

Erlang - funkcionalno rešenje za konkurentni svet

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Tijana Jevtić, Jelena Mrdak, David Dimić, Zorana Gajić
tijanatijanajevtic@gmail.com, mrdakj@gmail.com,
daviddimic@hotmail.com, zokaaa_gajich@bk.ru

6. april 2019.

Sažetak

U ovom radu je prikazan programski jezik Erlang iz različitih uglova. Kroz niz poglavlja i primera, ispričana je njegova istorija - kad, kako, gde i zašto je nastao, po čemu je karakterističan, šta ga to izdvaja od drugih programskih jezika, koji su to koncepti koji su svojevrsni Erlangu. Nakon čitanja rada, čitalac će imati globalnu sliku o jeziku i detaljniji pogled na neke važne koncepte, kao i uvid u korišćenu literaturu koju može konsultovati radi daljeg informisanja o temi.

Sadržaj

1 Uvod	2
2 Nastanak i istorijski razvoj	2
3 Osnovna namena, svrha i mogućnosti	3
4 Osnovne osobine	3
5 Okruženja (framework) i njihove karakteristike	4
6 Instalacija i pokretanje	4
7 Primeri kodova sa objašnjenjima	5
8 Specifičnosti	6
9 Zaključak	6
Literatura	6
A Dodatak	6

1 Uvod

Primer 1.1 *I tabele treba da budu u svom okruženju, i na njih je neophodno referisati se u tekstu. Na primer, u tabeli 1 su prikazana različita poravnanja u tabelama.*

Tabela 1: Različita poravnanja u okviru iste tabele ne treba koristiti jer su nepregledna.

centralno poravnanje	levo poravnanje	desno poravnanje
a	b	c
d	e	f

2 Nastanak i istorijski razvoj

1981. godine je oformljena nova laboratorija, Erikson CSLab (eng. *The Ericsson CSLab*) u okviru firme Erikson sa ciljem da predlaže i stvara nove arhitekture, koncepte i strukture za buduće softverske sisteme. Eksperimentisanje sa dodavanjem konkurentnih procesa u programski jezik Prolog je bio jedan od projekata Erikson CSLab-a i predstavlja začetak novog programskog jezika. Taj programski jezik je 1987. godine nazvan Erlang¹. Sve do 1990., Erlang se mogao posmatrati kao dijalekt Prologa. Od tada, Erlang ima svoju sintaksu i postoji kao potpuno samostalan programski jezik. Godine rada su rezultirale u sve bržim, boljim i stabilnijim verzijama jezika, kao i u nastanku standardne biblioteke OTP (eng. *The Open Telecom Platform*) [?]. Od decembra 1998. godine, Erlang i OTP su postali deo slobodnog softvera (eng. *open source software*) i mogu se slobodno preuzeti sa Erlangovog zvaničnog sajta [?]. Danas, veliki broj kompanija koristi Erlang u razvoju svojih softverskih rešenja. Neke od njih su: Erikson, Motorola, Votsap (eng. *Whatsapp*), Jahu (eng. *Yahoo!*), Amazon, Fejsbuk (eng. *Facebook*).

2.1 Uticaji

Erlang je funkcionalan i konkurentan programski jezik. Na njega, kao na funkcionalan jezik, uticao je Lisp funkcionalnom paradigmom koju je prvi predstavio. Na planu konkurentnosti Erlang je svojevrsan primer (detaljnije u poglavlju 4).

Na početku, Erlang je stvaran kao neki dodatak na Prolog, vremenom prerastao u dijalekt Prologa, a kasnije je zbog svoje kompleksnosti i sveobuhvatnosti evoluirao u potpuno novi programski jezik. Stoga je uticaj Prologa na Erlang bio neminovan. Sintaksa Erlanga u velikoj meri podseća na Prologovu (npr. promenljive moraju počinjati velikim slovom u oba jezika, svaka funkcionalna celina se završava tačkom), oba jezika u velikoj meri koriste poklapanje obrazaca (eng. *pattern matching*).

¹Erlang je jedinica saobraćaja u oblasti telekomunikacija i predstavlja kontinuirano korišćenje jednog kanala (npr. ako jedna osoba obavi jedan poziv telefonom u trajanju od sat vremena, tada se kaže da sistem ima 1 Erlang saobraćaja na tom kanalu).

Sa druge strane, Erlang je uticao na nastanak programskog jezika Elixir (eng. *Elixir*). Elik sir, uz izmenjenu Erlangovu sintaksu, dopunjenu Erlangovu standardnu biblioteku, uživa široku popularnost.

