Лекция 4: Паттерны проектирования и их представление в нотации UML

Аннотация: Паттерны объектно-ориентированного анализа и проектирования, их классификация. Паттерны проектирования в нотации языка UML. Полный список паттернов проектирования GoF. Паттерн Фасад, его обозначение в нотации языка UML и пример реализации. Паттерн Наблюдатель, его обозначение в нотации языка UML и пример реализации

Ключевые слова: объектно-ориентированный анализ и проектирование, атрибут, архитектурный паттерн, паттерн проектирования, паттерн анализа, паттерн тестирования, паттерн реализации, general, responsibility, assignment, software, pattern, ganged, предметной области, ARIS, architecture, information, уровень представления, IBM, оптимизация программного кода, UML, Паттерн, прямоугольник, Дополнение, список, factory method, @mediat, представление, кооперация, класс, интерфейс, базы данных, диаграммы классов, адрес, операции, диаграмма, subject, БД, произвольное, таблица, очередь, model, tools, Pascal, Java, визуальное моделирование, каноническая диаграмма, информация, ALM, extreme programming, экстремальное программирование

Паттерны, их классификация

При реализации проектов по разработке программных систем и моделированию бизнес-процессов встречаются ситуации, когда решение проблем в различных проектах имеют сходные структурные черты. Попытки выявить похожие схемы или структуры в рамках объектно-ориентированного анализа и проектирования привели к появлению понятия паттерна, которое из абстрактной категории превратилось в непременный атрибут современных CASE-средств

Паттерны ООАП различаются степенью детализации и уровнем абстракции. Предлагается следующая общая классификация паттернов по категориям их применения:

- Архитектурные паттерны
- Паттерны проектирования
- Паттерны анализа
- Паттерны тестирования
- Паттерны реализации

Архитектурные паттерны(Architectural patterns) - множество предварительно определенных подсистем со спецификацией их ответственности, правил и базовых принципов установления отношений между ними.

Архитектурные паттерны предназначены для спецификации фундаментальных схем структуризации программных систем. Наиболее известными паттернами этой категории являются паттерны **GRASP** (General Responsibility Assignment Software Pattern). Эти паттерны относятся к уровню системы и подсистем, но не к уровню классов. Как правило, формулируются в обобщенной форме, используют обычную терминологию и не зависят от области приложения. Паттерны этой категории систематизировал и описал К. Ларман.

Паттерны проектирования (Design patterns) - специальные схемы для уточнения структуры подсистем или компонентов программной системы и отношений между ними.

Паттерны проектирования описывают общую структуру взаимодействия элементов программной системы, которые реализуют исходную проблему проектирования в конкретном контексте. Наиболее известными паттернами этой категории являются паттерны **GoF** (Gang of Four), названные в честь Э. Гаммы, Р. Хелма, Р. Джонсона и Дж. Влиссидеса, которые систематизировали их и представили общее описание. Паттерны GoF включают в себя 23 паттерна. Эти паттерны не зависят от языка реализации, но их реализация зависит от области приложения.

Паттерны анализа (Analysis patterns) - специальные схемы для представления общей организации процесса моделирования.

Паттерны анализа относятся к одной или нескольким предметным областям и описываются в терминах предметной области. Наиболее известными паттернами этой группы являются паттерны бизнес-моделирования **ARIS** (Architecture of Integrated Information Systems), которые характеризуют абстрактный уровень представления бизнес-процессов. В дальнейшем паттерны анализа конкретизируются в типовых моделях с целью выполнения аналитических оценок или имитационного моделирования бизнес-процессов.

Паттерны тестирования (Test patterns) - специальные схемы для представления общей организации процесса тестирования программных систем.

К этой категории паттернов относятся такие паттерны, как тестирование черного ящика, белого ящика, отдельных классов, системы. Паттерны этой категории систематизировал и описал М. Гранд. Некоторые из них реализованы в инструментальных средствах, наиболее известными из которых является IBM Test Studio. В связи с этим паттерны тестирования иногда называют стратегиями или схемами тестирования.

Паттерны реализации (Implementation patterns) - совокупность компонентов и других элементов реализации, используемых в структуре модели при написании программного кода.

