Reaktivna paradigma

Natalija Drašković
natalija.draskovic@outlook.com

4. novembar 2018.

1 Uvod

Postoji veliki broj problema koji zahtevaju pisanje programskih sistema koji imaju komponente koje se izvršavaju međusobno asinhrono. Prethodno pomenuto asihrono izvršavanje podrazumeva da se određeni delovi programa izvršavaju nezavisno jedni od drugih, odnosno da ne postoji čekanje završetka izvršavanja jednog dela programa pre početka izvršavanja drugog, već se oni izvršavaju u istom vremenskom periodu. Na primer, kod distribuiranih izračunavanja postoje distribuirani čvorovi koji obrađuju podatke asinhrono, sistemi se servisno-orijentisanim arhitekturama moraju da obrađuju zahteve koje dobijaju od klijenata asinhrono, a u višejezgarnim heterogenim sistemima postoji više odvojenih zadataka koji se izvršavaju istovremeno radi što boljeg iskorišćenja hardverskih resursa. Čak i obični grafički programi imaju asinhrone komponente, odnosno grafičko korisničko okruženje ne sme biti zaglavljeno ni u jednom renutku, što znači da bez obzira na to šta program trenutno radi, ne sme da prestane da obrađuje i reaguje na ulazne događaje koji dolaze od korisnika. Programska paradigma koja omogućava ovakav način programiranja i bavi se problemima asinhronog programiranja je *reaktivna paradigma*.

Reaktivna paradigma je paradigma programiranja usmerena na tokove podataka u smislu prenošenja izmena prilikom promene podataka. Ovo znači da treba biti moguće da se jednostavno opišu statički ili dinamički tokovi podataka, vrši komunikacija sa povezanim izvršnim modelom i omogući automatsko propagiranje promena tokova podataka. Ovo programiranje uvodi novi način razmišljanja čija je ideja da se uspostave neke zavisnosti između različitih delova programa tako da promena jednog automatski propagira promenu drugom.

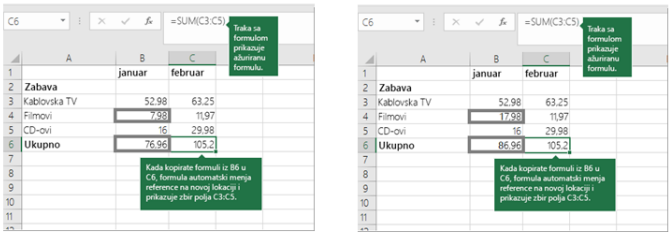
Primer 1.1 *Jedan od najjednostavnijih primera za razumevanje osnovnog principa reaktivnog programiranja jeste primer sabiranja dva broja. U proceduralnim programskim jezicima, kao što je C, ili u Javi, izraz a=b+c je komanda koja se izvršava dodelom vrednosti promenljivoj a na osnovu trenutnih vrednosti promenljivih b i c i kasnija promena vrednosti b ili c ne utiče na promenu vrednosti a. Kod reaktivnog programiranja, izraz a=b+c ima značenje da svaka promena vrednosti b ili c ima za rezultat automatsko ažuriranje vrednosti a, odnosno sa svakom promenom b ili c vrednost a se ponovo izračunava koristeći nove vrednosti.*

2 Rad sa tabelama

Primena računara u radu sa tabelama je velika. Postoji veliki broj programa koji nam omogućavaju da radimo sa tabelama i vršimo potrebna izračunavanja. Programiranje u okviru tabela je još jedan vid reaktivnog programiranja, a neki od programa koji se koriste za to su *VisiCalc*, *Excel*, *LibreOfficeCalc*.

