ЧАСТЬ II

Ограниченность традиционного объектно-ориентированного проектирования

В этой части

В этой части мы обсудим решение некоторых задач, взятых из реальной жизни, с использованием стандартных методов объектно-ориентированной разработки. Над обсуждаемой ниже проблемой я работал в то время, когда только приступал к изучению шаблонов проектирования.

Глава	Предмет обсуждения
3	• Описание характерной для систем автоматизированного проектирования (САПР) проблемы: выборка информации из больших производственных баз данных систем САПР для нужд сложных и дорогих прикладных программам анализа
	• Поскольку программное обеспечение систем САПР постоянно развивается и модернизируется, упомянутая выше проблема неразрешима без создания гибкого (настраиваемого) кода
4	• Первое решение связанной с системой САПР проблемы, выполненное с использованием стандартных объектно-ориентированных методов
	• Работая над указанной проблемой, я еще не полностью разобрался с теми принципами, которые были положены в основу многих шаблонов проектирования. В результате было предложено решение, от которого позднее

пришлось отказаться. Без особых затруднений я реализовал работающее решение, но впоследствии столкнулся с необходимостью учитывать множество особых случаев

- Найденное решение было связано с существенными проблемами: высокой стоимостью сопровождения и недостаточной гибкостью — а это как раз то, чего я намеревался избежать, обратившись к объектно-ориентированной технологии
- Позднее, в части 4, мы еще вернемся к этой проблеме (глава 12, *Решение задачи САПР с помощью шаблонов проектирования*). Там мы обсудим другое решение этой проблемы, базирующееся на использовании шаблонов проектирования для организации структуры приложения и реализующих его элементов. Такой подход позволяет получить намного более гибкое и простое в обслуживании решение

Очень важно прочесть эту часть книги, поскольку она иллюстрирует типичные проблемы, свойственные традиционному объектно-ориентированному проектированию — необходимость организации иерархии наследования классов, имеющей сильную связанность и слабую связность.

Глава 3

Проблема, требующая создания гибкого кода

Введение

Эта глава содержит краткий обзор проблемы, которую нам предстоит решить: извлечение информации из базы данных большой системы автоматизированного проектирования (САПР) с целью передачи ее на обработку в сложную и дорогостоящую программу анализа. Поскольку программное обеспечение САПР постоянно развивается и модернизируется, решение указанной задачи невозможно без создания гибкого (настраиваемого) программного кода.

В этой главе дается краткий обзор проблемы, связанной с САПР, вводится характерная для этой области терминология и описываются несколько важных особенностей данной проблемы.

Извлечение информации из базы данных САПР

Сначала обсудим мой первый проект, указавший мне дорогу к пониманию всех тех вещей, о которых идет речь в этой книге.

В то время я работал в проектном центре, в котором инженеры использовали САПР для создания чертежей различных деталей, вырезаемых из металлического листа. Примером может служить деталь, изображенная на рис. 3.1.

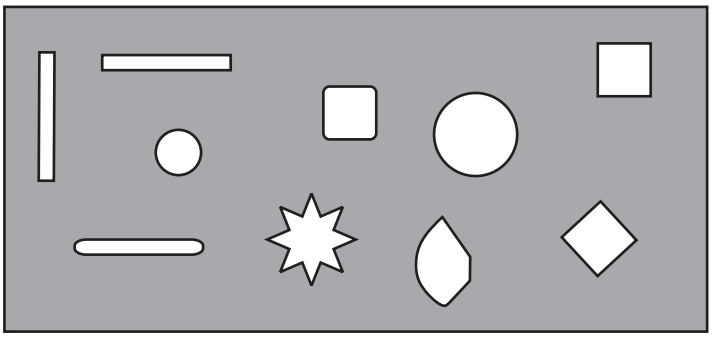

РИС. 3.1. *Пример детали, вырезанной из металлического листа*

Я должен был создать компьютерную программу, которая могла бы извлекать информацию из базы данных САПР и передавать ее экспертной системе в таком виде, чтобы последняя могла использовать полученные данные с определенной целью. Информация о детали нужна была экспертной системе для того, чтобы управлять ее изготовлением. Поскольку модификация самой экспертной системы — задача слишком сложная, а срок ее эксплуатации предполагался существенно бо́льшим, чем время жизни текущей версии программного обеспечения САПР, мне необходимо было написать программу так, чтобы ее можно было легко модифицировать для работы с последующими версиями САПР.

Что такое экспертная система

Экспертная система – это специальная компьютерная программа, которая использует правила, сформулированные экспертами-людьми, для автоматизированного принятия решений. Экспертная система создается в два этапа. На первом определяется набор правил, необходимых для решения поставленных задач. На втором этапе эти правила помещаются непосредственно в компьютер. Обычно для этого как инструмент используют одну из многих доступных коммерческих экспертных систем. Заметим, что для аналитика первый этап – несравненно более трудная задача.

Необходимый понятийный аппарат

Первая задача анализа состоит в том, чтобы усвоить терминологию, используемую пользователями и экспертами в предметной области задачи. Для нас сейчас важны те термины, которые описывают размеры и геометрию листа металла.

На рис. 3.1 показан фрагмент металлического листа определенного размера, в котором вырезаны отверстия различной формы. Эксперты называют эти прорези общим именем "элементы". Таким образом, в нашей задаче изготовленная из металлического листа деталь может быть полностью описана своими внешними размерами и указанием содержащихся в ней элементов.

Типы фигур-элементов, которые могут быть вырезаны в деталях, изготовляемых из металлического листа, описаны в табл. 3.1. Это как раз те элементы, с которыми будет работать экспертная система.

