УДК 004.051

Особенности реализации паттерна Observer средствами языка программирования C#

П. Е. Дадонов [1], О. А. Молокович [2]

[1] oZFdad@gmail.com, [2]o.molokovich@ya.ru

ФГБОУ ВО «Уфимский государственный авиационный технический университет» (УГАТУ)

Аннотация. В данной статье рассматриваются особенности реализации паттерна «Observer» средствами языка программирования C#. Приведены: «Классическая» реализация, реализация интерфейсов платформы .NET (IObserver, IObservable), использование, в качестве оповещения, событий средствами языка C# - event. Описываются основные детали этих реализаций. В качестве показателя эффективности реализации используется наиболее простой показатель - размер исходного кода. С точки зрения этого показателя наиболее эффективным оказалась реализация оповещения с помощью средств языка C# - event.

Ключевые слова: Паттерн; Шаблон проектирования; Наблюдатель; Observer; C#; .NET; Interface; delegate; event.

ВВЕДЕНИЕ

Понятие паттерна проектирования было введено в работе [1]. Существует множество определений данного термина, но наиболее удачное принадлежит Александеру К. В. [2]: «Любой паттерн описывает задачу, которая снова и снова возникает в нашей работе, а также принцип ее решения, причем таким образом, что это решение можно потом использовать миллион раз, ничего не изобретая заново». Это определение представляет интерес тем, что автор имел в виду архитектурные строительные паттерны, которые возникают при проектировании зданий и городов, но при этом данное определение верно и для паттернов объектно-ориентированного проектирования.

Преимущество использования паттернов при постановке той или иной задачи по проектированию программы или ее компонента заключается в том, что мы можем представить задачу в виде схемы, состоящей из классов, объектов и связей между ними, и применить один из существующих паттернов проектирования, тем самым не придумывая ничего нового, а применяя уже готовое, надежное решение проблемы. Такой подход повышает степень повторного использования готовых решений, но вместе с тем накладывает определенную ответственность при проектировании.

Необходимо отметить, что выбор языка напрямую влияет на выбор того или иного паттерна проектирования, так как трудоемкость реализации различных паттернов зависит от средств выбранного языка. Некоторые паттерны напрямую поддерживаются языками программирования. Так, в языке C# есть собственный инструмент «Event» для реализации оповещения подписчиков о конкретных изменений в «наблюдаемом» объекте, то есть фактически предоставляет один из вариантов обеспечения наблюдения за событиями. При этом в языке C# также реализованы интерфейсы IObserver, IObservable, что подразумевает их использование разработчиками.

В данной статье мы разберем достоинства и недостатки классической реализации паттерна «Observer», реализации встроенных в .NET интерфейсов IObserver, IObservable, а также использование инструмента языка C# «Event».

В качестве языка реализации выберем язык программирования C#, так как на данный момент это один из наиболее востребо-

ванных языков программирования как в Enterprise разработке, так и для отдельных веб-решений.

Паттерн Observer- поведенческий шаблон проектирования. Определяет зависимость «Один ко многим» между множеством наблюдателей и наблюдаемым объектом таким образом, что наблюдаемый объект оповещает наблюдателей о своих изменениях.

Паттерн Observer актуален на данный момент и, с учетом перспектив развития технологий, не потеряет своей актуальности в ближайшем будущем. Любое мобильное приложение, которое присылает push-уведомления использует ту или иную реализацию данного паттерна. Наиболее ярко, представить его работу мы можем на примере издательства журналов и его читателей. Читатели подписываются на определенный журнал у издателя, издатель хранит список подписчиков на конкретный журнал, при выходе нового журнала, извещает подписчиков, подписаных на этот, определенный журнал, о данном событие. Читатель идет на почту за новым журналом. При необходимости читатель может аннулировать подписку.

РЕАЛИЗАЦИЯ

Структура паттерна не зависит от выбранного языка программирования (C++, C#, Java), его UML диаграмма представлена на рис. 1.

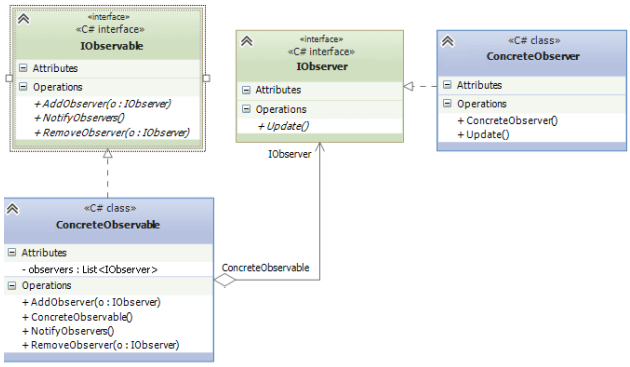

Рис. 1. Структура паттерна [3]

