ЧАСТЬ IV

Практическое применение шаблонов проектирования

Введение

В этой части рассказывается о новом подходе к объектно-ориентированному проектированию программных систем, основанному на применении шаблонов. Действенность этого подхода неоднократно проверена автором на практике. Предлагаемый подход мы применим к решению проблемы САПР, предложенной в главе 3, *Проблема, требующая создания гибкого кода.*

Предлагаемый подход подразумевает, прежде всего, попытку понять тот контекст, в котором были выделены объекты системы.

Глава	Предмет обсуждения
11	• Обсуждение идей Кристофера Александера и методов их использования опытными разработчиками при проектировании
12	• Применение предлагаемого подхода к решению задачи САПР, предложенной в главе 3.
	• Сравнение нового варианта решения с тем решением, которое было предложено в главе 4
13	• Подведение итогов обсуждения объектно-ориентированной идеологии и шаблонов проектирования.
	• Формулирование концепции *шаблонно-ориентированного проектирования*

Глава 11

Как проектируют эксперты

Введение

С чего следует начинать разработку проекта? Определить все возможные детали, а затем искать способ объединить их в единое целое? Или лучше предварительно составить общее представление о проекте и лишь затем приступить к его детализации? А может существует и другой путь?

Подход, предложенный Кристофером Александером, состоит в том, чтобы сначала сосредоточить внимание на отношениях концептуального уровня, а затем продвигаться от общего к частному. Прежде чем принять какое-либо проектное решение, он предлагает глубоко вникнуть в контекст той проблемы, которую это решение должно преодолеть, используя шаблоны проектирования для выявления существующих в этой среде отношений. Однако Александер предлагает не просто коллекцию шаблонов, а самостоятельную методологию проектирования. Предметной областью, о которой он пишет, является архитектура. Ее задача состоит в проектировании зданий, в которых люди живут и работают. Но предложенные Александером принципы вполне применимы и к проектированию программного обеспечения.

В этой главе мы выполним следующее.

- Обсудим подход Александера к проектированию.
- Проанализируем, как применить этот подход к разработке программного обеспечения.

Проектирование посредством добавления различий

После ознакомления с несколькими шаблонами проектирования пришло время выяснить, как они могут работать совместно. Александер полагает, что недостаточно просто описать отдельные шаблоны. Он использует их для построения новой парадигмы проектирования.

Книга Александера *The Timeless Way of Building* посвящена одновременно и описанию шаблонов, и обсуждению методов их совместного использования. Это превосходная книга, одна из моих самых любимых как с личной, так и с профессиональной точки зрения. Она помогла мне осознать то, что происходит в моей жизни, лучше понять тот мир, в котором я живу, а также научиться создавать более качественное программное обеспечение.

Как это стало возможным? Как может книга, посвященная проектированию зданий и городов, оказать столь глубокое влияние на разработку программного обеспечения?

Причина, как я полагаю, состоит в том, что в ней Александер представил новую парадигму, которая будет полезна *любому* проектировщику. Именно эта новая парадигма проектирования и является наиболее важным и интересным аспектом его книги.

К сожалению, я не могу похвастать тем, что принял предложенную Александером идеологию после первого же прочтения его книги. Первоначальной моей реакцией была такая мысль: "Это очень интересно. Возможно, все это даже имеет практический смысл". А затем я вновь возвратился к традиционным методам проектирования, которые использовал на протяжении многих лет.

Но иногда старые пословицы все же находят себе подтверждение в жизни. Не даром говорят: "Удача приходит к тем, кто ее ждет" или "На ловца и зверь бежит". Вскоре мне представился подходящий случай, благодаря которому все стало на свои места.

Спустя несколько недель после прочтения книги Александера жизнь заставила меня вновь вернуться к ней. Я был занят в проекте, который не поддавался традиционным подходам — они попросту не работали. Конечно, я мог бы предложить кое-какие решения, но все они были недостаточно хороши. Все мои старые и испытанные методы проектирования терпели неудачу, и я был очень расстроен. К счастью, мне хватило мудрости попробовать новый путь, предложенный Александером, и очень скоро можно было лишь восхищаться полученными результатами.

В следующей главе мы подробно обсудим все, что мной тогда было сделано. Но сначала посмотрим, какой же подход предлагает Александер.

> *Проектирование часто представляют себе как процесс синтеза, процесс соединения элементов в единое целое, процесс комбинирования.* Согласно этому представлению целое создается соединением отдельных частей, т.е. элементы являются первичными, а общая форма — вторичной.[1]

Это вполне естественно — проектировать, переходя от частного к общему, начиная с конкретных вещей, понятных и знакомых.

Когда я впервые прочитал это, то подумал: "Все это вполне соответствует моему взгляду на вещи. Сначала выяснить, что мне может пригодиться, а затем собрать все это в одно целое." Иначе говоря, при проектировании сначала идентифицируются требуемые классы, а затем уже устанавливается порядок их взаимодействия. После того как все требуемые элементы будут собраны в одно целое, можно вернуться на шаг назад и оценить, что же у нас получилось. Но и в этом случае при переключении внимания от частного (локального) к общему (глобальному) общая картина остается в нашем представлении состоящей из отдельных частей.

В объектно-ориентированной разработке элементами являются объекты и классы. Именно они идентифицируются на первом этапе, после чего определяется их поведение и интерфейсы. Однако, начав проектирование с частностей, мы, как правило, так и остаемся сосредоточенными на них до конца.

Вспомним исходный вариант решения проблемы с различными версиями САПР, предложенный в главе 4, *Стандартное объектно-ориентированное решение.* Мы начали с рассуждений о различных классах, которые могут нам потребоваться: классы для представления пазов, отверстий, просечек и т.д. Поскольку необходимо было связать их с системами версий V1 и V2, предполагалось наличие двух наборов классов, спе-

---

[1] *Alexander C., Ishikawa S., Silverstein M.* The Timeless Way of Building, *New York: Oxford University Press, 1979, c. 368.*

циализированных для каждой из систем. И только после определения всех требуемых классов внимание было перенесено на проблему их взаимодействия.

> *Тем не менее, простым соединением заранее определенных частей невозможно сформировать что-либо, имеющее естественный, законченный облик.*[2]

Тезис Александера состоит в том, что построение целого из отдельных частей — это далеко не лучший метод проектирования.

Несмотря на то что речь в книге Александера идет об архитектуре, многие признанные авторитеты в области разработки программного обеспечения согласятся с тем, что этот тезис верен и в их области. Я попытался понять, в чем состоит суть нового подхода к разработке. В результате в моей голове сложилась фраза, как будто бы произнесенная Александером: "Хорошее программное обеспечение не может быть создано простым соединением заранее определенных частей" (т.е. без предварительной оценки, насколько хорошо эти части будут подходить друг другу).

> *Когда элементы представляют собой отдельные модули, созданные прежде общего целого, то по определению они всегда будут идентичными, и нельзя ожидать, что каждая часть будет уникальной, отвечающей своему положению в общем целом.* И что еще более важно, с помощью любой комбинации модульных элементов просто невозможно получить все то множество шаблонов, которые должны быть одновременно представлены в любом месте, предназначенном для пребывания человека.[3]

Размышления Александера, связанные с модульностью, поначалу были для меня совершенно непонятны. Но в конце концов мне стало ясно, что если начать с создания модулей еще до уяснения того, как будет выглядеть общая картина, то модули в дальнейшем всегда будут одними и теми же, поскольку не существует никаких причин для их изменения в дальнейшем.

Похоже, именно в этом и состоит основная цель концепции многократного использования. Разве она не предполагает постоянного использования одних и тех же модулей? Именно так. Однако нам также требуется максимальная гибкость и устойчивость системы. Само по себе простое создание модулей не гарантирует этого.

Ознакомившись с использованием шаблонов проектирования по методологии Александера, я смог создавать многократно используемые и одновременно гибкие классы с гораздо большим успехом, чем прежде. Как проектировщик, я стал намного лучше.

> *Единственно возможный способ создать жизненное пространство, полностью отвечающее нуждам его обитателей, состоит в максимальном приспособлении каждой части общего целого к той позиции, которую она в нем занимает.*[4]

В нашем случае выражение *полностью отвечающее нуждам* следует понимать как *устойчивое и гибкое* программное обеспечение.

Выше упоминались слова Александера о том, что части должны быть уникальны — только в этом случае удастся реализовать все преимущества, определяемые их конкретным местоположением. Рассмотрим это утверждение подробнее. Каждый элемент создается посредством копирования эталона с последующим приспособлением новой ко-

---

[2] *Там же, с. 368.*

[3] *Там же, с. 368, 369.*

[4] *Там же, с. 369.*

пии к существующему окружению, что и придает общему целому уникальный, присущий только ему характер. Рассмотрим несколько примеров из области архитектуры.

- *Швейцарская деревня.* Перед глазами встает деревня, состоящая из расположенных близко друг к другу уютных коттеджей, очень похожих один на другой, но, тем не менее, каждый из них имеет что-то индивидуальное. Различия между домами вовсе не произвольны — каждый коттедж отражает финансовые возможности своего создателя и владельца и непременно удовлетворяет требованию гармонично вписаться в окружающую среду. Достигнутый эффект очень хорош — деревня в целом производит впечатление уюта и комфортабельности.
- *Американский пригород.* Все коттеджи выглядят как пряничные домики. Внимание к естественному окружению построек уделяется крайне редко. Установившиеся понятия и стандарты всячески поощряют подобное однообразие. В результате возникает эффект полного обезличивания зданий, и общая картина не способна вызвать каких-либо приятных эмоций.

Безусловно, в данный момент применение обсуждаемого подхода к проектированию программного обеспечения может показаться слишком уж концептуальным. Однако сейчас достаточно понять, что для создания устойчивых и гибких программных систем необходимо разрабатывать их элементы (классы или объекты) с учетом того окружения (контекста) в котором они будут функционировать.

Короче говоря, каждая часть обретает свою специфическую форму за счет ее нахождения в конкретном контексте более общего целого.