Таблица 3.1. Элементы, которые могут вырезаться в металлическом листе

Фигура	Описание
Паз	Прямая прорезь в листе металла постоянной ширины, ограниченная прямоугольными или закругленными торцами. Пазы могут быть ориентированы под любым углом. Обычно они вырезаются с помощью фрезерных станков. На рис. 3.1 показано три паза, расположенных с левой стороны листа. Один из пазов ориентирован вертикально, а два других — горизонтально
Отверстие	Отверстия круглой формы, прорезанные в металлическом листе. Чаще всего они вырезаются сверлами различного диаметра. На рис. 3.1 слева показано отверстие, окруженное тремя пазами, а также еще одно отверстие бо́льшего диаметра на правой стороне листа

Окончание таблицы

Фигура	Описание
Просечка	Отверстие с прямоугольными или закругленными краями — обычно выдавливается в листе с помощью пуансона. На рис. 3.1 показано три подобных отверстия, причем отверстие внизу справа повернуто под углом в 45 градусов
Отверстия специальной формы	Отверстия в листе металла, не являющиеся пазами, круглыми отверстиями или просечками. Для их быстрого изготовления обычно создается соответствующий пуансон. Типичным примером отверстия специальной формы является гнездо для электрического разъема. На рис. 3.1 примером отверстия специальной формы является просечка в виде звезды
Отверстие неправильной формы	Все остальные виды отверстий. Обычно они выполняются в листе металла с помощью комбинации различных инструментов. Пример отверстия неправильной формы приведен на рис. 3.1 справа внизу

Эксперты, работающие с САПР, пользуются также некоторой дополнительной терминологией, которая будет важна для понимания нашего дальнейшего обсуждения. Эта терминология приведена в табл. 3.2.

Таблица 3.2. Дополнительная терминология САПР

Термин	Описание
Геометрия	Описание внешнего вида детали, изготовленной из металлического листа: расположение каждого элемента и его размеры, а также характеристики внешней формы всей детали в целом
Деталь	Деталь, вырезанная из листа металла, сама по себе. Создаваемая программа должна позволять сохранять сведения о геометрии каждой детали
Набор данных или модель	Набор записей в базе данных САПР, хранящих сведения о геометрии детали
Станок с ЧПУ и ЧПУ-программа	Станок с числовым программным управлением, предназначенный для обработки металла с помощью набора различных режущих головок и работающий под управлением специальной компьютерной программы. Обычно компьютерная программа управляет станком в соответствии с описанием геометрии детали, составляя из отдельных команд обработки последовательность, образующую ЧПУ-программу

Описание задачи

Необходимо создать программу, которая позволит экспертной системе находить и считывать набор данных (модель), содержащий описание геометрии требуемой детали. Затем экспертная система должна будет сгенерировать команды, образующие программу управления станком с числовым программным управлением, изготовляющим данную деталь из металлического листа.

Здесь мы ограничимся рассмотрением только тех деталей, которые вырезаются из металлического листа, хотя САПР, безусловно, могут использоваться и во множестве других случаев.

На концептуальном уровне требуется, чтобы система выполнила следующие действия.

- Проанализировала характеристики детали, изготовляемой из металлического листа.
- Определила технологию изготовления детали, исходя из особенностей имеющихся в ней элементов.
- Подготовила соответствующий набор инструкций, предназначенных для управления технологическим оборудованием. Этот набор инструкций называется ЧПУ-программой и предназначен для управления работой станка с числовым управлением.
- Передала готовую ЧПУ-программу технологическому оборудованию, когда потребуется изготовить ту или иную деталь.

Основная трудность при программировании данной задачи состоит в том, что нельзя просто извлечь описание элементов из базы данных и сразу же сгенерировать ЧПУ-программу для соответствующего станка. Тип и порядок следования составляющих ЧПУ-программу команд зависит от характеристик элементов детали и их взаимного расположения.

Например, рассмотрим отверстие, представляющее собой комбинацию из нескольких элементов — просечки и двух пазов. Один из пазов пересекает просечку вертикально, а другой — горизонтально, как это показано на рис. 3.2.

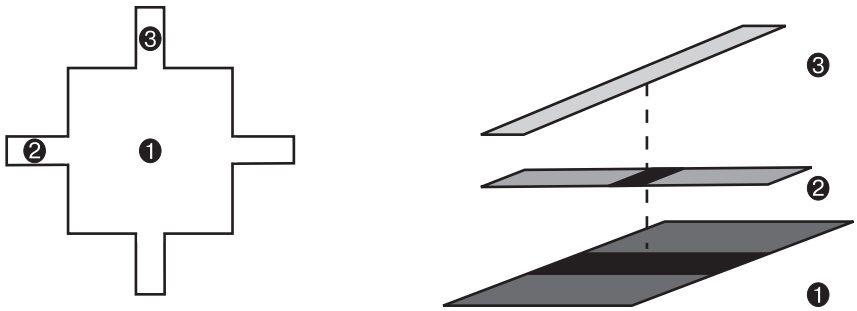

РИС. 3.2. ***Комбинация из просечки и двух пазов. Слева: конечный результат. Справа: составные части элемента***

Важно понять, что в действительности для получения элемента такой формы, которая показана слева, потребуется вырезать в детали три элемента, показанных на рисунке справа. Это происходит потому, что работающие с САПР инженеры, чтобы получить элемент сложной формы, обычно манипулируют простейшими элементами. Поступая так, они предполагают, что этим ускорят изготовление необходимых деталей.

Проблема состоит в том, что для реального технологического оборудования нельзя генерировать команды ЧПУ-программы изготовления всех трех элементов независимо друг от друга и при этом полагать, что деталь получится должным образом.

Существует понятие технологической последовательности изготовления детали, которую необходимо предварительно разработать. В нашем примере, если сначала вырезать в детали пазы, а затем попытаться сделать просечку (как показано на рис. 3.3), то при изготовлении просечки (напомним, что просечка делается с помощью пуансона под воздействием удара большой силы) металлический лист согнется, так как прочность металла в этом месте была ослаблена пазами.

