Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/02BE13Jeftic

RAZVOJ DISTRIBUIRANE, VISOKO SKALABILNE APLIKACIJE UZ *MICROSOFT ORLEANS*

DEVELOPMENT OF DISTRIBUTED, HIGHLY SCALABLE APPLICATION WITH *MICROSOFT ORLEANS*

Dušan Jeftić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – Primenjeno softversko inženjerstvo

Kratak sadržaj – *Pružen je opis kako uz pomoć jedinstvenog pristupa Microsoft Orleans radnog okvira da se na potpuno novi način modeluje distribuirana, visoko skalabilna aplikacija. Ovo je postignuto objašnjenjem glavnih koncepata ovog radnog okvira.*

Ključne reči: *radni okvir Micorosft Orleans, distribuirana aplikacija, skalabilnost, virtuelni akter model, pouzdanost*

Abstract – A *description of how to use Microsoft's framework Orleans, with its unique approach, to model a distributed, highly scalabel aplication is given. This was provided by explaining key features and concepts of this framework.*

Keywords: *Microsoft Orleans framework, distributed application, scalability, vritual actor model, reliability*

1. UVOD

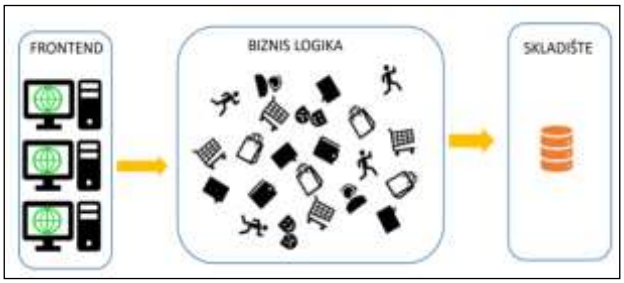
Distribuirane aplikacije su u današnje vreme od ključnog značaja, usled naglog razvoja tehnologije. Zajedno sa razvojem tehnologije javlja se i ogromna potreba za podrškom velikog broja korisnika. Sistemi koji koji nisu distribuirani, teško mogu da pruže podršku od na primer milion korisnika. Distribuirani sistemi predstavljaju mrežu autonomnih, računarskih sistema, koji međusobno razmenjuju informacije i resurse od značaja, kako bi se omogućilo odražavanje velikih sistema [2]. Ovi sistemi pružaju mogućnost skalabilnosti, pa je takvo rešenje lako proširiti dodavanjem novih komponenit. Kako uvek postoji više mašina koje mogu da rade isti posao, ukoliko je jedna u nemogućnosti da izvršava potreban posao, lako se dolazi do zamene [1]. Sve prethodno navedeno dovodi do pozitivnog efekta pouzdanosti sistema i poboljšanju performansi.

2. MICROSOFT ORLEANS I NJEGOV PRISTUP MODELIRANJU DISTRIBUIRANE APLIKACIJE

Microsoft Orleans predstavlja radni okvir sa jedinstvenim konceptom čija je svrha da omogući lakše i efikasnije pravljenje skalabilnih, pouzdanih i održivih distribuiranih aplikacija [3]. Kreiran je 2011. od strane Mikrosoftove istraživačke organizacije za korišćenje u oblak (*cloud)* sistemima [3]. Ono što je dobra strana Orleansa jeste što se lako dolazi do željenih rezultata, bez potrebnog znanja o pisanju konkurentnih i skalabilnih programa. Ovaj radni okvir je javno dostupan za uređivanje od Januara 2015. godine, što znači da je javnosti dostupan za javne kolaboracije, ispravke i dopune od strane svih zainteresovanih [4].

U toku prošlosti postojalo je više pokušaja da se napravi adekvatan model distribuirane aplikacije. Svaki od pristupa iz prošlosti su uspevali da reše problem prethodnog, ali su kao posledicu stvorili novi.