Это есть процесс дифференциации. Проектирование рассматривается как серия этапов *усложнения.* Простая структура превращается в нечто более общее, контролируемая и направляемая этим общим, а не за счет простого объединения мелких частей друг с другом. В процессе дифференциации целое порождает свои части, при этом формирование целого и его частей происходит одновременно. В целом процесс дифференциации напоминает развитие эмбриона.[5]

Усложнение — что означает здесь это слово? Ведь наша цель состоит в том, чтобы сделать процесс проектирования проще, а не сложнее!

Смысл его в том, что согласно подходу Александера обдумывание проекта должно начинаться с описания проблемы в простейших терминах и понятиях с последующим добавлением дополнительных особенностей (различий), в результате чего проект будет становиться все более и более сложным благодаря постоянному накоплению анализируемой информации.

Это совершенно естественный процесс. Мы постоянно применяем его на практике. Например, представим, что требуется договориться об аудитории для проведения лекции, причем ожидается, что слушателей будет около 40 человек. Описание соответствующих требований, скорее всего, будет выглядеть примерно так: "Мне понадобится комната размером 10 на 10 метров" (начинаем с простого). Далее: "Потребуются также стулья — 5 рядов по 8 стульев, установленные в виде полукруга" (добавление информации делает описание комнаты более сложным). Наконец: "Необходима так-

[5] *Там же, с. 370.*

же кафедра для лектора, расположенная перед слушателями" (еще более сложное описание).

> *Развертывание проекта в сознании его создателя, в терминах используемого им языка – это тот же самый процесс.*
>
> Каждый шаблон — это оператор, дифференцирующий пространство: т.е. он создает различия там, где прежде их не было. В языке операторы упорядочиваются в последовательности: в соответствии с тем, как они выполняются, один за другим. В результате рождается законченная форма, общая в том смысле, что она имеет структуру, одинаковую с другими сравнимыми элементами, и специфическая в том смысле, что она уникальна в соответствии с ее собственными обстоятельствами.
>
> Язык представляет собой последовательность таких операторов, в которой каждый оператор вносит дальнейшую дифференциацию в образ, являющийся порождением предыдущих дифференциаций.[6]

Итак, Александер утверждает, что проектирование должно начинаться с простейшей формулировки проблемы с последующей постепенной ее детализацией (усложнением) посредством добавления в эту формулировку новой информации. Вносимая информация принимает форму шаблона, поскольку, считает Александер, шаблоны определяют отношения между сущностями в проблемной области.

Для примера еще раз обратимся к шаблону "Внутренний двор", обсуждавшемуся в главе 5, *Первое знакомство с шаблонами проектирования*. Шаблон должен описывать сущности, присутствующие в среде внутреннего двора, и их взаимоотношения между собой. Такими сущностями являются следующие.

- Открытое пространство внутреннего двора.
- Пересекающиеся пути между окружающими помещениями.
- Внешний вид из внутреннего двора.
- И даже люди, которые будут пользоваться этим внутренним двором.

Осмысление проблемы в терминах того, как эти объекты должны взаимодействовать между собой, дает достаточно информации для проектирования внутреннего двора. Затем предварительный проект уточняется за счет применения других шаблонов, которые могут присутствовать в контексте шаблона "Внутренний двор", — например, крыльца или веранды, выходящих во внутренний двор.

Что же делает данный аналитический метод столь мощным, что он не нуждается в подтверждении моим личным опытом, интуицией и творческим потенциалом? Александер утверждает, что существование подобных шаблонов есть объективная реальность, и они существуют независимо от отдельных личностей. Организация пространства отвечает запросам его обитателей тогда, когда она удовлетворяет их естественным потребностям, а не просто потому, что автором проекта является гений. Если качество проекта зависит от его соответствия естественным процессам, не должно вызывать удивления, что качественные решения сходных проблем будут выглядеть очень похоже.

---

[6] *Там же, с. 372, 373.*

Основываясь на этих рассуждениях, Александер сформулировал следующие правила хорошего проектирования.

- *Строго по одному.* Шаблоны должны применяться только по одному, последовательно друг за другом.
- *Сначала следует формировать контекст.* Сначала следует применять те шаблоны, которые создают контекст для других шаблонов.

Шаблоны определяют отношения

Шаблоны, описываемые Александером, определяют отношения между сущностями в проблемной области. Для нас важны не сами шаблоны, а именно эти отношения, поскольку шаблоны просто предлагают способ их анализа.

Подход, предложенный Александером, вполне можно применить и к проектированию программного обеспечения. Конечно, не буквально, а концептуально. Что мог бы сказать Александер, обращаясь к разработчикам программного обеспечения? Возможный набор рекомендаций я поместил в табл. 11.1.

Таблица 11.1. Применение методологии Александера к разработке программного обеспечения

Последовательность разработки	Пояснения
Идентифицируйте шаблоны	Идентифицируйте шаблоны, присутствующие в контексте проблемы. Осмыслите проблему в терминах присутствующих в ней шаблонов. Помните, что назначение шаблонов состоит в определении отношений между сущностями в проблемной области
Начните с шаблонов, определяющих контекст	Идентифицируйте те шаблоны, которые создают контекст для других шаблонов. Именно с них следует начать разработку системы
Двигайтесь в направлении углубления в контекст	Рассмотрите остальные шаблоны и отыщите любые другие шаблоны, которые ранее остались незамеченными. Из полученного набора опять выделите те шаблоны, которые определяют контекст для оставшихся. Повторяйте эту процедуру до исчерпания всего набора шаблонов
Оптимизируйте проект	На этом этапе тщательно проанализируйте общий контекст, полученный в результате применения шаблонов
Реализуйте полученную схему	Реализация предусматривает воплощение всех деталей, требуемых каждым из использованных шаблонов

Личные впечатления автора от использования идей Александера при проектировании программного обеспечения

При первом использовании подхода Александера я воспринял его слова слишком буквально. Его концепции, сформулированные в отношении архитектуры, не всегда удается прямо применить к проектированию программного обеспечения (или другим видам проектирования). В некотором смысле при первых опытах применения шаблонов мне просто повезло, поскольку те проблемы, которые я решал, включали шаблоны проектирования, строго упорядоченные в отношении создания контекста. Однако это везение сработало и против меня, так как по наивности я решил, что так будет всегда (жизнь показала обратное).

Известно, что многие ведущие специалисты в области программного обеспечения поддерживали идею разработки "языка шаблонов" и пытались найти формальный способ применения идей Александера к разработке программного обеспечения. В моем понимании это был бы такой инструмент, который позволил нам прямо применять методы Александера к разработке программного обеспечения (лично я больше не верю, что это возможно). Поскольку Александер указал, что в архитектуре шаблоны обладают заранее определенным порядком образования контекста, я полагал, что и в программном обеспечении шаблоны также имеют некоторый предопределенный порядок. Иначе говоря, один тип шаблона всегда будет создавать контекст для шаблона другого типа. Я начал пропагандировать подход Александера именно так, как я понимал его тогда, и даже учить этому других. Но по прошествии нескольких месяцев и после применения этого подхода в нескольких проектах я столкнулся с серьезными проблемами. Обнаружились ситуации, в которых заранее установленный порядок контекстов не срабатывал.

Как человеку, имеющему математическое образование, мне было достаточно единственного исключения из правил для опровержения всей выдвинутой мной теории. Сложившаяся ситуация заставила меня заново критически пересмотреть весь подход к идеологии шаблонов проектирования. Ранее я всегда придерживался этого правила, но на этот раз просто забыл о нем в радостном возбуждении.

Начиная с самого первого этапа, я вновь проанализировал те *принципы*, на которых строится метод Александера. Несмотря на то, что они по-разному проявляются в архитектуре и в проектировании программ, эти принципы все же полностью применимы к разработке программного обеспечения. Они позволяют повысить качество создаваемых проектов, ускорить их разработку и выполнить более полный анализ. Подтверждение этому я вижу каждый раз при необходимости внесения очередных изменений в созданное мной программное обеспечение.

Резюме

Проектирование обычно понимается как процесс синтеза – процесс объединения отдельных частей в единое целое. При создании программного обеспечения обычный подход состоит в том, чтобы сначала выделить все необходимые объекты, классы и компоненты, и лишь затем подумать о том, как они будут взаимодействовать.

В книге *The Timeless Way of Building* Кристофер Александер предложил лучший подход, основанный на применении шаблонов проектирования. Этот подход предлагает следующее.

1. Начать с концептуального описания всей проблемы в целом, чтобы понять, какие требования в конечном счете должны быть удовлетворены.
2. Идентифицировать шаблоны, которые присутствуют в концептуальном описании проблемы.
3. Начать работу с тех шаблонов, которые создают контекст для остальных.
4. Применить эти шаблоны.
5. Повторить эту процедуру с оставшимися шаблонами, а также с любыми новыми шаблонами, которые, возможно, будут выявлены в процессе проектирования.

6. Наконец, оптимизировать проект и его реализацию в контексте, созданном за счет последовательного применения всех выявленных шаблонов проектирования.

Разработчик программного обеспечения не всегда имеет возможность применить предложенный Александером подход непосредственно. Однако проектирование методом добавления концепций в пределах контекста, образованного за счет представления предыдущих концепций, несомненно, доступно каждому. Помните об этом при изучении новых шаблонов в последующих главах книги. Многие шаблоны позволяют создавать надежное программное обеспечение, потому что они определяют контексты, в пределах которых классы, реализующие некоторое решение, могут успешно работать.

Глава 12

Решение задачи САПР с помощью шаблонов проектирования

Введение

В этой главе показано, как можно применить шаблоны проектирования для решения проблемы поддержки в приложении различных версий САПР, обсуждавшейся в главе 3, *Проблема, требующая создания гибкого кода.*

Здесь мы выполним следующее.

- Вспомним первый вариант решения проблемы САПР.
- Обсудим начальный этап проектирования второго варианта решения этой проблемы. Подробности реализации будут оставлены на усмотрение читателя.
- Сравним новый и предыдущий варианты решения проблемы САПР.

Повторный краткий обзор проблемы САПР

В главе 3, *Проблема, требующая создания гибкого кода*, была описана проблема, связанная с поддержкой в приложении двух версий САПР одновременно. По сути, это была первая практическая задача, для решения которой автор воспользовался концепцией шаблонов проектирования.

Проблемная область задачи представляет собой компьютерную систему обеспечения работы большой проектной организации, включающей, в частности, поддержку функционирования САПР.

