УДК 004.053+ 004.055+ 004.057.8

ПРОЕКТИРОВАНИЕ МНОГОАЛГОРИТМИЧЕСКИХ МЕТОДОВ НА ПРИМЕРЕ БИБЛИОТЕКИ КЛАССОВ ДЛЯ АНАЛИЗА И МОДЕЛИРОВАНИЯ ВРЕМЕННЫХ РЯДОВ

Кесиян Грант Арутович
аспирант

Уртенов Махамет Хусеевич
д.ф.-м.н., профессор
Кубанский государственный университет, Краснодар, Россия

В данной работе описывается модификация паттерна проектирования «Стратегия», которая информирует клиентов о различных типах алгоритмов (стратегиях) без раскрытия особенностей их реализации, а так же позволяет контролировать соответствие алгоритмов и методов. Кроме этого, подробно описывается архитектура алгоритмов и методов, которые используют вышеуказанную модификацию

Ключевые слова: СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ, МНОГОАЛГОРИТМИЧЕСКИЕ МЕТОДЫ, ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ, ДЕЛЕГИРОВАНИЕ

UDC 004.053+ 004.055+ 004.057.8

DESIGNING OF MULTIALGORITHMIC METHODS ON THE EXAMPLE OF LIBRARY OF CLASSES FOR THE ANALYSIS AND MODELLING OF TIME SERIES

Kesiyan Grant Arutovich
postgraduate student

Urtenov Makhamet Khuseevich
Dr.Sci.Phys.-Math., professor
Kuban State University, Krasnodar, Russia

In this article, we describe a modification of the design pattern of "Strategy", which informs customers about the different types of algorithms (strategies) without disclosing the features of their implementation, as well as allows you to control compliance with the algorithms and methods. In addition, the architecture of the algorithms and methods that use the above modification is described in details

Keywords: SYSTEM DESIGN, MULTIALGORITHMIC METHODS, DESIGN PATTERNS, DELEGATING

Введение

В наше время сфера автоматизации бизнеса диктует необходимость создания продукции для конечного потребителя в кратчайшие сроки. При этом мало кто задумывается над эффективностью применения системного проектирования, что, например, позволило бы использовать удачные архитектурные решения повторно и, в конечном счете, ускорило бы разработку программных продуктов.

В данной работе, в рамках программной библиотеки анализа и моделирования временных рядов, описывается архитектура классов, которая позволяет создавать многоалгоритмические методы. Основным достижением работы является представленная модификация паттерна проектирования «Стратегия», которая избавляет от недостатков

использования стандартного шаблона. Кроме того, мы описываем преимущества проектирования в соответствии с интерфейсами и абстрактными классами, а так же демонстрируем механизм делегирования и применения паттерна «Шаблонный метод».

1 Многоалгоритмические методы

Одним из центральных тезисов данной работы является положение об адаптации паттерна проектирования «Стратегия» («Strategy») к реализации разрабатываемой библиотеке анализа и моделирования временных рядов. Дело в том, что важной задачей при программировании математических методов является задача поддержания одновременно нескольких алгоритмов для каждого метода. Поэтому при создании библиотеки классов мы использовали шаблон «Стратегия», который позволит:

– иметь несколько разных вариантов алгоритма для каждого метода. Например, можно определить два варианта алгоритма, один из которых требует больше времени, а другой - больше памяти;

– скрыть от клиента данные алгоритма. Данное решение позволить не раскрывать сложные, специфичные для алгоритма структуры.

К сожалению, применение паттернов в чистом виде вызывает ряд сложностей. В нашем случае, использование паттерна «Стратегия» имеет два основных недостатка:

– клиенты должны «знать» о различных стратегиях. Для выбора подходящего алгоритма клиент должен понимать, чем они отличаются. Поэтому придется раскрыть клиенту некоторые особенности реализации;

– при решении нашей задачи необходимо контролировать соответствие алгоритмов и методов, так как паттерн «Стратегия» не запрещает одним экземплярам контекста (метода) использовать стратегии (алгоритмы), которые предназначались для других экземпляров.