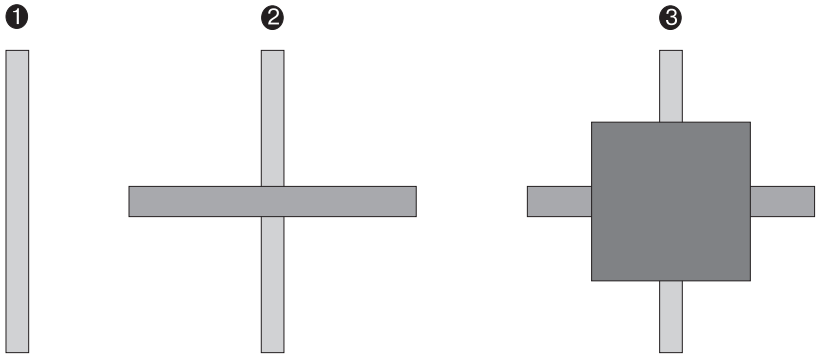

РИС. 3.3. ***Неправильная последовательность изготовления требуемого отверстия – в результате металлический лист согнется***

Отверстие, показанное на рис. 3.2, следует делать, начиная с операции просечки, и лишь затем переходить к прорезке пазов. В этом случае результат будет удовлетворительным, так как пазы вырезаются с помощью фрезерного станка, создающего боковое давление, а не поперечное. Фактически, предварительно сделав просечку, мы даже упростим работу фрезерному станку, как это показано на рис. 3.4.

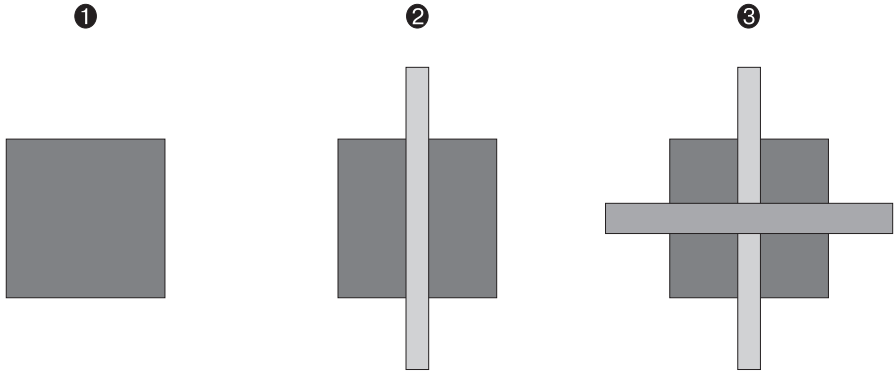

РИС. 3.4. ***Правильный подход к выполнению отверстия требуемой формы***

К счастью, правила для экспертной системы уже были разработаны ранее и мне не пришлось беспокоиться по этому поводу. Мы обсудили указанные проблемы лишь для того, чтобы лучше понять, какого рода информация была необходима экспертной системе.

Возможные подходы к решению проблемы

Программное обеспечение САПР постоянно развивается и совершенствуется. Моя задача состояла в том, чтобы помочь компании продолжать использовать имеющуюся у нее дорогую экспертную систему даже в том случае, если программное обеспечение САПР будет меняться.

На тот момент в компании использовалась САПР версии 1 (V1), но вскоре должна была поступить новая версия (V2). Несмотря на общего производителя, эти версии были несовместимы между собой.

По разным техническим и административным причинам невозможно было перенести данные из одной версии САПР в другую. Таким образом, от экспертной системы требовалось поддерживать обе версии САПР одновременно.

Фактически ситуация была даже несколько хуже: недостаточно было решить указанную проблему только для двух различных версий САПР. Я знал, что вскоре ожидался выпуск третьей версии этой системы, но не знал, когда именно это произойдет. Чтобы сохранить инвестиции компании в экспертную систему, я стремился создать систему с общей структурой, схематически представленной на рис. 3.5.

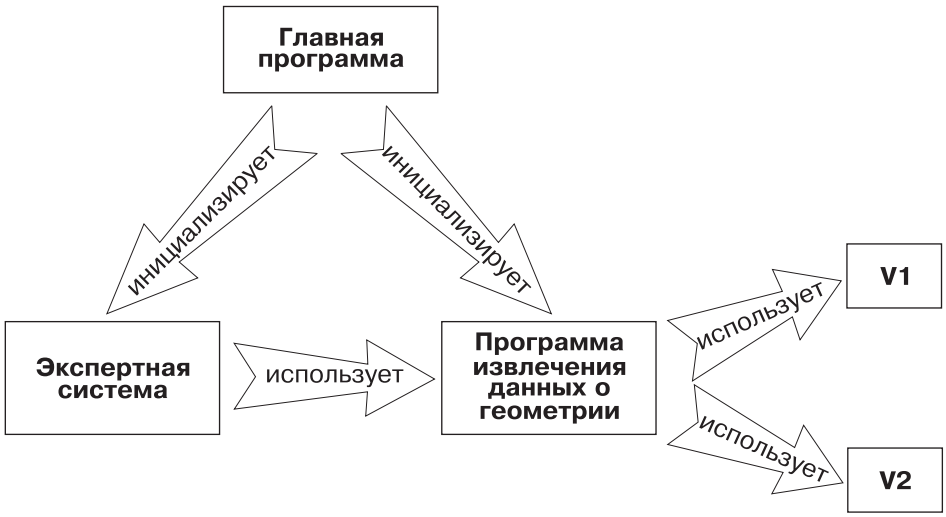

РИС. 3.5. *Концептуальная схема предлагаемого решения*

Согласно этой схеме приложение способно инициализировать работу с любой версией САПР и организовать работу экспертной системы с ней. Однако экспертная система должна быть способна использовать любую версию САПР. Следовательно, программа должна быть реализована так, чтобы версии V1 и V2 для экспертной системы представлялись неразличимыми.

Хотя полиморфизм определенно необходим на уровне получения геометрической информации, нет необходимости поддерживать его на уровне самих элементов. Суть в том, что экспертная система должна знать, с какими именно элементами она имеет дело. Тем не менее, даже при появлении 3-й версии САПР нежелательно вносить какие-либо изменения в саму экспертную систему.

Согласно основными концепциям объектно-ориентированного проектирования диаграмма классов верхнего уровня приложения должна была иметь вид, представленный на рис. 3.6.