3 Osnovna namena, svrha i mogućnosti

Sa početkom od 1981. godine, jedan od zadataka Eriksonove laboratorije za računarstvo je bio pronalaženje načina za bolje programiranje aplikacija za telekomunikacije [?]. Takve aplikacije su ogromni programi i od velike važnosti je da rade sve vreme (koliko je to moguće). Naravno, poznato je da će tolika količina koda zasigurno imati greške, ali u toj vrsti industrije, greške mogu biti fatalne. Na primer, šta se dešava ako je došlo do kvara na nekoj telefonskoj liniji, a telefon nam je hitno potreban (recimo, neko ima srčani udar). Jednostavno nije moguće zaustaviti takvu aplikaciju, popraviti je i nanovo pustiti u rad. Kako se izboriti sa greškama u softverskim sistemima kada su one neminovne je osnovna motivacija za razvoj Erlanga [?].

Tako, jedna od njegovih namena jeste pisanje što sigurnijih programa koje je moguće popraviti bez potrebe za isključivanjem čitavog sistema [?]. Vrlo brzi konkurentni i distribuirani programi su još jedna od Erlangovih specijalnosti. Poseban koncept konkurentnosti koji je implementiran u Erlangu (više u poglavlju 4), kao i funkcionalna paradigma omogućavaju lako skaliranje programa i pravljenje velikih konkurentnih i distribuiranih sistema. Velika zajednica koja se godinama razvijala je doprinela stvaranju velikog broja biblioteka i okruženja za Erlang, te proširila njegov inicijalni skup mogućnosti i namena [?].

4 Osnovne osobine

4.1 Podržane paradigme

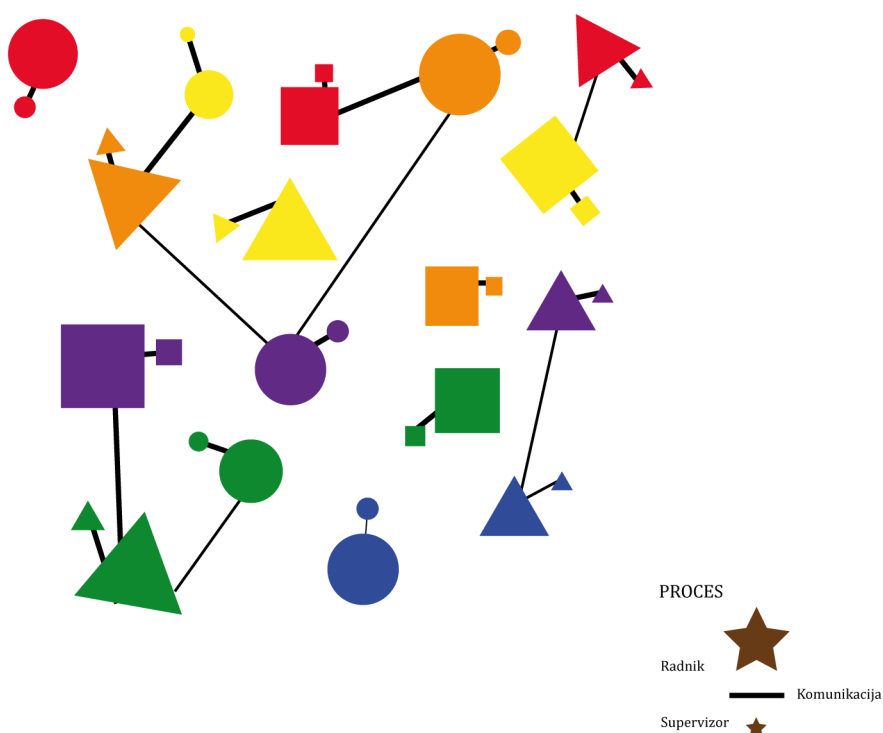
4.2 Koncepti

Jedna od osnovnih osobina Erlanga je konkurentnost i koncept na kom je zasnovana. Reč konkurentnost se odnosi na više radnji koje se dešavaju u istom trenutku. Posmatrajući svet oko sebe, uviđamo da je on u suštini konkurentan - u istom trenutku se dešava veliki broj procesa. [?] U tom istom trenutku, mi smo sposobni da takav svet pojmimo i odreagujemo na sve što se u njemu dešava. Dakle, mi prirodno razumemo konkurentnost. Tako se i prirodno nameće potreba za programskim jezikom koji bi omogućavao jednostavno modelovanje sveta kakav on stvarno jeste.

Koncept konkurentnosti implementiran u Erlangu se zove konkurentnost slanjem poruka (eng. *message passing concurrency*) i podrazumeva postojanje velikog broja procesa koji nikad ne dele memoriju, već komuniciraju isključivo asinhronim slanjem poruka [?]. Sva izračunavanja se obavljaju u okviru procesa i trebalo bi da sistem bude dizajniran tako da jedan proces radi jedan mali posao. Važno je napomenuti da procesi u Erlangu nisu procesi operativnog sistema, već Erlanga. To je moguće zbog toga što se programi napisani u Erlangu izvršavaju na BEAM virtualnoj mašini. Tako, procesi u Erlangu se prave i uništavaju jako brzo, zauzimaju samo onoliko memorije koliko je neophodno, u većini slučajeva

jako malo i ponašaju se isto na svim operativnim sistemima [?].