Для разрешения вышеуказанных минусов была предложена архитектура классов, показанная на рисунке 1:

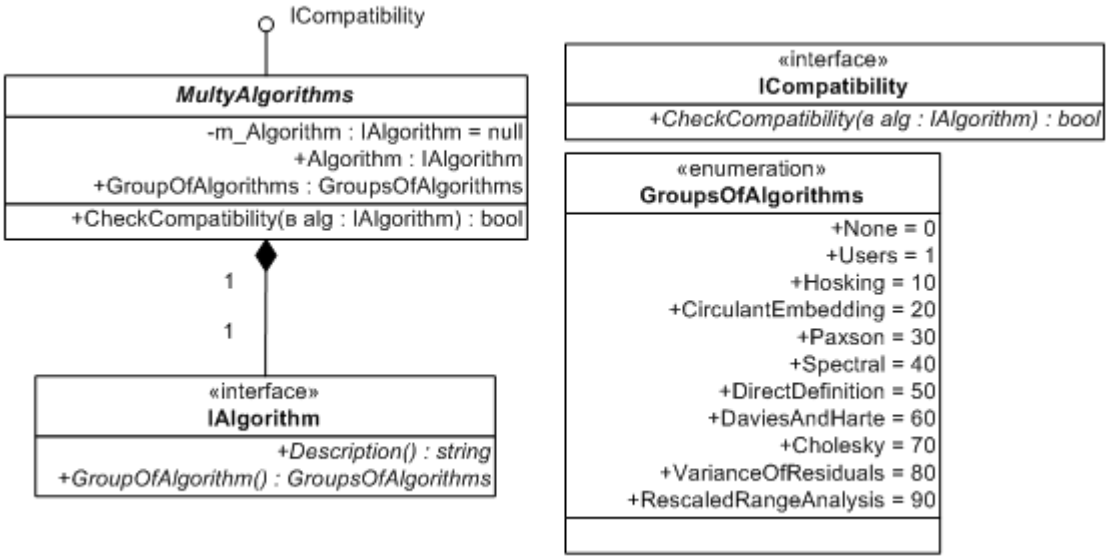

Рисунок 1. Диаграмма классов для методов, поддерживающих множество алгоритмов

Как видно из рисунка 1, абстрактный класс *MultyAlgorithms*, который выполняет роль контекста [1], наследуется от интерфейса *ICompatibility*, а также аргегирует интерфейс *IAlgorithm* (стратегия). В отличие от обычного паттерна «Стратегия», наш интерфейс *IAlgorithm* не содержит метода, который и представляет собой реализацию алгоритма. Это сделано для того, чтобы будущие потомки *IAlgorithm* содержали любую сигнатуру такого метода. Дело в том, что определить заранее какой набор параметров понадобится для того или иного метода невозможно, а универсализация набора параметров, например, списком объектов, приведет к неоднозначному восприятию клиентами сигнатур методов, а так же к сложным проверкам при их реализации.

Интерфейс *IAlgorithm* содержит сигнатуры следующих методов:

– *Description* – метод, возвращающий описание особенностей данного алгоритма в виде строки. Таким образом, информация о том или ином алгоритме закладывается уже на уровен интерфейса и доступна только через него. Поэтому клиент не будет знать ничего о реализации, но при этом он всегда сможет получить информацию о данном алгоритме, реализующем некоторый метод. Заметим, что вместо строкого типа можно было бы использовать перечислимый тип в качестве возвращаемого типа данного метода, который содержал бы набор привычных видов алгоритмов. Но такой подход ограничил бы возможность расширения видов алгоритмов, так как весь набор видов был бы фиксирован на уровне компиляции;

– *GroupOfAlgorithm* – данный метод возвращает значение перечислимого типа *GroupsOfAlgorithm*. Каждое такое значение свидетельствует о принадлежности данного алгоритма определенной группе. Другими словами, каждый алгоритм будет связан только с одним методом. Метод *GroupOfAlgorithm* и будет использоваться для того, чтобы контролировать соответствие алгоритмов и методов. На рисунке 1 продемонстрированы такие значения типа *GroupsOfAlgorithm* как None – неопределенная группа, Users – пользовательская, группы, реализующие методы генерации временных рядов с заданным параметром Херста (Hosking – метод Хоскинга, CirculantEmbedding - метод циркулянтных вложений и т.д.), группы, реализующие методы, оценивающие экспоненту Херста (VarianceOfResiduals - метод дисперсии остатков, RescaledRangeAnalysis – метод нормированного размаха).