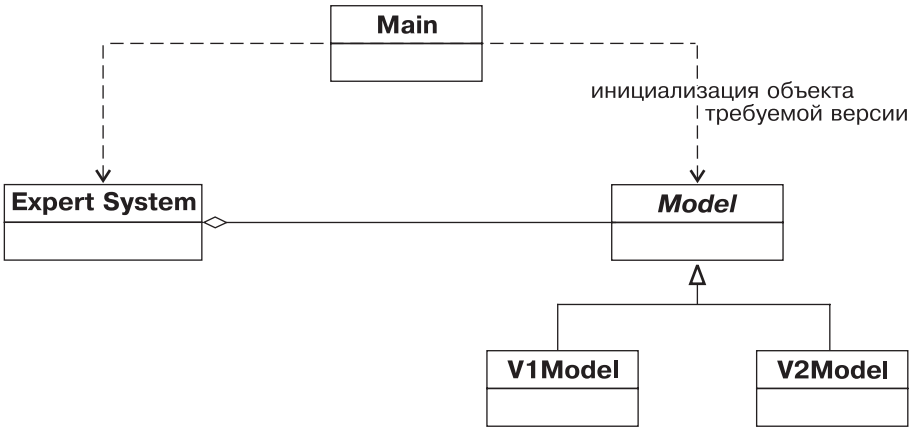

РИС. 3.6. *Диаграмма классов первой версии приложения*[1]

Из этой диаграммы следует, что экспертная система связывается с САПР через класс ***Model***. Класс **Main** отвечает за создание экземпляра объекта требуемой версии класса ***Model*** (т.е. класса **V1Model** или класса **V2Model**).

Остановимся на описании САПР и особенностей ее работы. К сожалению, основные характеристики обеих версий существенно отличались друг от друга.

Версия 1 по существу представляла собой набор библиотек подпрограмм. Для выборки информации из модели требовалось произвести ряд вызовов подпрограмм. Типичный набор запросов мог быть таким, как показано во врезке.

Последовательность действий при работе с САПР версии 1

1. Открыть модель детали `XYZ` и возвратить указатель на нее.
2. Сохранить этот указатель в переменной `H`.
3. Определить количество элементов в модели, указатель на которую хранится в переменной `H`, сохранить полученное значение в переменной `N`.
4. Для каждого элемента (от 1 до значения в переменной `N`) модели, указатель на которую хранится в переменной `H`, выполнить следующие действия:
   а) определить идентификатор i-го элемента модели и сохранить его в переменной `ID`;
   б) определить тип элемента модели, идентификатор которого хранится в переменной `ID`, и сохранить его в переменной `T`.

[1] *Эта и все другие диаграммы классов в данной книге построены с использованием нотации языка UML. Описание этой нотации можно найти в главе 2,* UML — унифицированный язык моделирования.

```
        в)  для каждого элемента модели, идентификатор которого хранится в переменной ID, определить координату
            по оси X и сохранить ее значение в переменной X (при этом значение в переменной T, используется для
            выбора подпрограммы, соответствующей данному типу элемента)…
…           …
```

Приведенная система несовершенна и в действительности не является объектно-ориентированной. Тот, кто работает с этой системой, должен вручную управлять контекстом каждого запроса. При каждом вызове функции необходимо знать, с каким именно элементом ведется работа.

Разработчики программного обеспечения САПР хорошо понимали недостатки, свойственные данному типу систем. Основным побуждением для создания версии V2 было стремление сделать систему объектно-ориентированной. Поэтому данные о геометрии в системе версии V2 хранятся в объектах. Когда к системе обращаются с запросом о выдаче модели, она возвращает объект, представляющий модель. Этот объект модели включает набор объектов, каждый из которых представляет отдельный элемент. Поскольку задача исходно формулировалась в терминах элементов, неудивительно, что система версии V2 использует набор классов, соответствующий типам элементов, обсуждавшихся нами выше, — паз, отверстие, просечка, отверстие специальной формы, отверстие произвольной формы.

Следовательно, в системе версии V2 можно сразу получить набор объектов, описывающих все элементы, которые присутствуют в детали, вырезаемой из металлического листа. Существующие классы элементов показаны на UML-диаграмме, представленной на рис. 3.7.

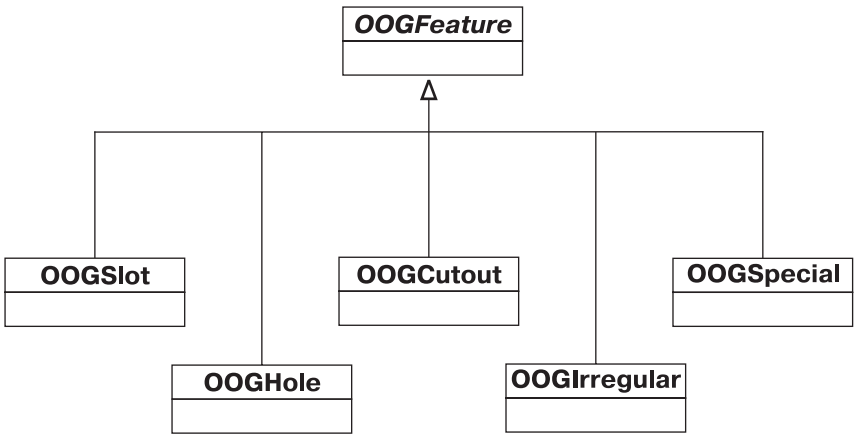

РИС. 3.7. *Классы элементов в системе версии V2*

Префикс `OOG` в имени классов означает "Объектно-ориентированная геометрия" — это просто напоминание о том, что система версии V2 является объектно-ориентированной.

Резюме

В этой главе описана проблема, связанная с эксплуатацией САПР.

- Информация, извлеченная из САПР различного типа, должна быть представлена в одном и том же виде. Это позволит сохранить большие инвестиции, вложенные компанией в экспертную систему, так как последняя сможет работать с различными САПР без дорогостоящих модификаций.
- Существуют две системы, построенные с использованием совершенно разных подходов, но содержащие по существу одну и ту же информацию.

Эта задача имеет много общего с другими проблемами, с которыми мне приходилось сталкиваться в различных проектах. Кратко ее можно сформулировать так: "Существует несколько различных реализаций некоторой системы, и требуется организовать работу так, чтобы другие объекты могли работать с ними одинаково".

