Upravljanje poslovnim procesima (UPP)

Nositelj: izv. prof. dr. sc. Darko Etinger
Asistent: Luka Blašković, mag. inf.

Ustanova: Sveučilište Jurja Dobrile u Puli, Fakultet informatike u Puli

(4) Smjernice u modeliranju, boundary eventi i predlošci tokova rada

Cilj ove skripte pružiti je sveobuhvatan pregled principa i smjernica u modeliranju poslovnih procesa kroz BPMN 2.0 notaciju. Do sad ste naučili osnovnu BPMN notaciju, uključujući osnovne elemente, tokove i događaje. Međutim, osim poznavanja elemenata (sintakse) važno je poslovni proces modelirati na način koji je razumljiv, konzistentan svim dionicima, ali i treba biti precizan s reprezentacijom stvarnog poslovnog procesa. U ovoj skripti ćemo se fokusirati na izvršivost procesa, pravilno imenovanje i upotrebu međudogađaja, modeliranje komunikacije, korištenje boundary eventa te ćemo predstaviti najčešće predloške tokova rada koji se koriste u modeliranju poslovnih procesa BPMN notacijom.

Posljednje ažurirano: 30.12.2025.

Sadržaj

1. Smjernice za modeliranje procesa

Do sad ste naučili da postoje 3 glavna objekata toka u BPMN notaciji, to su:

1. **Događaji** (*eng. Events*)
2. **Aktivnosti** (*eng. Activities*)
3. **Skretnice** (*eng. Gateways*)

Premda su razlike između ovih objekata jasne, ponekad je teško odrediti koji objekt koristiti za modeliranje određene radnje u procesu. Ukratko ćemo ponoviti definicije:

- **Događaji** označavaju određene trenutke u procesu koji označavaju promjenu stanja procesa, poput početka (*start event*), završetka (*end event*) ili ključnih točaka između (*intermediate event*) sljedova aktivnosti. Oni su **pasivni elementi** i ne **podrazumijevaju aktivnost**, već **signaliziraju** da se određeni uvjet ispunio ili stanje procesa promijenilo.
- **Aktivnosti** predstavljaju zadatke ili skup poslovnih zadataka koji se trebaju izvršiti kako bi proces "napredovao". Rekli smo da su za izvođenje aktivnosti potrebni neki **resursi** i **vrijeme**. Radi se o **operativnim elementima procesa**.
- **Skretnice** omogućuju donošenje odluka unutar procesa, usmjeravajući tijek rada prema različitim pravcima na temelju definiranih uvjeta. Smjernice su ključne za **razgranavanje** i **kontrolu toka procesa**, a do sad smo naučili nekoliko vrsta skretnica (ekskluzivne, paralelne, inkluzivne) te varijante grananja i spajanja.

1.1 Aktivnosti vs Događaji

Iako su definicije aktivnosti i događaja poprilično jasne, ponekad možemo biti u nedoumici koji objekt koristiti za modeliranje određenog dijela procesa. Da bismo to razjasnili, praktično je postaviti si nekoliko pitanja koja nam mogu pomoći u odabiru ispravnog objekta.

Primjerice, ako se pitate *"Što se događa u procesu?"*, tada je vjerojatno da je riječ o aktivnosti. S druge strane, ako se pitate *"Kada se nešto događa ili se dogodilo u procesu?"*, tada je vjerojatno da je riječ o događaju.

Za primjer uzmimo proces NARUČIVANJE PROIZVODA, tada bi se **aktivnosti** mogle odnositi na:

- "Unos podataka o kupcu",
- "Unos podataka o proizvodu",
- "Plaćanje",
- "Pakiranje",
- "Dostava"
- "Izrada računa"

Dok bi se **događaji** mogli odnositi na:

- "Primljen zahtjev za narudžbu",
- "Proizvod poslan",
- "Plaćanje izvršeno"

- "Kupac obaviješten putem e-maila"
- "Proizvod isporučen"

Kako ispravno imenovati **aktivnost**? Prema BPMN2.0 smjernicama, aktivnosti bi trebale biti imenovane na način koji jasno i precizno opisuje radnju koja se obavlja. Evo nekoliko smjernica:

- **Glagolska imenica** koja opisuje radnju, npr. "Unos", "Plaćanje", "Pakiranje", "Dostava", "Obrada"
- Naglašen **objekt** na kojeg se aktivnost odnosi, npr. "Unos podataka", "Pakiranje proizvoda", "Pakiranje robe", "Dostava paketa"
- Može biti i **glagol u infinitivu**, npr. "Obavijestiti kupca", "Poslati proizvod", "Pripremiti račun", "Provjeriti zalihe"
- **Nije uobičajeno navoditi subjekt**, budući da je subjekt implicitno jasan iz konteksta modela (polje/staze), npr. "Kupac unosi podatke", "Dostavljač dostavlja paket"
- **Uobičajeno je** koristiti infinitiv, glagolsku imenicu ili iznimno glagol u 3. licu (nikako u 1. i 2. glagolskom licu)
- **Nije uobičajeno** koristiti imperativ, npr. "Unesi podatke", "Pošalji proizvod", "Pripremi račun"
- Uobičajeno je koristiti **aktivni glagolski oblik**

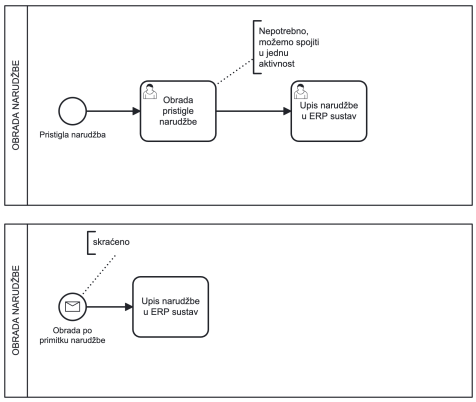
Kako ispravno imenovati **događaj**?

- **Glagolska imenica** koja opisuje **prošlo** (završeno) stanje aktivnosti ili slijeda aktivnosti, npr. "Primljena narudžba", "Plaćanje primljeno", "Proizvod poslan", "Pristigla narudžba"
- **Poželjno je** da sadržava informacije o subjektu i objektu, npr. "Kupac poslao narudžbu", "Dostavljač preuzeo paket", "Proizvod poslan kupcu", "Pristigao email od klijenta"
- Ako se radi o *timer eventu*, tada se koristi **vremenska oznaka**, npr. "Nakon 3 dana", "Svaki ponedjeljak", "Svaki mjesec", "Prošlo 5 minuta" itd.
- Radnja ne smije biti **u infinitivu** (kao kod aktivnosti), već se radi o **završenoj radnji** koja se dogodila u prošlosti, npr. "Proizvod poslan", "Plaćanje primljeno", "Narudžba zaprimljena"
- **Međudogađaje** je moguće imenovati i u **futuru**, npr. "Po primitku zahtjeva", "Kada stigne odgovor", "Jednom kad se dogodi..."
- Uobičajeno je koristiti **pasivan glagolski oblik**

> Napomena: Važno je držati se ovih smjernica imenovanja budući da će vam to olakšati pri odabiru ispravnog objekta toka (događaj vs aktivnost), ali će i poboljšati čitljivost i razumljivost modela svim dionicima kada se držite konzistentnog pristupa imenovanju prema BPMN smjernicama.

