https://doi.org/10.3359/oz1812029 1.04: STROKOVNI ČLANEK

članek

REAKTIVNO PROGRAMIRANJE

REACTIVE PROGRAMMING

Janko Žigart

Institut informacijskih znanosti, Maribor

Kontaktni naslov: janko.zigart@izum.si

Izvleček

Zaradi velikega števila naprav, povezanih v splet, in mnogih končnih uporabnikov, ki želijo nenehno razpoložljivost in hitro odzivnost aplikacij, so potrebni novi pristopi pri programiranju. V članku predstavljamo koncept reaktivnega programiranja, njegove temeljne lastnosti ter reaktivne rešitve in mikrostoritve, ki temeljijo predvsem na asinhronem pristopu. Članek poskuša odgovoriti na vprašanje *Zakaj uporabiti reaktivni pristop?* ter opisuje tudi rešitve in orodja, ki bodo že v osnovi pomagali pristopiti k reaktivnemu razvoju.

Ključne besede

reaktivno programiranje, reaktivne rešitve, mikrostoritve, podatkovni tokovi

Abstract

Due to a large number of web devices and many end users, who want constant availability and quick responsiveness of applications, new approaches to computer programming are required. The concept of reactive programming, its basic characteristics, reactive solutions and microservices based on asynchronous operation are presented. The answer to the question *Why use the reactive approach?* is searched for. Solutions and tools that will help take an approach to reactive development are described.

Keywords

reactive programming, reactive systems, microservices, data streams

UVOD

Obstaja veliko (tudi slabih ali modnih) razlag in definicij izraza reaktivno programiranje (angl. *reactive programming*). Po Wikipedii, ki podaja morda preveč splošno in teoretično razlago, je reaktivno programiranje deklarativna programska paradigma, ki se ukvarja s podatkovnimi tokovi in širjenjem sprememb (Wikipedia, 2018). The Reactive Manifesto (Reaktivni manifest) zveni kot dokument, ki ga pokažemo vodji projekta ali poslovnežem v podjetju. Microsoftova Rx-terminologija je po večini zelo kompleksna in težko razumljiva, kar povzroča zmedo. Izraz **reaktivno** (angl. *reactive*) ne označuje ničesar posebej drugačnega, kar že ne bi bilo značilno za tipičen priljubljen programski jezik (npr. asinhroni dogodki). Zato bi lahko izraz **reaktivno programiranje** v računalništvu še najbolje opredelili kot **"asinhroni programski pristop, ki v jedru uporablja podatkovne tokove (angl. *data streams*)"** (Staltz, 2014). Tokovi so lahko kar koli – tako podatkovni vhodi kot podatkovne strukture, lastnosti, trgovine, skladišča ipd.

To pomeni, da ne gre za novo arhitekturo, ampak za nov oziroma drugačen pristop k programiranju ali drugače: za miselni preskok, ki bolje izkorišča sodobne programske arhitekture. To bi lahko opisali s primerom seštevka dveh števil, ki seveda vplivata na rezultat v času izvedbe programskega stavka. Pri tem se pri klasičnem programiranju ob kasnejših spremembah seštevancev vsota ne spremeni več, pri reaktivnem programiranju pa se ob vsaki spremembi katere koli spremenljivke (seštevanca) spremeni tudi vsota (Wikipedia, 2018). Podoben primer je Excelova preglednica, kjer so v celicah formule in se ob spremembi vrednosti v eni celici spremeni celotna vsebina. Ta primer lahko prikažemo kot graf, ki prikazuje te spremembe po času. Seveda lahko to opravimo tudi s klasičnim programskim pristopom.

UPORABA REAKTIVNEGA PRISTOPA

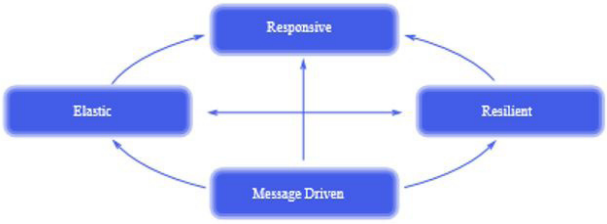
Danes ima dostop do spleta že prek 3,5 milijarde uporabnikov in ti uporabljajo različne naprave (računalnike, telefone, tablice …), ki podpirajo spletne rešitve. Pri tem uporabniki pričakujejo, da bodo sistemi in rešitve vsaj delujoči, če že ne hitro odzivni. Nekako se napoveduje, da bo do leta 2020 v splet povezanih prek 20 milijard naprav. In vsi uporabniki pričakujejo, da bodo storitve 100-odstotno