Формально реализация паттерна на языке C# выглядит так:

```
internal class Program
    {
        public static void Main(string[] args)
        {
            var observable = new Observerble();
            var observer = new Observer();
            observable.AddObserver(observer);
            observable.CallObserver();
            observable.RemoveObserver(observer);
        }
    }

    internal interface IObserver
    {
        void Update();
    }

    internal interface IObservable
    {
        void AddObserver(IObserver observer);
        void RemoveObserver(IObserver observer);
        void CallObserver();
    }

    internal class Observer : IObserver
    {
        public void Update()
        {
            Console.WriteLine("Получили новое уведомление");
        }
    }

    internal class Observerble : IObservable
    {
        private readonly List<IObserver> _observersList = new List<IObserver>();

        public void AddObserver(IObserver observer)
        {
            if (_observersList.Contains(observer)) return;

            _observersList.Add(observer);
            Console.WriteLine("Подписались на рассылку");
        }

        public void RemoveObserver(IObserver observer)
        {
            _observersList.Remove(observer);
            Console.WriteLine("Завершили наблюдение");
        }

        public void CallObserver()
        {
            foreach (var observer in _observersList)
            {
                observer.Update();
            }
        }
    }
```

Здесь интерфейс IObservable предлагает реализацию наблюдаемого объекта, определяя 3 базовых метода:

AddObserver - добавление наблюдателя

RemoveObserver - удаления наблюдателя

CallObserver - оповещение наблюдателей об изменении в наблюдаемом объекте

Класс Observable - конкретная реализация интерфейса IObservable

Интерфейс IObserver - предлагающий реализацию объекта наблюдателя, имеющего возможность получить оповещение от наблюдаемого объекта с помощью метода Update

Класс Observer - конкретная реализация интерфейса IObserver, определяет коллекцию объектов наблюдателей

Отметим, что некоторые языки программирования предлагают собственную реализацию данного паттерна. В том числе, в языке C# есть готовые интерфейсы IObservable <T>, подразумевает реализацию метода Subscribe, и IObserver <T>, который соответственно требует реализации методов OnNext, OnError и OnCompleted.

Реализация интерфейсов IObserver<T> и IObservable<T> выглядят следующим образом.

```
internal class Program
    {
        public static void Main(string[] args)
        {
            var observer = new Observer();
            var observable = new Observable();
            using (observable.Subscribe(observer))
            {
                observable.Update();
            }
        }
    }

    internal class Observer : IObserver<Observable>
    {
        public void OnNext(Observable value)
        {
            Console.WriteLine("Получили новое уведомление");
        }

        public void OnError(Exception error)
        {
            Console.WriteLine($"Получили ошибку {error}");
        }

        public void OnCompleted()
        {
            Console.WriteLine("Завершили наблюдение");
        }
    }

    internal class Observable : IObservable<Observable>
    {
        private readonly List<IObserver<Observable>> _observers = new List<IObserver<Observable>>();

        public IDisposable Subscribe(IObserver<Observable> observer)
        {
            _observers.Add(observer);
            Console.WriteLine("Подписались на рассылку");
            return new Subscription(_observers, observer);
        }

        public void Update()
        {
            foreach (var observer in _observers)
            {
                observer.OnNext(this);
            }
        }
    }

    internal class Subscription : IDisposable
    {
        private readonly List<IObserver<Observable>> _observers;
        private readonly IObserver<Observable> _observer;

        public Subscription(List<IObserver<Observable>> list, IObserver<Observable> observer)
        {
            _observers = list;
            _observer = observer;
        }

        public void Dispose()
        {
            _observers.Remove(_observer);
            _observer.OnCompleted();
        }
    }
```

Класс Observable реализует интерфейс IObservable<T>, метод Subscribe осуществляет подписку наблюдателя на наблюдаемый объект и возвращает объект класса Subscription.

Класс Observer реализует интерфейс IObserver<T>

-метод OnNext как правило используется наблюдаемым объектом для предоставления сведений об изменения состояния или передачи данных

-метод OnError - используется наблюдаемым объектом для оповещения наблюдателя о возникающих исключениях или ошибках

-метод OnCompleted - оповещает наблюдателя о завершении подписки

Класс Subscription реализует интерфейс IDisposable и фактически возвращает объект, с помощью которого может быть осуществлена отписка.