Глава 4

Стандартное объектно-ориентированное решение

Введение

В этой главе рассматривается первоначально выбранное решение проблемы, упомянутой в главе 3, *Проблема, требующая создания гибкого кода*. Вначале это решение выглядит приемлемым и быстро приводящим к требуемому результату. Однако в нем не учитывается важное требование, предъявляемое к создаваемой системе, — обеспечение работы с постоянно развивающейся САПР, выражающееся в поддержке нескольких ее версий одновременно.

В этой главе описывается решение проблемы, типичное для объектно-ориентированного подхода. Безусловно, оно не самое лучшее из возможных, но вполне работоспособное.

Замечание. В тексте главы примеры программного кода приведены на языке Java. Примеры соответствующего кода на языке C++ можно найти в конце главы.

Решение с обработкой особых случаев

Рассмотрим две различные версии САПР, описанные в главе 3, *Проблема, требующая создания гибкого кода*. Наша цель заключается в создании системы, способной извлечь информацию из базы данных таким образом, чтобы для экспертной системы не имело значения, какая именно версия системы САПР используется.

Обдумывая решение поставленной задачи, я полагал, что если будет найдено решение для обработки пазов, то его можно будет использовать также и для работы с другими элементами: просечками, отверстиями и т.д. Анализируя работу с пазами, легко заметить, что можно без труда создать производные классы для каждого из возможных вариантов. Другими словами, если для представления пазов определить класс ***`SlotFeature`***, то можно создать производный от него класс **`V1Slot`** для работы с первой версией системы САПР и класс **`V2Slot`** для работы со второй версией системы, — как показано на рис. 4.1.

Окончательное решение заключается в распространении данного подхода на все типы элементов, поддерживаемые в системе, — как показано на рис. 4.2.

Не вызывает сомнения, что подобное решение будет вполне работоспособно. Каждый из классов **`V1xxx`** будет работать с соответствующей ему библиотекой САПР версии V1. В свою очередь, каждый из классов **`V2xxx`** будет взаимодействовать с соответствующим объектом САПР версии V2.

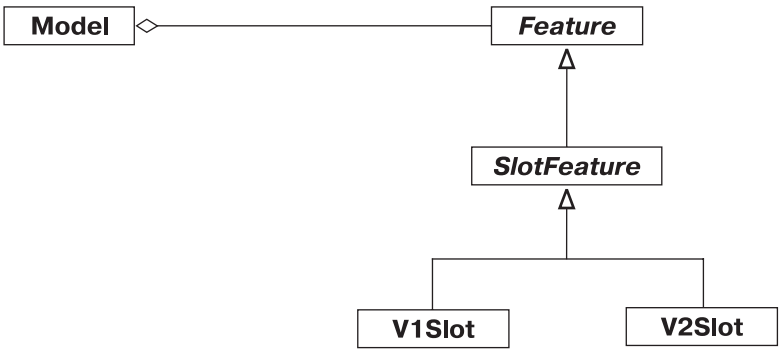

РИС. 4.1. *Проектное решение для обработки пазов*

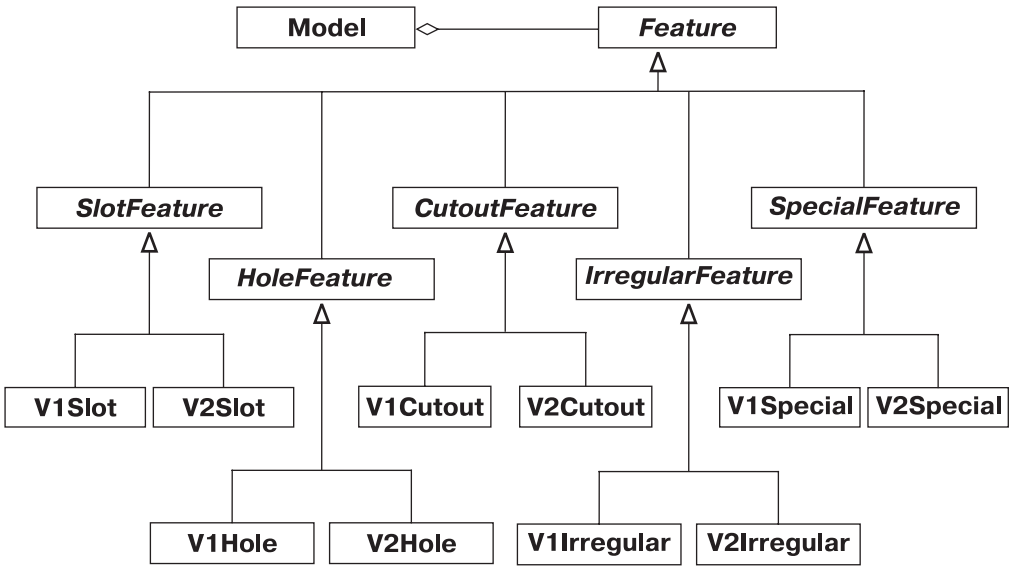

РИС. 4.2. *Исходный вариант решения проблемы извлечения информации*

Чтобы лучше представить себе работу всей системы, рассмотрим каждый класс в отдельности.

- Класс `V1Slot` в этом случае следует реализовать так, чтобы при создании его экземпляров запоминалась модель (тип детали) и ее идентификатор в системе V1. Кроме того, всякий раз при вызове одного из методов объекта класса `V1Slot` для получения определенной информации, этот метод, в свою очередь, должен вызывать некоторую последовательность подпрограмм системы V1, позволяющую достичь требуемого результата.

- Класс **`V2Slot`** следует реализовать подобным же образом, но за исключением того, что в этом случае каждый объект класса **`V2slot`** должен включать объект, представляющий соответствующий паз в системе V2. При этом, обращаясь к объекту класса **`V2Slot`** за информацией, он просто передает поступивший запрос объекту **`OOGSlot`** и возвращает полученный от него ответ объекту, обратившемуся к нему за информацией.

Общая структура приложения, обеспечивающего работу с САПР обеих версий, V1 и V2, показана на рис. 4.3.

В листингах 4.1–4.2 представлены примеры реализации отдельных пар классов в рассматриваемом проекте. Эти примеры помогут вам понять, как практически может быть реализован данный проект. Читатель, не нуждающийся в подобных примерах или предпочитающий работать с языком C++, может просто пропустить приведенный ниже Java-код или же обратиться к примерам программного кода на языке C++, помещенным в конце этой главы (листинги 4.5–4.8).