Интерфейс *ICompatibility* «наделяет» своих наследников возможностью проверять на соответствие алгоритмы и методы. Метод *CheckCompatibility*, который содерджиться в данном интерфейсе, имеет один аргумент – алгоритм, который необходимо проверить на возможность применения в текущем методе. Результат метода

CheckCompatibility это значение булевского типа, которое и говорит об искомой совместимости. Конечно, сами по себе интерфейсы мало чем полезны. Однако когда они реализуются в классах или структурах своим уникальным образом, они позволяют получать доступ к дополнительным функциональным возможностям за счет применения ссылки на них в полиморфной форме [1, 2]. Следовательно, мы предполагаем о будущем расширении библиотеки, в которой могут появиться другие классы, реализующие метод *CheckCompatibility*.

Программный код абстрактного класса *MultyAlgorithms* продемонстрирован на рисунке 2:

```
public abstract class MultyAlgorithms : ICompatibility
{
    private IAlgorithm m_Algorithm = null;
    public IAlgorithm Algorithm
    {
        get { return m_Algorithm; }
        set
        {
            if (!CheckCompatibility(value))
                throw new ArgumentException(@"Алгоритм не совместим с
            данным методом!");
            m_Algorithm = value;
        }
    }
    public abstract GroupsOfAlgorithms GroupOfAlgorithms { get; }
    public bool CheckCompatibility(IAlgorithm alg)
    {
        if (alg.GroupOfAlgorithm == GroupOfAlgorithms)
            return true;

        return false;
    }
}
```

Рисунок 2. Программный код абстрактного класса *MultyAlgorithms* (контекст в паттерне проектирования «Стратегия»)

Из рисунка 2 видно, что метод доступа set (мутатор) свойства *Algorithm* содержит проверку на соответствие данного

многоалгоритмического метода и алгоритма, который передается с помощью контекстно-зависимого ключевого слова value. При этом используется метод *CheckCompatibility*, реалзация которого содержит абстрактное свойство *GroupOfAlgorithms*. Таким образом, мы предоставляем возможность наследникам класса *MultyAlgorithms* самим решить о том, к какой группе алгоритмов относится данный метод. Такой прием носить название «Шаблонный метод» («Template Method»). «Шаблонный метод» - это паттерн поведения классов, который определяет основу алгоритма и позволяет подклассам переопределить некоторые шаги алгоритма, не изменяя его структуру в целом. Шаблонный метод - один из фундаментальных приемов повторного использования кода, который особенно важен в библиотеке классов, так как предоставляет возможность вынести общее поведение в библиотечные классы [1].

2 Архитектурное решение для алгоритмов

Рассмотрим создание наследуемой базы алгоритмов на примере проектирования алгоритмов, генерирующих временные ряды с заданным параметром Херста. Данное решение продемонстрировано на рисунке 3.

Для того чтобы проектировать в соответствии с интерфейсами, как показано на рисунке 3, был создан интерфейс *IAlgorithmGeneration*, который является родителем для всех алгоритмов генерации временных рядов с заданным параметров Херста. Этот интерфейс наследуется от интерфейса *IAlgorithm*, поэтому его могут агрегировать наследники абстрактного класса *MultyAlgorithms*. Кроме того, интерфейс *IAlgorithmGeneration* включает сигнатуру метода генерации *Execute*.

Метода *Execute* содержит два параметра:

– *H* – экспонента Херста, число вещественного типа;

– *T* – длина генерируемого ряда, целое число.

