Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/04BE37Kljajic

POSLOVNI INTEGRACIONI ŠABLONI I NJIHOVA UPOTREBA U APACHE CAMEL OKRUŽENJU

ENTERPRISE INTEGRATION PATTERNS AND THEIR USE WITHIN APACHE CAMEL FRAMEWORK

Marko Kljajić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Rad objašnjava šta predstavlja pojam integracije sistema i koje tipove integracije je moguće primeniti. Najveći deo rada posvećen je razmeni poruka kao jednom od načina integracije.*

Ključne reči: *Integracija sistema, Razmena poruka*

Abstract – *This paper discusses what system integration is and which types of integration exist. The major focus is on integration using messaging systems.*

Keywords: *System integration, Messaging*

1. UVOD

Programska rešenja se retko izvršavaju izolovano, česta potreba da se prilikom izrade nekog poslovnog sistema uspostavi komunikacija sa drugim sistemima radi razmene potrebnih informacija uvodi potrebu za integracijom sistema [1]. Sistemi koji u svom radu vrše komunikaciju sa drugim sistemima kako bi postigli određenu funkcionalnost iz specifikacije samog sistema nazivaju se integracionim sistemima.

-Prilikom njihove izrade nailazi se na nekoliko osnovnih problema kao što su nepouzdane i spore mreže u poređenju sa sistemima koji se izvršavaju na jednom računaru, razlike u operativnim sistemima, programskim jezicima, pa i samoj predstavi podataka, česte promene u sistemima što izaziva promene u svim sistemima koji učestvuju u komunikaciji.

Vremenom su nastajala različita rešenja koja su doprinosila da se osnovni problemi prebrode:

- Prenos fajlova – jedan sistem beleži podatke u fajl, drugi sistem čita zabeležene podatke. Potrebno je postići dogovor oko naziva fajl, njegovom formatu i lokaciji, vremenu kada će biti pisan i čitan i na kraju ko će vršiti uklanjanje istog.
- Deljenje baze podataka – potrebno je postići dogovor oko šeme baze podataka
- Udaljenje procedure – sistem izlaže procedure kojima je moguće pristupiti iz druge aplikacije (*web* servisi)
- Razmena poruka – informacije se dele upotrebom sistema za razmenu poruka navođenjem komunikacionog kanala preko kojeg se vrši prenos. Potrebno je postići dogovor oko naziva komunikacionog kanala i formata poruke.

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

2. INTEGRACIJA RAZMENOM PORUKA

Integracija razmenom poruka je najčešće korišćen tip integracije iz razloga što omogućava objavljivanje poruka i ukoliko drugi sistem nije u mogućnosti da ih čita. Iste bivaju dostupne drugom sistemu kada postane spreman. Komunikacija je nezavisna od platforme i programskog jezika u kom su sistemi pisani, dok asinhrona razmena omogućava da sistem koji proizvodi poruke ne mora čekati na odgovor primaoca, već može neometano nastaviti sa radom i proizvodnjom narednih poruka.

2.1. Komunikacioni kanali

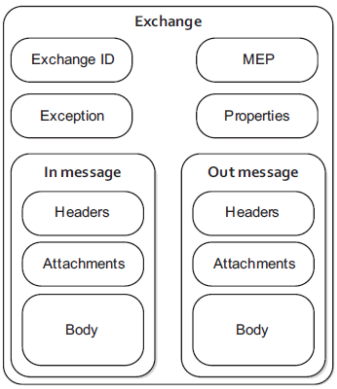
Nakon što se grupi aplikacija stavi na raspolaganje sistem za razmenu poruka, potrebno je definisati komunikacione kanale koji će biti korišćeni za razmenu poruka. Kao što je već rečeno potrebno je postići dogovor o njihovom nazivu. Pored naziva potrebno je definisati i njegov tip.
Komunikacija tačka-tačka omogućava da tačno jedan primalac konzumira jednu poruku iz kanala. Ukoliko je na kanalu prisutno više od jednog primaoca samo jednom od njih će biti dostavljena poruka.
Kod *publish-subscribe* tipa komunikacije naziv kanala predstavlja temu na koju se zainteresovane strane moraju prijaviti radi primanja poruka. Prilikom slanja poruke na određenu temu sve zainteresovane strane dobijaju svoju kopiju poruke.