razpoložljive. Tu pa seveda nastopi velik izziv za razvijalce in ponudnike storitev, ki morajo imeti tako ekonomski kot poslovni pozitivni rezultat. Ob tem se zastavlja vprašanje, kako zagotoviti rešitve za zadovoljitev zahtev in potreb uporabnikov ter za podporo različnih naprav in arhitekturnih rešitev, ki so danes v oblaku, pri tem pa jih kljub nenehnemu povpraševanju po storitvah kot dodatnih rešitvah zaradi skalabilnosti ne bo treba dograjevati in popravljati. Kot odgovor oziroma rešitev se ponuja reaktivni pristop. V preteklosti je bil odgovor na asinhrone storitve bolj negativen kot pozitiven in šele leta 2009 smo dobili arhitekturo, ki je to omogočala (WebSockets, Servlets 3.0) (Krajnc, et al., 2016). S tem so se možnosti asinhronega pristopa povečale in nekatere rešitve so bile tudi že izvedene. Ker pristop k reaktivnemu programiranju omogoča takšne tehnološke zahteve, je s tem upravičen.

ZNAČILNOSTI REAKTIVNEGA PROGRAMIRANJA

Koncept reaktivnega programiranja opisuje The Reactive Manifesto, ki je bil prvotno objavljen leta 2013 (The Reactive Manifesto (v2.0)) (Bonér, et al., 2014). Po tem manifestu so rešitve tega tipa (torej reaktivne rešitve) tiste, v katerih se kažejo naslednje štiri glavne lastnosti:

- odzivnost (angl. *responsive*),
- odpornost (angl. *resilient*),
- elastičnost (angl. *elastic*) in
- sporočilna usmerjenost (angl. *mesagge driven*).



Slika 1: Povezanost glavnih lastnosti reaktivnih rešitev (Vir: Bonér, et al., 2014)

Odzivnost pomeni, da se rešitve na uporabniške zahteve odzivajo dovolj hitro in v vsakem stanju sistema. Rešitve se po navadi odzivajo le, če je sistem v dobrem stanju. Da bo rešitev odzivna, sta nujno potrebni visoka skalabilnost in odpornost. Ponazorimo s primerom: aplikacija postane neodzivna, če podatkovna baza (strežnik) odpove ali je v zagonu in vse povezave do baze postanejo neaktivne. Če rešitev ni odzivna, uporabniki nimajo zaupanja v konsistentno delovanje aplikacije, s tem pa izgubljamo končne uporabnike ter posledično njihove namige in ideje.

Odpornost zagotavlja, da so storitve odzivne tudi v primeru odpovedi. Rešitve so načeloma sestavljene iz različnih komponent in referenc ter zunanjih storitev. Rešitev se mora ne glede na to, na kateri komponenti so se težave pojavile, pravilno odzvati. To dosegamo z replikacijo, izolacijo in delegacijo komponent.

Elastičnost pomeni, da se rešitve odzivajo tudi, ko je sistem pod velikimi obremenitvami. Reaktivni sistem reagira tako, da poveča ali zmanjša število virov, ki so potrebni za zadovoljevanje uporabniških zahtev. Skalabilnost povečamo s povečevanjem procesorjev (CPU) in pomnilnih enot (RAM), pri čemer gre za vertikalno skaliranje na vozliščih (računalniki, strežniki ipd.), in z dodajanjem oziroma odvzemanjem novih računalnikov ter strežnikov, kar je poznano kot horizontalno skaliranje (Krajnc, et al., 2016). Sistem je v takšnem primeru arhitekturno zelo težko postaviti, saj to zahteva znanja tako o programiranju kot o omrežjih ter tudi znanja o podatkovnih bazah, replikacijah baz in virtualizaciji strežnikov, in to v klasični in/ali oblačni izvedbi.

Sporočilna usmerjenost je temelj reaktivnega programiranja. Rešitve uporabljajo asinhrono pošiljanje sporočil. Asinhroni načini komunikacije so prisotni že v dosedanjih rešitvah, vendar le kot parcialni deli rešitve v določeni aplikaciji (npr. pošiljanje sporočil iz vrste, ko je določeno opravilo v delovnem krogu doseglo zaključek). V reaktivnem primeru rešitve pa to, da je vgrajena sporočilna usmerjenost v jedru rešitve, pomeni, da je zasnova rešitve bolj kompeksna in se že od začetka zasnove aplikacije tega zavedamo.