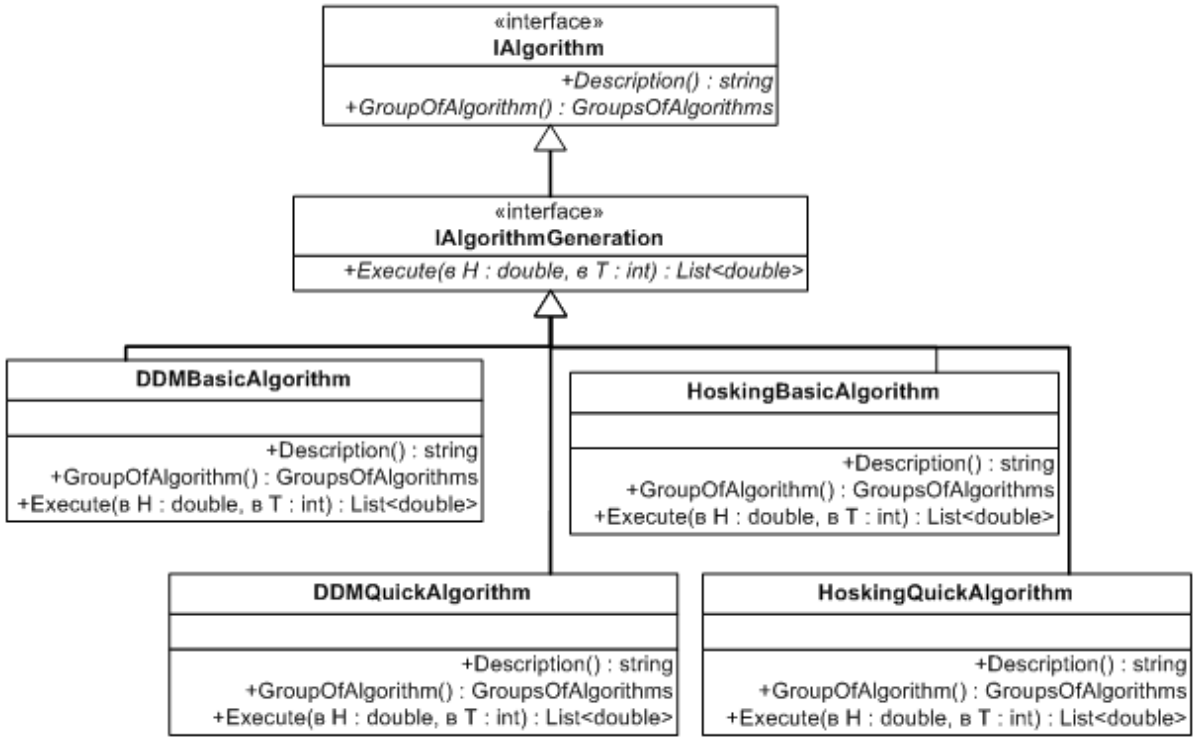

Рисунок 3. Проектирование алгоритмов, генерирующих временные ряды с заданным параметром Херста

Результатом выполнения вышеуказанного метода является список чисел вещественного типа, хранящий значения сгенерированного временного ряда длины *T*, корреляционна структура которого характеризуется параметром Херста - *H*.

Наследники интерфейса *IAlgorithmGeneration*, по своему реализуют ствойства *GroupOfAlgorithms*, *Description*, а так же метод *Execute*.

На рисунке 3 показана пара алгоритмов, которая генерирует временные ряды прямым методом определения (*DDMBasicAlgorithm* и *DDMQuickAlgorithm*) и пара алгоритмов, которая генерирует временные ряды методом Хоскинга (*HoskingBasicAlgorithm* и *HoskingQuickAlgorithm*).

Классы *DDMBasicAlgorithm* и *DDMQuickAlgorithm* одинаково реализуют свойство *GroupOfAlgorithms*. Это означает, что они принадлежат одной группе алгоримов. Однако при реализации свойства *Description*, указываются разные описания, характеризующие особенности

того или иного алгоритма. Например для класса *DDMBasicAlgorithm*, можно указать, что это «основной алгоритм», а для *DDMQuickAlgorithm* – «быстрый алгоритм».