Vrlo često u modeliranju želimo prikazati srodnu aktivnost odmah nakon događaja, tada je **praktično spojiti aktivnost i događaj u jedan objekt**. Tada možemo kombinirati događaj i aktivnost u događaj s dopunskim atributima (npr. *message start event* ili *timer intermediate catch event*).

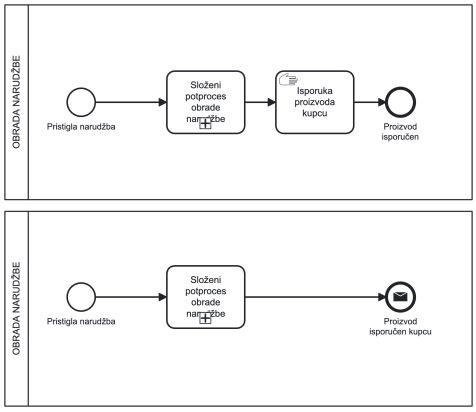
Primjer: start event naziva: "Obrada po primitku narudžbe" može zamijeniti *start event* "Pristigla narudžba" + aktivnost "Obrada narudžbe".



> Slika 1. Ponekad možemo događaj + aktivnost zamijeniti jednim događajem s dopunskim atributima - ali pripazite da ispravno imenujete takav događaj. Kako je novi objekt događaj, nećemo ga imenovati kao aktivnost, već koristimo **pasivan glagolski oblik**.

"Pristigla narudžba" prikazujemo kao *start event* jer se radi o završenoj radnji koja se dogodila u prošlosti. S druge strane, "Upis narudžbe u ERP sustav" prikazujemo kao *user task* jer se radi o radnji koju je potrebno poduzeti kako bi proces "napredovao".

Primjer: End event naziva: "Proizvod isporučen kupcu" može zamijeniti *manual task* "Isporuka proizvoda kupcu" + *end event* "Proizvod isporučen".



> Slika 2. Isto kao kod početnog događaja, možemo i završni događaj + aktivnost zamijeniti jednim završnim događajem s dopunskim atributima - ali pripazite da ispravno imenujete takav događaj. Kako je novi objekt događaj, nećemo ga imenovati kao aktivnost, već koristimo **prošli glagolski oblik**.

1.2 Kako odabrati ispravan međudogađaj?

Međudogađaji (*eng. Intermediate Events*) koriste se za označavanje **ključnih točaka (događaja) između početka i kraja procesa**. Preciznije, koriste se za modeliranje ključnih trenutaka u procesu koji **ne predstavljaju početak ili kraj procesa**, ali svakako mogu promijeniti tijek izvođenja.

Međudogađaje je moguće dodavati na **sekvencijalni tok** između aktivnosti ili unutar aktivnosti kao *boundary event*. Do sad smo ih većinom prikazivali na sekvencijalnom toku između aktivnosti, a u nastavku ćemo pokazati i kako se koriste kao *boundary events*, odnosno događaji vezani uz aktivnost.

Najčešći međudogađaji na sekvencijalnom toku su:

`Intermediate throw event` za označavanje **ključnih točaka** u procesu, tzv. **milestone**

- ✅ Primjeri **ispravnog** imenovanja: "Tijesto se dignulo", "Vrijeme isteklo", "Paket spreman za slanje", "Hrana spremna", "Vremenska prognoza prikladna", "Proizvod na zalihi", "Dokument odobren", "Račun izdan", "Faktura izrađena", "Knjiga pročitana"...
- ❌ Primjeri **neispravnog** imenovanja: "Spremanje tijesta", "Priprema paketa", "Pakiranje robe", "Pohrana bilješki"...

`Message Intermediate Catch event` - definiranje **nespecificiranog čekanja** u procesu, odnosno **čekanje na primitak vanjskog signala/poruke.** Proces nastavlja po primitku signala/poruke.

- ✅ Primjeri **ispravnog** imenovanja: "Primljen signal", "Primljena odbijenica", "Jednom kad pristigne odgovor", "Jednom kad je gotov", "Pristigao email potvrde", "Kada pristigne poruka klijenta", "Čekanje na primitak obavijesti o...", "Po primitku zahtjeva"...
- ❌ Primjeri **neispravnog** imenovanja: "Slanje odgovora korisniku", "Klijent provjerava email...", "Djelatnik obavještava...", "Račun spreman", "E-mail poslan"..., "Čeka email 3 dana"...

`Timer Intermediate Catch event` - označavanje **specificiranog čekanja** u procesu, ili **početak određenog vremenskog razdoblja**

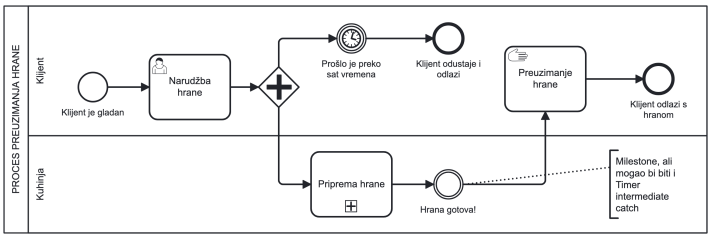
- ✅ Primjeri **ispravnog** imenovanja: "Nakon 3 dana", "Svaki ponedjeljak", "Svaki mjesec", "Prošlo 5 minuta", "Stigao ponedjeljak", "Prošlo je 2 tjedna", "90 minuta", "4 sata", "Pristizanje na red u koloni", "Narudžba došla na red za obradu nakon X vremena"...
- ❌ Primjeri **neispravnog** imenovanja: "Čekaj timer", "Čekanje na odgovor", "Čekanje na primitak poruke", "Čekaj", "Timer", "Nakon nekog vremena...", "Kad prođe vrijeme..."

`Message Intermediate throw event` - koristi se za **signalizaciju drugih aktora** u procesu, ili **okidanje drugih procesa**.

- ✅ Primjeri **ispravnog** imenovanja: "Proces X signaliziran", "Klijent obaviješten", "SMS poslan", "Poslan email kupcu", "Inicijaliziran proces narudžbe", "Račun dostavljen klijentu" itd.
- ❌ Primjeri **neispravnog** imenovanja: "Pošalji email", "Slanje računa kupcu", "Obavijesti klijenta", "Pošalji SMS djelatniku" (greška jer je imenovano kao aktivnost, a ne kao događaj)

> **Zapamtite**: Međudogađaje nastojite imenovati na način da jasno i precizno opisuju **trenutak** ili **stanje** u procesu koji se dogodio ili koji će se dogoditi (koji se očekuje). Koristite pasivan ton, izbjegavajte infinitiv te koristite pasivan glagolski oblik.
> **Izbjegavajte nazivati ove događaje kao čiste radnje tj. aktivnosti.**

Primjer: Klijent naručuje hranu iz restorana preko aplikacije, a potom čeka na dovršetak. Međutim, ako prođe preko sat vremena, klijent odustaje od narudžbe.



> Slika 2. Primjer modeliranog procesa sa sekvencijalnim međudogađajima

U ovom primjeru, koristi se paralelna skretnica (AND) te se proces dalje grana po principu: "ono što se prije dogodi".

