИМЕНЕНИЕ ПАТТЕРНОВ ПРОЕКТИРОВАНИЯ ПРИ ПОСТРОЕНИИ СИСТЕМ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ

О.А. Змеев, А.В. Приступа
Томский государственный университет

Паттерны проектирования – чрезвычайно удобный механизм, существенно упрощающий разработку объектно-ориентированных систем. Их применение повышает гибкость и возможности повторного использования программного обеспечения. Именно поэтому шаблоны (паттерны) проектирования сегодня по праву считаются одной из наиболее популярных концепций объектно-ориентированного программирования, подробно представленной в [1]. В данной статье мы рассмотрим, каким образом типичные проблемы, возникающие при построении систем имитационного моделирования, решаются с помощью паттернов проектирования.

Источники требований в имитационной модели СМО могут образовывать иерархичную структуру. Паттерн «Composite» («Компоновщик») описывает, как можно применить рекурсивную композицию в этой ситуации таким образом, что клиенту не придется проводить различие между простыми и составными объектами.

В ходе выполнения имитационной модели необходимо обрабатывать наступление различных событий из СБС (список будущих событий). При этом желательно, чтобы работа с этим списком велась независимо от событий, которые там хранятся. Этого можно добиться, используя паттерн «Command» («Команда»). В основе будет лежать абстрактный класс *Event*, в конкретных подклассах которого будут реализованы операции для каждого типа событий (рис. 1). При таком подходе можно будет легко добавлять новые типы событий, не изменяя при этом существующие классы. Таким образом, события будут представлять собой объекты, что позволит задавать параметры для их обработки, ставить их в очередь и протоколировать.

У каждого события теперь есть собственной обработчик *Execute*, реализованный в конкретном подклассе класса *Event*. Но он представляет лишь инвариантную часть алгоритма обработки (т.е. то, что должно быть выполнено в любом случае), хотя желательно иметь возможность реализации дополнительного поведения при наступлении события. Для этого используется паттерн «Template Method» («Шаблонный метод»), согласно которому в родительском классе объявляются абстрактные операции, которые могут быть замещены в подклассах (*BeforeExecute* и *AfterExecute* в нашем случае). Иллюстрация этого подхода приведена на рис. 1.

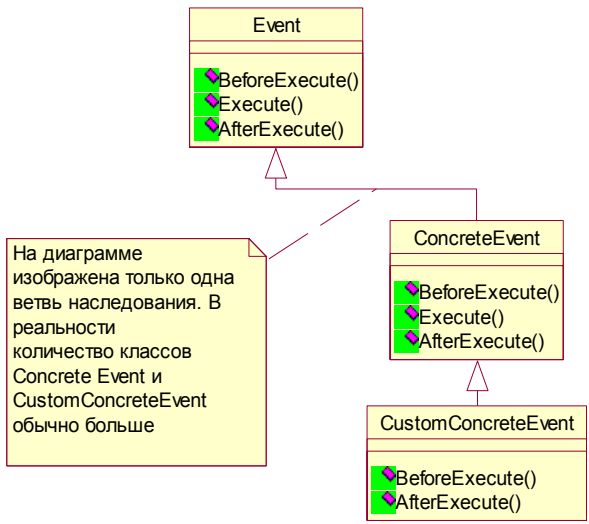

Рис. 1 – Иерархия классов событий

Зачастую события, возникающие в одном объекте имитационной модели, должны инициировать события, возникающие в других объектах. При этом хотелось бы, чтобы связи между объектами носили как можно более абстрактный характер, чтобы увеличить возможность повторного использования. Это достигается засчет применения паттерна «Observer» («Наблюдатель»), который при изменении состояния одного объекта гарантирует оповещение и обновление зависящих от него объектов. Однако необходимо учитывать, что объект-наблюдатель может одновременно следить за несколькими субъектами, и,

возможно, нуждается в обновлении только после того, как все они достигнут требуемых состояний. В этом случае применения одного лишь паттерна «Observer» недостаточно – нужно еще применить паттерн «Mediator» («Посредник») для того, чтобы производить обновление объекта-наблюдателя в нужное время. Согласно [1], такой посредник должен быть известен всем участникам. Следовательно, в данной ситуации удобно применить также паттерн «Singleton» («Одиночка»).

Еще одна особенность заключается в том, что типичные элементы систем имитационного моделирования могут поразному вести себя в зависимости от своего состояния. Это касается многих объектов, но мы рассмотрим только пример очереди. Если количество требований в очереди стало предельным, то очередь не принимает вновь поступающие требования (если лимит не достигнут - очередь работает в обычном режиме). Исходя из этого, здесь полезно применить паттерн «State» («Состояние»), который помещает поведение для каждого конкретного состояния в отдельный объект, тем самым облегчая добавление новых состояний и переходов.

Продолжая пример с очередью, можно отметить, что даже если она не имеет ограничения на длину, все равно алгоритм ее работы неоднозначен (очередь может быть устроена по LIFO-, FIFO- или случайному принципу). Применяя паттерн «Strategy» («Стратегия»), можно уйти от необходимости плодить условные операторы для выбора нужного алгоритма работы. Вместо этого надо определить классы, инкапсулирующие различные алгоритмы (стратегии).

Подводя итог, можно сказать, что здесь перечислены далеко не все, а только основные паттерны, которые можно применить при построении объектно-ориентированной системы имитационного моделирования. При более детальном анализе предметной области можно обнаружить и успешно применить и другие паттерны проектирования.

Литература

1. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб: Питер, 2001. – 368 с.