Исследование реактивного программирования как набора технологий и как завершенной парадигмы

В.А. Горбушин, Ю.А. Андриенко

Аннотация **- Целью данной статьи является исследование реактивного программирования (РП) с точки зрения реализующих его технологий, а также с точки зрения пунктов Манифеста Реактивных Систем и взгляда на РП, как на отдельную окончательно оформившуюся парадигму программирования. Констатируется, что успехи конкретных технологий РП подводят к необходимости его окончательного оформления в виде парадигмы путем внедрения в существующие языки особых синтаксических конструкций. Приведен прототип таких конструкций, работоспособность которых обеспечена транспилятором для языка Java.**

Ключевые слова **– парадигма, реактивное программирование, исследование технологий, транспилятор, многопоточность, асинхронность.**

I. ВВЕДЕНИЕ

В разработке систем обычно используют подход, когда поток приложения сначала делает запрос к ресурсу, ждёт ответ, и только после этого продолжает работу, при этом во время ожидания поток простаивает. Обычно для увеличения производительности системы добавляют больше потоков, но определить их оптимальное количество сложно из-за непредсказуемой нагрузки.

В этом контексте реактивное программирование предлагает более эффективный подход к использованию ресурсов системы, управлению масштабированием и повышению отзывчивости.

II. ИСТОРИЯ РАЗВИТИЯ ПАРАДИГМЫ РЕАКТИВНОГО ПРОГРАММИРОВАНИЯ

Раньше разработку приложений часто ассоциировали с тем, что со временем эти приложения смогут работать на большем количестве устройств благодаря как оптимизации самого кода, так и повышению производительности устройств за счет увеличения частоты их процессоров. Однако современные тенденции в развитии технологий указывают на изменение этого направления.

Исследования, проведенные Карлом Руппом, показывают изменения в эволюции процессоров. Он, использовал SpecINT, стандарт разработанный организацией SPEC для измерения производительности процессора в целочисленных операциях, и анализировал количество логических ядер, которое равно произведению физических ядер на количество потоков обработки, доступных для каждого из них. [1]

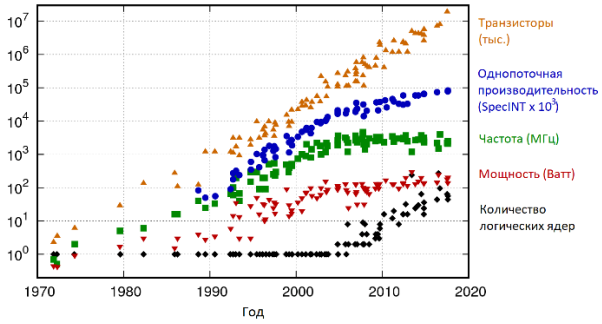

Рисунок 1. Данные о тенденциях развития процессов за последние 42 года.

Из анализа следует, что рост частоты процессоров прекратился из-за достижения физических пределов современных техпроцессов. Несмотря на это, производительность процессоров продолжает увеличиваться за счет внедрения многоядерных процессоров, что видно по росту количества логических ядер, обозначенных на графике как «Количество логических ядер».

Ядро процессора представляет собой независимый вычислительный блок, способный выполнять задачи последовательно. Если на ядро возлагается выполнение нескольких задач, оно будет переключаться между ними, что может замедлить обработку. В отличие от ядра, поток является виртуальной единицей, которая позволяет эффективнее использовать ресурсы ядра, обрабатывая два потока задач параллельно. Такая система делает вычисления более быстрыми, поскольку операционная система воспринимает каждый поток как отдельный вычислительный блок.

Для эффективного использования ядер и потоков разработчики используют такие подходы, как многопоточность и асинхронность, позволяющие оптимизировать процессы и улучшить производительность приложений. [2]

III. МНОГОПОТОЧНОСТЬ И АСИНХРОННОСТЬ

Многопоточность описывает возможность системы или приложения поддерживать работу нескольких потоков одновременно, что позволяет более эффективно использовать ресурсы компьютера за счет "параллельного" выполнения задач. Однако, многопоточность сопровождается трудностями в управлении потоками, такими как необходимость их синхронизации, риск возникновения состояния гонки и усложнение процесса отладки. Улучшение производительности системы благодаря многопоточности не всегда гарантировано и зависит от качества решения. [3]

Асинхронность представляет собой подход в программировании, при котором результат выполнения операции возвращается не мгновенно, а через некоторое время, изменяя тем самым стандартный порядок выполнения операций. Асинхронное программирование позволяет освобождать ресурсы и проводить параллельные вычисления, тем самым повышая эффективность работы программ. [4]