- ili će se narudžba dovršiti u roku od jednog sata i klijent će ju preuzeti
- ili će proći preko sat vremena i klijent odustaje

Kako znamo unaprijed vremensko razdoblje, možemo iskoristiti *Timer Intermediate Catch Event* za čekanje tih sat vremena ako se narudžba ne dovrši, dakle imamo specificirano vremensko razdoblje čekanja.

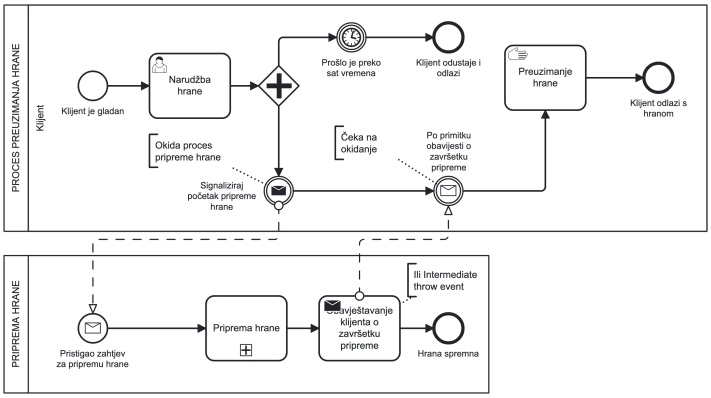
"Priprema hrane" je prikazana sklopljenim potprocesom koji traje neko vrijeme, a jednom kad je hrana spremna, ovisno o kontekstu, možemo definirati *milestone* (ključnu točku) "Hrana spremna" ili "Hrana gotova!" kao *Intermediate Throw Event*. Međutim, u ovom slučaju je moguće istu stvar prikazati i *Timer Intermediate Catch Event* budući da se radi o vremenskom razdoblju potrebnom za pripremu hrane gdje nam je procijenjeno vrijeme pripreme hrane unaprijed poznato.

> Napomena: *milestone* i ne mora biti nužno vezan uz vremensko razdoblje, već može označavati **ključnu točku u procesu**, npr. "Vremenska prognoza prikladna" → okini neki drugi proces ili nastavi slijed aktivnosti.

Vanjske procese uobičajeno je okinuti pomoću *Send Task* aktivnosti ili *Message Intermediate Throw Event* međudogađaja.

Komunikaciju između događaja prikazujemo kroz informacijski tok (*Message Flow*) isprekidanom strelicom, koji predstavlja samo **vezu informativnog karaktera** (ne utječe na sekvencijalni tijek procesa već samo pruža informaciju o komunikaciji između dva objekta).

> Napomena: Prema BPMN2.0 specifikaciji, informacijski tok nije nužno navoditi između dva objekta, ali je poželjno radi jasnije komunikacije i razumijevanja modela.



> Slika 3. Primjer modeliranog procesa u dva polja. Paralelnim grananjem se istovremeno signalizira početak procesa PRIPREMA HRANE, te započinje *timer* od sat vremena nakon čega klijent odustaje od narudžbe.

Na primjeru iznad, prikladno je koristiti *Send Task* aktivnost za signalizaciju aktivacije međudogađaja "Primanje obavijesti o završetku pripreme" budući da se radi o slanju nekog oblika poruke. Da to nije slučaj, prikladnije bi bilo koristiti generalni *Intermediate Throw Event* te definirati *milestone*, npr. "Hrana spremna" što opet implicira okidanje drugog procesa.

> Zapamtite: *Send Task*/*Message Intermediate throw event* implicira slanje poruke (npr. email, SMS, *realtime push* notifikacija, itd.) dok *Intermediate Throw Event* može signalizirati i bez slanja poruke (npr. okidanje nekog drugog procesa drugim mehanizmom ili računalnim protokolom, ali i *offline* komunikacijom poput fizičke dostave dokumenta).

Česta nedoumica: Kada koristiti Send/Receive Task, a kada Message Intermediate Throw/Catch Event?

Kao što smo već spomenuli, *Send Task* i *Receive Task* aktivnosti koriste se za **slanje i primanje poruka** (npr. email, SMS, push notifikacija, itd.) između različitih dionika ili sustava. Ove aktivnosti su **operativne radnje** koje zahtijevaju vrijeme i resurse za izvršenje.

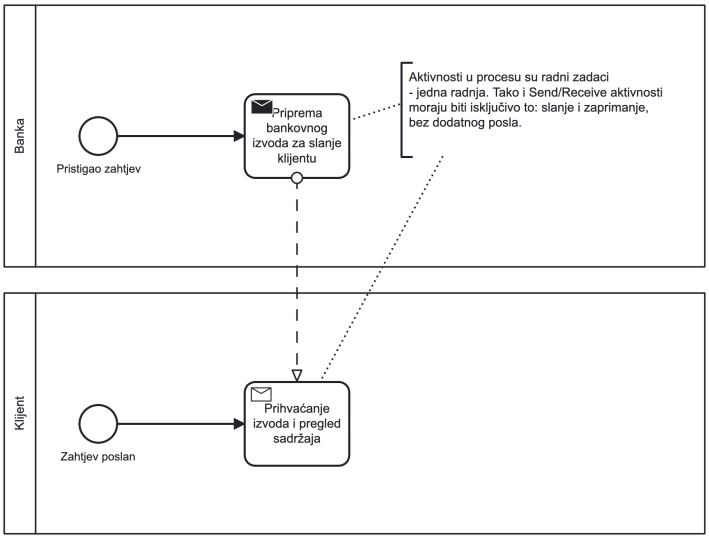
S druge strane, *Message Intermediate Throw Event* i *Message Intermediate Catch Event* su **događaji** koji označavaju **trenutke u procesu** kada se poruka šalje ili prima. Ovi događaji su **pasivni elementi** koji ne zahtijevaju vrijeme ili resurse za izvršenje, već samo signaliziraju da se određeni uvjet ispunio ili stanje procesa promijenilo.

Kod modeliranja poslovnih procesa, često ćemo doći u situaciju gdje možemo koristiti oba pristupa za prikazivanje komunikacije između procesa ili dionika. Ipak, prema BPMN smjernicama, postoje određene situacije kada je prikladnije koristiti jedan pristup pored drugog.

U skripti UPP2, smo naveli da želimo koristiti *send/receive* aktivnosti kod modeliranja komunikacija između dva entiteta odnosno polja, npr. između **Kupca** i **Veleprodaje** ili između **Veleprodaje** i **Skladišta**. Pritom je važno kako ćemo imenovati ove aktivnosti. Ukratko, moramo se pridržavati smjernica za imenovanje aktivnosti koje smo ranije naveli.

Primjer: Komunikacija između polja BANKA i KLIJENT gdje klijent šalje zahtjev za bankovni izvod. Banka priprema izvod i šalje ga klijentu koristeći *send* aktivnost **što je u redu**, ali pogreška je u imenovanju te aktivnosti: "Priprema bankovnog izvoda za slanje klijentu". Nije naglašeno da se radi o slanju poruke, već o pripremi izvoda. Osim toga, želimo imenovati aktivnosti na način da budu **elementarne radne jedinice** - prisjetite se prve skripte gdje smo rekli da aktivnosti još nazivamo i **radnim koracima**.

Drugim riječima, aktivnost treba jasno opisivati **što se radi** u toj aktivnosti, a ne kombinirati više radnji u jednoj aktivnosti.