Эта категория паттернов делится на следующие подкатегории: паттерны организации программного кода, паттерны оптимизации программного кода, паттерны устойчивости кода, паттерны разработки графического интерфейса пользователя и др. Паттерны этой категории описаны в работах М. Гранда, К. Бека, Дж. Тидвелла и др. Некоторые из них реализованы в популярных интегрированных средах программирования в форме шаблонов создаваемых проектов. В этом случае выбор шаблона программного приложения позволяет получить некоторую заготовку программного кода.

Паттерны проектирования в нотации языка UML

В сфере разработки программных систем наибольшее применение получили паттерны проектирования GoF, некоторые из них реализованы в популярных средах программирования. При этом паттерны проектирования могут быть представлены в наглядной форме с помощью рассмотренных обозначений языка UML.

Паттерн проектирования в контексте языка UML представляет собой параметризованную кооперацию вместе с описанием базовых принципов ее использования.

При изображении паттерна используется обозначение параметризованной кооперации языка UML (рис. 4.1), которая обозначается пунктирным эллипсом. В правый верхний угол эллипса встроен пунктирный прямоугольник, в котором перечислены параметры кооперации, которая представляет тот или иной паттерн.

Изображение паттерна в форме параметризованной кооперации

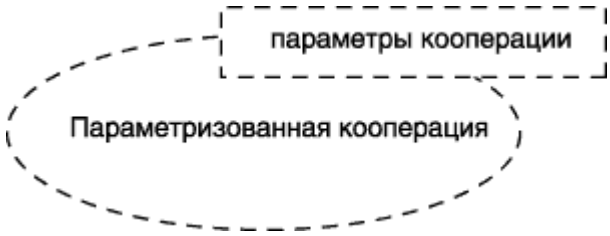

Рис. 4.1. Изображение паттерна в форме параметризованной кооперации

В последующем параметры паттерна могут быть заменены различными классами, чтобы получить реализацию паттерна в рамках конкретной кооперации. Эти параметры специфицируют используемые классы в форме ролей классов в рассматриваемой подсистеме. При связывании или реализации паттерна любая линия помечается именем параметра паттерна, которое является именем роли соответствующей ассоциации. В дополнение к диаграммам кооперации особенности реализации отдельных паттернов представляются с помощью диаграмм последовательности.

Паттерны проектирования позволяют решать различные задачи, с которыми постоянно сталкиваются проектировщики объектно-ориентированных приложений. Ниже представлен полный список паттернов проектирования GoF и краткое описание назначения каждого из них (таблица 4.1).

Таблица 4.1. Полный список паттернов проектирования GoF

№	Название паттерна	Перевод	Назначение паттерна
1	Abstract Factory	Абстрактная фабрика	Предоставляет интерфейс для создания множества связанных между собой или независимых объектов, конкретные классы которых неизвестны.
2	Adapter(синоним - Wrapper)	Адаптер (Обертка)	Преобразует существующий интерфейс класса в другой интерфейс, который понятен клиентам. При этом обеспечивает совместную работу классов, невозможную без данного паттерна из-за несовместимости интерфейсов.
3	Bridge	Мост	Отделяет абстракцию класса от его реализации, благодаря чему появляется возможность независимо изменять то и другое.
4	Builder	Строитель	Отделяет создание сложного объекта от его представления, позволяя использовать один и тот же процесс разработки для создания различных представлений.
5	Chain of Responsibility	Цепочка обязанностей	Позволяет избежать жесткой зависимости отправителя запроса от его получателя, при этом объекты-получатели связываются в цепочку, а запрос передается по цепочке, пока какой-то объект его не обработает.
6	Command	Команда	Инкапсулирует запрос в виде объекта, обеспечивая параметризацию клиентов типом запроса, установление очередности