Orleans pokušava da objedini pozitivne stvari starih modela i da uvede sopstvene novine uvođenjem modela virtuelnog aktera [3]. Ono što Orleansov pristup čini jedinstvenim, je što srednji sloj, koji se nalazi između krajnjeg korisnika i skladišta, koji će činiti mnoštvo nezavisnih, jedinstvenih objekata, koji postoje u svakom trenutku. Ovakvi objekti iako izolovani, u mogućnosti su da komuniciraju međusobno, a njihovom izolovanošću od strane programera, ne mora se voditi računa o tome da li su objekti na istim ili različitim serverima [3]. Takođe je od značaja napomenuti da se nad ovakvim objektima radi asinhrono.



Slika 1. *Isti primer slike sa manjim dimenzijama*

Ovaj pristup takođe može da se zamisli kao gomilu malih specificiranih radnika, gde svaki radnik ima svoja zaduženja i koja je u mogućnosti da izvršava zavisno ili nezavisno od drugih radnika. Virtuelni akteri predstavljaju one činioce sistema, koji su svakog momenta postojani u memoriji, a predstavljaju podskup svih ikada kreiranih učesnika [4]. Sa stanovišta programera, na ovaj način više nije na njemu da vodi računa o životnom veku i fizičkoj aktivaciji aktera, već to radi sam Orleans automatski, zahvaljujući ovom vidu apstrakcije aktera [4]. Akteri predstavljaju jedinice izolacije i distribucije u samom Orleansu. Svaki akter ima 128-bitni jedinstveni GUID, koji biva kreiran na osnovu

---

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Aleksandar Kupusinac, vanr. prof.

njegovog tipa i primarnog ključa [10]. Kako su akteri izolovani, oni međusobno ne mogu da dele memoriju, odakle sledi da jedini način na koji dva aktera mogu da komuniciraju je putem poruka [10].

3. ZRNA

Zrno predstavlja glavnu egzekucionu jedinicu radnog okvira Orleans, i ono je zapravo predstavnik samog virtuelnog aktera. Dva zrna međusobno mogu da interaguju tako što će pozivati metode zrna sa kojim žele da ostvare komunikaciju [6]. Takođe je moguće da skladište reference ka drugim zrnima [6]. Jedan od najvećih izazova je obezbediti konzistentnost podataka unutar jednog velikog sistema sa velikim brojem korisnika.

Ovo se postiže zabranom deljenja podataka između zrna, osim putem poruka [6]. Takođe, Orleans garantuje da će svaka akcija nad zrnom uvek biti izvršana nekonkurentno [6].

3.1. Kreiranje zrna

Identitet i funkcionalnost svakog zrna postiže se kroz interfejse, koje će zrno da implementira [7]. Kao što je prethodno pomenuto, jedno zrno će komunicirati sa drugim zrnom ukoliko pozove metode drugog zrna, koje su mu od koristi. Odatle sledi, da je za kreiranje zrna potrebno kreirati klasu zrna, i implementirati interfejse, koji će obuhvatati željeno ponašanje [7].

Pre nego što budu objašnjeni sami detalji pravljenja intrfejsa zrna, bitno je objasniti šta je to klasa *Task*, koja pripada Microsoftovom .NET radnom okviru. Klasa *Task* predstavlja izolovanu operaciju, koja će se izvršavati asinhrono, nezavisno od glavne niti [7]. Ova klasa zbog svog asinhronog ponašanja, omogućava pristup stanja određenog zadatka (*Task)*, kako bi se videlo da li je stanje otkazano, završeno ili se desila greška [7]. Ono što je bitno napomenuti, jeste da sve metode interfejsa, koje će zrno da implementira, moraju da vraćaju objekat tipa *Task* za sve metode koje nemaju povratni tip vrednosti [7]. Za metode koje imaju proizvoljan povratni tip *T*, očekuje se da vraćaju objekat tipa *Task*<*T*> [7]. Slika 2 pokazuje primer jednog zrna koji predstavlja mušteriju.

```
public interface ICustomerGrain : IGrainWithGuidKey
{
    Task LogIn(IShopGrain shop);
    Task LogOut(IShopGrain shop);
    Task BuyProduct(IShopGrain shop, IItemGrain item);
    Task<IShopGrain> GetCurrentShop();
    Task<IItemGrain> GetCurrentItem();
}
```

Slika 2. *Interfejs mušterije zrna*

Nakon što je kreiran željeni interfejs ponašanja, sledeći korak je da se kreira klasa koja će da implementira ovaj interfejs. Pored toga što će ova klasa morati da nasledi prethodno kreiran interfejs, ona će takođe morati da nasledi osnovnu klasu zrna, koja se zove *Grain.* Posle toga ove iste metode je potrebno implementirati.