Ovi programi omogućavaju nam da, definišemo zavisnosti između određenih ćelija tabele ili čitavih kolona ili vrsta te tabele. Zavisnosti između delova tabele ostvaruju se funkcijama ili nekim izrazima koje mi sami definišemo. Promenom vrednosti neke od ćelija ažurira se čitava tabela, odnosno sve one ćelije tabele koje zavise od promenjene ćelije. U tom automatskom propagiranju promene svih vrednosti tabele koje je neophodno promeniti i međusobne zavisnosti određenih ćelija ogleda se reaktivan način reagovanja programa na promenu.

Primer 2.1 *Ako bismo na levom delu slike 1 vrednost u polju za filmove za januar uvećali za 10, tada bi se i ukupna suma za januar uvećala za 10, dok bi suma za februar ostala ista zbog toga što ne zavisi od date vrednosti, što se može videti na desnom delu slike 1.*



Slika 1: Tabele u Excelu

3 Jezici za opis hardvera

Jezik za opis hardvera je specijalizovani jezik koji se koristi za opisivanje osobina i ponašanja elektronskih kola i digitalnih logičkih kola. On omogućava precizan, formalan opis elektronskog kola koji dozvoljava automatsku analizu i simulaciju elektronskog kola i olakšava konstrukciju integrisanih kola, što dalje omogućava lakši i brži razvoj hardvera, lakše i brže testiranje i verifikaciju hardverske implementacije, kao i portabilnost realizovane implementacije na platforme različitih proizvođača.

Jedan kod jezika za opis hardvera odgovara opisu dizajna jednog hardverskog bloka (modula) koji izvršava određenu logičku funkciju (ili skup logičkih funkcija). Kada se opisuje jedan hardverski modul važne su dve stvari. Prvu stvar predstavljaju interfejsi modula prema okolini i preko kojih se dotični modul povezuje (komunicira) sa drugim modulima ili čipovima, odnosno stvaraju se određene zavisnosti između različitih modula i čipova. Ovaj deo je bitan za upotrebu modula i njegovu interakciju sa ostalim hardverom u složenim sistemima, jer je korisniku ili projektantu složenog sistema bitna samo funkcija modula, a ne i interna struktura modula tj. kako se zaista izvršava ta funkcija.

Za njih je modul poput 'crne kutije' - nije bitno šta je u 'crnoj kutiji', već samo šta ona radi. Druga stvar je interna struktura modula koja definiše princip rada modula, a samim tim i funkciju (ili skup funkcija) koje modul obavlja. Ovaj deo definiše kako se izvršava funkcija koju modul treba da obavi i ona je važna dizajnerima modula, jer je cilj realizovati što optimalniju implementaciju interne strukture modula da bi modul što efikasnije obavljao svoju funkciju. Reaktivan način programiranja ogleda se u tome što na osnovu formiranih zavisnosti i funkcija modula, često promena u jednom od modula automatski izaziva promenu u drugom zavisnom modulu, a zatim on izaziva promenu nekog modula koji zavisi od njega i tako sve dok postoji međusobna zavisnost dva modula, čime data promena utiče ne samo na jedan modul već na čitav skup zavisnih modula.

Ovi jezici dosta liče na programske jezike kao što je C, ali razlika između većine programskih jezika i jezika za opis hardvera je u tome što jezik za opis hardvera eksplicitno uključuje pojam vremena u čitav proces, što ga stavlja u programe reaktivne paradigme. Primena ovog tipa programiranja ogleda se u tome što se izmena jednog kola u dizajnu propagira na celo kolo, a neki od jezika koji se koriste su *Verilog* i *VHDL*.

4 Funkcionalno reaktivno programiranje

Funkcionalno reaktivno programiranje [6] je vrsta programiranja koja kombinuje funkcionalno programiranje sa reaktivnom paradigmom. Kombinovanjem ove dve paradigme dobijamo funkcionalno programiranje koje se može koristiti za aplikacije u realnom vremenu. Takođe, funkcionalno reaktivno programiranje omogućava da možemo programirati takozvane hibridne sisteme (eng. *hybrid systems*) [5] koji omogućavaju rad i sa kontinuiranim i sa diskretnim komponentama. Kako bismo govorili o ovoj vrsti programiranja navedimo prvo definiciju i osnovne osobine funkcionalnog programiranja.

