DOI https://doi.org/10.15407/usim.2019.05.062
УДК 004.9

O.S. BULGAKOVA, PhD (Ehg.) Sciences, associate professor,
V.O. Sukhomlynsky Mykolaiv National University,
Nikolska str., 24, Mykolaiv, 54000, Ukraine,
sashabulgakova2@gmail.com

V.V. ZOSIMOV, PhD (Ehg.) Sciences, Head of the Department ,
V.O. Sukhomlynsky Mykolaiv National University,
Nikolska str., 24, Mykolaiv, 54000, Ukraine,,
zosimovvv@gmail.com

REACTIVE PROGRAMMING PARADIGM FOR DEVELOPMENT USER INTERFACES

The article discusses the relevance of using the paradigm for user interfaces development. The essence of the reactive programming paradigm, its features and disadvantages are described. The technology of reactive programming ReactiveX is considered. Features of the reactive systems implementation are described as a result of an in-depth study of the main development tools and language structures. The reactive paradigm possibilities of reducing labor costs for building valid models, minimizing errors, and efficient and quick solution of the tasks are shown on the example of the application implementation.

Keywords: *reactive programming, GUI development, NoSQL database, MongoDB, ReactiveX technology, MVC model.*

Introduction

In last years, reactive programming in general, and technology in particular, is becoming increasingly popular in developer's area. Some developers are already using all benefits of this approach, while others are only beginning to learn this technology. In the world of programming there are two fundamentally different ways of organizing large systems: according to objects and states that exist in the system, and according to streams of data that pass through it.

The reactive programming paradigm assumes the easing expression of data streams, and expansion of changes through these streams. In imperative programming, the assignment operation denotes the end of the result, whereas in the reactive paradigm a value will be recalculated when new inputs are received. Stream concept is the basis of the program. The stream of values passes a series of transformations that are necessary for solving a certain task. Operations with streams are making system expandable and asynchronous, and the correct response to emerging exception provide a fault-tolerant ability [1–2].

One of the main differences between reactive programming and the functional is that it operates not continuously changing values, but discrete values that are "released" over a long period of time. This paradigm allows to create asynchronous and event-oriented programs that use observable sequences [1–3].

Reactive paradigm and its features

Reactive programming is a programming paradigm that focuses on data streams and the expansion of changes. This means that there should be an opportunity to easily express static and dynamic data streams, as well as the fact that underlying proces-

sing model should automatically distribute changes due to a data stream.

Modern table processors are an example of reactive programming. The table columns may contain the string values or a formula such as "$= B_1 + C_1$", the value of which will be calculated based on values in corresponding cells. When the value of one of dependent cells is changed, the cell value will be automatically updated.

Another example is Hardware Description Language (HDL), such as Verilog [4]. Reactive programming allows to simulate changes in the form of its distribution inside the model.

Reactive programming was offered as a way for easy creation of user interfaces, animations, or modeling of systems that change in time.

For example, in MVC architecture using reactive programming, it can be implemented an automatic display of changes from Model to View and conversely from View to Model.

Object-oriented reative programming (OORP) — is a combination of object-oriented approach with the reactive paradigm [2]. Probably the most natural way to do this is that instead of methods and fields in objects there are reactions that automatically changed the values, and other reactions depend on changes in these values.

Functional programming is the most natural basis for an implementation of reactive architecture, which well combined with parallelism.

Main advantages of this paradigm are [5]:

1. Responsiveness. The application should give the result in a minimum amount of time. This also applies to principle of fast fail — that is, when something goes wrong, it need to return the user an error message, then to keep wait.

2. Scalability — a way to provide responsiveness under load.

3. Fault-tolerance. In a case that something goes wrong in the system, it is necessary to foresee the processing of errors and methods of working capacity restoration.

4. Architecture based on reactive streams.

The reactive paradigm exploits an idea of application's streams. Stream is a sequence, what consist of continuous events sorted by time. It can contain three types of messages: values (some type data), errors and shutdown signals.

Programmers are given the opportunity to create streams of something, not just events or movement of the cursor. Streams are light and used everywhere: variables, custom input, properties, cache, data structures, etc. For example, a news feed on a social network can be present as a stream of data along with a number of user interface events. That is, you can view a stream and react to each event in it.