Основное требование — создать компьютерную программу, которая будет извлекать из базы данных САПР сведения об отдельных элементах деталей и передавать установленной в организации экспертной системе данные, необходимые ей для эффективного проектирования технологических процессов их изготовления. Предполагалось, что эта программа будет изолировать экспертную систему от САПР. Проблема осложнялась тем, что используемая САПР была подвержена частым изменениям. Весьма вероятно, что одновременно будет использоваться несколько различных версий САПР, с каждой из которых экспертная система должна будет работать без каких-либо осложнений.

После предварительных опросов мной были сформулированы общие требования к создаваемой системе, приведенные в табл. 12.1 и схематически представленные на рис. 12.1.

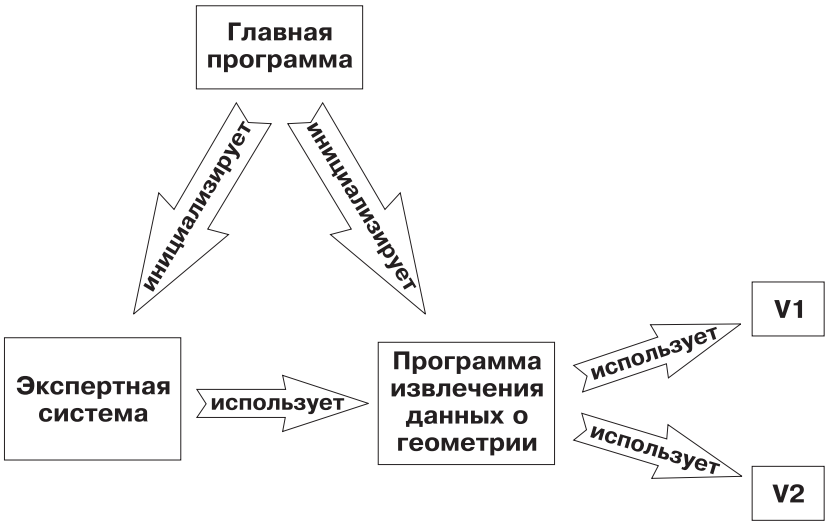

РИС. 12.1. *Концептуальная схема предлагаемого решения*

Таблица 12.1. Общие требования к программе

Требования	Пояснение
Получить информацию о детали от САПР и извлечь из нее описание отдельных элементов	• Система должна быть способна получать от САПР и анализировать информацию о деталях, вырезаемых из металлического листа • Экспертная система должна спроектировать технологическую последовательность операций обработки металлического листа и сгенерировать набор соответствующих инструкций для станка с числовым программным управлением
Поддерживать обработку большого количества различных типов деталей	• Первоначально внимание концентрируется на деталях, изготовляемых из металлического листа • Каждая такая деталь может состоять из множества элементов различных типов (пазы, отверстия, просечки, отверстия специальной формы и отверстия неправильной формы). Маловероятно, что в будущем появятся какие-либо другие типы элементов

Окончание таблицы

Требования	Пояснение
Поддержка нескольких версий САПР	• На рис. 12.1 показано, что программа должна позволять работать с САПР различных версий, причем переход на другую версию не должен требовать внесения каких-либо изменений в экспертную систему

Переосмысление проблемы с применением шаблонов

Мы уже познакомились с несколькими шаблонами проектирования и новой методологией проектирования, предложенной Александером. Согласно этой методологии начать работу следует с анализа самой общей картины, а затем добавлять к ней отдельные детали. Реализация этого подхода при разработке программных проектов состоит из следующих этапов.

1. Отыскать шаблоны, присутствующие в проблемной области, и проанализировать полученный набор.
2. Для полученного набора шаблонов выполнить такие действия:
   а) выбрать шаблон, который в наибольшей степени формирует контекст для всех остальных шаблонов;
   б) применить этот шаблон к самой высокоуровневой схеме реализации проекта;
   в) выявить любые дополнительные шаблоны, которые могут появиться в полученной схеме, и добавить их к набору, который был получен в результате предыдущего анализа;
   г) повторно применить указанный процесс к оставшемуся набору шаблонов, выделенных в результате анализа.
3. Внести в проект все необходимые дополнительные детали. Записать определение классов и их методов.

Следует отметить, что этот подход применим только тогда, когда в терминах шаблонов можно составить представление о всей проблемной области в целом. К сожалению, так получается далеко не всегда. Часто шаблоны проектирования позволяют лишь начать работу, после чего следует самостоятельно наполнить проект содержанием, идентифицируя отношения между концепциями в проблемной области. Методы, позволяющие решить эту задачу, строятся на анализе общности и изменчивости, но их обсуждение выходит за рамки этой книги. Дополнительную информацию о методах анализа общности и изменчивости (Commonality/Variability Analysis, CVA) желающие могут получить на Web-странице по адресу `http://www.netobjectives.com/dpexplained`.

Переосмысление проблемы с применением шаблонов. Этап 1

В предыдущих главах в проблеме САПР были идентифицированы четыре следующих шаблона:

- Abstract Factory;
- Adapter;
- Bridge;
- Facade.

Другие шаблоны на этом этапе нам не потребуются, но я готов обсудить любые дополнительные шаблоны, которые, возможно, будут найдены читателями.

Переосмысление проблемы с применением шаблонов. Этап 2, *a*

Мы будем последовательно применять шаблоны, выбирая их на основании того, в какой степени каждый из них создает контекст для остальных шаблонов.

Для того чтобы определить, какие шаблоны создают контекст для других в обсуждаемой проблемной области, воспользуемся самой простой технологией — сравним все возможные пары шаблонов. В данном случае существует всего шесть различных пар, представленных на рис. 12.2.

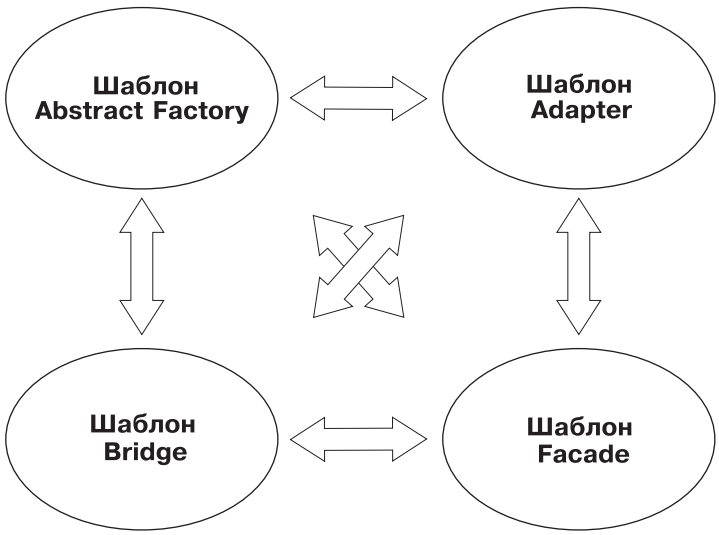

РИС. 12.2. ***Все возможные связи между четырьмя шаблонами***

Если задача содержит еще несколько шаблонов, то подобный анализ может затянуться надолго, что крайне нежелательно. После небольшой практики вы быстро обнаружите, что многие из выделенных шаблонов могут быть легко исключены из про-

цедуры сравнения с важнейшими шаблонами. Чаще всего вам придется иметь дело с пятью шаблонами или около того.

В нашем случае возможны всего несколько комбинаций, рассмотреть которые не составит особого труда.

Что же конкретно подразумевается под выражением "один шаблон создает контекст для другого"? Одно из определений понятия *контекст* гласит, что последний представляет собой взаимосвязь условий, в которых нечто существует или проявляется — т.е. его среду, окружение.

При обсуждении примера с внутренним двором в главе 11, *Как проектируют эксперты*, Александер утверждает, что крыльцо существует в контексте внутреннего двора. Внутренний двор определяет среду или окружение, в котором присутствует крыльцо.

В программной системе один шаблон часто оказывается связанным с другими шаблонами, формируя их контекст. Очень важно и полезно проанализировать, где и как каждый шаблон соотносится с другими шаблонами. Следует выявить контексты, которые этот шаблон создает или предоставляет для других шаблонов, а также те, в которых этот шаблон существует. Вероятно, в некоторых случаях найти желаемые взаимосвязи окажется невозможно, однако такой анализ, несомненно, позволит создать более качественный проект.

Поиск контекста — это важнейший инструмент, который непременно следует добавить к вашему набору инструментов проектирования и анализа.

Правило для выделения контекста

Работая над одним из проектов, я решил проанализировать используемый мной подход к проектированию. В результате я обнаружил в собственной работе одну особенность, которая имела место всегда и практически неосознанно. Я никогда не думаю о том, как создавать экземпляры используемых в программе объектов до тех пор, пока окончательно не уясню для себя, какими эти объекты должны быть. Главное внимание уделяется отношениям между объектами, причем так, как будто они уже существуют. Предполагается, что создание таких объектов не вызовет никаких проблем – в любой момент, когда это потребуется.

Причина состоит в том, что я всегда стремился минимизировать количество информации, которую необходимо держать в голове при проектировании. Как правило, подобный подход характеризуется минимальным риском. Если слишком рано начать беспокоиться о том, как будут создаваться экземпляры требуемых объектов, то это приведет лишь к снижению продуктивности разработки. Лучше подождать с этим до тех пор, пока станет окончательно ясно, что именно требуется создать. Отложим на завтра то, что может подождать, – по крайней мере, до того момента, когда дело действительно дойдет до создания экземпляров объектов!

Возможно, эти рассуждения покажутся вам вполне обоснованными. Но я никогда ранее не слышал, чтобы они воспринимались как некое правило, и потому решил удостовериться в их справедливости, прежде чем возвести в ранг универсального метода. Я доверяю собственной интуиции проектировщика, но, безусловно, тоже могу ошибаться. Поэтому я выяснил мнение по данному вопросу других опытных разработчиков и должен сказать, что все они без исключения следуют этому же правилу. Все это позволяет мне рекомендовать подобный подход и читателям.

***Правило**. Уясните, что должна содержать разрабатываемая система, прежде чем думать о том, как это можно создать.*

Это утверждение совпадает с правилом выявления контекста, предложенным Александером: "Если существует шаблон проектирования, предусматривающий создание объектов, то эти объекты определяют контекст для данного шаблона".