REAKTIVNE REŠITVE IN MIKROSTORITVE

Po definiciji je sistem množica elementov in povezav med njimi. Če si rešitev zamislimo kot informacijsko rešitev, potem elemente določajo storitve, povezave pa so klici med njimi oziroma komunikacija med storitvami, ki jih ponujamo uporabnikom. Pri sodobnih rešitvah govorimo o servisno orientiranih rešitvah, pri reaktivnih rešitvah pa o mikrostoritvah ali mikroservisih. Že sama beseda pove, da gre za manjše sklope rešitev, katerih značilnosti so majhnost, enostavnost in opravljanje ene funkcije. S tem se nekako približamo definiciji in značilnostim reaktivnega programiranja, kar pomeni, da morajo mikrostoritve zagotavljati izolacijo, enojno odgovornost, lastništvo podatkov in asinhronost (Krajnc, et al., 2016). Mikrostoritve med seboj po navadi v zadnjem času uporabljajo komunikacijski protokol **HTTP+REST+JSON**, ki pa sam po sebi ni asinhron (Krajnc, et al., 2016).

Izolacija pomeni, da morajo biti mikrostoritve neodvisne od drugih, s tem so šibko sklopljene. Izolacija v tem primeru pomeni, da predvsem lažje obravnavamo odpoved sistema in/ali nadzor, ki se nanaša le na eno komponento.

Enojna odgovornost pomeni, da storitev opravlja samo eno stvar, torej mora po principu za to storitev obstajati samo en razlog. Če obstaja več razlogov, ima storitev več odgovornosti (Bonér, et al., 2014).

Lastništvo podatkov je v domeni mikrostoritev, ki so komponente s stanjem in morajo same poskrbeti za svoja stanja podatkov in shranjevanja v podatkovne baze.

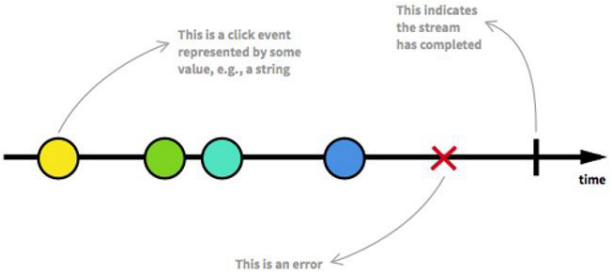
Asinhronost je privzeta komunikacija pri mikrostoritvah. Zaradi hitrejše in boljše odzivnosti je treba mikrostoritve razvijati na osnovi asinhronega protokola.

Napačno je razmišljanje, da lahko velike monolitne rešitve enostavno razbijemo na neke logične oziroma manjše enote, ker s tem ne dobimo pravih mikroservisov z zgoraj opisanimi lastnostmi, ampak mikrolite, ki imajo še zmeraj slabe lastnosti velikih monolitov.

PODATKOVNI TOKOVI

Tok je zaporedje trenutnih dogodkov, ki so naročeni v nekem časovnem obdobju. Ti lahko oddajajo tri različna stanja, in sicer *vrednost, napako* ali *uspešen konec*. Ta stanja ujamemo, ker dogodki ta stanja asinhrono oddajajo, opredeljujejo pa jih določene funkcije (*se je izvršilo*, *napaka* in *končano*). Podatkovni tok (zaporedje dogodkov po časovni komponenti) lahko uporabimo kot dodatek k drugim tokovom. Celo več, tokovi se lahko uporabljajo kot vložki v druge tokove. Lahko združimo dva tokova. Lahko filtriramo tok, da bi tako dobili še enega s samo tistimi dogodki, ki nas zanimajo. Lahko preslikamo vrednosti podatkov iz enega toka v drugi tok ipd.

Primer podatkovnega toka s klikom na gumb prikazuje slika 2.



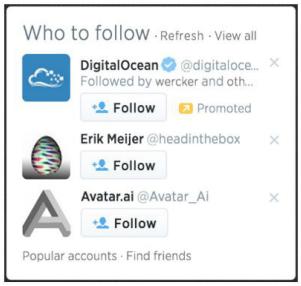
Slika 2: Prikaz podatkovnega toka (Vir: Staltz, 2014)

Te emitirane dogodke zajemamo le asinhrono, tako da definiramo funkcijo, ki se bo izvedla, ko se oddaja vrednost, nato drugo funkcijo, ki se bo izvedla, ko se oddaja napaka, in tretjo funkcijo, ki se bo izvedla, ko je oddajanje končano. Včasih se ti zadnji dve funkciji izpustita in se lahko osredotočimo le na določanje funkcije za vrednosti. Poslušanje toka se imenuje ***naročanje***. Funkcije, ki jih definiramo, pa so ***opazovalci***. Tok je predmet, ki ga opazujemo.