Аналогом паттерна Observer, может являться использование такого средства языка C# как Event (Далее событие). События используются для сигнализации системе о том, что произошло определенное действие

```
internal class Program
    {
        public static void Main(string[] args)
        {
            var observable = new Observable();
            var observer = new Observer();

            observer.SubscribeTo(observable);
            observable.Change();
            observer.UnsubscribeFrom(observable);
        }
    }

    internal class Observer
    {
        public void SubscribeTo(Observable observable)
        {
            observable.OnChange += ObservableOnChange;
            Console.WriteLine("Подписались на рассылку");
        }

        public void UnsubscribeFrom(Observable observable)
        {
            observable.OnChange -= ObservableOnChange;
            Console.WriteLine("Завершили наблюдение");
        }

        private void ObservableOnChange(Observable value)
        {
            Console.WriteLine("Получили новое уведомление");
        }
    }

    internal class Observable
    {
        public delegate void ChangeEventHandler(Observable value);

        public event ChangeEventHandler OnChange;

        public void Change()
        {
            OnChange?.Invoke(this);
        }
    }
```

В классе Observer реализованы методы:

SubscribeTo - Осуществление подписки

UnSubscribeFrom - Осуществление отписки

ObservableOnChange - Реакция на действие наблюдаемого объекта

В классе Observable реализован единственный метод Change, который вызывает событие OnChange.

Вначале мы определяем делегат ChangeEventHandler, который принимает на вход один параметр типа Observable, после с помощью ключевого слова event мы определяем событие с именем OnChange и типом ChangeEventHandler.

В качестве обработчика события выступает метод ObservableOnChange класса Observer, добавление обработчика выполняется с помощью условной конструкции «+=», отписка с помощью конструкции «-=»

ЗАКЛЮЧЕНИЕ

«Классическая» реализация паттерна «Обсервер» на C# занимает 63 строки программного кода. Реализация интерфейсов IObserver и IObservable занимает 74 строки. Оповещение с помощью событий занимает 49 строк.

Таким образом самая компактная реализация основана на использовании встроенного механизма событий. Основной недостаток такого подхода заключается в том, что он не учитывает появление ошибок и исключений, обработку которых, необхо-

димо реализовывать отдельно. Так же, нужно понимать, что использование делегата подразумевает подписку метода наблюдателя на события объекта, а не подписку самого объекта наблюдателя.

Использование интерфейсов .NET самое объемное, при этом, при подписке «наблюдателя» на «наблюдаемый» объект, метод подписки обязан вернуть объект, с помощью которого можно осуществить отписку «наблюдателя». Для отписки «наблюдателя» необходимо использовать условную конструкцию «Using» или же хранить объект отписки. Так же, реализация интерфейса IObserver<>, позваляет обрабатывать ошибки и исключения объекта «наблюдателя».

В «классической» реализации паттерна наблюдаемый объект содержит в себе список объектов наблюдателей. Обработка ошибок и исключений требует отдельной реализации.

Как показывает практика, в Enterprise разработке, в качестве метода оповещения подписчиков о конкретных изменений в «наблюдаемом» объекте, используют именно метод событий (event). Выбор обуславливается компактностью кода, а так же возможностью подписываться на события без реализации дополнительных классов-подписчиков

СПИСОК ЛИТЕРАТУРЫ

1. Сайт о программировании. [Электронный ресурс] URL: https://metanit.com/sharp/patterns/3.2.php (Дата обращения 02.02.2020). [Programming site (2020, Feb. 2). Available: https://metanit.com/sharp/patterns/3.2.php]

2. **Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.** «Приёмы объектно-ориентированного проектирования. Паттерны проектирования». СПб.: Питер, 2015 - 368 с. [E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, (in Russian). SPb.: Piter, 2015]

3. **Александер К.В., Исикаева С.,Силверстайн М**. “Язык шаблонов. Города. Здания. Строительство”. Москва: Издательство студии Артемия Лебедева, 2014 - 1096 с.[C Alexander, S. Ishikawa, M. Silverstein, A Pattern Language: Towns, Buildings, Construction, (in Russian). Moscow: Publisher Studio Artemy Lebedev, 2014]

ОБ АВТОРАХ

ДАДОНОВ Павел Евгеньевич, бакалавр 2-го курса ФИРТ.

МОЛОКОВИЧ Ольга Александровна, магистрант 2-го курса ФИРТ.

METADATA

Title: Implementation features of Observer pattern by tools of C# programming language

Authors: P. E. Dadonov [1], O. A. Molokovich [2]

Affiliation:

Ufa State Aviation Technical University (UGATU), Russia.

Email: [1] ozfdad@gmail.com, [2] o.molokovich@ya.ru,

Language: Russian.

Source: Molodezhnyj Vestnik UGATU (scientific journal of Ufa State Aviation Technical University), no. 1 (22), pp. 56-60, 2020**.** ISSN 2225-9309 (Print).

Abstract: This article discusses the implementation features of the “Observer” pattern using the C # programming language. A few implementations are given. Implementation efficiency is evaluated. As an indicator of implementation efficiency is used the size of the source code. From this indicator, the most effective was the implementation of alerts using C# language tools - event.

Key words: Pattern; observer; observable; C#; .NET; interface; delegate; event.

About authors:

DADONOV, Pavel Yevgenyevich, bachelor student 2 year, Ufa state aviation technical University

MOLOKOVICH, Olga Alexandrovna, master student 2 year, Ufa state aviation technical University