Moreover, the paradigm implies a creation and filtration of each of these streams. Streams can be used as input parameters of each other stream. Even a multiple stream can be used as an input argument for another stream. It is possible to combine multiple streams, filter one stream, then get another that contains only an actual data, combine data from one stream with data of another to get another one.

Reactive paradigm tools

The reactive paradigm is not a basis of any programming language, in contrast to the object-oriented paradigm, so special tools are needed to work with it.

In most cases, program calls a method and receives a result on an output. But some processes are built in a different way. For example, a method can be process over a long period of time. Or, worse, the method not only makes calculation for a long time, but also irregularly returns some results during execution. Current technologies are needed exactly for this case.

Further, the basic tools and language constructs designed for the implementation of reactive systems will be considered [5].

Green streams

Green streams are simulation of threads. Virtual machine takes care of switching between different green streams, and the machine itself works as one operating system's stream. It gives some benefits. Operating system streams are a bit expensive. In addition, switching between system streams is much slower than between green streams. All this means that in some situations, green streams are much more profitable than common ones. The system can support a much larger number of green streams than operating system threads.

Green streams are asynchronous and nonblocking, but they do not support message transfer. They do not support horizontal scaling on different network nodes. Also, they do not provide any mechanisms to ensure fault tolerance, so developers are forced to independently handle the processing of potential failures.

Communicating Sequential Processes

Communicating Sequential Processes (CSP) is a mathematical encapsulation several processes or threads in a single process that interact with the transmission of a message. The uniqueness of CSP lies in the fact that two processes or threads should not know about each other, they are perfectly separated in sending and receiving messages, but still related with a transmitted value. Instead of collecting messages in a queue, the rendezvous principle applies: for exchanging messages, one side must be ready to send it, and the other one — receiving. Therefore, messaging is always synchronized.

The CSP model is asynchronous and nonblocking. It supports messaging in the Rendezvous style and is can be enlarge for multi-core systems, but it does not support horizontal scaling. Also, it does not provide mechanisms for fault-tolerance.

Future and Promise descriptors

Future — a reference to a value or error code that may become readable (read-only) at some point in time.

Promise — a corresponding descriptor with possibility of a single record that provides access to a value. Future and Promise object analogues are implemented in most popular languages: Java, C #, Ruby, Python, etc.

The function that returns a result asynchronously creates a Promise object, initializes asynchronous processing, sets a final callback function, which at one time fills a Promise data, and returns a component that called a Future object associated with the Promise. Then a calling component can attach to the Future some code that will be executed when values in this object appear.

In all Future implementations, there is a mechanism for converting a code block into a new Future object so that it executes the code passed to it in another thread and puts a final result inside the Promise object associated with it. Thus, Future provides an easy way to make the code asynchronous. Future objects return either the result of successful calculations or an error.

The Future and Promise objects are asynchronous and nonblocking, but they do not support messaging. However, they are capable of scaling vertically, but they can not be scaled horizontally. They also provide fault-tolerance mechanisms at the level of individual Future objects.

Reactive extension

If there is a process that runs for a long time and occasionally returns results, then Reactive Extensions will handle following results every time they arrive. Code becomes easier when reactive extensions used, and also gets more rich functionality.

Reactive Extensions (Rx) — is a library, more precisely, a set of libraries that allow to work with events and asynchronous calls in compositional style. The library came from the .NET world. Then it was moved to popular platforms: Java, Ruby, JS, Python, etc. This library combines two stream control templates: an observer and an iterator. Both are related to the processing of potentially unknown elements or events.

The essence of the Rx library is to write a cyclic design that responds to events occurring. In its semantics, it is similar to stream processing, when incoming data is continuously processed by iterations.

The Rx library represents tools for processing data streams asynchronously. Its current implementation is scaled vertically, but not horizontally. It does not provide mechanisms for delegating fault handling, but it can break a failed streaming container.

ReactiveX technology

ReactiveX framework is a reactive programming tool that works with all popular object-oriented programming languages. Authors call it a multiplatform API for asynchronous development [6].

One of the main differences between the ReactiveX library and functional reactive programming is that it operates not continuously changing, but discrete values that are «released» over a long period of time.