ZAKAJ UPORABITI REAKTIVNI PRISTOP?

Reaktivno programiranje dviguje raven abstrakcije v kodi, tako da se lahko osredotočimo na soodvisnost dogodkov, ki opredeljujejo poslovno logiko, namesto da se nenehno ukvarjamo z veliko količino podrobnosti glede izvedbe. Prednost je bolj očitna v sodobnih internetnih rešitvah in mobilnih aplikacijah, ki so močno interaktivno povezane z množico uporabnikov in dogodkov, le-ti pa s podatkovnimi dogodki. Pred desetimi leti je bila interakcija s spletnimi stranmi v bistvu pošiljanje dolgih obrazcev v izvajanje na strežnik, preprosta prikazovanja rezultatov pa so bila uporabniku na voljo na odjemalcu. Aplikacije so se sčasoma razvile v aplikacije v realnem času, tako da se spreminjanje polj obrazcev lahko samodejno funkcijsko kaže v ozadju, kot npr. všečkanje, in se nekatere vsebine lahko prikazujejo v realnem času pri drugih povezanih uporabnikih, ki so v omrežju (Staltz, 2014).

Danes aplikacije ponujajo nepregledno množico vseh vrst dogodkov v realnem času, ki omogočajo zelo interaktivno izkušnjo za uporabnika. Zato potrebujemo orodja in paradigmo za njihovo ustrezno obravnavo, reaktivno programiranje pa je lahko odgovor na takšno zahtevo. Čeprav so nekateri koncepti za reaktivno programiranje poznani in v uporabi že nekaj časa, gre za novo paradigmo, ki jo potrebujemo za nove sodobne rešitve v sodobnih arhitekturah. Kot primer navajamo Twitterjev uporabniški vmesnik za spremljanje ljudi oziroma avtomatiziran seznam priporočenih uporabniških računov Who to follow (Staltz, 2014). Naš namen ni opisovati funkcije in kode, ki je sicer na voljo na spletnem naslovu http://jsfiddle.net/staltz/8jFJH/48/, kjer lahko delovanje tudi preizkusimo, ampak gre za opis realnega primera. Opisati želimo, kako z nekaj razmeroma preprostimi klici funkcij in na osnovi miselnosti, da je vse podatkovni tok, izvedemo funkcije osveževanja in brisanja, ki so prikazane na sliki 3. Pri tem moramo vedeti, da primer ne deluje z vmesniki proizvajalcev programske opreme (npr. Twitterjev API), ker so lahko le-ti iz različnih razlogov (poslovnih, ekonomskih, zakonskih ipd.) zaprti oziroma javnosti nedostopni.



Slika 3: Twitterjev avtomatiziran seznam priporočenih uporabniških računov Who to follow (slov. *Komu slediti*) (Vir: Staltz, 2014)

REŠITVE IN ORODJA

Reaktivne rešitve z asinhronim pristopom so zelo težko izvedljive, če prej ne napravimo miselnega preskoka, saj lete od razvijalcev v osnovi zahtevajo drugačno razmišljanje. Seveda je to lažje priporočiti kot izvesti. Vpeljava novega pristopa ni vedno preprosta, ker smo ljudje postavljeni v okolja, kjer že obstajajo ustaljeni postopki in rešitve. Zato potrebujemo nova orodja, ki bodo imela v jedru vgrajeno reaktivnost oziroma bodo izkoriščala reaktivnost programskega jezika. Tako je danes najbolj razširjen programski jezik go (https://www.golang-book.com/books/intro), ki razvijalcem v osnovi pomaga pristopiti k reaktivnemu razvoju.

Ker količine podatkov neprestano rastejo, bo obdelovanje v realnem času postalo dejstvo, paketne obdelave (angl. *offline/batch processing*) pa preteklost. Tu se pojavljajo različna orodja za obdelavo procesiranja tokov podatkov, kot so Apache Spark, Apache Kafka, Riak ipd. Pri procesiranju tokov se pojavljajo različne obremenitve, zato se predlaga standard (Reactive Streams 1.0.0) za asinhrono obdelavo podatkov. Eden od konceptov reševanja procesiranja tokov je povratni pritisk (angl. *back pressure*), ko sistem izvoru javi, da ne more obdelovati takih količin podatkov, izvor pa nato začne pošiljati manjše količine podatkov (angl. *pull*). Ko sistem zazna, da je manj obremenjen, javi viru, da naj pošilja več podatkov (angl. *push*).

