INTERFACE TECNOLÓGICA

UM ESTUDO SOBRE PROGRAMAÇÃO REATIVA

A STUDY ON REACTIVE PROGRAMMING

Eder Ferreira de Matos – eder.matos@fatec.sp.gov.br
Faculdade de Tecnologia de Taquaritinga – Taquaritinga – São Paulo – Brasil

Jederson Donizete Zuchi – jederson.zuchi@fatec.sp.gov.br
Faculdade de Tecnologia de Taquaritinga – Taquaritinga – São Paulo – Brasil

DOI: 10.31510/infa.v18i2.1287
Data de submissão: 14/09/2021
Data do aceite: 03/11/2021
Data da publicação: 30/12/2021

RESUMO

Com o avanço das tecnologias nos últimos anos, a demanda por informações em tempo real vem crescendo cada vez mais. Em conjunto à necessidade de maior interação com o usuário, surgem diferentes formas de resolver determinados problemas, e uma dessas formas é utilizando a programação reativa. Diante disso, esse estudo tem como propósito realizar uma revisão bibliográfica junto ao conceito de programação reativa. Grandes empresas de tecnologias estão adotando a programação reativa em seu desenvolvimento, visto isso pode-se entender qual o motivo, e ter uma noção do que essas empresas querem para o futuro.

Palavras-chave: Programação Reativa. Arquitetura. Reatividade. Observer, Mensageria

ABSTRACT

With the advancement of technologies in recent years, the demand for real-time information is growing more and more. With this, different ways to solve certain problems arise, and one of them is using reactive programming. Therefore, this study aims to carry out a literature review on the concept of reactive programming. Large technology companies are adopting reactive programming in their development, with this we can understand why, and get a sense of what these companies want for the future.

Keywords: Reactive Programming. Architecture. Reactivity. Observer, Message Communication

1 INTRODUÇÃO

Décadas atrás, foi identificada a necessidade de evoluir os paradigmas de programação existentes, visto que a programação procedural exigia códigos mais extensos para realizar determinadas atividades. Com isso, a programação orientada a objetos ganhou espaço na área de desenvolvimento, resolvendo alguns problemas da programação procedural (SCHACH, 2010).

Hoje, esses sistemas evoluíram, assim como as pessoas e a forma de comunicação. Todos os dias, diversos eventos acontecem de forma simultânea. Alguns deles de forma independente, mas em outros casos por conta de uma concorrência, esses eventos podem sofrer com lentidão, isso tanto no dia a dia das pessoas como em sistemas (SAMOYLOV E NIELD, 2020).

Mudando a forma de comunicação, também mudou o tempo de espera nessas comunicações. Os sistemas atuais possuem uma vasta quantidade de eventos em tempo real, sendo muito mais interativos se comparados aos do passado. Essa interação se tornando cada vez mais utilizada, fez com que se tornasse difícil a utilização dos paradigmas de programação atuais. Isso porque não era mais possível prever nenhum tipo de ordem desses eventos acionados por usuários. A programação orientada a objetos já não supria mais a necessidade (BAINOMUGISHA, 2013).

Com essa dificuldade, começou-se a trabalhar com mensageria, fluxos assíncronos e não bloqueantes, mas não era uma tarefa fácil, por mais que as linguagens de programação evoluíssem (TURINI, 2017).

Portanto, esse estudo tem como objetivo trazer um paradigma de programação orientado a eventos, tornando o processo de interação com o usuário mais eficaz, rendendo vantagens também aos desenvolvedores que não necessitam mais se preocupar com a ordem de eventos, nem em aprender uma outra linguagem.

2 METODOLOGIA

A metodologia desse estudo se define pela revisão de livros e artigos a fim de entender os conceitos, processos e principais características, tanto teóricas como práticas, de desenvolvedores que utilizam programação reativa em seu dia a dia.

O que se espera dessa revisão bibliográfica é que possa servir como referência para compreender a viabilidade do uso da programação reativa, gerando grandes benefícios sem dificuldades para os desenvolvedores.