Определяя, какой шаблон создает контекст для других, всегда следует начинать с шаблона Abstract Factory. Контекст шаблона Abstract Factory всегда определяется теми объектами, экземпляры которых требуется создавать, — как следует из приведенных ниже рассуждений.

- Шаблон Abstract Factory требует определить набор методов создания объектов, причем реализация каждого из них будет включать возврат указателя на новый объект xxx.
- В настоящее время еще неизвестно, каким именно будет этот объект xxx.
- Что должен представлять собой xxx, определяется объектами, которые будут использоваться в системе.
- Объекты, используемые в системе, определяются с помощью остальных шаблонов.

Поэтому шаблон Abstract Factory будет просто невозможно выявить, пока не станет известен набор классов, определяемых другими шаблонами. Очевидно, что шаблон Abstract Factory не является шаблоном высшего уровня (т.е. создающим контекст для всех остальных шаблонов). Следовательно, это не тот шаблон, с которого следует начать общий анализ системы.

Фактически, шаблон Abstract Factory будет *последним* применяемым шаблоном из всех, присутствующих в проекте (если только на первых этапах проектирования системы не обнаружится какой-нибудь другой шаблон создания объектов — тогда оба эти шаблона будут соперничать за право оказаться последним).

Шаблоны высшего уровня ограничивают остальные шаблоны

Шаблон высшего уровня (supermost) – это мой термин, обозначающий один или два шаблона, задающие контекст для всех остальных шаблонов в системе. Этот шаблон накладывает ограничения на то, что могут делать остальные шаблоны. Иначе можно было бы назвать этот шаблон, например, *внешним* или *контекстно-задающим*.

У нас осталось три пары шаблонов, которые необходимо рассмотреть:

- Adapter–Bridge;
- Bridge–Facade;
- Facade–Adapter.

Не имея большого опыта работы с шаблонами, довольно трудно понять, какой шаблон зависит от другого, так же, как и найти шаблон, который устанавливает контекст для всех остальных.

Когда отсутствует очевидный выбор, следует последовательно рассмотреть все комбинации шаблонов, стараясь дать ответ на следующие два вопроса.

- Можно ли утверждать, что один шаблон определяет поведение другого шаблона?
- Можно ли утверждать что оба шаблона взаимно влияют друг на друга?

Как мы уже знаем, шаблон Adapter — это шаблон, изменяющий интерфейс класса так, что в результате получается другой интерфейс, отвечающий ожиданиям класса-клиента. В нашем случае адаптируемым интерфейсом является `OOGFeature`. Шаблон Bridge отделяет множество конкретных вариантов абстракции от их реализации. В нашем случае абстракцией является класс ***`Feature`***, а ее реализациями — системы V1 и V2. Получается, что шаблон Bridge нуждается в шаблоне Adapter для модификации интерфейса `OOGFeature`, отсюда можно сделать вывод о том, что Bridge будет использовать шаблон Adapter.

Теперь ясно, что между шаблонами Bridge и Adapter существуют определенные отношения.

Однако можно ли определить один из этих шаблонов без другого, или же они совершенно необходимы друг другу?

Внимательный взгляд на эти шаблоны подсказывает нам, как следует поступить.

- Мы можем говорить о шаблоне Bridge применительно к отделению класса ***`Feature`*** от систем V1 и V2 даже без каких-либо конкретных знаний о том, как будут использоваться эти системы V1 и V2.
- Однако невозможно сказать что-либо об использовании шаблона Adapter для модификации интерфейса программы V2, не имея информации о том, к какому именно виду его следует привести. Но без шаблона Bridge этот интерфейс не существует. Можно сделать вывод, что шаблон Adapter необходим для приведения интерфейса программы V2 к интерфейсу реализации, определяемому шаблоном Bridge.

Таким образом, шаблон Bridge создает контекст для шаблона Adapter. Этот вывод позволяет исключить шаблон Adapter из кандидатов на роль шаблона высшего уровня.

Взаимосвязь между созданием *контекста* и отношением *использования*

Часто оказывается, что когда один шаблон использует другой, то используемый шаблон существует только в контексте того шаблона, который его использует. Вероятно, существуют и исключения из этого правила, но в большинстве случаев оно является верным.

Теперь нам осталось проанализировать только две пары шаблонов: Bridge–Facade и Facade–Adapter.

Сначала рассмотрим отношения между шаблонами Bridge и Facade, поскольку в случае, если шаблон Bridge окажется первичным, необходимость рассматривать отношения в паре Facade–Adapter просто отпадет (напомню, что сейчас мы пытаемся идентифицировать шаблон высшего уровня).

Должно быть совершенно очевидно, что те рассуждения, которые мы применили к паре шаблонов Bridge–Adapter, распространяются и на пару Bridge–Facade.

- Шаблон Facade используется с целью упрощения интерфейса системы V1.
- Но кто же будет использовать новый интерфейс? Видимо, одна из реализаций шаблона Bridge.

Можно сделать вывод, что шаблон Bridge создает контекст для шаблона Facade. А сам шаблон Bridge и является шаблоном высшего уровня.

Следуя рекомендациям Александера, начнем работу с анализа проблемы в целом. Однако, возвратившись к самому началу, мы обнаружим, что контекст для шаблона Bridge все еще не определен.

Переосмысление проблемы с применением шаблонов. Этап 2, *б*

Итак, восстановим последовательность этапов проектирования для поиска контекста, в котором присутствует шаблон Bridge. Нам необходимо разработать систему, предназначенную для преобразования информации о деталях, выбираемой из базы данных САПР, в набор команд для станка с числовым программным управлением, позволяющий изготовить эту деталь (рис. 12.3).

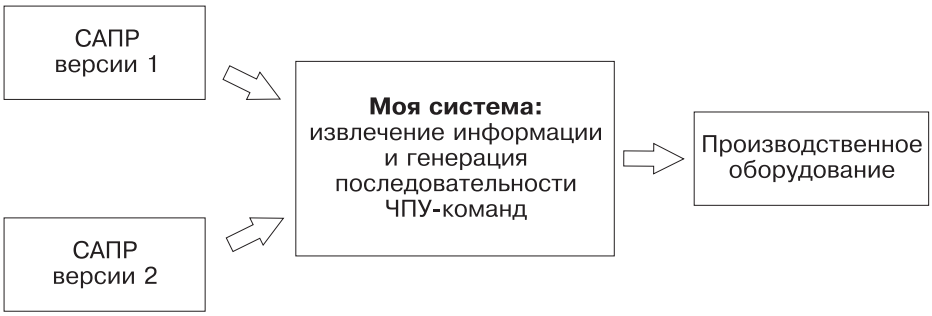

РИС. 12.3. ***Концептуальное представление разрабатываемой системы***

Далее представление о проекте было расширено благодаря использованию технологии объектно-ориентированного проектирования. В частности, предоставление экспертной системе информации о детали будет обеспечивать абстрактный класс ***`Model`***. Причем класс ***`Model`*** в нашем случае должен иметь две конкретные версии — по одной для каждой из используемых версий САПР. Соответствующая схема представлена на рис.12.4.

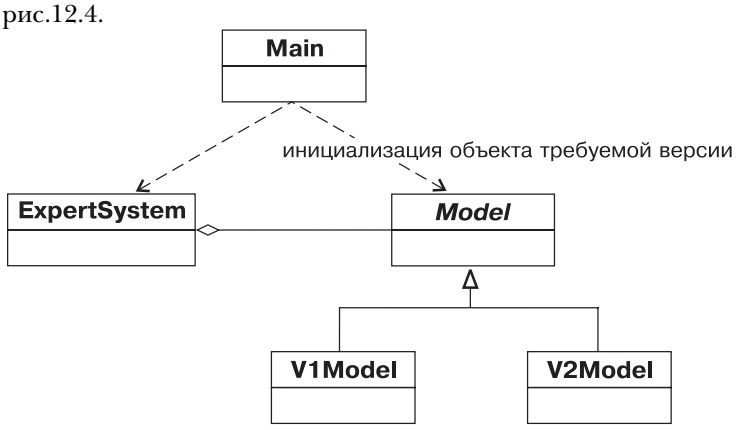

РИС. 12.4. ***Классы, осуществляющие генерацию набора команд для станка ЧПУ***

Напомним, что нам не нужно разрабатывать экспертную систему, а следует просто воспользоваться уже готовой. Поэтому основное внимание можно сосредоточить на проектировании класса **`Model`**. Известно, что класс **`Model`** должен извлекать из базы

данных информацию об отдельных элементах детали, представленных абстрактным классом ***Feature***, как показано на рис. 12.5.[1]

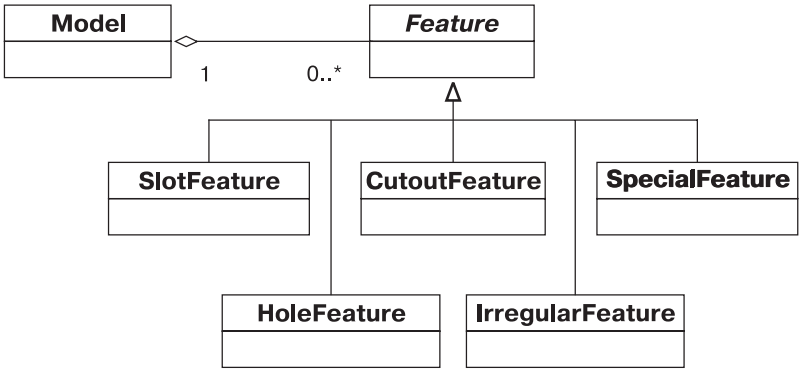

РИС. 12.5. *Схема функционирования класса* ***Model***

Похоже, что именно теперь мы можем применить шаблон Bridge. Очевидно, что в системе существует некоторое множество элементов, описываемых конкретными подклассами абстрактного класса ***Feature*** (абстракция) и несколько версий САПР (реализации). Именно эти объекты образуют контекст для шаблона Bridge.