Несмотря на широкое применение многопоточности и асинхронности в современных технологиях, эти подходы могут иметь свои недостатки, особенно когда возникает нехватка потоков для обработки данных или блокировка потоков в ожидании ресурсов. В таких случаях может оказаться более предпочтительной реактивная система, элементы которой не простаивают в ожидании, а активно реагируют на поступающие события, обеспечивая, тем самым, более гибкое и эффективное использование ресурсов. [5]

IV. РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

Реактивность определяется как способность отвечать на стимулы или воздействия из внешней среды. В контексте реактивного программирования, эта концепция претерпевает расширение: она фокусируется на управлении потоками данных и автоматическом распространении изменений в этих потоках. Это предполагает, что система должна уметь эффективно обрабатывать как изменяемые, так и неизменные данные, а также гарантировать, что любые изменения в данных автоматически отражаются по всей модели исполнения. [6]

Парадигма реактивного программирования – это парадигма, основанная на работе с асинхронными потоками данных. Эти потоки, представляющие собой упорядоченные во времени последовательности событий, могут включать в себя всё: от переменных и пользовательского ввода до свойств, кэша и структур данных.

Reactive Streams является международным проектом, целью которого является создание унифицированного стандарта для асинхронной обработки данных. В рамках этой инициативы определены основные интерфейсы, методы и протоколы, необходимые для обработки данных в режиме реального времени без блокировок, благодаря механизму неблокирующего обратного давления. Этот стандарт позволяет на его основе строить разнообразные реализации на различных языках программирования. [7]

V. ТЕХНОЛОГИИ РЕАКТИВНОГО ПРОГРАММИРОВАНИЯ

ReactiveX представляет собой API для реализации реактивного программирования на различных платформах, включая Java, JavaScript, C#, Scala, Clojure, Swift и другие. Этот API уникален тем, что основывается на работе с последовательностями дискретных значений, которые генерируются со временем.

Ключевым элементом в обработке асинхронных потоков данных в ReactiveX являются реактивные операторы. Эти функции обрабатывают данные, поступающие от наблюдаемых (Observable) объектов, преобразуя их и возвращая новые наблюдаемые объекты. Они могут также создавать новые потоки данных в процессе обработки.

Observer в этой парадигме выступает как получатель данных, с реакцией на события посредством вызова специфических методов: onNext() для каждого элемента потока, onComplete() при завершении потока и onError() в случае возникновения ошибки.

В ReactiveX представлено множество типов источников данных (например, ConnectableObservable, Flowable, Single, Maybe) и операторов (как empty(), never(), map(), buffer(), debounce()), обеспечивающих гибкое управление потоками данных. [8]

В различных языках программирования существуют альтернативные реализации реактивного программирования, такие как RxJava и Spring Reactor с типом Flux для Java, а также Publisher, предложенный в спецификации Java 9, показывая многообразие подходов к асинхронной обработке данных в современном программировании [9].

ObservableComputations представляет собой библиотеку для C#, разработанную для выполнения вычислений над объектами, поддерживающими интерфейсы INotifyPropertyChanged и INotifyCollectionChanged, аналогично тому, как это происходит в LINQ. Эта библиотека сопоставима с такими инструментами, как Obtics, OLinq, NFM.Expressions, BindableLinq и ContinuousLinq. [10].

Akka Streams является частью экосистемы Akka.NET, C# и предлагает реализацию концепции Reactive Streams, упрощая работу с асинхронным потоковым программированием и уменьшая сложности, связанные с управлением акторами [11].

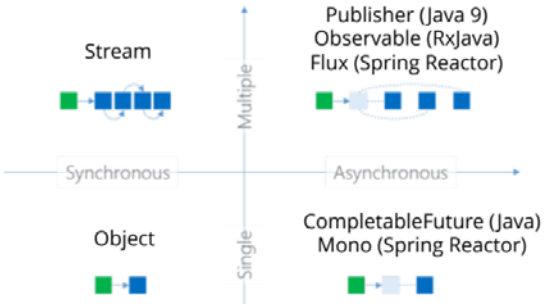

Рисунок 2. Типы возвращаемых значений, с учётом синхронности и количества потоков.

VI. FUNCTIONAL REACTIVE PROGRAMMING

FRP, или Functional Reactive Programming, представляет собой парадигму программирования, в основе которой лежит функциональный подход. В FRP акцентируется работа с функциями вместо переменных, что делает эту парадигму особенно подходящей для реактивного программирования. Примерами языков, использующих этот подход, являются Elm и Haskell с фреймворком Reflex.

Elm — это функциональный язык для создания веб-приложений, работающих по принципу The Elm Architecture (TEA), который предполагает разделение кода на модель (model), представление (view) и обновление (update), где каждый из этих компонентов выполняет свою роль в структуре приложения. [12]

Reflex, в отличие от Elm, предлагает большую гибкость в организации архитектуры веб-приложений на Haskell, предоставляя разработчикам абстракции Event, Behavior и Dynamic для эффективного управления состоянием. [13]

Среди других реализаций FRP стоит упомянуть Mrr и Bacon для JavaScript, а также reactive-banana для Haskell, каждая из которых предлагает свой подход к реактивному программированию.