			запросов, протоколирование запросов и отмену выполнения операций.
7	Composite	Компоновщик	Группирует объекты в иерархические структуры для представления отношений типа "часть-целое", что позволяет клиентам работать с единичными объектами так же, как с группами объектов.
8	Decorator	Декоратор	Применяется для расширения имеющейся функциональности и является альтернативой порождению подклассов на основе динамического назначения объектам новых операций.
9	Facade	Фасад	Предоставляет единый интерфейс к множеству операций или интерфейсов в системе на основе унифицированного интерфейса для облегчения работы с системой.
10	Factory Method	Фабричный метод	Определяет интерфейс для разработки объектов, при этом объекты данного класса могут быть созданы его подклассами.
11	Flyweight	Приспособленец	Использует принцип разделения для эффективной поддержки большого числа мелких объектов.
12	Interpreter	Интерпретатор	Для заданного языка определяет представление его грамматики на основе интерпретатора предложений языка, использующего это представление.
13	Iterator	Итератор	Дает возможность последовательно перебрать все элементы составного объекта, не раскрывая его внутреннего представления.
14	Mediator	Посредник	Определяет объект, в котором инкапсулировано знание о том, как взаимодействуют объекты из некоторого множества. Способствует уменьшению числа связей между объектами, позволяя им работать без явных ссылок друг на друга и независимо изменять схему взаимодействия.
15	Memento	Хранитель	Дает возможность получить и сохранить во внешней памяти внутреннее состояние объекта, чтобы позже объект можно было восстановить точно в таком же состоянии, не нарушая принципа инкапсуляции.
16	Observer	Наблюдатель	Специфицирует зависимость типа "один ко многим" между различными объектами, так что при изменении состояния одного объекта все зависящие от него получают извещение и автоматически обновляются.
17	Prototype	Прототип	Описывает виды создаваемых объектов с помощью прототипа, что позволяет создавать новые объекты путем копирования этого прототипа.

18	Proxy	Заместитель	Подменяет выбранный объект другим объектом для управления контроля доступа к исходному объекту.
19	Singleton	Одиночка	Для выбранного класса обеспечивает выполнение требования единственности экземпляра и предоставления к нему полного доступа.
20	State	Состояние	Позволяет выбранному объекту варьировать свое поведение при изменении внутреннего состояния. При этом создается впечатление, что изменился класс объекта.
21	Strategy	Стратегия	Определяет множество алгоритмов, инкапсулируя их все и позволяя подставлять один вместо другого. При этом можно изменять алгоритм независимо от клиента, который им пользуется.
22	Template Method	Шаблонный метод	Определяет структуру алгоритма, перераспределяя ответственность за некоторые его шаги на подклассы. При этом подклассы могут переопределять шаги алгоритма, не меняя его общей структуры.
23	Visitor	Посетитель	Позволяет определить новую операцию, не меняя описаний классов, у объектов которых она вызывается.

В качестве примеров рассматриваются два паттерна проектирования, которые нашли наибольшее применение при проектировании программных систем: паттерны **Фасад** и **Наблюдатель**.

Паттерн Фасад и его обозначение в нотации языка UML

Паттерн Фасад предназначен для замены нескольких разнотипных интерфейсов доступа к определенной подсистеме некоторым унифицированным интерфейсом, что существенно упрощает использование рассматриваемой подсистемы. Общее представление паттерна проектирования Фасад может быть изображено с помощью следующей диаграммы параметризованной кооперации (рис. 4.2).
Общее представление паттерна проектирования Фасад:

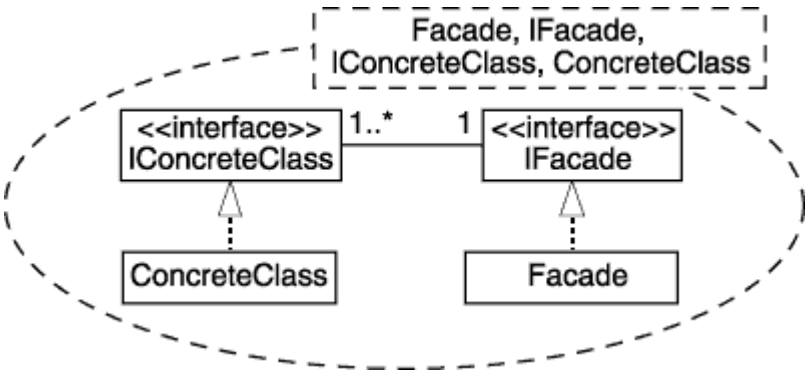

Рис. 4.2. Общее представление паттерна проектирования Фасад

Изображенная параметризованная кооперация содержит 4 параметра: класс Facade (Фасад), интерфейс IFacade, интерфейсы IConcreteClass и конкретные классы ConcreteClass, в которых

реализованы интерфейсы IConcreteClass. Пунктирная линия со стрелкой в форме треугольника служит для обозначения отношения реализации (не путать с отношением обобщения классов).