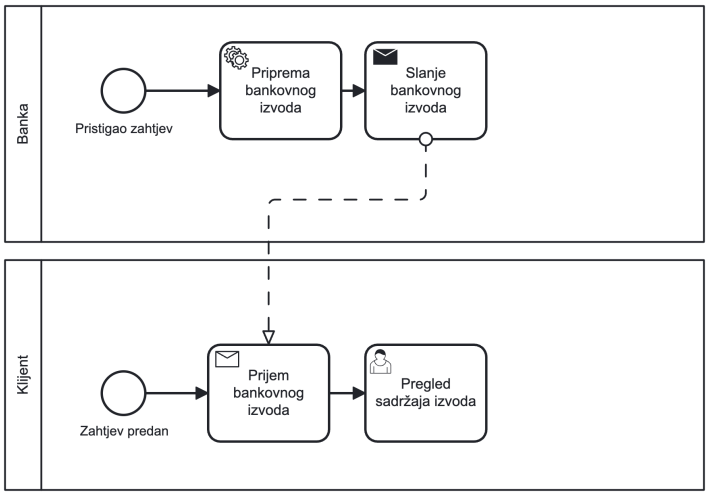


> Slika 4. Primjer **neispravnog imenovanja** *Send/Receive Task* aktivnosti (*send task* ne naglašava da se radi o slanju izvoda, već o pripremi istog). *Receive task* naglašava i zaprimanje i pregled sadržaja, što bi bilo bolje podijeliti u dvije aktivnosti.

- ✅ **Dobar dio**: Korištenje *send/receive* aktivnosti za modeliranje komunikacije između dva polja (BANKA i KLIJENT) je u redu.
- ❌ **Manje dobar dio**: Imenovanje aktivnosti nije u skladu sa smjernicama za imenovanje aktivnosti (aktivnosti nisu elementarne radne jedinice, već kombiniraju više radnji u jednoj aktivnosti). Također, "prihvaćanje bankovnog izvoda" nije baš aktivnost koja zahtijeva vrijeme i resurse, već se radi o trenutku kada klijent zaprima izvod pa je bolje koristiti međudogađaj.

Podijelit ćemo ove složene *send/receive* aktivnosti u dva elementarnija radna zadatka:

1. "Priprema bankovnog izvoda" (*service task*) → "Slanje bankovnog izvoda" (*send task*)
2. "Prijem bankovnog izvoda" (*receive task*) → "Pregled sadržaja izvoda" (*user task*)

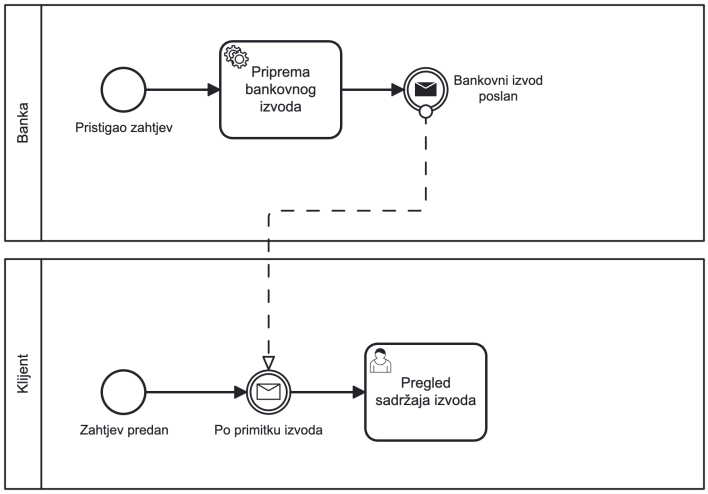


> Slika 5. Ispravno modeliranje komunikacije između dva entiteta koristeći *Send/Receive* aktivnosti s ispravnim imenovanjem i podjelom u elementarnije radne zadatke.

- ✅ **Dobar dio**: Ispravno imenovanje *send/receive* aktivnosti prema smjernicama za imenovanje aktivnosti te podjela na elementarnije radne zadatke.
- ❌ **Manje dobar dio**: Ostaje pitanje da li je prikladno koristiti *receive* aktivnost u ovom kontekstu? Bolje bi bilo koristiti međudogađaj iz ranije navedenog razloga.

Pitanje: Možemo li ovo modelirati koristeći *Message Intermediate Throw/Catch Event* umjesto *Send/Receive Task* aktivnosti? Odgovor je da, i to smo do sada i radili. Međutim, **važno je ispravno imenovati te događaje prema smjernicama za imenovanje događaja** koje smo ranije naveli.

Pretvorit ćemo *send/receive* aktivnosti u *message intermediate throw/catch event* događaje, i **izmijeniti njihove nazive prema smjernicama za imenovanje događaja**.



> Slika 6. Primjer modeliranja komunikacije između dva entiteta koristeći *Message Intermediate Throw/Catch Event* međudogađaje s ispravnim imenovanjem.

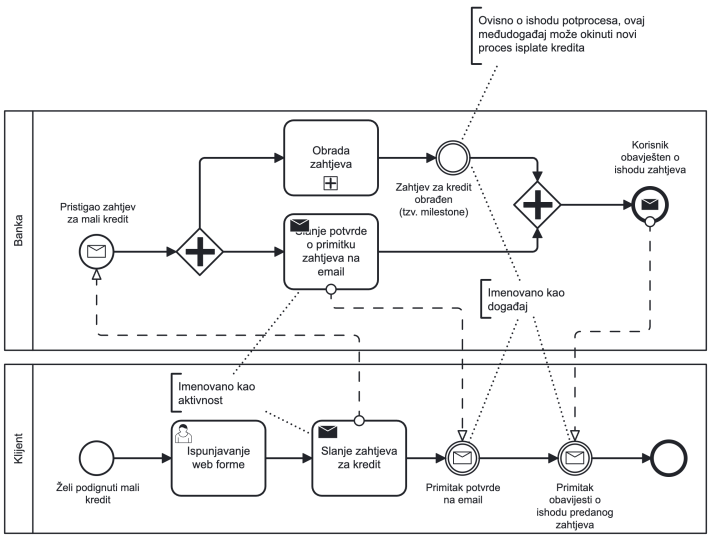
- ✅ **Dobar dio**: Ispravno imenovanje, podjela na elementarnije radne zadatke te korištenje međudogađaja za zaprimanje poruke.
- ❌ **Manje dobar dio**: "Bankovni izvod poslan" je prikladan kada se radi o automatiziranom sustavu koji šalje izvod bez ljudske intervencije, dok bi *send task* "Slanje bankovnog izvoda" bio prikladniji kada djelatnik banke ručno šalje izvod klijentu. **Sve ovisi o kontekstu poslovne logike koja se modelira**.

> Dodatno pojašnjenje: Navedeni primjeri su ekvivalentni i oba su ispravna pristupa su sintaksno točna. Međutim, drugi način je prikladniji u ovoj situaciji budući da "Prijem bankovnog izvoda" nije baš aktivnost koja zahtijeva vrijeme i resurse, već se radi o trenutku kada banka zaprima izvod - iz tog razloga bolje je koristiti međudogađaj. S druge strane, međudogađaj "Bankovni izvod poslan" je prikladan kada se radi o automatiziranom sustavu koji šalje izvod bez ljudske intervencije, dok bi *send task* "Slanje bankovnog izvoda" bio prikladniji kada djelatnik banke ručno šalje izvod klijentu.