2.2. Poruka

Poruka predstavlja nedeljiv skup informacija koje je potrebno preneti od pošiljaoca do primaoca. Sastoje se iz dva osnovna dela [2]:

- Zaglavlje - služi da opiše njeno poreklo, tip i destinaciju…
- Telo – sam sadržaj koji se prenosi

Postoje različiti tipovi poruka kao što su komandne poruke koje pošiljalac šalje kada ima potrebu da inicira funkcinalnost na strani primaoca. Poruke sa dokumentom primaocima samo prosleđuju sadržaj. Poruke sa događajem obaveštavaju primaoce o nastalim promenama. Zahtev-odgovor poruke se koriste ukoliko je potrebno da pošiljalac na svaki svoj zahtev dobije određen odgovor. U *Apache Camel* okruženju svi navedeni tipovi poruka mogu se predstaviti instancom *Exchange* klase. Ulazna poruka je obavezna za sve tipove poruka i predstavlja primljenu poruku sa komunikacionog kanala. Polje *MEP* označava o kom tipu komunikacije se radi i sadrži jednu od vrednoti *InOnly* i *InOut*. Izlazna poruka se koristi ukoliko je potrebno poslati odgovor za primljenu poruku i u tom slučtaju *MEP* ima vrednost *InOut*.



Slika 1. Sadržaj poruke u *Apache Camel* okruženju [3]

Identifikator razmene jednoznačno određuje poruku, ukoliko nije unapred naveden on biva automatski generisan. *Exception* polje nosi sa sobom informaciju o grešci ukoliko je do nje došlo prilikom obrade. Zaglavlje poruka predstavlja mapu vrednosti gde je ključ tipa *String*, a sama vrednost je tipa *Object*. Ovo znači da *Camel* ne zahteva određen tip za elemente zaglavlja. *Properties* polje je slično kao i zaglavlje, s tom razlikom što postoji dok god postoji objekat razmene. Poruka takođe može posedovati i priloge koji se uglavnom koriste za *web* servise i email komponente. Telo poruke je tipa *Object*, što korisnicima daje slobodu za tip poruke koji će biti prenet od pošiljaoca ka primaocu.

2.3. Filteri i cevi

U većini poslovnih scenarija jedan događaj može inicirati niz akcija koje je potrebno izvršiti s ciljem uspešne obrade. Korišćenje cevi i filtera omogućava podelu složene poslovne logike na manje celine koje mogu biti ponovo iskorišćene. Celine moraju biti nezavisne i izvršavaju se u redosledu navođenja. Akcije su međusobno povezane cevima.

Svaka akcija mora da implementira isti interfejs čime se garantuje da će čitati poruke sa ulazne cevi, obraditi ih na proizvoljno definisan način i postaviti ih na izlaznu cev.

```
public class MyProcessor implements Processor {
  public void process(Exchange exchange) throws Exception {
    // do something...
  }
}
```

Slika 2. Filter u *Apache Camel* okruženju [4]

Za definisanje akcija u *Apache Camel* okruženju potrebno je implementirati *Process* interfejs koji sadrži jednu metodu čiji je argument referenca na instacu *Exchange* klase i nema povratni tip. Dakle sve izmene se vrši nad prosleđenim objektom razmene. Samim tim što svaka od akcija implementira isti interfejs moguće je zameniti redosled izvršavanja ukoliko je potrebno.

2.4. Rutiranje poruka

Rutiranje poruka se bavi problemom kako proslediti poruku jednom od mogućih filtera u zavisnoti od definisanog uslova. Razlika između filtera i rutera je u tome što se izlaz rutera povezuje na nekoliko izlaznih kanala. Zahvaljujući arhitekturi filtera i cevi komponente koje okružuju ruter poruka nisu o njemu svesne. Sam ruter nema pravo da menja sadržaj poruke, njegov zadatak je samo prosleđivanje poruke na jedan od mogućih izlaza.

Jedan od najčešće korišćenih rutera je ruter na osnovu sadržaja. Ruter prihvata ulaznu poruku na osnovu čijeg sadžaja utvrđuje kojem od postojećih filtera se prosleđuje poruka, odnosno na koju izlaznu cev se ona postavlja.

```
from("seda:a")
    .choice()
        .when(header("foo").isEqualTo("bar"))
            .to("seda:b")
        .when(header("foo").isEqualTo("cheese"))
            .to("seda:c")
        .otherwise()
            .to("seda:d");
```

Slika 3. Rutiranje na osnovu sadržaja poruke [4]

Filter poruka je specijalni tip rutera, čiji zadatak je da eliminiše one poruke za čiji sadržaj sistem nije zainteresovan. Sadrži samo jednu izlaznu cev u koju se smeštaju poruke koje zadovoljavaju kriterijum zadat filterom. Poruke koje nisu zadovoljile kriterijum bivaju odbačene.

```
from("seda:a")
    .filter(header("foo").isEqualTo("bar"))
        .to("seda:b");
```

Slika 4. Filter poruka [4]

Dinamički ruter obezbeđuje nezavisnost samog sistema od uslova rutiranja, ovo znači da kod samog sistema neće u sebi sadržati uslove rutiranja. Za postizanje ove funkcionalnosti potrebno je da ruter poseduje kolekciju koja će sadržati destinacije kao i definisane uslove za rutiranje. Zainteresovani sistemi da bi primali poruke moraju specijalnim komunikacionim kanalom dostaviti uslov rutiranja.