3 REVISÃO BIBLIOGRÁFICA

Nessa seção, serão detalhados os conceitos da programação reativa e de sistemas reativos. Entendendo desde padrões de projeto implementados, até métodos de tratamento dos eventos e *frameworks*.

3.1 Conceito de programação reativa

Programação reativa pode ser definida como um paradigma de programação orientada a fluxos de dados e propagação de mudanças de forma assíncrona. Isso quer dizer que, de acordo com algumas ações, outras serão executadas como forma de reação (TSVETINOV, 2015).

O conceito de programação reativa é um tema que gera muita discussão e divergências, pois diversos autores definem programação reativa de maneira diferente. Há quem chame de modelo de programação, modelo arquitetural ou paradigma de programação. O termo programação reativa foi citado pela primeira vez em um artigo escrito por Gérard Berry em 1989, o "*Real time programming: special purpose or general purpose languages*", ou "Programação em tempo real: Linguagens de propósito especial ou propósito geral".

A programação reativa pode ser vista como uma programação funcional de forma mais abstrata e voltada para sistemas que lidam com uma cadeia de dados assíncronos e com estado distribuído (ODERSKY, 2016).

Desenvolvedores web estão acostumados a utilizar programação reativa com Javascript, isso porque, ao implementar *event listeners,* naturalmente está sendo usado a programação reativa para reagir a determinados eventos, seja de mouse ou teclado no contexto web (RHYNES, 2016).

Uma ideia importante sobre a programação reativa é que dados são eventos, e eventos são dados. A composição desses eventos e dados permite transformações e filtros de maneira radicalmente eficaz para construir grandes aplicações com consumo de dados em tempo real (SAMOYLOV E NIELD, 2020).

3.2 O design pattern observer

A programação reativa tem como característica a implementação do padrão de projeto "*Observer*". Com o uso desse padrão de projeto, é criada uma entidade que faz emissão de eventos ao longo do tempo, então sempre que tal evento ocorrer essa entidade notificará os interessados (SOUZA, 2018).

O *observer* é um padrão de projeto comportamental, que permite a definição de um mecanismo de assinatura, permitindo notificar os interessados sobre alguma ação relacionada ao objeto que está sendo observado (GAMMA, *et.al, 2008*).

Um exemplo simples, que facilita o entendimento sobre esse padrão de projeto, é imaginar uma loja online, que após um pagamento de uma compra ser aprovado, deve realizar algumas ações independentes. Sem utilizar o *observer,* poderia ser feita uma validação sobre o estado do pagamento de tempos em tempos, para assim, quando identificado um estado diferente, realizar alguma ação. Mas essa aprovação pode demorar, o que torna tal validação muito custosa. A maneira que o *observer* resolve tal problema é simples: ao invés de perguntar algo, o sistema é notificado sobre algo. Dessa forma, o processamento de qualquer rotina será menor, pois só será executado quando realmente algo acontecer, ou seja, quando for necessário. Utilizando o *observer* é criado um *publisher,* uma classe que sabe quando um evento acontece, e sabe também quem são os interessados nesses eventos. Então, quando o evento ocorrer, essa classe irá notificar os interessados, que tem o nome de *subscribers.*

Um sistema reativo implementa a ideia do *observer,* visto que existem ações que são assíncronas, logo não ocorrem em sequência. Portanto, com o uso do observer, tem-se a capacidade de notificar um consumidor sobre algum evento, seja de sucesso ou de erro com mais eficácia (TSVETINOV, 2015).

3.3 Manifesto Reativo

O manifesto reativo é um documento escrito em setembro de 2014, que conta com algumas recomendações para ter sistemas reativos. Escrito por Jonar Bonér, Dave Farley, Roland Kuhn e Martin Thompson, esse documento traz o que os autores entendem por sistemas reativos e o que esses sistemas devem obedecer para serem flexíveis, desacoplados, escaláveis e altamente responsivos, dando aos usuários um feedback interativo (TSVETINOV, 2015).