Ipak, **u praksi se često kombiniraju oba pristupa** ovisno o kontekstu poslovne logike koja se modelira.

Primjer: Klijent predaje zahtjev za mali kredit u banci. Banka obrađuje zahtjev nekoliko dana, a zatim šalje odluku klijentu. Banka prvo šalje klijentu obavijest da je zahtjev zaprimljen, a nakon obrade (složenog potprocesa) šalje konačnu odluku i popratnu dokumentaciju. Navedeno možemo modelirati kombiniranjem *send/receive* aktivnosti i *message intermediate throw/catch event* međudogađaja kako bi jasno definirali ključne trenutke u ovom procesu.

Iz ovog primjera najvažnije je uočiti **kako su aktivnosti/događaji imenovani** prema smjernicama za imenovanje aktivnosti i događaja te kako smo odabrali ispravan objekt toka (aktivnost vs događaj) ovisno o kontekstu poslovne logike.



> Slika 7. Primjer modeliranja procesa predaje zahtjeva za kredit koristeći kombinaciju *Send/Receive Task* aktivnosti i *Message Intermediate Throw/Catch Event* međudogađaja s ispravnim imenovanjem.

- ✅ **Dobar dio**: Ispravno imenovanje aktivnosti i događaja prema smjernicama za imenovanje aktivnosti i događaja. "Slanje zahtjeva za kredit" je definitivno aktivnost koja zahtijeva vrijeme i resurse pa smo ju modelirali kao *send task*, dok su "Primitak potvrde na email" te "Primitak obavijesti o ishodu predanog zahtjeva" trenuci u procesu koji ne zahtijevaju vrijeme i resurse pa smo ih modelirali kao *message intermediate catch event*. "Zahtjev za kredit obrađen" je generalni intermediate throw event koji predstavlja ključnu točku u procesu (milestone) te ga možemo i ne moramo dodavati - ovisno koliko želimo detaljno modelirati proces.
- ❌ **Manje dobar dio**: "Slanje potvrde o primitku zahtjeva na email" je dosta generalno. Npr, ako modeliramo stazu djelatnika banke - onda ovo može ostati *send task* jer djelatnik zapravo šalje email nakon što je primio zahtjev. Međutim, ako modeliramo stazu IT sustava banke koji automatski šalje email potvrde te ne želimo naglašavati

ljudski faktor, tada bi bilo bolje koristiti *message intermediate throw event* "Potvrda o primitku zahtjeva poslana" budući da se radi o trenutku u procesu kada sustav šalje email, a ne o aktivnosti koja zahtijeva vrijeme i resurse.

> Napomena: Radi se o nijansama koje ovise o kontekstu poslovne logike koja se modelira, međutim kod izrade *business-process-oriented* aplikacija, onda ove nijanse postaje puno važnije zbog prirode same implementacije i egzekucije procesa.

Komunikacija između dionika u procesu

U BPMN modelima često moramo modelirati **komunikaciju između različitih procesa** (npr. jedan proces pokreće/okida drugi) ili inter-procesnu komunikaciju gdje se informacije razmjenjuju **između različitih staza**.

Jedna od najčešćih grešaka u modeliranju procesa odnosi se upravo na neispravno modeliranje komunikacije između dva procesa ili staze.

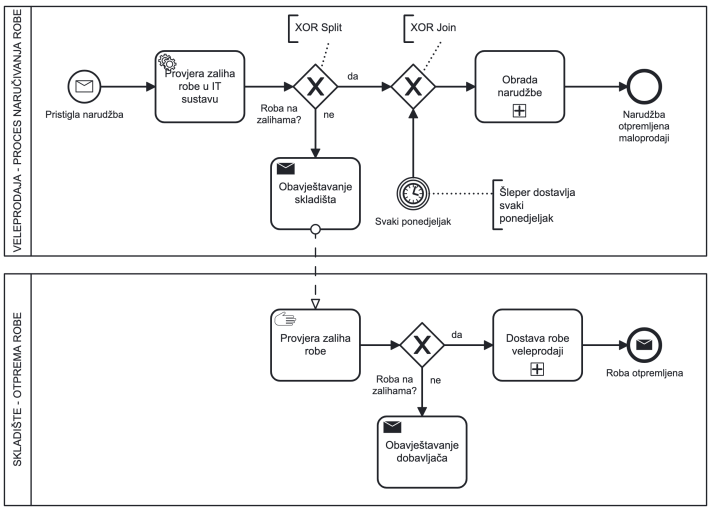
Za modeliranje bilo kojeg oblika poslovne komunikacije, dovoljni su nam sljedeći BPMN elementi:

- Informacijski tok/slijed (*Message Flow*)
- Sekvencijalni tok/slijed (*Sequence Flow*)
- *Message Intermediate Throw Event*
- *Message Intermediate Catch Event*
- *Send Task*
- *Receive Task*

Primjer: Komunikaciju između dionika ćemo objasniti na primjeru **Veleprodaje** i **Skladišta** na procesima naručivanja i otpreme robe.

Veleprodaju predstavljamo kao zasebni proces u zasebnom polju (VELEPRODAJA - PROCES NARUČIVANJA ROBE). Proces započinje kad veleprodaja zaprimi narudžbu. Evidencijom zaliha zaključuju da nedostaje robe pa moraju kontaktirati skladište kako bi provjerili dostupnosti i naručili što nedostaje. Roba se dostavlja veleprodaji svakog ponedjeljka.

Skladište u ovom kontekstu je *outsourcani* partner čiji ćemo proces nazvati OTPREMA ROBE.

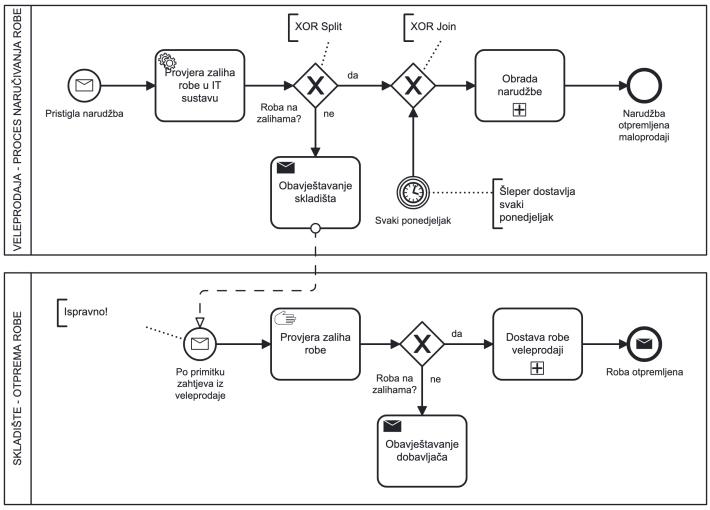


> Slika 8. Primjer modeliranih procesa naručivanja i otpreme robe s pogrešnom komunikacijom između dva polja/procesa. Drugi proces (SKLADIŠTE - OTPREMA ROBE) nema početni događaj.

Idemo identificirati što je dobro, a što pogrešno u ovom modelu.