The technology is based on the “Observer” pattern. The “Observer” pattern exists the same time

as the object-oriented programming languages. An object whose state may change is called the publisher. All other members interested in these changes are subscribers.

Subscribers are registered with the publisher to receive messages, clearly indicating their ID. The publisher occasionally generates messages that are sent to them by the list of registered subscribers.

Technology operates with such concepts as: Observer, Observable, Subject, Fig. 1. The Observable model is a data source and allows processing of asynchronous events streams in a similar way to what is used for arrays. And all this is instead of callback, which means the code becomes more readable and less inclined to errors.

The publisher generating the message is set here using the Observable class. A publisher can have multiple subscribers, and they need to use the Subscriber class to implement them. Standard behavior for Observable is to release one or more messages to subscribers, and then complete their work. Messages can be both variables and objects, Fig. 2.

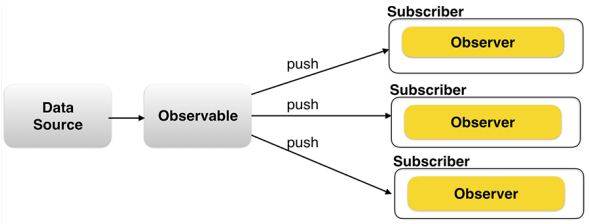

Fig. 1. ReactiveX simple data processing scheme

There are tasks for which need to connect Observer and Observable to receive event reports and report them to subscribers. There is a Subject that has, besides the standard one, several more implementations [6]:

- ReplaySubject has the ability to cache sent to him data, and when a new subscriber arrives, firstly give all this sequence, and then work in the normal mode.
- BehaviorSubject saves the last value, by analogy with ReplaySubject, giving it an emerging subscriber. When it is created, it gets the default

Fig. 2. Currently available operators

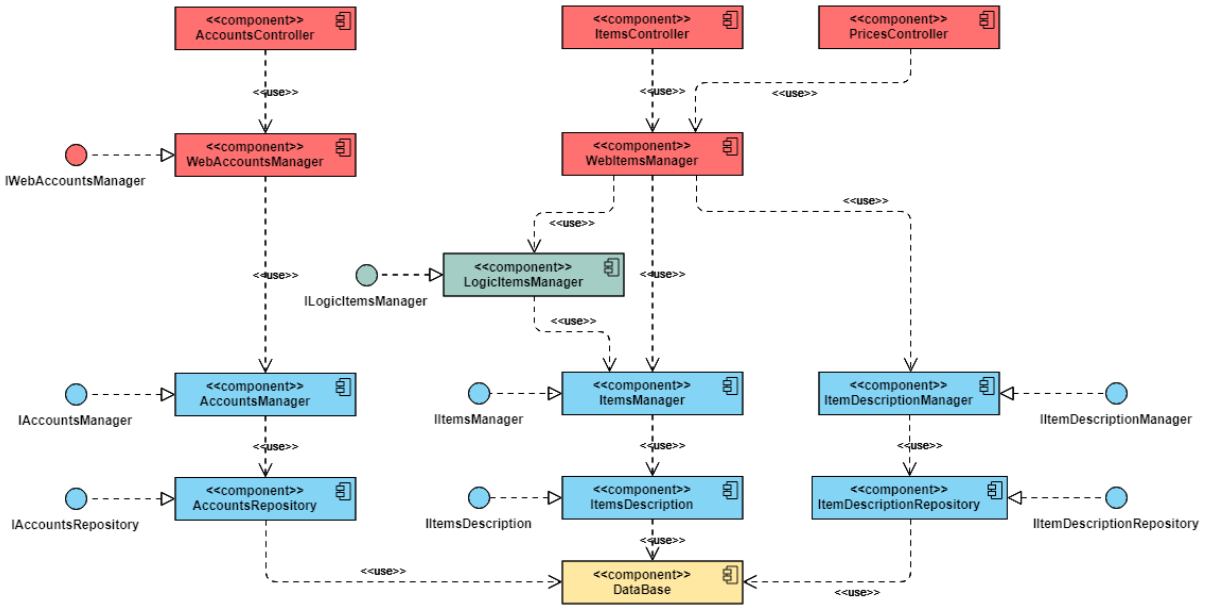

Fig. 3. Application server part components diagram

Fig. 4. The MVVM pattern Scheme

value that each new subscriber will receive if the last value has not yet been received.