Шаблон Bridge связывает класс ***Feature*** с различными реализациями САПР. Класс ***Feature*** в нашей схеме соответствует классу ***Abstraction*** шаблона Bridge, тогда как САПР версии V1 и V2 представляют конкретные реализации его абстрактного класса ***Implementor***. Но что можно сказать в отношении абстрактного класса ***Model***? Может, и здесь присутствует шаблон Bridge? Безусловно, нет. Конкретные типы класса ***Model*** можно определить с помощью механизма наследования, так как единственное, что изменяется в отношении абстрактного класса ***Model***, — это используемая реализация. В данном случае для каждой из версий САПР можно создать конкретные классы, производные от класса ***Model*** (рис. 12.6). Попытка применить для класса ***Model*** шаблон Bridge приводит к получению схемы, представленной на рис. 12.7.

Обратите внимание на то, что в схеме на рис. 12.7 шаблон Bridge в действительности отсутствует, поскольку класс **Model** никогда не изменяется, за исключением момента реализации. Что касается класса ***Feature***, то для него действительно существуют различные конкретные типы, дополнительно имеющие различные типы реализаций, поэтому шаблон Bridge здесь присутствует.

Приступим к воплощению шаблона Bridge, используя класс ***Feature*** как абстракцию и системы V1 и V2 как основу реализации. Чтобы перевести нашу задачу в термины шаблона Bridge, обратимся к стандартной схеме этого шаблона и заменим ее классы теми, которые требуются в нашем случае. На рис. 12.8 показана стандартная, упрощенная (иногда ее называют *канонической*) схема шаблона Bridge.

---

[1] *Различия между классами* ***V1Model*** *и* ***V2Model*** *относительно невелики. Поэтому в дальнейших рассуждениях мы будем говорить об обобщенном классе* ***Model***.

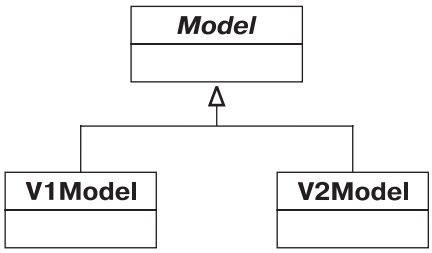

РИС. 12.6. *Использование наследования для представления двух версий описания деталей*

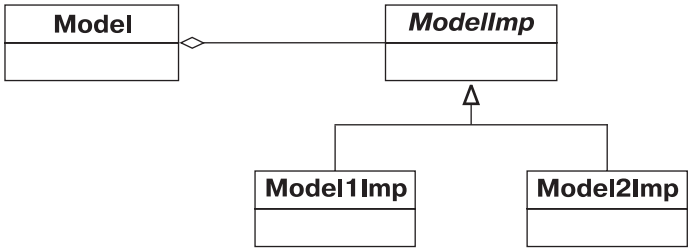

РИС. 12.7. *Использование шаблона Bridge для представления двух версий описания деталей*

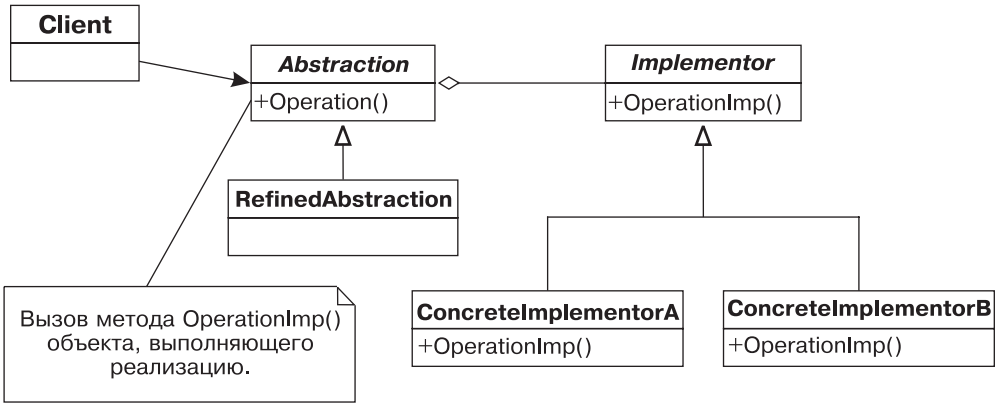

РИС. 12.8. *Каноническая схема шаблона Bridge*

В нашем случае класс ***`Abstraction`*** соответствует классу ***`Feature`***. Существует пять различных типов элементов: пазы, отверстия, просечки, отверстия специальной формы и отверстия неправильной формы. Реализацией являются системы версий V1 и V2. Назовем классы, представляющие эти реализации, **`V1Imp`** и **`V2Imp`**, соответственно. Результат подстановки этих классов в каноническую схему шаблона Bridge представлен на рис. 12.9.

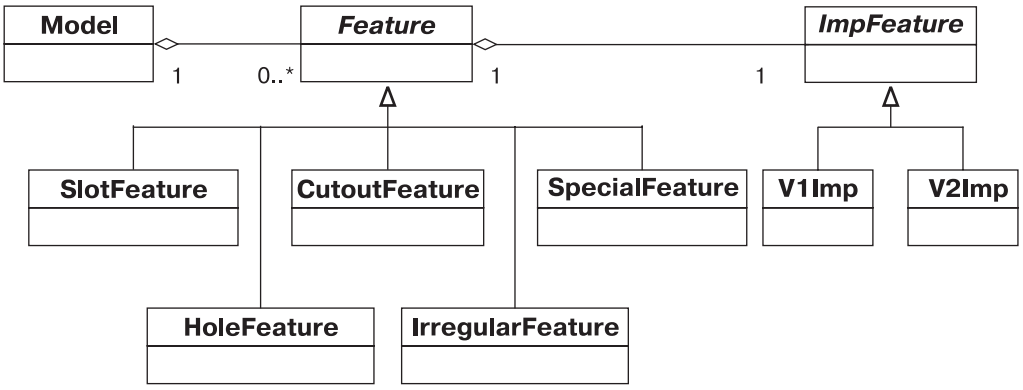

РИС. 12.9. *Применение шаблона Bridge к нашей задаче*

На рис. 12.9 абстрактный класс ***Feature*** реализуется с помощью абстрактного класса ***ImpFeature***, который может быть представлен конкретными классами **V1Imp** или **V2Imp**. В нашем случае класс ***ImpFeature*** должен включать интерфейс, позволяющий классу ***Feature*** получить любую информацию, которая ему потребуется, чтобы передать классу **Model** те сведения, которые он запросил. Таким образом, класс **ImpFeature** должен содержать интерфейс, включающий следующие методы:

- метод `getX` для получения координаты X элемента класса **Feature**;
- метод `getY` для получения координаты Y элемента класса **Feature**;
- метод `getLength` для получения длины элемента класса **Feature**.

Кроме того, он должен включать методы, необходимые только определенным типам классов **Feature**:

- метод `getEdgeType` для получения сведений о типе торца элемента класса **Feature**.

Замечание. Последний метод должны вызвать только те элементы, которые нуждаются в данной информации. Ниже мы обсудим, как использовать подобную контекстную информацию при отладке кода.

Переосмысление проблемы с применением шаблонов. Этап 2, *в*

Возможно, сейчас еще непонятно, как можно довести разработку проекта до полного завершения. Но не стоит заранее беспокоиться — ведь нам предстоит применить еще и другие шаблоны.

Посмотрим на схему, представленную на рис. 12.9, и подумаем, присутствуют ли на ней какие-нибудь другие шаблоны, которые остались неидентифицированными до настоящего момента. Лично я не вижу никаких других шаблонов. Единственная нерешенная пока задача состоит в подключении нашей системы к САПР версий V1 и V2. Именно эту задачу предстоит решить с помощью шаблонов Facade и Adapter.

Переосмысление проблемы с применением шаблонов. Этап 2, *г* (шаблон Facade)

Теперь необходимо проверить, создает ли какой-либо из оставшихся шаблонов контекст для другого шаблона. В нашем случае уже понятно, что шаблоны Facade и Adapter связаны с разными частями проекта и не зависят один от другого. Поэтому применять их можно в любом порядке. Выберем первым шаблон Facade. Результат его применения в проекте представлен на рис. 12.10.

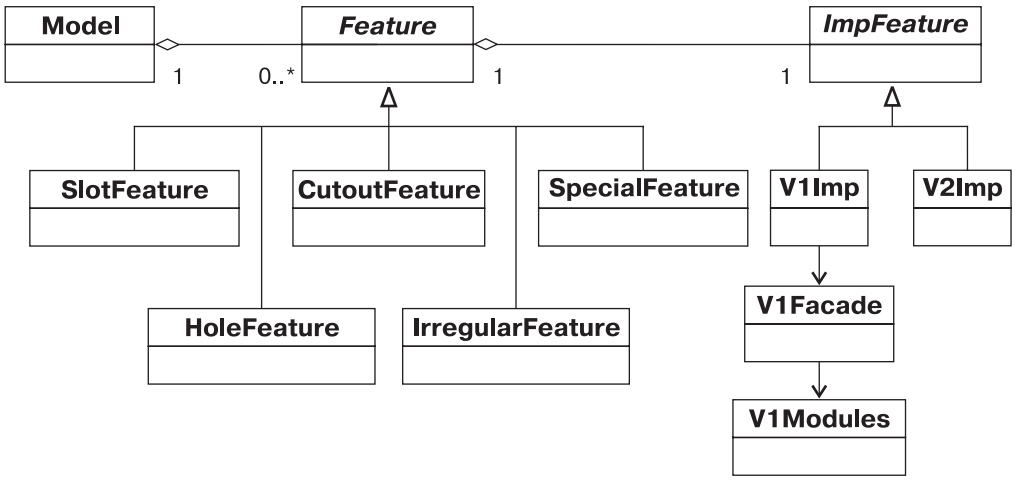

РИС. 12.10. *Результат применения шаблонов Facade и Bridge*

Применение шаблона Facade приводит к тому, что между модулем V1 и классом `V1Imp`, который его использует, будет вставлен промежуточный класс `V1Facade`. Этот класс включает реализацию методов, в выполнении которых нуждается объект `V1Imp`. Каждый подобный метод класса `V1Facade` представляет собой некоторую последовательность вызовов функций системы V1.