При решении конкретных задач проектирования данный паттерн может быть конкретизирован. В этом случае вместо параметров изображенной кооперации должны быть указаны классы, предназначенные для решения отдельных задач.

Ниже приведен пример, который иллюстрирует использование паттерна Фасад для выполнения операций по заданию и считыванию адресов из базы данных сотрудников. Фрагмент соответствующей диаграммы классов содержит 2 класса: Адрес и интерфейс к операциям этого класса IАдрес (рис. 4.3). При задании адреса нового сотрудника необходимо обратиться к этому интерфейсу и последовательно выполнить операции: задатьУлицу(), задатьДом(), задатьКорпус(), задатьКвартиру(), используя в качестве аргумента идентификационный номер нового сотрудника. Для получения информации об адресе сотрудника, необходимо также обратиться к этому интерфейсу и последовательно выполнить операции: прочитатьУлицу(), прочитатьДом(), прочитатьКорпус(), прочитатьКвартиру(), используя в качестве аргумента идентификационный номер интересующего сотрудника.

Фрагмент диаграммы классов до применения паттерна Фасад

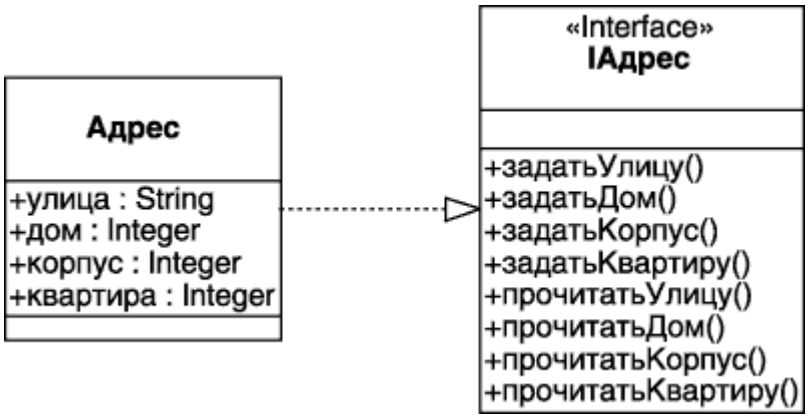

Рис. 4.3. Фрагмент диаграммы классов до применения паттерна Фасад

Очевидно, отслеживать при каждом обращении правильность выполнения этих последовательностей операций неудобно. С этой целью к данному фрагменту следует добавить еще один интерфейс, реализацию паттерна Фасад для рассматриваемой ситуации. Соответствующий фрагмент модифицированной диаграммы классов будет содержать 4 класса (рис. 4.4), изображенные таким образом, чтобы иллюстрировать реализацию параметрической кооперации (рис. 4.2).