Poređenja radi, koncept konkurentnosti korišćen od strane većine programskih jezika je takozvani koncept konkurentnosti deljenih stanja (eng. *shared state concurrency*) gde procesi menjaju memoriju (nasuprot tome, u Erlangu vrednost promenljivoj može biti dodeljena samo jednom) [?]. U slučaju da više procesa žele da menjaju istu memoriju, tzv. kritična sekcija, potrebno je nekako zaštititi taj deo memorije (muteksi, katanci i dr.). U slučaju da do greške dođe baš u kritičnoj sekciji, ostali procesi ne znaju kako da se nose sa datom situacijom i u najboljem slučaju sistem prekida sa radom.



Slika 1: Konkurentnost u Erlangu

5 Okruženja (framework) i njihove karakteristike

6 Instalacija i pokretanje

Postoji više načina da se instalira Erlang sa neophodnim paketima. U ovom odeljku će biti predstavljena instalacija korišćenjem prekompajliranih binarnih fajlova za neke operativne sisteme zasnovane na Linuksu.

kernelu i pokretanje na jednom od njih, kao i instalacija za Windows.

6.1 Linux

Na operativnim sistemima zasnovanim na *Ubuntu*, Erlang se može instalirati sa: `sudo apt-get install erlang`.

Nakon uspešne instalacije, Erlang kod je moguće kompajlovati ili interpretirati i pokretati u interpretatoru. Interpretator se pokreće kucanjem komande `erl` u terminalu, a iz istog se izlazi sa `Ctrl+G` iza kog sledi `q` [?]. Erlang interpretator ima u sebi ugrađen editor teksta koji je baziran na *emacs-u* [?]. Kod iz datoteke se kompajluje komandom `erlc` i navođenjem imena fajla sa ekstenzijom `erl`. Nakon toga se dobija izvršna datoteka sa ekstenzijom `beam` koja se može pokrenuti uz navođenje adekvatnih flegova.

6.2 Windows

7 Primeri kodova sa objašnjenjima

Počecemo od primera "Hello World".

```
% hello world program
-module(helloworld).
-export([start/0]).

start() ->
    io:fwrite("Hello, world!\n").

> Hello, world!
```

Nekoliko napomena:

- Znak `%` označava linijski komentar.
- Modul je ekvivalent namespace-u u drugim jezicima.
- Export funkcija se koristi da bi funkcija definisana u programu mogla da se koristi. Mi definišemo funkciju `start`, a da bismo mogli da je koristimo potreban nam je `export`. `/0` označava da funkcija `start` prima 0 argumenata.
- Da bismo željeni tekst prikazali u konzoli, koristimo `io` modul koji sadrži potrebne IO funkcije u Erlangu.

Pošto smo videli kako string možemo ispisati na standardni izlaz, sada ćemo videti kako broj možemo ispisati. Sledeći program prikazuje zbir dva intiger-a.

```
-module(helloworld).
-export([start/0]).

start() ->
    io:fwrite("~w", [1+1]).

> 2
```

Na sličan način možemo ispisivati i razne druge tipove.

Kao i većina funkcionalnih jezika, i Erlang podržava shvatanje listi (eng. list comprehensions), što ilustrujemo narednim primerima.

```
> [X || X <- [1,2,a,3,4,b,5,6], X > 3].
[a,4,b,5,6]
```

Notacija `X <- [1, 2, a, ...]` je generator, dok je izraz `X>3` filter.

Možemo primeniti više filtera.

```
> [X || X <- [1,2,a,3,4,b,5,6], integer(X), X > 3].
[4,5,6]
```

Takođe, moguće je kombinovati i generatore. Na primer, Dekartov proizvod dve liste možemo napisati kao

```
> [{X, Y} || X <- [1,2,3], Y <- [a,b]].
[{1,a},{1,b},{2,a},{2,b},{3,a},{3,b}]
```

Algoritam QuickSort u Erlangu se može implementirati na sledeći način:

```
sort([Pivot|T]) ->
  sort([ X || X <- T, X < Pivot]) ++
  [Pivot] ++
  sort([ X || X <- T, X >= Pivot]);
sort([]) -> [].
```

Izraz `[X || X <- T, X < Pivot]` je lista svih elemenata iz `T` koji su manji od pivot. Slično, `[X || X <- T, X >= Pivot]` je lista svih elemenata iz `T` koji su veći ili jednaki od pivot.

Neizostavna funkcija svih funkcionalnih programskih jezika jeste `map`. `map(F, List)` je funkcija koja prima funkciju `F` i listu `L` i vraća novu listu dobijenu primenom funkcije `F` na svaki element liste `L`.

```
map(F, [H|T]) -> [F(H)|map(F, T)];
map(F, []) -> [].

double(L) -> map(fun(X) -> 2*X end, L).

> double([1,2,3,4]).
[2,4,6,8]
```

8 Specifičnosti

9 Zaključak

A Dodatak