De acordo com Bonér *et.al* (2014), sistemas reativos devem obedecer a quatro princípios: responsividade, resiliência, elasticidade e orientação a mensagens. Tais princípios serão descritos a seguir:

- **Responsivo:**

Um sistema responsivo responde em tempo hábil, de forma consistente, simplificando o tratamento de possíveis erros dando mais confiança ao seu usuário. Responder em um tempo hábil não necessariamente significa responder rápido, visto que rapidez pode ser relativa de acordo com o sistema.

- **Resiliente:**

Um sistema resiliente continua respondendo mesmo em caso de falhas. Esse ponto se aplica a qualquer sistema, visto que se o sistema não for resiliente ele ficará indisponível em caso de falhas. O ponto principal desse tema é que sempre é importante dar um feedback a quem está consumindo o sistema, mesmo que a resposta seja informando que não é possível realizar tal ação em determinado momento.

- **Elástico:**

Um sistema elástico permanece respondendo independente da carga, ou seja, tendo um pico de requisições de forma inesperada ou tendo uma quantidade muito pequena, o sistema deve permanecer respondendo a todas essas requisições.

- **Orientado a mensagens:**

Sistemas reativos devem ser baseados em transferência de mensagens assíncronas, tendo assim um baixo acoplamento e isolamento. Comunicação utilizando mensageria permite gerenciar a carga, controle de fluxo, elasticidade e tratamento de erros de forma mais simples.

3.4 Síncrono X Assíncrono X Não-Bloqueante

É comum se deparar com esses três termos ao ler sobre programação reativa, isso porque um dos pontos positivos dela é ser eficiente, ou seja, conseguir realizar alguma ação ou algum processamento, de forma que consuma menos recurso e se possível em menos tempo. E para isso, é importante entender como as ações que o código irá realizar serão executadas, seja essa ação de forma síncrona, assíncrona ou não bloqueante (SOUZA, 2018).

- **Síncrono:**

Uma das formas que um site tem para obter dados é por meio de requisições HTTP, que é um protocolo de comunicação. Supondo que seja necessário obter os dados de um usuário, é feita uma requisição e espera-se que a resposta sejam os dados desse usuário. Mas, enquanto a resposta dessa requisição não for obtida e tratada, a conexão relacionada à requisição permanece travada ou bloqueada. Portanto, algo que é síncrono permanece parado enquanto a ação não é finalizada. Se uma thread executar alguma ação síncrona, ela fica travada até essa ação ser finalizada, portanto não pode ser utilizada para outros processos (RHYNES, 2016).

- **Assíncrono:**

Essa mesma requisição citada acima, pode ser consumida de maneira diferente, simplesmente definindo um *callback*, ou seja, uma função que deve ser executada após a requisição ser completada. Com isso, o código pode realizar outros processamentos e mesmo assim conseguirá tratar a resposta da requisição (RHYNES, 2016).

- **Não bloqueante:**

Um processamento não bloqueante significa que algum tipo de processamento com interação com I/O (atividade que não ocorre em cpu) não força uma espera até o final desse processamento. Isso significa que um mesmo código pode ser executado multiprocessadores de memória compartilhada, e uniprocessadores, se tornando muito importante diante de paralelismo e multiprocessadores (CHAKRABORTY, 2017).

Um exemplo é uma chamada para banco de dados, onde deve-se buscar algumas informações podendo demandar tempo. Ao utilizar um processamento bloqueante, outros interessados em obter dados devem esperar a finalização de uma execução anterior. Ao tornar não bloqueante, é como se fosse feito um agendamento de execução, sendo notificado ao obter os dados necessários, e não sendo necessário esperar execuções anteriores, já que todos serão notificados ao final de cada uma, não é necessário uma espera.

3.5 Tratamento de eventos

Uma das características de um sistema reativo é o uso da programação funcional, sendo diferente de uma programação imperativa, onde o código é executado linha a linha (TSVETINOV, 2015).