- AsyncSubject also stores the last value, but does not give data until the entire sequence is complete.

Observable and Observer are just the beginning of ReactiveX technology. They do not carry all the power, which are operators that allow to transform, combine, manipulate sequences of elements that give Observable.

Using reactive technologies for GUI creation

As an example, a software application (GUI — graphic user interface) was developed, on the basis of which could be visually explored the principles of reactive programming technology.

Server solution

The server part of the solution was built using ASP. NET Core. Query processing now uses the new HTTP conveyor, which is based on Katana components and the OWIN specification. And its modularity makes it easy to add own components, Fig. 3.

As a database, was used the NoSQL database, namely MongoDB. MongoDB implements a new databases building approach without tables, schemas, SQL queries, external keys, and many other things that are inherent in object-relational databases [7]. Unlike relational databases, MongoDB offers a document-oriented data model, which makes MongoDB work faster, has better scalability, makes it easier to use. The functionality of MongoDB allows to place multiple databases on multiple physical servers, and these databases will be able to easily exchange data and maintain integrity. For the creation of the automatic documentation of API used a tool called Swagger — is a framework for the RESTful API specification. It gives an opportunity not only to interactively view the specification, but also to send queries to the Swagger UI.

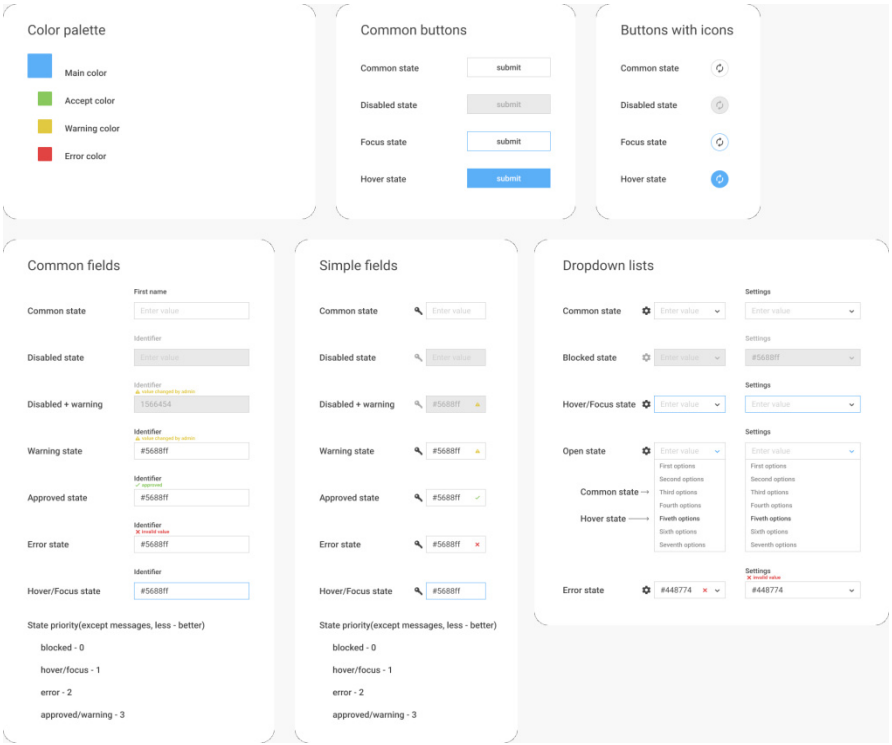

Fig. 5. The light style design example

Client solution

A server project is a significant part of the work, but for interacting with it the client it is needed an application with a graphical interface, so it was built using reactive technologies. Modern user does not want to work with a slow or unstable application, so the use of this technology is a priority on the client side.

As the basic technology for this work, it was chosen an Angular framework from Google, in this case, the interaction with the user has improved, because this framework works directly in a browser, so it does not need to be downloaded separately, but the complexity of deployment and increased load on the physical server prompted to choose another technology. So the Microsoft's WPF framework was chosen for the current works.