3.2. Životni ciklus zrna

Kao i svaki objekti u objektno orijentisanom programiranju, tako i zrna imaju svoj životni vek. Svako zrno prolazi kroz 4 stanja unutar svog životnog ciklusa. Ova 4 stanja su: aktivno u memoriji, deaktivirano, perzistirano i aktivirano [6]. Zrno prolazi kroz sva 4 stanja u toku jednog ciklusa i to prelazi iz jednog stanja u drugo, redom koji je naveden prethodno a prikazan na slici 3.

Slika 3. *Životni ciklus zrna*

Sva zrna žive u egzekucionim kontejnerima koji se nazivaju silos i predstavljaju Orleansove servere. Ovi silosi formiraju klastere, kombinujući mnoštvo fizičkih ili virtualnih mašina i na taj način pružaju skalabilnost i stabilnost distribuirane aplikacije [6]. Kada dođe do određenog zahteva, potrbeno je aktivirati zrno. Da bi se to obavilo, zrno se prvo traži na nekom od silosa unutar klastera, ukoliko potraga bude neuspešna, napraviće se nova instanca tog zrna, koja će posle biti postojana na nekom od silosa [6].

Dokle god to zrno obavlja određene zahteve, ili sam šalje svoje zahteve drugim zrnima, to zrno će biti aktivno u memoriji, i biće spremno za obradu [6]. Ukoliko nakon dužeg, unapred konfigurisanog vremena, neko zrno ne bude niti slalo, niti primalo, niti obavljalo ikakve zadatke, pokreće se proces deaktiviranja zrna, gde se oslobađa memorija koju je to zrno zauzelo, a nakon toga zrno prelazi u perzistentno stanje [6].

Ovo perzistentno stanje, znači da će instance tog zrna postojati negde, na nekom od silosa, ali što se tiče same programerske strane, može se uvek smatrati kao da je zrno uvek aktivno i spremno za korišćenje, jer ostale stvari Orleans radi sam u pozadini [6].

4. KLIJENT

Klijentom se smatra deo sistema, koji će obavljati interakciju sa zrnima radi ostvarivanja određenih ciljeva, ali sam klijent nije deo logike zrna [8]. Ono što se misli pod time, da taj deo sitema nije deo logike zrna, jeste da se kod klijenta neće izvršavati na silosima, kao što je to slučaj kod zrna [8]. Klijent je uglavnom web servis, koji može biti napisan u proizvoljnoj tehnologiji, ali tako da je sposoban da se konektuje na Orlenasov servis tj. silose, kako bi bio u mogućnosti da ostvaruje komunikaciju sa željenim zrnima.

4.1. Inicijalizacija klijenta

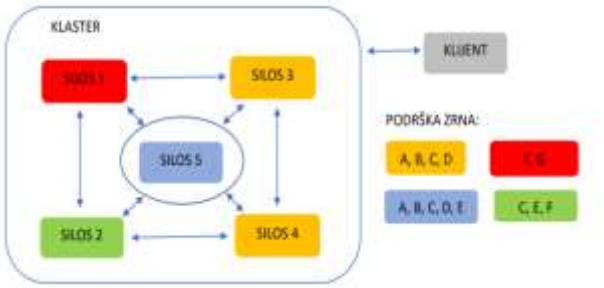
Kao što je prethodno spomenuto, neophodno je ostvariti komunikaciju između klijentskog i serverskog dela, kako bi klijent bio u stanju da obavlja interakciju sa potrebnim zrnima sistema. Konfiguracija klijenta se vrši pomoću

objekta ClientConfiguratioin, koji sadrži mnoštvo podešavanja, koji omogućavaju pravilnu konfiguraciju klijenta [8].

Prilikom razvoja aplikacije, smatraće se da će se aplikacija izvršavati na toj mašini pa će zbog toga koristiti localhost za pokretanje silosa. Nakon toga, potrebno je izgraditi *IClusterClient* objekat pomoću prethodno kreirane konfiguracije i klase *ClientBuilder* [8]. Kada je sve prethodno odrađeno, potrebno je konektovati klijenta sa klasterom, i takođe je potrebno sačekati potvrdu izvršavanja ovog zadatka.