Листинг 4.1. Реализация представления элементов для системы версии V1 на языке Java

```
// Реализация представления элементов деталей.
// Данный код не был протестирован – это лишь иллюстрация
// возможного проектного решения.

// Каждый представляющий элемент объект для выборки
// запрашиваемой информации должен знать номер модели
// и ID представляемого элемента.
// Обратите внимание на то, как эта информация передается в
// конструктор каждого объекта.

   // Открытие модели
   modelNum = V1OpenModel (modelName);

   nElements = V1GetNumberofElements (modelNum);

   Feature features[]= new Feature[MAXFEATURES];

   // Выполнить для каждого элемента в модели.
   for (i = 0; i < nElements; i++) {
      // Определить тип элемента и создать соответствующий
      // этому элементу объект.
      switch(V1GetType(modelNum, i)) {
         case SLOT:
            features[i]= new V1Slot(modelNum,
                                    V1GetID(modelNum, i));
            break;
         case HOLE:
            features[i]= new V1Hole(modelNum,
                                    V1GetID( modelNum, i));
            break;
         ...
      }
...}
```

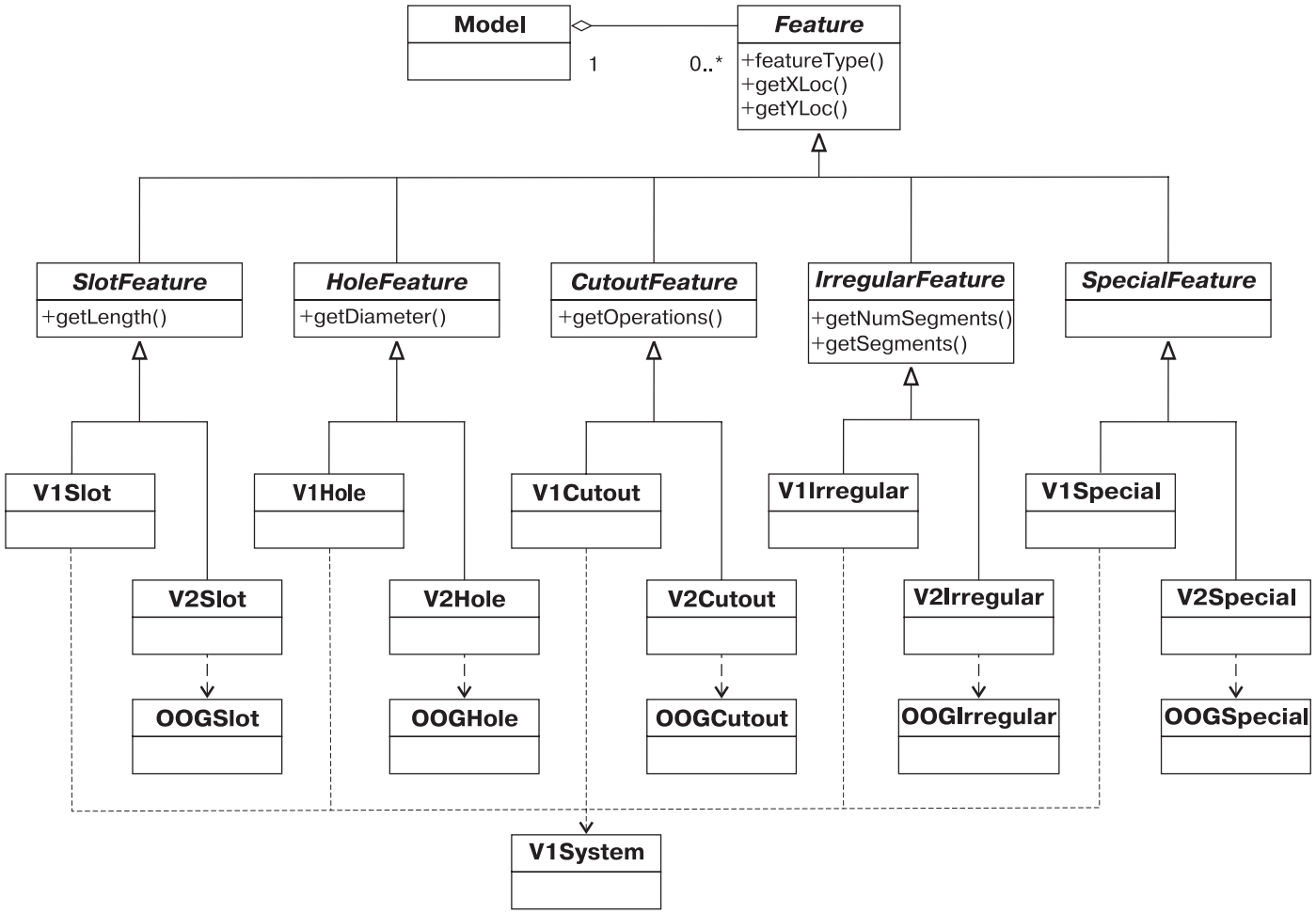

РИС. 4.3. *Первый вариант решения проблемы*

Листинг 4.2. Реализация методов для системы версии V1 на языке Java

```
// ModelNum и myID – закрытые переменные-члены класса,
// содержащие информацию о модели и элементе (в системе V1),
// которому этот объект соответствует.

class V1Slot {
   Double getX () {
      // Вызов соответствующего метода версии V1 с целью
      // получения требуемых данных.
      // Замечание: в действительности для получения
      // информации этот метод может вызывать несколько
      // методов в системе V1.
      return V1GetXforSlot (modelNum, myID);
   }
}

Class V1Hole {
   Double getX () {
      // Вызов соответствующего метода версии V1 с целью
      // получения требуемых данных.
      // Замечание: в действительности для получения
      // информации этот метод может вызывать несколько
      // методов в системе V1.
      return V1GetXforHole (modelNum, myID);
   }
}
```