Traditional Windows applications has the operating system definitions, such as User32.dll (standard graphics library) and GDI + (cross-platform library of user interface) for drawing controls and graphics, then WPF applications are based on DirectX technology. This is the key feature of playing graphics in WPF: much of the work on displaying graphics, from simple buttons to complex 3D models, falls on a video card graphics processor, which also allows to use hardware graphics acceleration.

One of the important features is the usage of the XML declarative XAML markup language: it is possible to create a rich graphical interface using interface declaration or a code in a managed language, such as C #.

WPF works using MVVM architectural pattern, Fig. 4, which led to formation of the solution assemblies.

The decision is made up of five assemblies, namely:

- RedSharp.Minecraft.Common — shared resources collection.
- RedSharp.Minecraft.Model — collection that contains tools for interacting with the server. In the MVVM pattern, the form of the model is not strictly defined in the solution, so in this case, the

model performs functions of communication with the server: the build queries, and receive answers.

- RedSharp.Minecraft.ViewModel — collection designed to hold and prepare data from a model for use in the user interface.
- RedSharp.Minecraft.View — contains a built-in user interface and ways to interact with the user.
- RedSharp.Minecraft.Styles — designed to build user interface styles.

Below is an example of possible blocks for interface design using reactive solutions, Fig. 5. The interface changes immediately after selecting one of the styles.

The reactive solutions described in this paper work at the Model level and the ViewModel level as commands. They are responsible for the asynchronous processing of data that came from the server request and processing user-generated commands, as an example of an authorization attempt on the server. The processing in this way avoids a large amount of work on the valid models construction and avoiding errors, efficiently and fast.

Using ReactiveX is not so hard because it looks very similar to the processing of collections using .Net LINQ technology, namely the Flow Ideas Chains. Asynchronous processing in this style allows to build the user interface directly while response processing, which makes the program more responsive. ReactiveX technology is well integrated into the MVVM pattern used in the work.

Conclusion

The paper considers the reactive programming paradigm and its differences from imperative and functional programming. The basic technology used to create the reactive programming based applications is investigated. The example shows the advantages of using the reactive paradigm, which allows to reduce work input for building valid models, minimizing errors, allowing to solve the problem efficiently and quickly. This project, developed using ReactiveX, ASP.NET and WPF technologies, can be used as a visual application of the considered technology.

Today, reactive programming tools allows to improve the creation and support of existing projects, especially those focused on the intensive user interaction with the system.

REFERENCES

1. *Demetrescu C., Finocchi I., Ribichini A.* “Reactive Imperative Programming with Dataflow Constraints”. Proceedings of the International Workshop ACM, 2011.

2. *Salvaneschi G., Mezini M.* “Towards Reactive Programming for Object-oriented Applications”. Transactions on Aspect-Oriented Software Development XI, 2014, pp. 227–261.

3. *Zosimov, V., Khrystodorov, O., Bulgakova, O.* “Dynamically changing user interfaces: software solutions based on automatically collected user information”. Programming and Computer Software, 2018, 44 (6), pp. 492–498.

4. *Harris D. Harris S.* “Hardware Description Languages”. Digital Design and Computer Architecture, 2013, 2, pp. 172–237.

5. *Bonér J., Klang V.* “Reactive programming vs. Reactive systems”, [online]. Available at: https://www.oreilly.com/radar/reactive-programming-vs-reactive-systems/ [Accessed 03 Sept. 2019].

6. *ReactiveX* — Introduction. ReactiveX.io., [online]. Available at: http://reactivex.io/ [Accessed 15 Apr. 2019].