5. KLASTERI I SILOSI

Sva zrna žive i egzistiraju unutar silosa, a mnoštvo silosa, čini jedan klaster. Sami silosi su heterogeni, i mogu podržavati više različitih tipova zrna, ali je onda neophodno naglasiti koja zrna želimo da smeštamo na koji silos [9]. Svi tipovi zrna koja se definišu na silosima, zapravo definišu i koje tipove zrna sam klaster podržava.



Slika 4. *Heterogeni silosi*

Iz priloženog primera može da se vidi da tipovi zrna A i B mogu da se nalaze na silosima 3, 4 i 2. Tipovi zrna C mogu da se nađu na silosima 2, 3, 4 i 5. Tipovi zrna D mogu da se nađu na silosima 3, 4 i 5. Tipovi zrna E mogu da se nađu na silosima 2 i 5. Tipovi zrna F mogu da se nađu na silosima 1 i 2. Na kraju, tipovi zrna G mogu da se nađu samo na klasteru 1. Ono što je ovde bitno napomenuti, jeste da definicije, koji tipovi zrna smeju da se nalaze na kom silosi, govori samo o njihovom životu, ali to ne znači da silosi ne mogu da dobiju informacije o drugom tipu zrna od drugog silosa, radi izvršavanja određene operacije.

Takođe se vidi da sam klijent nije svestan ove podele, već on izvršava samo komunikaciju sa klasterom, dok se ostale stvari dešavaju u pozadini [9]. Izuzetno je važno znati, da svi silosi treba da referenciraju interfejse svih tipova zrna, ali konkretne implementacije klase tih zrna treba da referenciraju samo silosi, koji su predviđeni da će ih skladištiti [9]. Jedna od ključnih stvari gde mogu nastati greške jesu same implementacije tipova zrna na silosima. Naime, ukoliko dva silosa podržavaju određeni tip zrna, onda implementacija tog tipa zrna na silosima mora biti ista na oba silosa, inače će doći do greške [9]. Ograničenja ovakvog pristupa ogledaju se u tome što u toku klijentovog rada sa klasterom, ukoliko je samo jedan silos bio zadužen za određen tip zrna, korisnik će dobiti OrleansException [9]. Greška takođe može da se desi ukoliko se u toku klijentove komunikacije doda novi silos, koji je zadužen za korisnika od interesa tip zrna, doći će do greške ArgumentException [9].

6. ORLEANSOV TOK PODATAKA

Orleansov tok ili tunel, predstavlja mehanizam za slanje različitih tipova informacija ka različitim tipovima zrna, sa jednom velikom prednošću da omogućava da jedan isti tok obrađuje različite stvari u zavisnosti od potrebe konkretnih zrna, što znači da se ovim drastično može smanjiti sama topologija slanja tokova podataka [10] .Ono što Orleansov tok izdvaja od drugih jeste da ispunjava 4 glavna zahteva postavljena od strane potreba programerske zajednice [10]:

- Fleksibilna logika za obradu toka,
- Podrška za visoko dinamičke topologije,
- Dobra granuliranost toka,
- Distribucija.

Pod fleksibilnošću logike za obradu toka smatra se da se pruži podrška za bilo koji od načina za izražavanje logike obrade toka kao na primer: deklarativni upitnici, funkctionalno programiranje, protok podataka itd. [10]. Podrška visoko dinamičke topologije podrazumeva, da će tok biti sposoban da se dinamički menja i prilagođava potrebama na primer menjanjem logike [10].