Аналогично предыдущим рассуждениям, классы *HoskingBasicAlgorithm* и *HoskingQuickAlgorithm* одинаково реализуют свойство *GroupOfAlgorithms*, так как относятся к одной группе алгоритмов (выполняют генерацию временных рядов по методу Хоскинга).

Расмотренное архитектурное решения для алгоритмов позволяет:

– создавать алгоритмы генерации с другой сигнатурой методов;

– информировать пользователя об особенностях того или иного алгоритма;

– динамически подменять алгоритмы, которые относятся к одной группе.

3 Архитектурное решение для методов

Для использования преимуществ вышеуказанной модификации паттерна «Стратегия», нами была предложена архитектура классов, пример которой продемонстрирован на рисунке 4.

Как показано на рисунке 4, абстрактный класс *BaseGenerationMethod*, который отвечает за методы генерации временных рядов с заданным параметром Херста, наследуется от базового класса многоалгоритмических методов *MultyAlgorithms*, а так же от двух интерфейсов: *IGenerationMethod* и *ISeries*. *IGenerationMethod* включает сигнатуру метода генерации *ExecuteMethod*, который содержит такие же параметры, как и метод *Execute* интерфейса *IAlgorithmGeneration* (*H* и *T*). Интерфейс *ISeries* содержит сигнатуры методов для работы с рядами *Series*:

– GetSeriesByName – позволяет получить ряд по его имени;

– GetSeriesByInd - позволяет получить ряд по его индексу;

– GetSeriesesNames – возвращает список имен рядов;

– Count – возвращает количество рядов.