Тип информации, которая необходима для вызова этих функций, определяет способ реализации класса `V1Imp`. Например, при использовании системы V1 необходимо сообщить ей, какая деталь нас интересует, и указать идентификаторы ее элементов. Следовательно, все объекты класса `V1Imp`, использующие объект класса `V1Facade`, должны обладать этой информацией. Поскольку данная информация зависит от конкретной реализации, поступать она должна независимо, т.е. эта информация не может присутствовать в запросе от класса ***`Feature`***. Таким образом, в системе V1 каждому объекту класса ***`Feature`*** должен быть определен соответствующий ему объект класса `V1Imp` (для хранения специфической для этой системы информации об элементах). Мы еще вернемся к этому моменту и обсудим его более подробно, когда закончим определение общей структуры проекта.

Использование специфических сведений для отладки кода

Выше в этой главе уже упоминалось, что некоторые из методов на стороне реализации должны вызваться только отдельными типами объектов класса ***Feature***. Зная, кто и что должен вызывать, можно вставить соответствующие проверки в программный код. Конечно, поступать так вовсе не обязательно, и впоследствии при изменении требований к системе подобные проверки может потребоваться удалить. Однако на первых этапах отладки данная практика может оказаться весьма полезной.

Например, в нашей задаче с поддержкой различных версий САПР имеются объекты класса ***Feature***, содержащие соответствующий объект реализации. Один из методов в объектах реализации называется `getEdgeType`. Его выполнение имеет смысл только тогда, когда объект класса ***Feature*** представляет паз или просечку. Никаким другим типам объектов класса ***Feature*** информация о виде торца не требуется. Если их реализация будет выполнена правильно, то метод `getEdgeType` будет вызываться только для элементов, представляющих собой паз или просечку. Можно организовать проверку выполнения этого правила, включив в код метода `getEdgeType` оператор, анализирующий тип вызывающего объекта класса ***Feature***.

Переосмысление проблемы с применением шаблонов. Этап 2, *г* (шаблон Adapter)

Внедрив в проект шаблон Facade, можно поместить в него и шаблон Adapter. Полученный результат представлен на рис. 12.11.

Переосмысление проблемы с применением шаблонов. Этап 2, *г* (шаблон Abstract Factory)

Осталось рассмотреть последний шаблон — Abstract Factory. Однако можно заметить, что теперь в этом шаблоне нет никакой необходимости. Смысл применения шаблона Abstract Factory состоял в получении гарантии, что *все* объекты стороны реализации будут иметь либо тип V1, либо тип V2, в строгом соответствии с версией используемой системы. Однако объект **`Model`** всегда будет знать тип используемой версии. Нет никакого смысла реализовывать данный шаблон, если какой-нибудь другой объект легко может инкапсулировать в себе правила создания новых объектов. Я сохранил шаблон Abstract Factory в наборе используемых шаблонов лишь потому, что в начале проектирования у меня сложилось впечатление, что он действительно может понадобиться. Это также пример того, что предположение о наличии в проекте шаблона, когда в действительности это не так, необязательно приводит к снижению производительности.

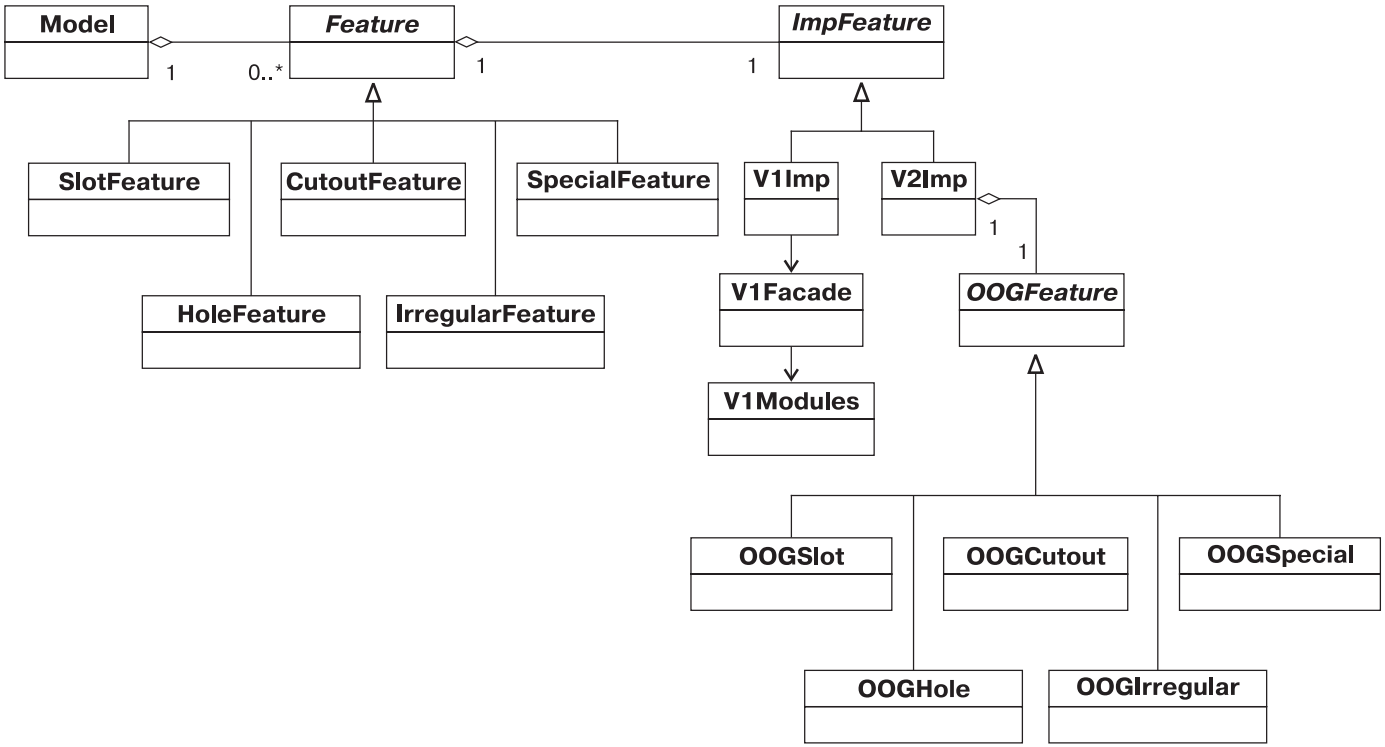

РИС. 12.11. *Результат применения шаблонов Bridge, Facade и Adapter*

Переосмысление проблемы с применением шаблонов. Этап 3

Некоторые детали проекта все еще требуют определенной проработки, и мы продолжим работу, следуя рекомендации Александера проектировать с учетом контекста. Например, решая, как следует реализовать класс **SlotFeature** или класс **V1Imp**, необходимо учитывать методы использования помещенных в проект шаблонов. Так, в нашем случае следует помнить, что благодаря шаблону Bridge методы на стороне абстракции не зависят от стороны реализации. А это означает, что класс шаблона ***Abstraction*** (в нашем случае это класс ***Feature***) и все производные от него конкретные классы (**SlotFeature**, **HoleFeature** и т.д.) не содержат никакой информации о стороне реализации. Все сведения о реализации абстракции должны быть помещены в класс шаблона ***Implementation*** (в нашем случае это класс ***ImpFeature***).

Вышесказанное означает, что классы, производные от абстрактного класса ***Feature***, будут включать универсальные методы, например, `getLocation` или `getLength`, в то время как класс ***ImpFeature*** будет отвечать за конкретные способы извлечения требуемой информации. Например, объект класса **V1Imp** должен знать идентификаторы элементов в системе V1. Поскольку каждый элемент в этой системе имеет уникальный идентификатор, это означает, что потребуется по одному объекту реализации для каждого объекта класса **Feature**. Методы объекта класса **V1Imp** будут использовать этот идентификатор, посылая запрос к объекту класса **V1Facade** для получения информации об объекте.

Сопоставимое решение существует и в отношении реализации для системы V2. В этом случае объекты класса **V2Imp** будут содержать в запросе ссылку на требуемый объект класса **OOGFeature**.

Сравнение полученных результатов с предыдущим решением

Сравним вновь полученное решение, представленное на рис. 12.11, с нашим первым решением, которое еще раз показано на рис. 12.12.

Хороший способ сравнить два решения состоит в том, чтобы *прочитать* их. Другими словами, диаграммы визуально представляют отношения наследования (отношения ***is-a***) и композиции (отношения ***has-a***). Можно прочитать каждую диаграмму, используя, соответственно, слова "является" и "включает" для обозначения присутствующих на ней отношений указанного типа.

В первоначальном решении представление детали включало набор объектов класса ***Feature***. Класс ***Feature*** представляет любые элементы — пазы, отверстия, просечки, отверстия специальной формы и отверстия неправильной формы. Элемент-паз (класс ***SlotFeature***) может быть представлен как объектом системы V1, так и объектом системы V2. Класс **V1Slot** использует систему V1, тогда как класс **V2Slot** использует класс **OOGSlot**. Элемент-отверстие (класс ***HoleFeature***) может быть представлен как объектом системы V1, так и объектом системы V2. Класс **V1Hole** использует систему V1, тогда как класс **V2Hole** использует класс **OOGSlot**. И так далее — утомительно, не правда ли?