Листинг 4.3. Реализация представления элементов для системы версии V2 на языке Java

```
// Реализация представления элементов деталей.
// Данный код не был протестирован – это лишь иллюстрация
// возможного проектного решения.

// Каждый представляющий элемент объект для выборки
// запрашиваемой информации должен знать номер
// представляемого элемента в системе V2.
// Обратите внимание на то, как эта информация передается в
// конструктор каждого объекта.

   // Открытие модели
   MyModel = V2OpenModel(modelName);

   nElements = myModel.getNumElements();
   Feature features[]= new Feature[MAXFEATURES];
   OOGFeature oogF;
   // Выполнить для каждого элемента в модели.
      for (i = 0; i < nElements; i++) {
         // Определить тип элемента и создать
         // соответствующий этому элементу объект.
         oogF = myModel.getElement(i);

         switch (oogF.myType()) {
            case SLOT:
               features[i]= new V2Slot(oogF);
               break;

            case HOLE:

               features[i]= new V2Hole( oogF);
```

```
                break;

            ...

        }
    }
```

Листинг 4.4. Реализация методов для системы версии V2 на языке Java

```
// Метод oogF ссылается на объект, представляющий в
// системе V2 элемент, соответствующий данному.

Class V2Slot {
   double getX() {
      // Вызвать соответствующий метод oogF с целью
      // получения требуемой информации.
      return oogF.getX();
   }
}

Class V2Hole {
   double getX() {
      // Вызвать соответствующий метод oogF с целью
      // получения требуемой информации.
      return oogF.getX();}
   }
}
```

На рис. 4.3 представлено несколько методов, необходимых для обработки элементов различных типов. Обратите внимание на то, что они различаются в зависимости от типа элемента. Это означает отсутствие полиморфизма в представлении элементов. Однако это не помеха, поскольку экспертной системе в любом случае необходимо точно знать тип обрабатываемого элемента, так как для описания разных элементов необходима различная информация.

Из сказанного можно сделать вывод, что мы не столько заинтересованы в полиморфизме при представлении элементов, сколько в способности экспертной системы без каких-либо изменений работать с различными версиями САПР.

Однако поставленная цель — прозрачная (для экспертной системы) работа с несколькими различными версиями САПР — выявляет наличие в предложенном выше решении нескольких недостатков.

- *Избыточное количество методов.* Совершенно очевидно, что методы, выполняющие обращение к системе V1, будут во многом похожи друг на друга. Например, методы `V1getX` для класса **`Slot`** и класса **`Hole`** мало чем отличаются друг от друга.
- *Беспорядочность.* Это не самый важный показатель, но, тем не менее, один из факторов, усугубляющих неудовлетворенность найденным решением.
- *Сильная связанность.* Найденное решение характеризуется сильной связанностью, поскольку все элементы косвенно связаны друг с другом. Эта избыточная связанность проявляется в том, что потребуется модификация представления практически *всех* элементов в системе при возникновении одного из следующих событий:

  - появление новой версии САПР;
  - очередная модификация существующей версии САПР.
- *Слабая связность*. Уровень связности в системе невысок потому, что методы, реализующие основные функции системы, рассеяны среди множества различных классов.

Однако наибольшее беспокойство вызывает предполагаемое появление третей версии САПР. Умножение количества классов по правилам комбинаторики фактически приведет к краху проекта. Чтобы понять это, достаточно просто проанализировать количество классов на нижнем уровне диаграммы, представленной на рис. 4.3.

- На этом уровне представлены пять существующих типов элементов.
- Представление каждого типа включает два класса, по одному для каждой версии САПР.
- При появлении третьей версии каждое представление будет включать уже не два, а три класса.
- Поэтому в приложении будет уже не десять, а пятнадцать классов.

В результате будет получено столь громоздкое приложение, что его практически невозможно будет сопровождать.

Данный вариант был моим первым решением и я сразу же понял, что его нельзя считать удовлетворительным. Это ощущение было скорее интуитивным, а не каким-то логическим выводом, сделанным на основе приведенных выше рассуждений. Я чувствовал, что данный вариант имеет существенные недостатки.

Недостатки проведенного анализа: слишком рано чрезмерное внимание было уделено подробностям

Одна из самых распространенных ошибок предварительного анализа состоит в том, что слишком рано чрезмерное внимание уделяется мелким деталям проектируемых процессов. Однако это вполне закономерно, поскольку работать с деталями проще. Решения для реализации отдельных деталей проекта лежат на поверхности, но это не самый лучший подход к началу работы над ним. Излишней детализации проекта следует избегать как можно дольше.

В нашем случае я стремился к одной цели – создать единый API для извлечения информации об элементах деталей. Кроме того, я формулировал определение объектов с точки зрения возлагаемых на них обязанностей, рассматривая все существующие возможности по отдельности. Если обнаруживался новый особый случай, он подлежал независимой реализации. В результате процедура сопровождения создаваемой системы чрезвычайно усложнилась.

Не вызывало сомнения, что существует некое лучшее решение. Однако спустя два часа мне так и не удалось его найти. Как выяснилось позднее, проблема состояла в самом *общем подходе*, который был выбран мной. Подтверждение этому вы найдете ниже, на страницах этой книги.

Руководствуйтесь своей интуицией

Интуитивная оценка – это на удивление эффективный индикатор качества создаваемого проекта. Я полагаю, что будущих разработчиков программного обеспечения следует приучать полагаться на свою интуицию.

Под интуицией я здесь понимаю то особое ощущение, которое возникает у вас при взгляде на что-то такое, что вам не нравится. Я понимаю, что подобное определение звучит крайне ненаучно (и это действительно так), однако мой личный опыт убедительно доказывает, что если проектное решение мне интуитивно не нравится, то где-то рядом обязательно имеется лучшее. Безусловно, иногда можно предложить сразу несколько возможных направлений совершенствования, и выбор лучшего из них бывает не вполне очевиден.

Резюме

В этой главе было показано, как можно быстро решить поставленную задачу, выделяя каждый возможный вариант в особый случай. Требуемое решение совершенно очевидно. Оно позволяет добавлять дополнительные методы без изменения уже существующих. Однако подобному решению свойственно несколько недостатков: высокая избыточность, слабая связность, большое количество классов (с учетом будущих расширений).