Imperativnu paradigmu karakteriše postojanje naredbi, odnosno izvšavanje programa svodi se na izvršavanje naredbi. Takođe, izvršavanje programa može se svesti i na evaluaciju izraza, a ako su ti izrazi funkcije dobijamo funkcionalno programiranje. *Funkcionalna paradigma* se zasniva na pojmu matematičkih funkcija i njen osnovni cilj je da oponaša matematičke funkcije, a rezultat toga je pristup programiranju koji je u osnovi drugačiji od imperativnog programiranja. Ova paradigma zasniva se na izračunavanju izraza kombinovanjem funkcija, tj. program je niz definicija i poziva funkcija, a izvršavanje programa evaluacija funkcija.

Funkcionalno reaktivno programiranje je programiranje kod koga su programi funkcionalni ali se argumenti funkcije mogu menjati tokom vremena, a te promene se propagiraju na izlaz. Ova vrsta programiranja nasleđuje tradicionalno funkcionalno programiranje dodajući rad sa tokovima podataka čime omoćava pisanje interaktivnih programa u deklarativnom stilu. Programi pisani korišćenjem ovakve vrste programiranja kreiraju dinamičke grafove zavisnosti podataka i na promene reaguju propagiranjem ažuriranja kroz graf [3]. Ova vrsta programiranja je prvi put prikazana u Fran-u [4], jeziku specifične namene (eng *Domain Specific Language*) za grafiku i animacije razvijenog od strane Konal Eliota (eng. *Conal Eliott*) tokom Majkrosoftovog (eng. *Microsoft*) istraživanja. Takođe, ovaj vid proramiranja se koristi i kod programiranja za

sisteme sa ugrađenim računarom (eng. *embedded*), na primer za programiranje robota, za šta je razvijen jezik specifične namene u Haskell-u nazvan Yampa [5].

Primer 4.1 *REScala je reaktivni jezik koji integriše koncepte programiranja zasnovanog na događajima (eng. event-based) i funkcionalno reaktivnog programiranja u svet objektno orijentisanog programiranja. On podržava razvoj reaktivnih aplikacija podsticanjem funkcionalnog i deklarativnog stila koji dopunjuje objektno orijentisano programiranje i dodaje mu još neke korisne osobine. Programi pisani na ovakav način reaguju na spoljašnje promene kao što su korisnički unos podataka i poruke dobijene sa mreže.*

5 Reaktivno programiranje i biblioteka Rx

U programiranju *reaktivna proširenja* (eng. *reactive extensions*), poznata i kao *ReactiveX* [2], su skup alata koji omogućavaju jezicima imperativne paradigme da rade se nizovima podataka bez obzira da li su oni sinhroni ili asinhroni. Ovo je jedna implementacija reaktivnog programiranja i obezbeđuje alate neophodne za reaktivno programiranje u mnogim imperativnim jezicima.

Rx je biblioteka za kreiranje asinhronih programa i programa zasnovanih na događajima (eng. *event-based programs*) [1] koja za sve nizove podataka predstavlja kao nizove na koje je moguće motriti (eng. *observable sequences*) i LINQ redove (eng. *language integrated query*). Aplikacija se zatim može povezati tako da prima asinhrona obaveštenja kada računar primi novi podatak u neku od ovih struktura. Ova biblioteka predstavlja jedno reaktivno proširenje imperativnih jezika i dostupna je kod razvoja aplikacija u Javi, .NET-u, JavaScript-u, C-u, Python-u, PHP-u, Swift-u, Scali itd. Prethodno pomenuti tipovi podataka i kolekcije operatora nad njima podržavaju rad sa nizovima podataka ili događaja i omogučavaju nam deklarativan pristup apstrahovanjem stvari kao što je rad sa nitima, sinhronizacija, konkurentni tipovi podataka itd.