Standard tako definira minimalno množico vmesnikov ter metod in protokolov za doseganje asinhronega procesiranja podatkov, kar je implementirano v orodjih, kot so *Akka Streams, MongoDb, Ratpack, Reactive Rabbit* ipd. Specifikacija je odprtokodna rešitev (licenca *Creative Commons*).

Pri razvoju reaktivnih rešitev se pojavljajo različni pristopi. Za reaktivne rešitve so primerni funkcionalni jeziki, ki imajo vgrajene nekatere koncepte za obdelavo podatkov. Eden od načinov je uporaba knjižnice *ReactiveX* in se pojavlja tudi v programskem jeziku java kot *RxJava*, precej reaktivnih rešitev pa je tudi v jeziku *go (RxGo)*. Pri reaktivnih rešitvah je treba razmišljati tako na strani strežnika kot na strani odjemalca. Na podlagi izkušenj predstavljamo knjižnico *RxJava*, ki vsebuje med drugim razred *Observable* in vmesnik *Observer* (za obdelavo tokov), imena metod pa so *OnCompleted, OnError* in *OnNext.* Vse metode se kličejo kot povratni klici (angl. *callback*). Za vsak dogodek se kliče metoda *OnNext*; če so dogodki končani, se kliče metoda *OnCompleted,* sicer pa metoda *OnError*.

ZAKLJUČEK

Ob uporabi vse večjega števila pametnih naprav in ob vse večjem številu uporabnikov na eni strani ter storitev na drugi strani se nam zdi ključnega pomena razmišljanje, da se v okolje, kjer nimamo lastnosti reaktivnega pristopa, uvede uporaba konceptov reaktivnega programiranja, kot sta asinhronost in povratni pritisk. S tem ne mislimo, da se morajo spremeniti rešitve, kajti te lastnosti lahko uporabljamo tudi v monolitnih rešitvah. Kakor koli, zanimivo bi bilo npr. na spletnih straneh uporabnikom postopno ponuditi različne informacije, ki so na voljo v danem trenutku. Je pa potreben razmislek o tem, kaj uporabiti za razvoj in dostavo mikroservisov (npr. Docker, https://www.docker.com/) ter orkestracijo (npr. Kubernetes, https://kubernetes.io/) (Krajnc, et al., 2017), kajti gre za nova orodja, ki zahtevajo veliko truda, da se jih spozna, pravilno uporablja in integrira v obstoječi razvoj.

Reference

Bonér, J., Farley, D., Kuhn, R. in Thompson, M., 2014. *The Reactive Manifesto (v2.0)*. [online] Dostopno na: http://www.reactivemanifesto.org/ [13. 2. 2019].

Krajnc, A., Štok, B. in Petr, C., 2016. Reaktivni model spleta in programiranja. V: Heričko, M. in Kous, K. ur. *Sodobne tehnologije in storitve: OTS 2016 : zbornik enaindvajsete konference, Maribor, 14. in 15. junij 2016*. Dostopno na: http://ots.si/Prejsnje_konference/OTS_2016/OTS2016/wp-content/uploads/2016/06/OTS_2016_Zbornik_Web_z.pdf [13. 2. 2019].

Krajnc, A., Zaletel, U., Štok, B. in Petr, C., 2017. Orkestracija mikroservisov z uporabo Kubernetes. V: Heričko, M. in Kous, K. ur. *Sodobne tehnologije in storitve: OTS 2017 : zbornik dvaindvajsete konference, Maribor, 13. in 14. junij 2017*. Dostopno na: https://www.ots.si/Prejsnje_konference/OTS_2017/OTS2017/wp-content/uploads/2017/06/OTS_2017_zbornik_splet-zasciten.pdf [13. 2. 2019].

Staltz, A., 2014. *The introduction to Reactive Programming you've been missing*. [blog] Dostopno na: https://gist.github.com/staltz/868e7e9bc2a7b8c1f754 [13. 2. 2019].

Wikipedia, 2018. *Reactive programming.* [online] Dostopno na: https://en.wikipedia.org/wiki/Reactive_programming#Definition_of_Reactive_Programming [13. 2. 2019].