- **Controle de estado:**

Um dos conceitos principais da programação funcional é a imutabilidade. Significa que o dado que trafega pelo sistema não pode ser alterado. Porém, normalmente é preciso fazer algum tipo de manipulação nesse dado. Para isso, existem operadores que criam novos objetos imutáveis mas com a alteração que se precisava, e assim é propagado um novo estado para um próximo consumidor (TSVETINOV, 2015).

- **Ciclo de vida:**

Existem diversos frameworks com suporte a programação reativa, e para isso também existem algumas padronizações para que todas essas formas de se trabalhar com programação reativa fiquem mais padronizadas. Uma prática que todas essas implementações seguem é que existem ao menos três eventos que podem ocorrer durante um processamento. Esses eventos têm o nome de "*onNext*", "*onError*" e "*onComplete*" (SAMOYLOV E NIELD, 2020).

É possível fazer uma ligação com um "*try*", "*catch*" e "*finally*", onde algo irá acontecer com o dado, gerando um erro ou não. A ação nesse caso é o "*onNext*", quando todas as ações terminaram de executar, o evento "*onComplete*" deve ser executado, indicando finalização, mas caso algum erro ocorra, então o evento "*onError*" deve ser executado.

- **Transformação de um estado:**

Esses frameworks que suportam programação reativa possuem meios de transformação de dados. Muito utilizado em lista, existem os métodos "*map*", "*flatMap*", "*reduce*", "*filter*" que trabalham sobre os dados (URMA, FUSCO E MYCROFT, 2018).

Seguindo os métodos já existentes, esses frameworks oferecem métodos para tratar esses dados. Sendo esses métodos:

- Operadores condicionais
- Operadores de transformação
- Operadores booleanos
- Operadores de coleta de erros

- Operadores de redução.

Cada tipo de operador é responsável por tratar um evento e criar um novo evento para ser passado para o próximo consumidor (SAMOYLOV E NIELD, 2020).

3.6 Backpressure

A partir da característica de sistemas reativos serem orientados a mensagens, pode-se ter alguns problemas no controle de fluxos de dados. Como sistemas reativos têm seus produtores e seus consumidores, um dos problemas que podem ocorrer é quando um produtor envia um número maior de mensagens que um consumidor consegue processar. Isso pode ocorrer quando os produtores atuam em diferentes threads, o que significa diferentes velocidades. Para isso, sistemas reativos têm o conceito de contrapressão, ou *backpressure.* O conceito de *backpressure* é essencialmente uma maneira para que o consumidor indique para o produtor que ele não consegue lidar com a velocidade ou volume dos eventos emitidos (PHELPS, 2019).

Com o *backpressure* têm-se o poder de implementar algum algoritmo que trará um comportamento quando tal problema acontecer, já que eventos serão gerados mesmo que não consumidos. Os *frameworks* facilitam a implementação e controle dessas ações junto ao *backpressure,* logo os desenvolvedores conseguem fazer o tratamento necessário sem grandes dificuldades.

3.7 Frameworks reativos

Existem muitos frameworks no mercado que trabalham com programação reativa. Linguagens como Java, Javascript, Dotnet, tem bibliotecas que auxiliam no desenvolvimento. E isso é incrível para quem desenvolve os sistemas, pois essas pessoas não precisam se preocupar em saber como o processador conseguirá receber diversas requisições na mesma thread ou coisas tipo. As duas principais bibliotecas com suporte a programação reativa serão descritas a seguir:

- **ReactiveX:**

De acordo com sua documentação, ReactiveX ou Rx é um conjunto de bibliotecas que fornecem um conjunto de interfaces e métodos que ajudam os desenvolvedores a escrever

códigos limpos e simples, disponível em 18 linguagens, como Java, Javascript, C#, Kotlin (REACTIVEX, 2021)

- **Project Reactor:**

Segundo sua documentação, project reactor é a principal biblioteca reativa para o ecossistema do Java com Spring, apresentando o Spring Webflux como forma de programar utilizando a programação reativa, de maneira simples (PROJECT REACTOR, 2021).