U pojedinim slučajevima postoji potreba da se jedna poruka dostavi do više primaoca. Prvo je potrebno utvrditi listu primaoca kojima se sve poruka prosleđuje. Najčešće se lista utvrđuje dinamički odnosno predstavlja rezultat neke obrade, ali može biti i statička. Drugi korak predstavlja samo postavljanje u komunikacioni kanal svakog od primaoca. Lista primaoca ne menja sadržaj poruke.

```
from("seda:a")
    .multicast().to("seda:b", "seda:c", "seda:d");
```

Slika 5. Statička lista primaoca [4]

Dinamička lista primaoca koristi kolekciju u kojoj se čuvaju informacije o destinaciji i uslovi za primanje poruka. Svaki od zainteresovanih primaoca putem kontrolnog kanala dostavlja željeni uslov.

```
from("direct:a").recipientList(
        header("recipientListHeader").tokenize(","));
```

Slika 6. Dinamička lista primaoca [4]

Mnoge poruke koje prolaze kroz integracione sisteme sačinjene su od nekoliko elemenata. Primer je porudžbenica koja sadrži stavke. Za svaku od stavki potrebno je izvršiti proveru dostupnosti gde same stavke ne moraju biti sadržane unutar istog sistema. Svaki od sistema treba da primi samo onu stavku za koju treba da odradi proveru.

Za obradu ovakvih poruka moguće je razdvojiti glavnu poruku u niz manjih poruka (po stavki porudžbenice). Dakle poruke koje poseduju ponavljajuće elemente mogu biti podeljene u niz manjih poruka gde će svaka poruka sadržati jedan element iz niza.

```
from("seda:a")
    .split(body(String.class).tokenize("\n"))
        .to("seda:b");
```

Slika 7. Podela ulazne poruke [4]

Poruke koje su u određenoj zavisnosti ponekad je potrebno grupisati u jednu poruku koja se zatim postavlja na izlazni kanal. Dakle, od jedne ili više ulaznih poruka potrebno je napraviti jednu izlaznu poruku. Predstavlja suprotnu operaciju razdvajanju. Problem predstavlja utvrđivanje tačnog broja poruka koje je potrebno spojiti. Za uspešno grupisanje poruka koriste se tri osnovne komponente:

- Identifikatora zavisnosti – jedinstveno identifikuje poruke koje su u međusobnoj zavisnosti
- Stanje kompletnosti – kada je potrebno objaviti poruku na osnovu prikupljenih podataka
- Algoritam sakupljanja – opisuje način spajanja poruka

2.6. Transformacija poruka

Poruke koje se razmenjuju između različitih sistema često nisu u željenom formatu za svaki od sistema, prilikom čega je potrebno obezbediti mehanizme transformacije s ciljem prilagođavanja formata.

Zahtevanjem svih sistema, koji komunikaciju obavljaju preko sistema za razmenu poruka, da koriste isti format poruke vrši se „vezivanje“ jednog sistema za drugi čime je jedna od glavnih odlika integracije narušena. Samim tim upotreba istog formata poruke od svih sistema onemogućava svakom od sistema da vrši nezavisne izmene.

U slučaju da sistem koji prima poruke zahteva više informacija nego što se nalazi u samoj poruci moguće je upotrebiti mehanizme za proširivanje sadržaja. Potreban sadržaj moguće je dobaviti izračunavanjem, gde se nedostajeći sadržaj izračunava na osnovu postojećih informacija.

Nedostajeći sadržaj je moguće dobaviti iz okruženja, kao npr. sistemsko vreme. Najčešće korišćeni pristup predstavlja dobavljanje informacija od drugih sistema, npr. baza podataka, datoteka, ili čak manuelni unos.

Potreba da se sadržaj poruke umanji, odnosno da se pojedina polja uklone vrši se filterom sadržaja. Ovaj mehanizam uveden je prevashodno kako bi se što efikasnije iskoristila mreža, kao i iz bezbednosnih razloga gde pojedini sistemi nemaju prava pristupa poverljivim informacijama.

U određenim scenarijima neophodno je samo privremeno ukloniti određeni sadržaj iz poruke. Npr. primljena poruka poseduje polja koja nisu neophodna svim komponentama sistema za uspešnu obradu ali određenom malom broju jesu.

Kako bi se smanjila veličina poruke i povećala efikasnost sistema, pojedina polja se privremeno čuvaju u trajnom skladištu i uklanjaju iz same poruke. Komponenta kojoj je potreban uklonjen sadržaj vraćanje istog vrši proširivanjem sadržaja.