Конкретная реализация паттерна проектирования Фасад

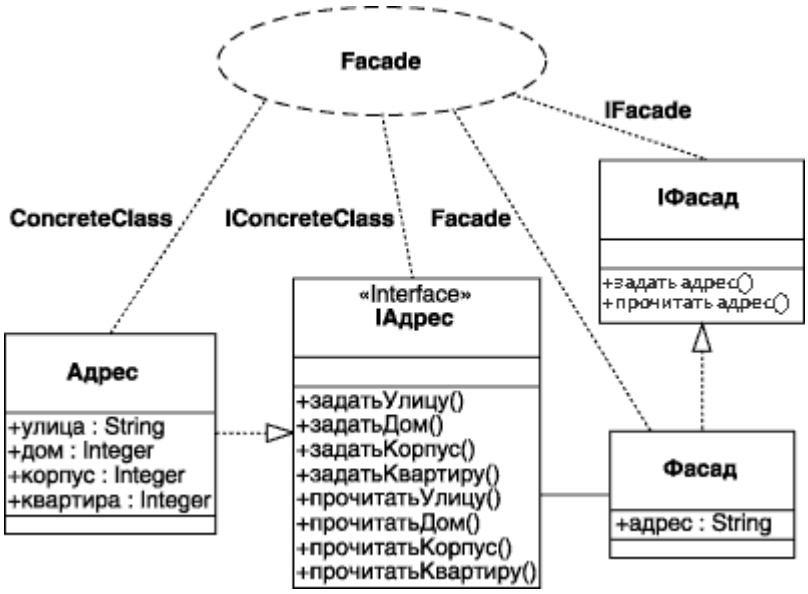

Рис. 4.4. Конкретная реализация паттерна проектирования Фасад

При задании адреса нового сотрудника в этом случае достаточно обратиться к интерфейсу IФасад и выполнить единственную операцию: задатьАдрес(), используя в качестве аргумента идентификационный номер нового сотрудника. Для получения информации об адресе сотрудника также достаточно обратиться к этому интерфейсу и выполнить единственную операцию: прочитатьАдрес(), используя в качестве аргумента идентификационный номер интересующего сотрудника. Реализацию данных операций следует предусмотреть в классе Фасад. Взаимодействие объектов этих классов может быть представлено с помощью диаграммы последовательности (рис. 4.5).

Диаграмма последовательности для выполнения операции задания адреса

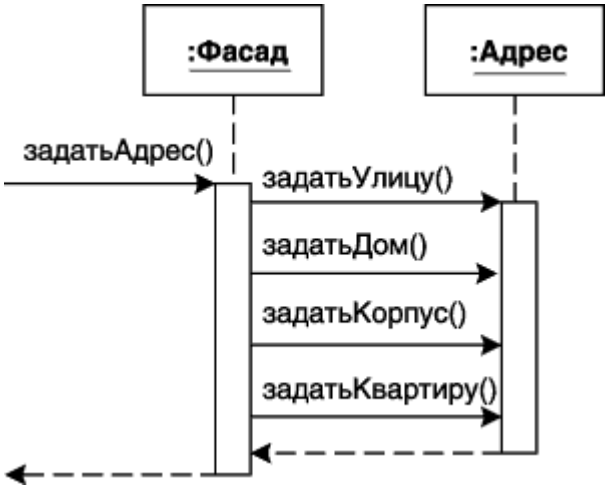

Рис. 4.5. Диаграмма последовательности для выполнения операции задания адреса

Аналогичная диаграмма последовательности может быть построена для выполнения операции по чтению адреса. Использование паттерна Фасад обеспечивает для клиента не только простоту доступа к информации об адресах, но и независимость представления объектов класса Адрес от запросов клиентов. Это обстоятельство особенно актуально при изменении формата представления информации или смене соответствующей базы данных. В этом случае потребуется внести изменения только в реализацию операций класса Фасад.

Паттерн Наблюдатель и его обозначение в нотации языка UML

Паттерн Наблюдатель предназначен для контроля изменений состояния объекта и передачи информации об изменении этого состояния множеству клиентов. В общем случае паттерн Наблюдатель также может быть изображен в виде параметризованной кооперации (рис. 4.6).

Общее представление паттерна проектирования Наблюдатель

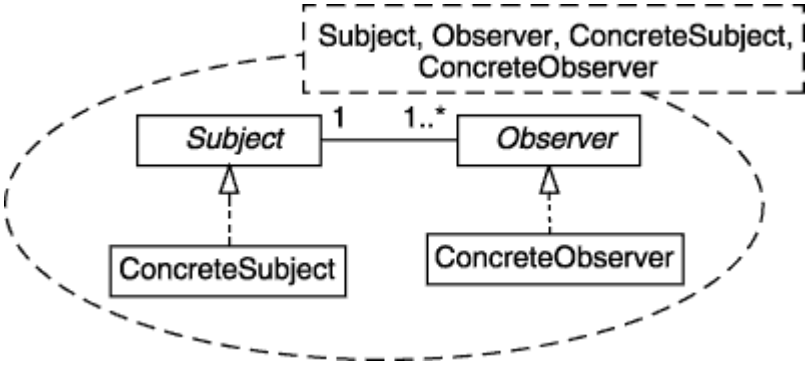

Рис. 4.6. Общее представление паттерна проектирования Наблюдатель

Изображенная параметризованная кооперация содержит 4 параметра: абстрактный класс Subject (Субъект), класс ConcreteSubject (Конкретный Субъект), абстрактный класс Observer (Наблюдатель) и класс ConcreteObserver (Конкретный Наблюдатель). Пунктирная линия со стрелкой в форме треугольника служит для обозначения отношения обобщения классов.