- ✅ **Dobar dio**: Budući da se radi o slanju poruke skladištu koristimo *send Task* aktivnost za slanje poruke o nedostatku robe na zalihama u veleprodajnom skladištu
- ❌ **Pogreška**: Međutim, SKLADIŠTE - OTPREMA ROBE je **proces za sebe**, definiran u **vlastitom polju**, a nema početni događaj. **Svaki proces (definiran u vlastitom polju) ili potproces mora imati početni** (i završni) događaj.

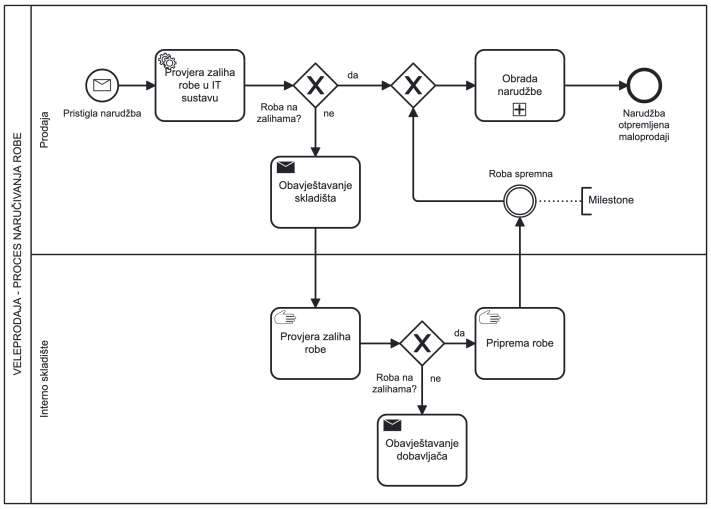
Jednostavno ćemo dodati početni događaj u polje SKLADIŠTE - OTPREMA ROBE.



> Slika 9. Primjer modeliranih procesa naručivanja i otpreme robe s ispravnom komunikacijom između dva polja/procesa. Drugi proces (SKLADIŠTE - OTPREMA ROBE) sada ima početni događaj.

Međutim, što ako se radi o internom skladištu koje je dio iste veleprodaje? U tom slučaju komunikaciju **ne želimo modelirati kao slanje poruke** (prisjetimo se emaila, SMS-a i sl.) Dakle, možemo maknuti *send Task* aktivnost i samo nastaviti sekvencijalni tok prema sljedećoj aktivnosti.

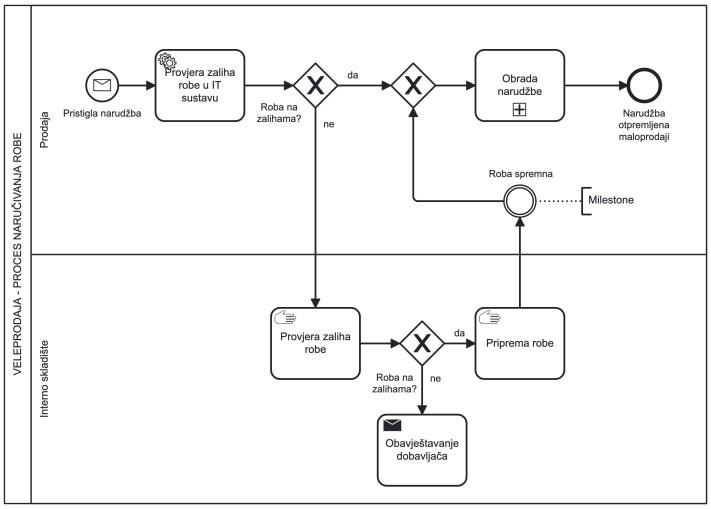
Zašto? Vjerojatno provjeravamo dostupnost robe u internom skladištu putem istog IT sustava, ERP-a ili jednostavno fizički odlazimo i provjeravamo stanje. U tom slučaju, ***send/receive* aktivnosti su nam redundantne** i samo **nepotrebno kompliciraju model**.



Slika 10. Primjer modeliranog procesa VELEPRODAJA - PROCES NARUČIVANJA ROBE s redundantnom komunikacijom unutar istog polja - između dvije staze.

- ✅ **Dobar dio**: Obzirom da se radi o internom skladištu, uklanjamo *Timer Intermediate Catch event* "Svaki ponedjeljak", već na sekvencijalni slijed između pripreme robe i XOR spajanja možemo ubaciti *milestone* događaj "Roba spremna", iako je to više opcionalno, njime bolje pojašnjavamo trenutak nastavka procesa i **naglašavamo da postoji neko vremensko razdoblje potrebno** za pripremu robe.
- ❌ **Pogreška**: Radi se o internom skladištu, ne modeliramo "slanje maila, SMS-a ili sl. poruke" već samo "provjeravamo" dostupnost robe u tom skladištu putem istog IT sustava, ERP-a ili sl. *Send task* nam je ovdje reduntantna aktivnost.

Možemo jednostavno ukloniti *Send Task* aktivnost i samo nastaviti sekvencijalni tok prema sljedećoj aktivnosti "Provjera zaliha robe" u stazi internog skladišta.

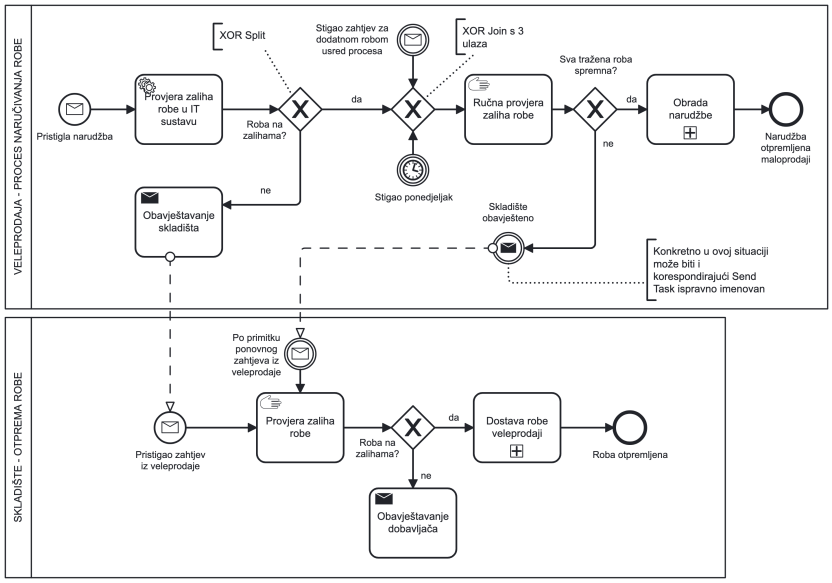


Slika 11. Primjer modeliranog procesa VELEPRODAJA - PROCES NARUČIVANJA ROBE bez redundantne komunikacije unutar istog polja - između dvije staze.

Razmotrimo ponovno scenarij s eksternim skladištem. Što se događa ako se, nakon dolaska šlepera s robom, pojavi potreba za dodatnom narudžbom robe?

Primjer poslovnog slučaja: maloprodaja šalje narudžbu veleprodaji. Veleprodaja naručuje robu od eksternog skladišta svakog ponedjeljka te je proces SKLADIŠTE - OTPREMA ROBE instanciran. Međutim, nakon što šleper stigne s robom, maloprodaja šalje dodatnu narudžbu veleprodaji (npr. zbog neočekivane potražnje). Tada se već aktivni proces VELEPRODAJA - PROCES NARUČIVANJA ROBE mora na neki način **ponovno okinuti** kako bi se obradila nova narudžba, a samim time i proces SKLADIŠTE - OTPREMA ROBE ako je potrebno.