2.7. Krajnje tačke poruka

Ovom grupom šablona rešava se problem povezivanja sistema sa samim komunikacionim kanalima. Sistem za razmenu poruka i sistemi koji komuniciraju razmenom poruka su zasebni sistemi. Jedan sistem rešava domenski problem korisnika, dok drugi brine o komunikacionim kanalima kojima se razmenjuju poruke kao i o samoj razmeni. Iz razloga što su to nezavisni sistemi potrebno je obezbediti način na koji će se uspostaviti njihova komunikacija. Šablon definiše potreban API kako bi same korisničke aplikacije izbegle implementaciju dodatne logike potrebne da se izvrši uspešna konekcija sa sistemom za razmenu poruka.

Ukoliko sistem ima potrebu da kontroliše trenutak u kom će primiti sledeću poruku na raspolaganju mu je primalac baziran na prozivanju. Pošiljalac dostavlja poruke proizvoljnom učestalošću, dok primalac diktira tempo njihog preuzimanja. Prisutnost poruka se utvrđuje proverom komunikacionog kanala, ukoliko se utvrdi prisutnost poruke ista biva prosleđena primaocu. U suprotnom se nastavlja provera. Ovaj proces se naziva prozivanjem. Dakle primalac proziva sledeću poruku kada je spreman da je obradi.

```
Endpoint endpoint = context.getEndpoint("activemq:my.queue");
PollingConsumer consumer = endpoint.createPollingConsumer();
Exchange exchange = consumer.receive();
```

Slika 8. Prozivanje poruka [4]

Za sisteme koji zahtevaju da poruka bude obrađena čim je pre moguće, postoje primaoci bazirani na događajima. Umesto učestale provere na pristigle poruke, komunikacionom kanalu je omogućeno da objavi događaj o pristizanju poruke. Dakle, umesto da primalac vrši konstantu proveru novih poruka, komunikacioni kanal prosleđuje poruku primaocu čim je ona postavljena.

```
from("file://local/router/messages/foo")
        .to("jms:queue:foo");
```

Slika 9. Primaoc baziran na događaju [4]

Sistem iz komunikacionog kanala ponekad ne želi da preuzme sve postavljene poruke, potrebno je omogućiti primaocu način na koji će naglasiti za kakve poruke je zainteresovan. Za jedan komunikacioni kanal koji osluškuje jedan korisnički sistem podrazumevano je preuzimanje svih postavljenih poruka.

Poželjno je da za svaki tip poruke postoji zaseban kanal kojim će se vršiti njen prenos do sistema primaoca. Ovim pristupom se sistema primaoca lišava pisanje logike koja određuje kojim tokom će poruka biti prosleđena. Kada je broj tipova koji se razmenjuju prevelik ovo nije moguće postići.

Dakle, više različitih tipova poruka moguće je postaviti na isti komunikacioni kanal. U ovom slučaju potrebno je koristi šablon otpravnika poruka koji će za svaku pristiglu poruku utvrditi tok kojim će ona biti obrađena.

3. ZAKLJUČAK

Sve veći broj sistema vrši razmenu poruka kao način obavljanja komunikacije, uzrokujući stručna lica da usavršavaju postojeća znanja i šablone kao i definisanje novih. Vremenom se na svakom od šablona vrši manji broj izmena, dok razvoj implementacije ne prestaje kako bi se krajnjim korisnicima što više olakšala njihova upotreba i omogućio što brži razvoj sistema. Integracija sistema je veoma važna zbog postojanja kompleksnih sistema čija se rešenja mogu nalaziti na različitim platformama i u različitim tehnologijama.

Šabloni predstavljaju odličnu pomoć obezbeđujući fleksibilnu strukturu rešenja. Oblast integracije je veoma opširna i ovim radom nisu obuhvaćeni svi njeni aspekti i šabloni.

4. LITERATURA

[1] Gregor Hohpe, Bobby Woolf, "Enterprise Integration Patterns", 2004.

[2] Li Da Xu, *"Enterprise Integration and Information Architecture"*, 2011.

[3] Claus Ibsen, Jonathan Anstey, *"Camel in Action"*, 2011.

[4] http://camel.apache.org/manual/camel-manual-2.24.0.html (pristupljeno u maju 2019.)

Kratka biografija:

Marko Kljajić rođen je u Bačkoj Topoli 1994. god. Odbranio je diplomski rad na Fakultetu tehničkih nauka iz oblasti Elektro-tehnike i računarstva na temu „Implementacija bezbednosnih mehanizama u sistemu platnog prometa" u septembru 2017. godine.

kontakt: kljajic77 @gmail.com