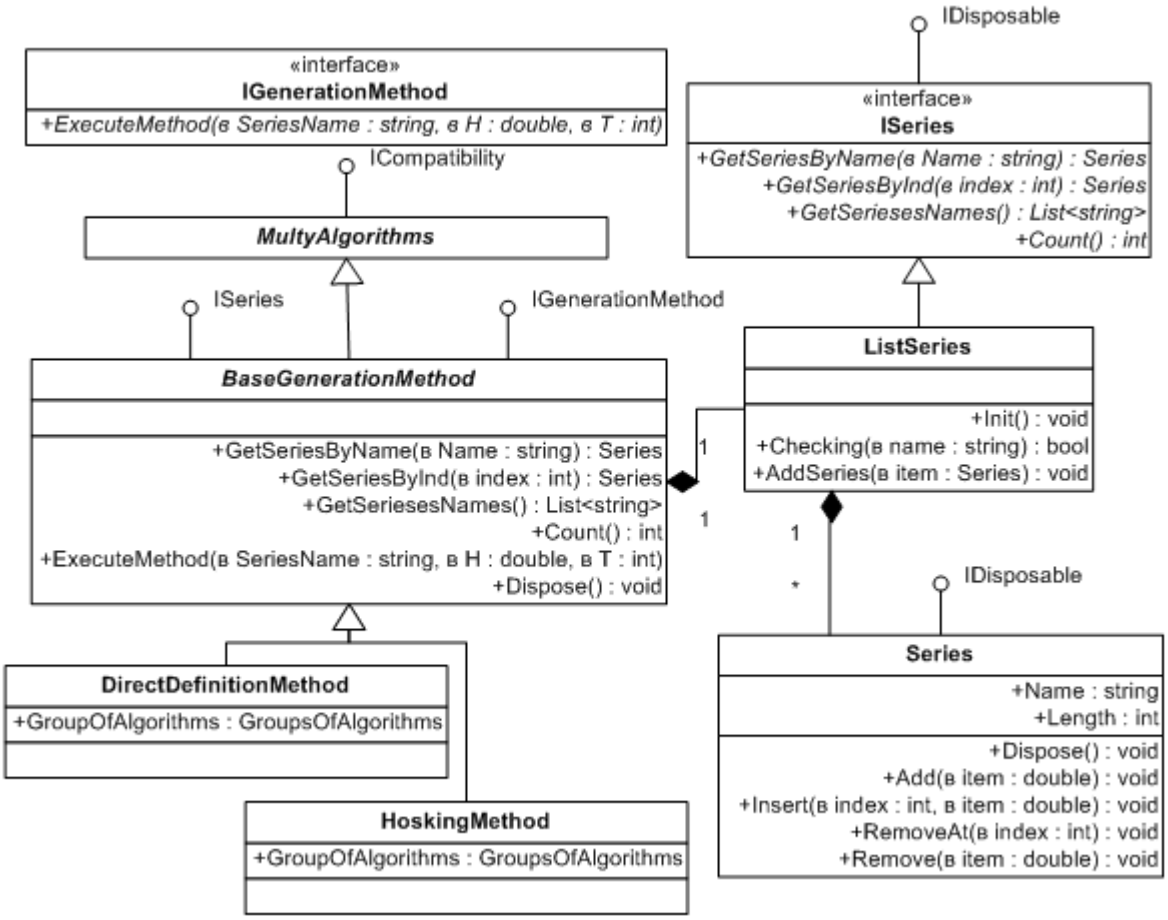

Рисунок 4. Проектирование алгоритмов, генерирующих временные ряды с заданным параметром Херста

Класс *Series* служит для внутреннего представления данных. Кроме методов для работы со списком вещественных чисел, класс *Series* содержит название ряда, что позволяет удобно идентифицировать его в системе.

Функционал для работы со списком рядов в данной архитектуре выделен в отдельный класс *ListSeries*, который реализует интерфейс *ISeries*, а так же содержит два дополнительных метода: Checking, который позволяет проверить наличие того или иного ряда и AddSeries, который добавляет ряд в список имеющихся. Внутри себя метод AddSeries

содержит проверку с использованием метода Checking, таким образом, запрещая добавлять ряды с именами, которые уже есть в наличии.

Класс *BaseGenerationMethod* агрегирует *ListSeries* и делегирует ему методы, реализующие интерфейс *ISeries*.

Реализация метода ExecuteMethod продемонстрирована на рисунке 5:

```
private ListSeries m_ListSeries = null;
public void ExecuteMethod(string SeriesName, double H, int T)
{
    if (m_ListSeries.Checking(SeriesName))
        return;

    Series _series = new Series(SeriesName);

    _series.Init(
        (Algorithm as AlgorithmsGeneration.IAlgorithmGeneration).Execute(H, T)
        );

    m_ListSeries.AddSeries(_series);
}
```

Рисунок 5. Пример реализации исполнения многоалгоритмического метода

Как видно из рисунка 5, метод ExecuteMethod передает выполнение непосредственного исполнения генерации методу Execute текущего алгоритма, который является наследником интерфейса *IAlgorithmGeneration*.

Рассмотренная композиция классов позволяет:

– создавать другие методы генерации и динамически их подменять;

– эффективно создавать и использовать различные структуры внутреннего представления данных

– быстро расширять набор методов генерации временных рядов.

Заключение

Хорошее проектирование – это возможность, которая есть у одних и, которой нет у других. Использование навыков объектно-

ориентированного проектирования позволяет развивать и совершенствовать систему.

В данной статье была представлена модификация паттерна проектирования «Стратегия», предназначенная для создания многоалгоритмических методов, которая:

– информирует клиентов о различных типах алгоритмов (стратегиях) без раскрытия особенностей их реализации;

– позволяет контролировать соответствие алгоритмов и методов,

Кроме этого, была подробно описана реализация вышеуказанной модификации на примере проектирования библиотеки анализа и моделирования временных рядов, которая использует механизм делегирования, а так же паттерн проектирования «Шаблонный метод».

Библиографический список

1. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб.: Питер, 2008. — 366 с.
2. Иан Грэхем Объектно-ориентированные методы. Принципы и практика. — 3-е изд. — М.: Издательский дом «Вильямс», 2004. — 880 с
3. Эндрю Троелсен. C# и платформа .NET. Издательство «Питер». Петербург 2002.
4. UML. Классика CS. Буч Г., Якобсон А., Рамбо Дж., Орлов С.А. 2-е изд. 2005 год. ISBN 5-469-00599-2.