Чрезмерное увлечение традиционными методами разработки будет иметь следствием высокую стоимость сопровождения системы (по крайней мере, об этом говорит мой личный опыт).

Примеры программного кода на языке C++

Листинг 4.5. Реализация представления элементов для системы версии V1 на языке C++

```
// Реализация представления элементов деталей.
// Данный код не был протестирован – это лишь иллюстрация
// возможного проектного решения.

// Каждый представляющий элемент объект для выборки
// запрашиваемой информации должен знать номер модели
// и ID представляемого элемента.
// Обратите внимание на то, как эта информация передается в
// конструктор каждого объекта.

   // Открытие модели
   modelNum = V1OpenModel(modelName);

   nElements = V1GetNumberofElements(modelNum);

   Feature *features[MAXFEATURES];

   // Выполнить для каждого элемента в модели
   for (i = 0; i < nElements; i++) {
      // Определить тип элемента и создать соответствующий
      // этому элементу объект.
      switch(V1GetType(modelNum, i)) {
         case SLOT:
            features[i] =
               new V1Slot(modelNum,
                          V1GetID(modelNum, i));
            break;
```

```
         case HOLE:
            features[i] =
               new V1Hole(modelNum,
                        V1GetID(modelNum, i));
            break;

         …
      }
   }
```

Листинг 4.6. Реализация методов для системы версии V1 на языке C++

```
// ModelNum и myID – закрытые переменные-члены класса,
// содержащие информацию о модели и элементе (в системе V1),
// которому этот объект соответствует.

Double V1Slot::getX () {
   // Вызов соответствующего метода версии V1 с целью
   // получения требуемых данных.
   // Замечание: в действительности для получения
   // информации этот метод может вызывать несколько
   // методов в системе V1.
   return V1GetXforSlot(modelNum, myID);
}

Double V1Hole::getX() {
   // Вызов соответствующего метода версии V1 с целью
   // получения требуемых данных.
   // Замечание: в действительности для получения
   // информации этот метод может вызывать несколько
   // методов в системе V1.
   return V1GetXforHole(modelNum, myID);
}
```

Листинг 4.7. Реализация представления элементов для системы версии V2 на языке C++

```
// Реализация представления элементов деталей.
// Данный код не был протестирован – это лишь иллюстрация
// возможного проектного решения.

// Каждый представляющий элемент объект для выборки
// запрашиваемой информации должен знать номер
// представляемого элемента в системе V2.
// Обратите внимание на то, как эта информация передается в
// конструктор каждого объекта.

   // Открытие модели
   MyModel = V2OpenModel(modelName);

   nElements = myModel -> getNumElements();
   Feature *features [MAXFEATURES];

   OOGFeature *oogF;
   // Выполнить для каждого элемента в модели.
   for (i = 0; i < nElements; i++) {
```

```
      // Определить тип элемента и создать
      // соответствующий этому элементу объект.
      oogF = myModel -> getElement(i);

      switch(oogF -> myType()) {
         Case Slot:
         features[i] = new V2Slot(oogF);
         break;

         case HOLE:
         features[i]= new V2Hole(oogF);
         break;

      ...

      }
   }
```

Листинг 4.8. Реализация методов для системы версии V2 на языке C++

```
// Метод oogF ссылается на объект, представляющий в
// системе V2 элемент, соответствующий данному.

Double V2Slot::getX() {
   // Вызвать соответствующий метод oogF с целью
   // получения требуемой информации.
   return oogF -> getX();
}

Double V2Hole:getX() {
   // Вызвать соответствующий метод oogF с целью
   // получения требуемой информации.
   return oogF -> getX();
}
```

ЧАСТЬ III

Шаблоны проектирования

В этой части

В этой части мы познакомимся с шаблонами проектирования — узнаем, что они собой представляют и как используются. Здесь описываются четыре шаблона, позволяющих успешно решить задачу о системе САПР, обсуждавшуюся в главе 3, *Проблема, требующая создания гибкого кода.* Мы рассмотрим каждый из них в отдельности, а затем в применении к нашей задаче. Описание этих шаблонов опирается на объектно-ориентированные концепции, изложенные в основополагающей работе "банды четырех" (Гамма, Хелм, Джонсон и Влиссайдес) *Design Patterns: Elements of Reusable Object-Oriented Software.*

Глава	Предмет обсуждения
5	• Знакомство с шаблонами проектирования
	• Концепция шаблонов проектирования, их первоначальное появление в архитектуре и последующий перенос этой концепции в область проектирования программного обеспечения
	• Мотивы для изучения шаблонов проектирования
6	• Шаблон **`Facade`** (фасад), его определение, использование и реализация
	• Шаблон **`Facade`** в применении к задаче, связанной с САПР
7	• Шаблон **`Adapter`** (адаптер), его определение, использование и реализация
	• Сравнение шаблонов **`Facade`** и **`Adapter`**
	• Шаблон **`Adapter`** в применении к задаче, связанной с САПР
8	• Некоторые важные концепции объектно-ориентированного программирования: полиморфизм, абстракция, классы и инкапсуляция. Обсуждение ведется на основе материала, рассмотренного в главах 5–7

9 • Шаблон **Bridge** (мост). Этот шаблон несколько сложнее двух предыдущих, однако намного полезнее, поэтому и рассматривается более подробно

• Шаблон **Bridge** в применении к задаче, связанной с САПР

10 • Шаблон **Abstract Factory** (абстрактная фабрика), позволяющий создавать семейство объектов, его определение, использование и реализация

• Шаблон **Abstract Factory** в применении к задаче, связанной с САПР

Ознакомившись с материалом этой части, читатель узнает, что такое шаблоны проектирования, чем они полезны, и получит представление о четырех конкретных шаблонах. Кроме того, он узнает, как эти шаблоны можно применить к нашей задаче о системе САПР. Однако полученной информации будет все еще недостаточно для создания проекта лучшего, чем самый первый, основанный на простейшем механизме наследования. Успеха мы достигнем лишь тогда, когда научимся использовать шаблоны проектирования способом, отличным от того, который применяет большинство разработчиков-практиков.