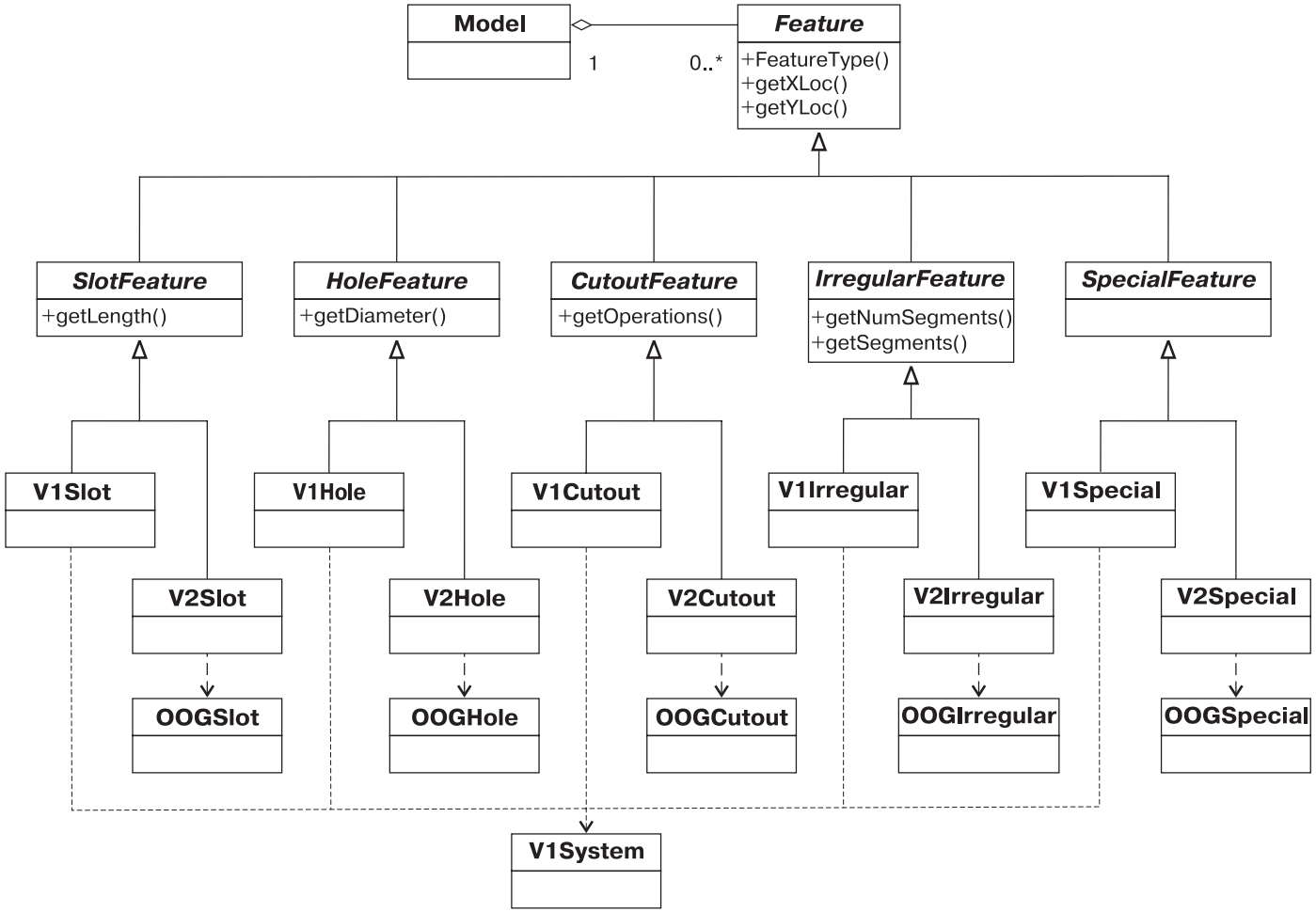

РИС. 12.12. *Первый вариант решения задачи САПР*

А теперь попробуем прочитать новый вариант решения. Существует деталь (класс **`Model`**), включающая набор элементов (класс ***`Feature`***). Класс ***`Feature`*** представляет любые элементы — пазы, отверстия, просечки, отверстия специальной формы и отверстия неправильной формы. Все объекты, представляющие отдельные элементы, включают объекты реализации — это реализация либо для системы версии V1, либо для системы версии V2. Реализация для системы V1 использует класс **`V1Facade`** для доступа к САПР версии V1, а реализация для системы V2 с помощью объекта класса **`OOGFeature`** адаптирует конкретные типы объектов элементов в САПР версии V2. Вот и все. Несомненно, что описание этого варианта звучит намного лучше первого.

Резюме

В этой главе было показано, как стандартный подход к проектированию часто приводит к созданию систем, которые потом очень трудно сопровождать. По причине чрезмерной сосредоточенности на деталях будущей системы — ее классах — теряется общая перспектива, и за отдельными деревьями проектировщик просто не видит леса.

Кристофер Александер предложил лучший путь. Применяя шаблоны проектирования к проблемной области, можно осмыслить задачу при различной степени детализации. Начиная с общей картины, мы постепенно расширяем ее новыми деталями и понятиями. Каждый шаблон добавляет новую информацию к той, которая была накоплена прежде.

Выбрав шаблон, который создает самую общую картину — контекст всей системы, мы последовательно добавляем к нему другие важные шаблоны. Подобным образом удается разработать такую структуру приложения, которую невозможно увидеть, сосредоточившись на одних только классах. Такой подход учит нас проектировать от контекста, а не предпринимать малоэффективные попытки соединить в общее целое отдельные детали, которые были предварительно идентифицированы в проблемной области.

Вспомните двух плотников, о которых шла речь в главе 5, *Первое знакомство с шаблонами проектирования*. Они пытались сделать выбор между различными способами соединения деревянных деталей. Контекст, вот что должно формировать структуру проекта. В процессе проектирования мы часто погружаемся в детали и забываем об общем контексте системы. Детали затуманивают общую картину, заставляя нас сосредоточиться на мелких, локальных решениях. Шаблоны же позволяют нам подняться над множеством деталей и действительно учесть особенности контекста в принимаемых решениях. Используя шаблоны, можно обнаружить и учесть все факторы, действующие в проблемной области, а также воспользоваться наработками других проектировщиков. Поэтому применение шаблонов помогает создавать устойчивые, эффективные и удобные в сопровождении системы.

Глава 13

Обработка возможных вариаций с помощью шаблонов проектирования

Введение

В предыдущих главах было показано, каким образом шаблоны проектирования могут применяться как на локальном, так и на глобальном уровнях. На локальном уровне они демонстрируют, как решить определенную проблему в рамках контекста соответствующего шаблона. На глобальном уровне шаблоны представляют схему взаимосвязей компонентов приложения. Один из способов освоения шаблонов проектирования состоит в изучении наиболее эффективных методов их использования как на глобальном, так и на локальном уровнях, которые в совокупности представляют собой прекрасный инструмент решения проблем.

Другой способ изучения шаблонов проектирования заключается в ознакомлении с теми закономерностями, принципами и алгоритмами, которые положены в их основу. Полученные знания помогут вам существенно развить свои способности как аналитика и проектировщика. Изучение этих принципов и алгоритмов поможет найти выход даже в тех ситуациях, когда требуемые шаблоны проектирования еще не созданы, поскольку набор строительных блоков, необходимых для решения проблемы, уже будет вам известен.

В этой главе мы выполним следующее.

- Познакомимся с принципом открытости-закрытости, положенным в основу многих шаблонов проектирования.
- Обсудим принцип проектирования от контекста, который является важнейшим звеном идеологии шаблонов Александера.
- Рассмотрим принцип включения вариаций.

Принцип открытости-закрытости

Очевидно, что программное обеспечение должно быть расширяемым. Однако всякое изменение программного обеспечения связано с риском внесения ошибок. Для устранения этой дилеммы Бертран Мейер (Bertrand Meyer) предложил принцип открытости-закрытости[1]. Упрощенно данный принцип можно сформулировать так:

---

[1] *Meyer B.* Object-Oriented Software Construction, *Upper Saddle River, N.J.: Prentice Hall, 1997, c. 57.*

"Модули, методы и классы должны быть открыты для расширения, но закрыты для модификации".[2] Другими словами, программное обеспечение следует проектировать таким образом, чтобы можно было расширять его возможности, не изменяя то, что уже существует.

Как бы противоречиво это не звучало поначалу, мы, тем не менее, уже встречались с подобными примерами. При обсуждении шаблона Bridge была продемонстрирована возможность добавления новых реализаций (т.е. расширения возможностей программного обеспечения) без модификации какого-либо из существующих классов.

Принцип проектирования от контекста

Александер рекомендует проектировать, отталкиваясь от контекста, сначала создавая общую картину, а потом переходя к проектированию деталей образующих ее элементов. Большинство шаблонов следуют именно такому подходу, одни в большей степени, другие в меньшей. Из тех четырех шаблонов, с которыми мы уже познакомились, шаблон Bridge является наиболее ярким примером применения данного правила.

Взгляните еще раз на схему шаблона Bridge, приведенную в главе 9, *Шаблон Bridge* (рис. 9.13). Принимая решение о том, как проектировать классы на стороне реализации, учитывайте их контекст — т.е. каким образом классы, производные от класса **`Abstraction`**, будут их использовать.

Например, если разрабатывается система, предназначенная для вывода изображения разнообразных геометрических фигур на оборудование различного типа, ей обязательно потребуется несколько различных типов классов реализаций, для чего целесообразно использовать шаблон Bridge. Общая схема шаблона Bridge свидетельствует, что классы геометрических фигур будут использовать классы реализации (т.е. различные версии графических программ) через общий интерфейс. В данном случае проектирование от контекста, как рекомендует Александер, означает, что сначала следует рассмотреть, что представляют собой фигуры, которые необходимо изобразить — т.е. выяснить, что, собственно, предстоит отображать. Именно эти требования будут определять поведение классов стороны реализации. Так, от этих классов (графических программ), безусловно, потребуется способность отображать линии, окружности и т.д.

Применение методов анализа общности/изменчивости в приложении к тому контексту, в пределах которого существуют интересующие нас классы, позволяет выявить различные случаи их использования (прецеденты), как существующие, так и потенциальные. В результате можно будет принять обоснованное решение о желательном уровне обобщения (генерализации) на стороне реализации, основываясь на предполагаемых затратах, которых потребует поддержка того или иного уровня обобщения. Такой подход часто позволяет найти более общее решение для стороны реализации, чем это ожидалось поначалу, при этом требующее минимальных дополнительных издержек.

Например, для отображения геометрических фигур, на первый взгляд, вполне достаточно линий и окружностей. Однако давайте зададим себе вопрос: "Отображение каких фигур не может быть выполнено с помощью только прямых линий и дуг окружностей?". Ответ очень прост — это эллипсы. Теперь нам потребуется сделать выбор между следующими вариантами.

---

[2] *Прекрасную статью "The Open-Closed Principle" Роберта Мартина (Robert C. Martin) можно найти на Web-странице* `http://www.netobjectives.com/dpexplained`.