При решении конкретных задач проектирования данный паттерн также может быть конкретизирован. В этом случае вместо параметров изображенной кооперации должны быть указаны классы, предназначенные для решения отдельных задач.

Теперь можно рассмотреть пример, который иллюстрирует использование паттерна Наблюдатель для отслеживания изменений в таблице БД и отражении этих изменений на диаграммах. Для определенности можно использовать таблицу БД MS Access и две диаграммы - круговую и столбиковую. Фрагмент соответствующей диаграммы классов содержит 5 классов (рис. 4.7).

Конкретная реализация паттерна проектирования Наблюдатель

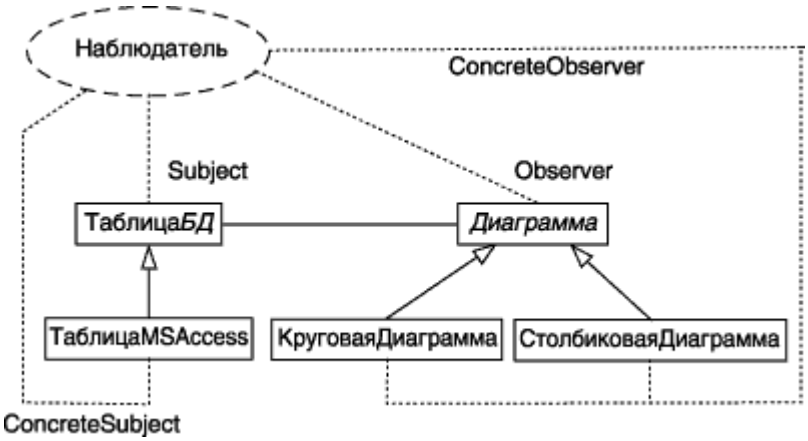

Рис. 4.7. Конкретная реализация паттерна проектирования Наблюдатель

В этом случае за субъектом Таблицей MS Access может "следить" произвольное число наблюдателей, причем их добавление или удаление не влияет на представление информации в БД. Класс Таблица MS Access реализует операции по отслеживанию изменений в

соответствующей таблице, и при их наличии сразу информирует абстрактного наблюдателя. Тот в свою очередь вызывает операции по перерисовке соответствующих диаграмм у конкретных наблюдателей, в качестве которых выступают классы Круговая Диаграмма и Столбиковая Диаграмма.

Использование паттерна Наблюдатель не только упрощает взаимодействие между объектами соответствующих классов, но и позволяет вносить изменения в реализацию операций классов субъекта и наблюдателей независимо друг от друга. При этом процесс добавления или удаления наблюдателей никак не влияет на особенности реализации класса субъекта.

В настоящее время паттерны проектирования реализованы в инструментальном средстве Model Maker 7 компании ModelMaker Tools BV (http://www.modelmakertools.com), которое поддерживает нотацию языка UML и позволяет генерировать программный код на языке Delphi Pascal. Паттерны проектирования также реализованы в CASE-средстве Together 2005 компании Borland (http://www.borland.com), которое поддерживает нотации языка UML версий 1.4 и 2.0 и позволяет генерировать программный код на языке Java. Описание этих средств и особенностей реализации в них нотаций языка UML будет рассмотрено в отдельных курсах лекций.

В заключение следует отметить, что язык UML представляет собой нотацию для визуального моделирования программных систем и бизнес-процессов. В то же время описание языка UML не содержит сведений относительно того, каким образом и в какой последовательности следует разрабатывать канонические диаграммы при выполнении конкретных проектов. Соответствующая информация относится к области методологии проектирования программных систем. В настоящее время наиболее известны следующие методологии:

Rational Unified Process (RUP), разработанная и поддерживаемая компанией IBM Rational Software
Microsoft Solutions Framework (MSF), разработанная и поддерживаемая компанией Microsoft
Application Lifecycle Management (ALM), разработанная и поддерживаемая компанией Borland
Extreme Programming (XP) - экстремальное программирование, поддерживаемое открытым сообществом независимых разработчиков
Описание этих методологий отводится на самостоятельную работу и на индивидуальное задание.