Primer 5.1 *Na primeru koda pisanog korišćenjem biblioteke RxJava prikazane su neke od klasa i metoda sadrzanih u toj biblioteci. Klasa Observable sadrzi one entitete koje posmatramo i koristi se kada imamo relativno malo predmeta posmatranja tokom vremena, dok je u slučaju puno predmeta posmatranja neophodno koristiti klasu Flowable. Metod just koristimo kako bismo napravili objekat klase Observable koji sadrzi dati predmet koji želimo da posmatramo, dok metod subscribe povezuje posmatrača i predmet posmatranja klase Observable kako bi bilo moguće da posmatrač vidi dati predmet i primi obaveštenja ili greške tokom rada. Posmatranjem narednog koda vidimo da se u njemu kreira objekat klase Observable i povezuje sa standardnim ulazom kao posmatračem na koji emituje predmet posmatranja, u ovom slučaju nisku "Hello world".*

```
package rxjava.examples;

import io.reactivex.*;

public class HelloWorld {
    public static void main(String[] args) {
        Observable.just("Hello world").subscribe(System.out::println);
    }
}
```

Ponekad se ovaj način programiranja naziva funkcionalno reaktivno programiranja, što nije tačno, programiranje korišćenjem reaktivnih proširenja jeste funkcionalno i jeste reaktivno, ali ne pripada funkcionalno reaktivnom programiranju. Najveća razlika između ovog načina i funkcionalno reaktivnog programiranja jeste u tome što funkcionalno reaktivno programiranje operiše i nad vrednostima koje se kontinuirano menjaju tokom vremena, dok se korišćenjem reaktivnih proširenja operiše samo nad diskretnim veličinama koje se emituju tokom vremena.

Primer 5.2 *Imperativno programiranje bez korišćenja reaktivnih proširenja može se posmatrati kao kada posećujemo biblioteku kako bismo pročitali neku knjigu koju ta biblioteka poseduje. Međutim, ako nakon toga želimo neku drugu knjigu moramo ponovo otići u biblioteku kako bismo je pročitali. Slučaj korišćenja reaktivnih proširenja možemo poistovetiti sa učlanjenjem u neki književni klub u kome biramo određeni žanr koji nas interesuje, nakon čega knjige koje nas interesuju automatski dobijamo da pročitamo nakon što su objavljene. Najvažnija razlika je u tome što ne moramo čekati u redu da dobijemo knjigu koju želimo, već nam ona biva dostavljena odmah nakon objavljvanja. Ovaj primer objašnjava ne samo način funkcionisanja ovih biblioteka već kompletan koncept i ideju reaktivnog programiranja.*

6 Zaključak

Reaktivno programiranje je pre svega predstavljeno kao način da se pojednostavi izrada interaktivnih korisničkih interfejsa, animacija u sistemima koji rade u realnom vremenu, ali je danas jedna od programskih paradigmi koja se dosta koristi i sve više nalazi svoju primenu i u okviru nekih drugih programskih paradigmi.

Literatura

[1] Reactive Extensions. on-line at: https://docs.microsoft.com/en-us/previous-versions/dotnet/reactive-extensions/hh242985(v%3dvs.103).

[2] ReactiveX. on-line at: http://reactivex.io/.

[3] Kimberley Burchett, Gregory H. Cooper, and Shriram Krishnamurthi. Lowering: a static optimization technique for transparent functional reactivity. 2007.

[4] Conal Elliott and Paul Hudak. Functional reactive animation. *SIGPLAN Not.*, 32(8):263–273, August 1997.

[5] Antony Hudak, Pauland Courtney, Henrik Nilsson, and John Peterson. *Arrows, Robots, and Functional Reactive Programming.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[6] Peter Van Roy. Programming paradigms for dummies: What every programmer should know.