4 CONCLUSÃO

A programação reativa tem se tornado cada vez mais relevante para determinados projetos. Grandes empresas têm adotado esse paradigma pensando em aproveitar o máximo de seu hardware, desde que os conceitos façam sentido para os softwares desenvolvidos.

Tornar sistemas reativos retrata a necessidade de adquirir agilidade sobre mudanças para atender de forma mais rápida às demandas. Os quatro pilares da programação reativa, tem semelhanças com outros conceitos, como PWA (*Progressive Web Applications*), sendo que ambos buscam tornar os softwares melhores para seus usuários e gestores. Tendo que ser responsivos, deixando o usuário ciente do que está acontecendo, elásticos o suficiente para aguentar grandes cargas, resilientes para ter uma total transparência sobre eventuais problemas, e dirigido por mensagens a fim de notificar quem for necessário.

Embora tenha que se pensar de forma diferente do convencional, desenvolvedores tendem a ter outra visão sobre o desenvolvimento de software ao utilizar programação reativa, contribuindo para o crescimento profissional, agregando um grande valor para o produto.

Portanto, pode-se concluir que a programação reativa tende ser cada vez mais usada em sistemas, por trazer grandes benefícios para essas aplicações, e por tratar muito bem a comunicação assíncrona e eventos.

REFERÊNCIAS

BAINOMUGISHA, Engineer. CARRETON, Andoni. CUSTEM, Tom. MOSTINCKX, Stijn. **A survey on Reactive Programming**. 2013

BONÉR, Jonas. (2014) FARLEY, Dave. KUHN, Roland. THOMPSON, Martin. **The Reactive Manifest.** Disponível em: <https://bit.ly/2WHLOCQ> . Acesso em: 02 set. 2021.

Chakraborty, Rivu. **Reactive Programming in Kotlin: Design and Build Non-blocking, Asynchronous Kotlin Applications with RXKotlin, Reactor-Kotlin, Android, and Spring. Reino Unido: Packt Publishing.** 2017

GAMMA, E. HELM, R. JOHNSON, R. VLISSIDES, J. **Padrões de Projeto – Soluções reutilizáveis de software orientado a objetos.** 2008

JAVA 9 REACTIVE STREAMS. Disponível em: <https://bit.ly/3not6eW>. Acesso em: 10 set. 2021.

ODERSKY, Martin, **Functional Program Design in Scala. 2016**

OBSERVER**.** Disponível em: <https://bit.ly/3DRC4Hf> . Acesso em: 07 set. 2021.

PHELPS, Jay. **Backpressure explained — the resisted flow of data through software**. 2019. Disponível em: <https://bit.ly/3lfcEL5>. Acesso em 11 set 2021.

PROJECT REACTOR**.** Disponível em: <https://bit.ly/3yVRNkS> . Acesso em: 07 set. 2021.

REACTIVEX. Disponível em: <https://bit.ly/3zU5e66> . Acesso em: 07 set. 2021.

RHYNES, John. **Reactive Programming Basics: Starting Reactive in an easy way.** 2016.

SAMOYLOV, Nick; NIELD, Thomas. **Learn RxJava, Second Edition.** 2020.

SCHACH, Stephen. **Engenharia de Software: Os paradigmas clássico & orientado a objetos**. 2010

SOUZA, Evandro. **ReactiveX – Transformando sua linguagem favorita**. 2018. Disponível em <https://bit.ly/2XdwSMT>. Acesso em 11 set 2021

TSVETINOV, Nickolay. **Learn Reactive Programming with Java 8**. 2015, Disponível em: <https://bit.ly/3C4foBT>. Acesso em 11 set 2021

TURINI, Rodrigo. **Java 9, Interativo, Reativo e Modularizado**. Casa do Código, 2017

URMA, Raoul-Gabriel. FUSCO, Mario. MYCROFT, Alan. **Modern Java in action. Lambdas, streams, functional and reactive programming**. 2018