Treći zahtev podrazumeva da postoje što manje jedinice kontrole radi boljeg upravljanja [10]. Četvrti zahtev predstavlja zapravo ciljeve svakog distribuiranog sistema a to su: skalabilnost, elastičnost, stabilnost, efikasnost i odzivnost [10]. Ove tokove podataka 7 osobina izdvaja u odnosu na druge [10]:

- Orleansov tok podataka se nikad ne kreira i ne uništava eksplicitno, već on uvek postoji jer je virtuelan, a shodno tome ne može nikada ni da otkaže.
- Ovi tokovi podataka su jedinstveno identifikovani sa kombinacijom GUID-a i proizvoljnog stringa
- Omogućenost razdvajanja samog procesiranja podataka od vremena i mesta, što pruža stabilnost prilikom razmene podataka između vremenski i geografski udaljenih servera
- Orleansovo okruženje za tokove omogućava održavanje velikog broja tokova
- Povezivanje zrna na tokove je dinamičko, što omogućava podršku čestih uključenja i isključenja zrna
- Orleansovo okruženje za tokove vodi računa o životnom veku tokova i njihovoj upotrebi
- Orleansov tok podataka radi uniformno sa svim zrnima i Orleans klijentima.

7. ZAKLJUČAK

Iz svega priloženog, može se doći do zaključka da Microsoft Orleans framework pruža potpuno jedinstveno rešenje za distribuirane aplikacije. Ne samo to, već novi principi, koji su ovom tehnologijom uvedeni pružaju potpuno novi pogled na celokupan razvoj jedne takve aplikacije.

Prilikom samog razvoja ovakve aplikacije, verovatno će trebati vremena prihvatiti ovakav pristup, ali nakon toga postaje potpuno prirodno razmišljanje kroz spektar zrna i silosa.

Kao što su i sami ljudi koji razvijaju ovaj framework primetili, celokupan pogled virtuelnog aktera je neophodno sada na ostale delove sistema primeniti, kako bi se postigle pune performanse ovog modela.

Pored toga što se trenutno radi na distribuiranoj transakciji, geo-distribuciji i indeksiranoj bazi virtuelnih aktera, potrebno je takođe dodatno raditi na sigurnosti i bezbednosti ovakvih podataka, jer je to ključna stvar kod sistema, koji za cilj imaju pravljenje aplikacije sa milionskim korisnicima. Ovo je posebno bitno s obzirom da je ova tehnologija namenjena za cloud platforme, koje su i dalje u razvoju, i još uvek sa dosta propusta.

8. LITERATURA

[1]https://www.ibm.com/support/knowledgecenter/en/SSAL2T_9.1.0/com.ibm.cics.tx.doc/concepts/c_wht_is_distd_comptg.html (pristupljeno u avgustu 2018.)

[2]https://www.techopedia.com/definition/18909/distributed-system (pristupljeno u avgustu 2018.)

[3]https://dotnet.github.io/orleans/Documentation/index.html#sidetoggle (pristupljeno u avgustu 2018.)

[4] George Pallis, "Developing Cloud Services Using the Orleans Virtual Actor Model" septembar-oktobar 2016

[5] Philip A. Bernstein, Sergez Bzkov, Alan Geller, Gabriel Kliot, Jorgen Thelin, "Orleans: Distributed Virtual Actors for Programmability and Scalability", 2014

[6]https://dotnet.github.io/orleans/Documentation/core_concepts/index.html (pristupljeno u septembru 2018.)

[7]https://dotnet.github.io/orleans/Documentation/grains/index.html (pristupljeno u septembru 2018.)

[8]https://dotnet.github.io/orleans/Documentation/clusters_and_clients/index.html (pristupljeno u septembru 2018.)

[9]https://dotnet.github.io/orleans/Documentation/clusters_and_clients/heterogeneous_silos.html (pristupljeno u septembru 2018.)

[10]https://dotnet.github.io/orleans/Documentation/streaming/streams_why.html (pristupljeno u septembru 2018.)

Kratka biografija:

Dušan Jeftić je rođen je 21.06.1994. godine u Novom Sadu. Završio je srednju školu Gimnazija Jovan Jovanović Zmaj u Novom Sadu 2013. godine. Fakultet tehničkih nauka u Novom Sadu je upisao 2013. godine. Ispunio je sve obaveze i položio je sve ispite predviđene studijskim programom, nakon čega je sa Diplomskim radom "Obfuskacija programskog koda i detekcija obfuskovanog malvera" ostvario titulu Diplomiranog inženjera Elektrotehnike i računarstva. Ubrzo posle toga je upisao master studije takođe na Fakultetu tehničkih nauka u Novom Sadu.

.