Kako bismo cijeli proces učinili jasnijim i preglednijim, možemo ga modelirati **korištenjem međudogađaja** *Message Intermediate Throw Event* i *Message Intermediate Catch Event* kako bismo bolje naglasili navedene ključne trenutke u procesu:



> Slika 12. Primjer modeliranog procesa VELEPRODAJA - PROCES NARUČIVANJA ROBE s ispravnom komunikacijom između dva polja/procesa koristeći međudogađaje za signalizaciju potrebe za dodatnom narudžbom robe.

- ✅ **Ispravno**: U ovom slučaju, koristimo *Message Intermediate Throw Event* za signalizaciju potrebe za dodatnom narudžbom robe. Kada se dogodi taj međudogađaj, ponovno se stvara instanca procesa VELEPRODAJA - PROCES NARUČIVANJA ROBE. Koristimo korespondirajući *Message Intermediate Catch Event* za hvatanje tog signala i pokretanje procesa SKLADIŠTE - OTPREMA ROBE.
- ✅ **Ispravno**: Definirali smo i *Message Intermediate Catch Event* "Stigao zahtjev za dodatnom robom usred procesa", kako bi jasno definirali trenutak kad se za vrijeme trajanja procesa VELEPRODAJA - PROCES NARUČIVANJA ROBE pojavila potreba za dodatnom narudžbom robe. Potreba se pojavila usred procesa, dok se čeka na dolazak šlepera s prvotnom narudžbom - XOR merge će nastaviti prvim dolaznim signalom.

Nekoliko korisnih smjernica za ispravno modeliranje poslovne komunikacije

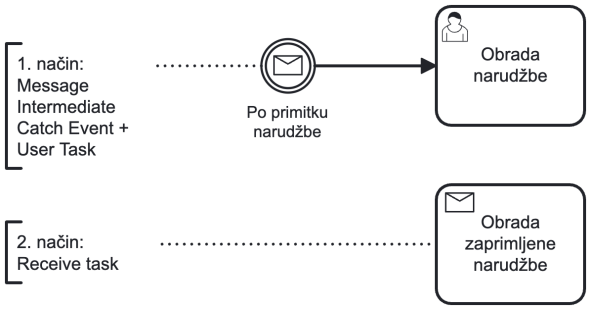
1. Koristite *Send Task* za eksplicitno slanje poruka dionicima procesa (npr. email, SMS, itd.) koje zahtijevaju vrijeme i resurse za izvršenje (npr. ljudske intervencije).
2. Koristite *Receive Task* za obradu poruka koje dolaze od dionika procesa (*Receive Task* je ustvari ekvivalentna kratica za Message Intermediate Catch event + Task`).
3. Koristite *Message Intermediate Throw Event* za signalizaciju drugim procesima da nešto

učine (okidanje drugih procesa) slanjem poruke/signala/notifikacije.

4. Koristite *Message Intermediate Catch Event* za hvatanje signala ili poruka od drugih procesa (okidanje procesa ili nastavak procesa) bez potrebe za ljudskom intervencijom.
5. Koristite *Intermediate Throw Event* za signalizaciju ključnih točaka u procesu (*milestone*) bez eksplicitnog slanja poruka.
6. Koristite *Timer Intermediate Catch Event* kada proces stagnira na način da čeka specificirano vremensko razdoblje.

Receive Task je nešto rjeđa aktivnost u BPMN modeliranju, međutim ima svoju svrhu. Radi se o kratici (skraćenici) koja kombinira dva BPMN elementa: *Message Intermediate Catch event* i neku popratnu aktivnost - najčešće obradu dobivenog signala/poruke. Dakle, *Receive Task* predstavlja aktivnost koja čeka na primitak poruke ili signala od drugog entiteta prije nego nastavi s obradom te poruke/signala.

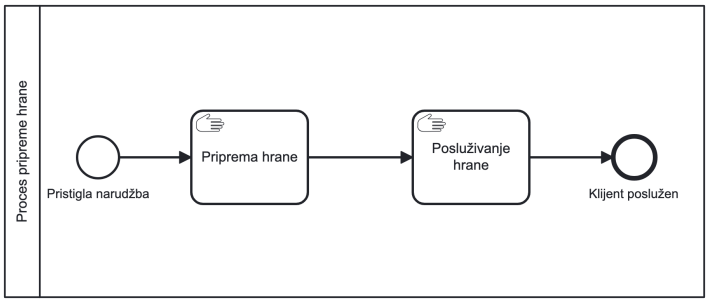
Primjer: "Obrada zaprimljene narudžbe" može biti modelirana kao `Receive Task` koja čeka na primitak narudžbe od kupca (trenutak) i njenu obradu (aktivnost).



> Slika 12. Ponekad je korisno koristiti `Receive Task` za modeliranje aktivnosti koja uključuje primitak poruke ili signala te popratnu aktivnost obrade iste.

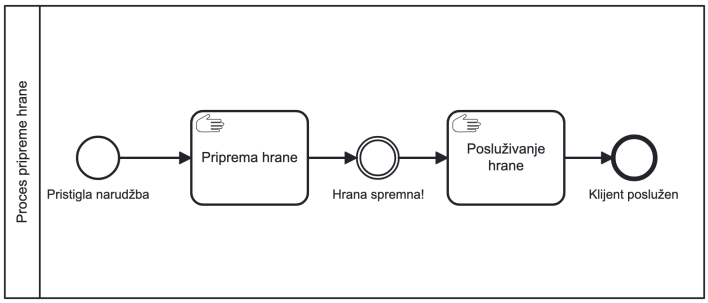
Intermediate Throw Event je korisno koristiti kada želimo **naglasiti ključne točke u procesu**, tzv. *milestone* (npr. "Roba spremna", "Vrijeme isteklo", "Proizvod na zalihi", "Hrana spremna"). Bez obzira, procesni tijek je moguće definirati i bez njih, ali na ovaj način možemo značajno **poboljšati čitljivost i razumljivost procesa**.

Primjer: Proces pripreme hrane u restoranu.



> Slika 13. Primjer modeliranog procesa pripreme hrane u restoranu bez *milestone* događaja.

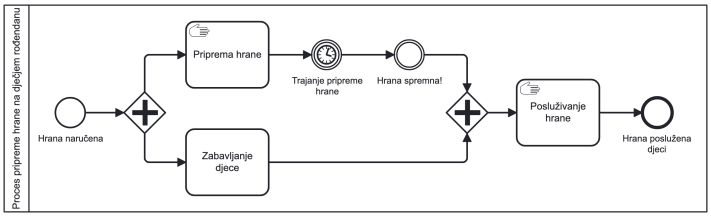
Možemo dodati *Intermediate Throw Event* "Hrana spremna" kako bismo naglasili ključnu točku u ovom procesu. Ovaj događaj ne utječe na sekvencijalni tijek procesa, već samo signalizira da je hrana spremna za posluživanje te na taj način postižemo bolju čitljivost procesa.