7. The database for modern applications, [online]. Available at: https://www.mongodb.com/ [Accessed 25 Apr. 2019].

Received 17.11.2019

О.С. Булгакова, канд. техн. наук, доцент,
Миколаївський національний університет імені В.О. Сухомлинського,
вул. Нікольська, 24, Миколаїв, 54000, Україна,
sashabulgakova2@gmail.com

В.В. Зосімов, канд. техн. наук, завідувач кафедри,
Миколаївський національний університет імені В.О. Сухомлинського,
вул. Нікольська, 24, Миколаїв, 54000, Україна,
zosimovvv@gmail.com

ВИКОРИСТАННЯ ПАРАДИГМИ РЕАКТИВНОГО ПРОГРАМУВАННЯ ДЛЯ РОЗРОБКИ КЛІЄНТСЬКИХ ІНТЕРФЕЙСІВ

Вступ. Парадигма реактивного програмування передбачає легкість вираження потоків даних, а також поширення змін завдяки цим потокам. В імперативному програмуванні операція присвоювання означає кінцівку результату, тоді як в реактивному — значення буде перераховано при отриманні нових вхідних даних. У центр програми ставитися поняття потоку. Потік значень проходить в системі ряд трансформацій, які необхідні для вирішення певної задачі. Оперування потоками дозволяє системі бути розширюваною та асинхронною, а правильна реакція на виникаючі помилки — відмовостійкою.

Мета статті. Дослідити актуальність використання парадигми реактивного програмування, як основи для розробки клієнтських інтерфейсів.

Результати. В роботі розглянуто парадигму реактивного програмування, її відмінності від імперативного та функціонального програмування. Досліджена основна технологія, яка використовується сьогодні для створення додатків на основі реактивного програмування. На прикладі показано використання реактивної парадигми, яка дозволяє знизити трудовитрати на побудову валідних моделей, мінімізувати помилки, дозволяючи вирішувати поставлену задачу ефективно і швидко.

Висновки. На сьогоднішній день інструменти реактивного програмування дозволяють поліпшити створення і підтримку існуючих проектів, особливо, орієнтованих на інтенсивну взаємодію користувача з системою. Представлений в статті проект, розроблений за допомогою технології *ReactiveX*, *ASP.NET* і *WPF* може бути використаний як наочне застосування даної технології.

***Ключові слова**: реактивне програмування, розробка GUI, бази даних NoSQL, MongoDB, технологія ReactiveX, модель MVC.*

*А.С. Булгакова***,** канд. техн. наук, доцент кафедры информационных технологий,
Николаевский национальный университет имени В.А. Сухомлинского,
ул. Никольская, 24, Николаев 54000, Украина,
sashabulgakova2@gmail.com

В.В. Зосимов, канд. техн. наук, заведующий кафедрой информационных технологий,
Николаевский национальный университет имени В.А. Сухомлинского,
ул. Никольская, 24, Николаев 54000, Украина,
zosimovvv@gmail.com

ИСПОЛЬЗОВАНИЕ ПАРАДИГМЫ РЕАКТИВНОГО ПРОГРАММИРОВАНИЯ ДЛЯ РАЗРАБОТКИ КЛИЕНТСКИХ ИНТЕРФЕЙСОВ

Введение. Парадигма реактивного программирования предусматривает легкость выражения потоков данных, а также распространение изменений благодаря этим потокам. В императивном программировании операция присваивания означает конечность результата, тогда как в реактивном — значение будет пересчитано при получении новых входных данных. В центр программы ставится понятие потока. Поток значений проходит в системе ряд трансформаций, которые необходимы для решения определенной задачи. Оперирование потоками позволяет системе быть расширяемой и асинхронной, а правильная реакция на возникающие ошибки — отказоустойчивой.

Цель статьи. Исследовать актуальность использования парадигмы реактивного программирования, как основы для разработки клиентских интерфейсов.

Результаты и выводы. В работе рассмотрена парадигма реактивного программирования и ее отличия от императивного и функционального программирования. Исследована основная технология, используемая сегодня для создания приложений на основе реактивного программирования. На примере показано использование реактивной парадигмы, которая позволяет снизить трудозатраты на построение валидных моделей, минимизировать ошибки, позволяя решать поставленную задачу эффективно и быстро.

На сегодняшний день инструменты реактивного программирования позволяют улучшить создание и поддержку существующих проектов, в особенности, ориентированных на интенсивное взаимодействие пользователя с системой. Представленный в статье проект, разработанный с помощью технологий *ReactiveX*, *ASP.NET* и *WPF* может быть использован как наглядное применение рассматриваемой технологии.

***Ключевые слова**: реактивное программирование, разработка GUI, базы данных NoSQL, MongoDB, технология ReactiveX, модель MVC.*