VII. СИНТАКСИЧЕСКОЕ ОФОРМЛЕНИЕ ПАРАДИГМЫ

Несмотря на то, что понятие «парадигма программирования» носит несколько расплывчатый характер, интуитивное понимание этого термина есть у каждого программиста. Формальным признаком оформления новой парадигмы, на наш взгляд, может считаться появление в языке программирования некоторых существенно новых синтаксических элементов. Так, например, объектно-ориентированное программирование привнесло новые синтаксические элементы в виде класса и конструктора, функциональное программирование – стрелочные функции и лямбда-выражения, асинхронное программирование – синтаксис async/await. Каждая парадигма, за счет появления новых синтаксических элементов, способствовала как развитию нового стиля кодирования, так и появлению нового стиля мышления, упрощая и ускоряя процесс разработки.

С этой точки зрения, реактивное программирование пока не может считаться окончательно оформленной парадигмой, поскольку его реализация в конкретных библиотеках происходит с помощью синтаксических конструкций обычного ООП.

Важно также отметить, что в реактивном программировании, несмотря на то, что наибольшую пользу оно приносит при использовании в асинхронных и многопоточных приложениях, ключевым элементом является вовсе не асинхронность, а отзывчивость на изменения. Реактивность не всегда подразумевает асинхронность. Например, в Project Reactor предусмотрены инструменты для работы с синхронными источниками, позволяющие использовать одну логику для различных потоков данных. С другой стороны, однопоточные системы (например, Node.js с его реактивными элементами) активно применяются для множества задач. [14]

Суть реактивной парадигмы заключается не столько в новых технических решениях, сколько в новом стиле мышления, позволяющем решать на новом уровне определённые задачи. Это может даже снизить производительность решений, но значительно ускорить разработку продуктов.

Чтобы окончательно определиться в качестве самостоятельной парадигмы, реактивное программирование, по нашему мнению, должно определить свой уникальный синтаксис. До того, как синтаксис будет принят в стандартах тех или иных языков, он может быть реализован путем создания транспиляторов.

Транспилятор – это инструмент, который трансформирует код из одного языка программирования в код другого. В примере на рисунке 3, исходный код написан на некотором «новом» Java-подобном языке, а целевой язык – Java.

Символ «$=» означает инициализацию «реактивной переменной», которая автоматически изменяет своё значение при изменении других реактивных переменных, использованных при ее задании.

```
public class Example {
    public static void test() {
        var x $= 1;
        var y $= 2;
        var z $= 0;
        z = x + y;
        System.out.println(z); // вывод: 3
        x = 3;
        System.out.println(z); // вывод: 5
        var w $= 0;
        w = z + y;
        System.out.println(w); // вывод: 7
        y = 3;
        System.out.println(w); // вывод: 9

        var nonReactive1 = 1;
        var nonReactive2 = 2;
        var nonReactive3 = nonReactive1 + nonReactive2;
        System.out.println(nonReactive3); // вывод: 3
        nonReactive1 = 3;
        System.out.println(nonReactive3); // вывод: 3
    }
```

Table

x	y	z	w
1	--	--	--
1	2	--	--
1	2	0	--
1	2	3.0	--
3	2	5.0	--
3	2	5.0	0
3	2	5.0	7.0
3	3	6.0	9.0

Рисунок 3. Демонстрация синтаксиса

Такой синтаксис позволяет сократить код и повысить его читаемость, но, главное, абстрагироваться от непосредственной реализации взаимодействия «издатель-подписчик». Инкапсуляция реализации в новом простом и коротком синтаксисе может открыть путь к формированию нового стиля кодирования и мышления.

IX. ЗАКЛЮЧЕНИЕ

Данная статья фокусируется на создании и программной реализации проектов, основанных на реактивной парадигме программирования. В тексте статьи проведен анализ разнообразных технологий, выделены их преимущества и ограничения. При правильном использовании каждая из этих технологий способна значительно увеличить производительность.

В свою очередь, реактивная парадигма уже достаточно развита для того, чтобы быть закрепленной в языках программирования с помощью уникальных для нее синтаксических конструкций, пример которых реализован с помощью описанного в данной работе транспилятора.

БЛАГОДАРНОСТИ

Авторы выражают благодарность Высшей инжиниринговой школе НИЯУ МИФИ за помощь в возможности опубликовать результаты выполненной работы.