- Дополнительно к линиям и окружностям реализовать отображение эллипсов.
- Учитывая, что эллипс является обобщением понятия окружности, реализовать отображение эллипсов вместо окружностей.
- Отказаться от реализации отображения эллипсов, если связанные с этим издержки не компенсируются полученными преимуществами.

Приведенный выше пример иллюстрирует еще одну важную концепцию проектирования — наличие возможности реализовать что-либо вовсе не означает, что это обязательно должно быть выполнено. Мой опыт работы с шаблонами проектирования показывает, что они позволяют очень хорошо изучить характеристики проблемной области. Однако я крайне редко учитываю все обнаруженные ситуации и чаще всего отказываюсь от написания кода для поддержки тех из них, которые в данный момент еще не представлены на практике. Тем не менее, проектирование от контекста с применением шаблонов позволяет предвидеть и учесть появление возможных изменений, создавая систему, продуманно разделенную на классы и заранее приспособленную к ожидаемым изменениям. Шаблоны проектирования помогают понять, в каких местах возможны изменения, но не дают конкретных указаний о том, какими они будут. Однако хорошо продуманный интерфейс будет не только эффективно работать с уже существующими вариациями, но позволит учесть потребности новых потенциальных требований.

Шаблон Abstract Factory — еще один хороший пример проектирования от контекста. Вполне очевидно, что объект-фабрика некоторого типа будет использоваться для координации процесса создания семейств (или наборов) экземпляров объектов. Однако существует несколько различных способов реализовать эту задачу (табл. 13.1).

Таблица 13.1. Возможные варианты реализации шаблона Abstract Factory

Вариант	Описание
Использование производных классов	Классическая реализация шаблона Abstract Factory требует определения производных классов для каждого из существующих наборов объектов. Это несколько громоздкий вариант, но он имеет весомые преимущества, позволяя добавлять новые классы, не затрагивая ни один из уже существующих
Использование одного объекта с переключателями	Если согласиться на изменение класса Abstract Factory по мере необходимости, то можно просто создать один объект, который будет включать все правила. Хотя этот подход противоречит принципу открытости-закрытости, все правила будут реализованы в одном месте, и такую систему будет достаточно легко обслуживать
Использование файла конфигурации и переключателей	Это более гибкий способ по сравнению с предыдущим, но здесь так же время от времени потребуется вносить изменения в программный код
Использование файла конфигурации совместно с механизмом RTTI	Механизм RTTI (RunTime-Type-Identification — определение типа во время выполнения) включает средства создания экземпляров объектов с выбором их типа на основании имени объекта, помещенного в строковую переменную. Реализации этого типа присуща максимальная гибкость, так как новые классы и новые комбинации могут быть добавлены в систему без изменения какого-либо программного кода

Какие же рекомендации можно дать по выбору оптимального варианта реализации шаблона Abstract Factory? Ответ прост — решение нужно принимать исходя из того контекста, в котором он присутствует. Каждый из четырех вариантов реализации имеет свои преимущества в зависимости от следующих факторов.

- Вероятность будущих изменений.
- Важность сохранения неизменности существующей системы.
- Доступность для модификации классов, входящих в отдельные семейства (кто является их создателем — вы или другая группа программистов).
- Используемый язык программирования.
- Наличие и доступность базы данных или файла конфигурации.

Безусловно, этот список не полон, так же, как и список вариантов реализации. Однако каждому должно быть понятно, что попытка решить, как следует реализовать шаблон Abstract Factory, без учета того, как создаваемая система будет использоваться (т.е. без понимания контекста), выглядит, по меньшей мере, глупо.

Как принимаются проектные решения

Делая выбор между альтернативными вариантами реализации, многие разработчики стремятся получить ответ на вопрос: "Какая из возможных реализаций является лучшей?". Однако на самом деле так ставить вопрос нельзя. Проблема заключается в том, что очень редко одна реализация бывает лучше другой во всех отношениях. Предпочтительнее рассматривать каждую альтернативу в отдельности, задаваясь следующим вопросом: "При каких обстоятельствах данная альтернатива будет лучше, чем другие?". Затем следует ответить на такой вопрос: "Какие из этих обстоятельств более всего характерны для заданной проблемной области?". При необходимости легко можно остановиться и вернуться на шаг назад. Подобный подход заостряет внимание проектировщика на возможных вариациях и проблемах масштабирования системы в заданной проблемной области.

Шаблон Adapter иллюстрирует принцип проектирования от контекста потому, что сам он почти всегда обнаруживается в пределах контекста. По определению шаблон Adapter применяется для приведения существующего интерфейса к некоторому другому интерфейсу. Напрашивается следующий вопрос: "Как узнать, к какому виду следует привести существующий интерфейс?". Как правило, до окончательного формирования контекста (определения, к какому именно классу требуется адаптация) ответить на этот трудно.

Выше уже обсуждалось, как шаблон Adapter может быть использован для адаптации класса к роли, предусмотренной для него шаблоном. В частности, при решении проблемы поддержки в приложении нескольких версий САПР потребовалось адаптировать определенную существующую реализацию к виду, необходимому для использования ее в шаблоне Bridge.

Шаблон Facade в терминах контекста очень напоминает шаблон Adapter. Чаще всего он присутствует в контексте других шаблонов или классов. Следовательно, прежде чем приступать к разработке интерфейса, необходимо подождать, пока не будет установлено, кто же его будет использовать.

При первых попытках использования шаблонов проектирования я полагал, что всегда можно отыскать шаблон, который создает контекст для других шаблонов. Во всяком случае, Александеру в его книге всегда удавалось сделать это, правда он

имел дело с шаблонами в области архитектуры. Поскольку множество людей принимало участие в обсуждении проблемы создания языка шаблонов для разработки программного обеспечения, я также задумался: "Почему бы и мне не попробовать?". Ведь кажется абсолютно очевидным, что шаблоны Adapter и Facade всегда будут определяться в контексте чего-то другого.

Однако это оказалось не так. Те разработчики программного обеспечения, которые, как и я, занимаются преподаванием, обладают одним большим преимуществом. Оно состоит в том, что преподаватели принимают участие в гораздо бо́льшем количестве проектов, чем обычные разработчики. В начале моей преподавательской деятельности, связанной с шаблонами проектирования, я полагал, что в последовательности определения контекста шаблоны Adapter и Facade всегда рассматриваются после других шаблонов, не связанных с созданием объектов. Чаще всего именно так и происходит. Однако некоторые системы включают требование создания некоторого специфического интерфейса. В этом случае шаблон Facade или Adapter (единственный из многих шаблонов, присутствующих в системе) может оказаться в системе шаблоном высшего уровня.

Принцип включения вариаций

Несколько человек независимо указали на определенное сходство всех моих проектов, с которыми им приходилось сталкиваться. Дело в том, что иерархия наследования классов в моих проектах редко содержит более двух уровней в глубину. Так происходит потому, что мои проекты имеют типичную для шаблонов проектирования структуру из двух уровней для основных и абстрактных классов. (Правда, существуют и исключения — например, шаблон Decorator, обсуждаемый в главе 15, использует три уровня классов.)

Основная причина заключается в том, что при проектировании я придерживаюсь правила никогда не создавать классов, включающих несколько подверженных вариациям элементов, которые так или иначе связаны между собой. Обсуждавшиеся нами выше шаблоны демонстрируют различные способы эффективного включения вариаций.

Шаблон Bridge являет собой превосходный пример включения вариаций. Все реализации, присутствующие в шаблоне Bridge, различны, но доступ к ним организуется через общий интерфейс. Новые реализации легко могут быть осуществлены в пределах этого интерфейса.

Шаблон Abstract Factory включает вариации в отношении наборов или семейств тех объектов, экземпляры которых создаются. Существует несколько различных путей реализации этого шаблона. Хочу особо обратить ваше внимание на то, что даже если первоначально был выбран некоторый вариант реализации, а позднее было установлено, что имеется другой, лучший путь, реализация шаблона может быть изменена без оказания какого-либо влияния на остальные части системы (интерфейс фабрики остается неизменным, меняется только способ ее реализации). Таким образом, сам принцип построения шаблона Abstract Factory скрывает все вариации в отношении того, как объекты создаются.

Шаблон Adapter — это инструмент, обеспечивающий использование различных по происхождению объектов через общий интерфейс. Это часто бывает необходимо в отношении интерфейсов, вызываемых из многих шаблонов. Таким образом, шаблон Adapter предназначен для сокрытия вариаций в интерфейсах классов.

Шаблон Facade, как правило, не включает изменений. Однако практика дает множество примеров использования шаблона Facade для работы с конкретными подсистемами. В этом случае при появлении новой подсистемы для нее строится собственный шаблон Facade с тем же самым интерфейсом. Этот новый класс представляет собой комбинацию шаблонов Facade и Adapter. Однако, если изначально эти шаблоны использовались для упрощения, то теперь они позволяют сохранить прежний интерфейс и избежать изменения существующих клиентских объектов. Подобное применение шаблона Facade позволяет скрыть вариации в используемых подсистемах.

Однако шаблоны предназначены не только для включения вариаций. Они также определяют отношения между отдельными вариациями. Подробнее об этом речь пойдет в следующих главах. В отношении шаблона Bridge можно дополнительно указать, что он не только определяет и включает вариации в абстракции и реализации, но и определяет отношения между ними.

Резюме

В этой главе мы обсудили, как в шаблонах проектирования иллюстрируется применение двух мощных стратегий проектирования:

- проектирование от контекста;
- включение вариаций в классы.

Использование этих стратегий позволяет отложить принятие решения до тех пор, пока не будут выявлены все возможные варианты. Тщательный анализ контекста решаемой задачи позволяет найти лучшие проектные решения.

Посредством включения вариаций в классы удается учесть потенциальные изменения, которые могут остаться необнаруженными, если предварительно не взглянуть на проект с более общей точки зрения. Это особенно важно для таких проектов, которые не обеспечиваются всеми требуемыми ресурсами в полном объеме (другими словами, для всех проектов). Корректное включение вариаций позволяет ограничиться реализацией только тех функций, которые требуются на текущий момент, не ставя под угрозу сохранение качества проекта в будущем. Не стоит забывать, что попытки выявить *все* потенциальные вариации и обеспечить их поддержку в системе обычно ведут не к созданию лучшей системы, а к краху проекта вообще. Это явление обычно называют параличом от анализа.