БИБЛИОГРАФИЯ

[1] Morgillo I. Grokking ReactiveX / I. Morgillo, F. Chignoli, S. Sekulic. – Manning Publications, 2007. – 30 с.
[2] Long J. Reactive Spring. John Long, 2020.
[3] Машнин Т. Многопоточное программирование в Java. Ridero, 2021.
[4] Samson C. F. Asynchronous Programming in Rust: Learn asynchronous programming by building working examples of futures, green threads, and runtimes. Packt Publishing, 2024.
[5] Дейли Б. Разработка веб-приложений с помощью Node.js, MongoDB и Angular: исчерпывающее руководство по использованию стека MEAN / Б. Дейли, Б. Дейли, К. Дейли. – Санкт-Петербург: «Диалектика-Вильямс», 2020. – 656 с.
[6] Tate B. Seven More Languages in Seven Weeks: Languages That Are Shaping the Future / B. Tate, I. Dees, F. Daoud, J. Moffitt. – Pragmatic Bookshelf, 2014. – 291 p.
[7] Кун, Р. Реактивные шаблоны проектирования / Роланд Кун. – СПб.: Питер, 2018. — 405 с.
[8] Шилдт, Г. Java. Полное руководство / Герберт Шилдт. – 10 изд. – М.: «Диалектика», 2018. – 1488 с.
[9] Julien Ponge, Vert.x in Action: Asynchronous and Reactive Java, 2020
[10] Lamis Chebbi, Reactive Patterns with RxJS for Angular: A practical guide to managing your Angular application's data reactively and efficiently using RxJS 7, 2022
[11] Peter Royal, Building Modern Business Applications: Reactive Cloud Architecture for Java, Spring, and PostgreSQL, 2022
[12] Энтони Уильямс. Параллельное программирование на C++ в действии. Практика разработки многопоточных программ, 2022
[13] Гетц Брайан, Java Concurrency на практике, 2020
[14] Eric Freeman, Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software 2nd Edition, 2021

Статья получена 5 апреля 2024.
Горбушин Владимир Алексеевич, Национальный Исследовательский Ядерный Университет МИФИ, магистрант, gorbushin.volodja@mail.ru
Андриенко Юрий Анатольевич, Национальный Исследовательский Ядерный Университет МИФИ, доцент, yand@outlook.com

Research of reactive programming as a set of technologies and as a completed paradigm

V.A. Gorbushin, Yu.A. Andrienko

Abstract. The purpose of this paper is to study reactive programming (RP) from the point of view of technologies that implement it, as well as from the point of view of the Reactive Systems Manifesto, and of the RP as a separate programming paradigm. We conclude that successes of specific RP technologies lead to the necessity to finalize RP as a paradigm by introducing special syntactic constructions into existing languages. We propose a prototype of such construction, whose operability is provided by a transpiler for the Java language.

Keywords – paradigm, reactive programming, technology research, transpiler, multithreading, asynchrony.

REFERENCES

[1] Morgillo I. Grokking ReactiveX / I. Morgillo, F. Chignoli, S. Sekulic. – Manning Publications, 2007. – 30 s.
[2] Long J. Reactive Spring. John Long, 2020.
[3] Mashnin T. Mnogopotochnoe programmirovanie v Java. Ridero, 2021.
[4] Samson C. F. Asynchronous Programming in Rust: Learn asynchronous programming by building working examples of futures, green threads, and runtimes. Packt Publishing, 2024.
[5] Dejli B. Razrabotka veb-prilozhenij s pomoshh'ju Node.js, MongoDB i Angular: ischerpyvajushhee rukovodstvo po ispol'zovaniju steka MEAN / B. Dejli, B. Dejli, K. Dejli. – Sankt-Peterburg: «Dialektika-Vil'jams», 2020. – 656 s.
[6] Tate B. Seven More Languages in Seven Weeks: Languages That Are Shaping the Future / B. Tate, I. Dees, F. Daoud, J. Moffitt. – Pragmatic Bookshelf, 2014. – 291 p.
[7] Kun, R. Reaktivnye shablony proektirovanija / Roland Kun. – SPb.: Piter, 2018. — 405 s.
[8] Shildt, G. Java. Polnoe rukovodstvo / Gerbert Shildt. – 10 izd. – M.: «Dialektika», 2018. – 1488 s.
[9] Julien Ponge, Vert.x in Action: Asynchronous and Reactive Java, 2020
[10] Lamis Chebbi, Reactive Patterns with RxJS for Angular: A practical guide to managing your Angular application's data reactively and efficiently using RxJS 7, 2022
[11] Peter Royal, Building Modern Business Applications: Reactive Cloud Architecture for Java, Spring, and PostgreSQL, 2022
[12] Jentoni Uil'jams. Parallel'noe programmirovanie na C++ v dejstvii. Praktika razrabotki mnogopotochnyh programm, 2022
[13] Getc Brajan, Java Concurrency na praktike, 2020
[14] Eric Freeman, Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software 2nd Edition, 2021