> Slika 14. Primjer modeliranog procesa pripreme hrane u restoranu s *milestone* događajem "Hrana spremna".

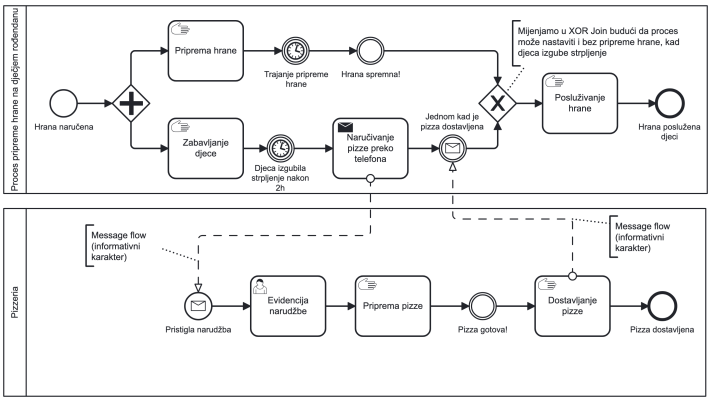
Primjer: Što ako su naši klijenti djeca na nekoj rođendanskoj proslavi? Npr. želimo animirati djecu dok čekaju hranu.
Možemo dodati paralelnu aktivnost gdje zabavljamo djecu dok hrana nije gotova, a samo čekanje na spremanje hrane naglasiti kroz *Timer Intermediate Catch Event* "Trajanje pripreme hrane". Po završetku vremenskog razdoblja, proces se nastavlja.

> Napomena: U nastavku ćemo pokazati kako ovo modelirati bolje koristeći *boundary event*.



> Slika 15. Primjer modeliranog procesa pripreme hrane u restoranu s paralelnom aktivnošću zabavljanja djece.

Primjer: Zakomplicirat ćemo još malo stvari. Što ako nam hrana izgori i nemamo više ideja kako zabavljati djecu (primjerice prođe preko 2 sata)? U tom slučaju, ćemo naručiti pizzu iz obližnje pizzerije. Komunikaciju prema pizzeriji možemo prikazati kroz *send Task* aktivnost koja se izvršava jednom kad se okine *Timer Intermediate Catch Event* - "Djeca izgubila strpljenje nakon 2 sata".



> Slika 16. Primjer modeliranog procesa pripreme hrane u restoranu s paralelnom aktivnošću zabavljanja djece i narudžbom pizze nakon što djeca izgube strpljenje.

1.3 Entiteti na informacijskim (message) tokovima

Uobičajeno je dodati entitete na informacijskim tokovima kako bi se dodatno pojasnila komunikacija između objekata. Primjerice, možemo dodati entitet "Narudžba" na informacijskom toku između *send taska* "Naručivanje pizze preko telefona" i *message start eventa* "Pristigla narudžba" kako bi naglasili da je poruka koja se šalje upravo **narudžba s detaljima o naručenim pizzama**.

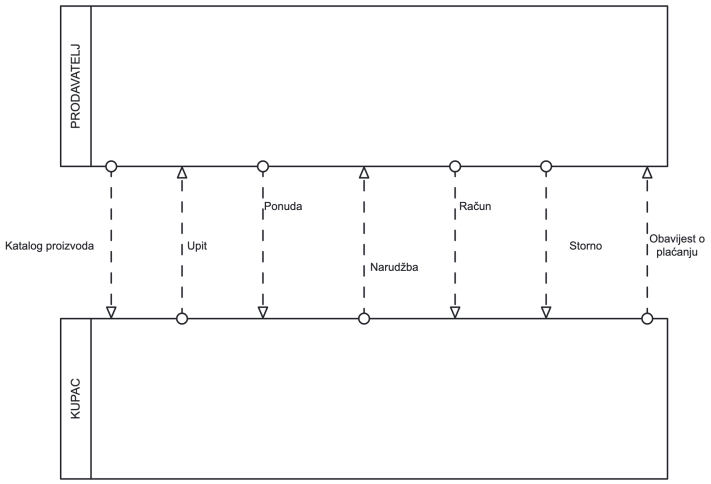
Jednako tako možemo na informacijskom toku između *manual taska* "Dostavljanje pizze" i *Message Intermediate Catch Event*: "Jednom kad je pizza dostavljena", dodati entitet "Naručene pizze i račun" kako bi naglasili da se informacijskim tokom dostavljaju upravo ti entiteti.

> Slika 17. Primjer modeliranog procesa pripreme hrane u restoranu s paralelnom aktivnošću zabavljanja djece i narudžbom pizze nakon što djeca izgube strpljenje, uz dodatne entitete na informacijskim tokovima.

Entiteti na informacijskom toku su korisni jer:

- **pojašnjavaju** što se šalje između objekata
- **jasno definiraju** što se očekuje od poruke/signala
- **poboljšavaju čitljivost** i **razumijevanje** modela procesa

Ilustracija ispod prikazuje komunikaciju između PRODAVATELJA i KUPCA te različite entitete koji se šalju između njih, a koje definiramo na informacijskim tokovima.



> Slika 18. Informacijske tokove možemo dodatno pojasniti definiranjem entiteta koji se šalju između BPMN objekata.

> Napomena: **Entitete možemo dodati i na sekvencijalnim tokovima**, premda je to manje uobičajeno. Entiteti na sekvencijalnim tokovima mogu biti korisni kada želimo naglasiti što se prenosi između aktivnosti, npr. koji su ulazni podaci u određenu aktivnost i/ili koji su izlazni podaci nakon izvršenja aktivnosti. Prisjetite se procesa produljenja registracije motornog vozila iz skripte UPP3 gdje smo dodavali entitete na sekvencijalne tokove kako bismo naglasili koji se dokumenti i/ili podaci prenose između aktivnosti.

2. Boundary event (Međudogađaji na aktivnostima)

U BPMN modeliranju, *boundary event* (međudogađaj na aktivnosti) je poseban tip događaja koji je vezan za određenu aktivnost i koristi se za hvatanje (*eng. catch*) određenih eskalacija, signala, poruka ili vremenskih okidača koji se mogu dogoditi tijekom izvođenja te aktivnosti.

U BPMN literaturi na hrvatskom, *boundary event* se često prevodi kao *međudogađaj na aktivnosti* ili **rubni događaj**. U nastavku ove skripte, koristit ćemo termin *boundary event*.

Boundary eventi su korisni za modeliranje iznimnih situacija ili uvjeta koji mogu utjecati na tijek procesa, **a ne zahtijevaju prekid same aktivnosti**.

Boundary eventi se prikazuju krugovima s dvostrukim, koncentričnim kružnicama (identično kao međudogađaji), ali su **priključeni na rub aktivnosti** - zato se nazivaju *boundary event* (međudogađaj na aktivnosti).

`Boundary event` (priključen na aktivnost)

Prema Camunda 8 BPMN specifikaciji, postoje više različitih tipova *boundary eventa*, od kojih se većina koristi prilikom razvoja procesne aplikacije. Međutim, u kontekstu modeliranja poslovnih procesa, koristi se samo nekolicina njih.

Postoje 2 glavna tipa *boundary eventa*:

1. **Interrupting boundary event** (prekidajući *boundary* međudogađaj) - kada se dogodi ovaj *boundary* događaj, **aktivnost na koju je vezan se prekida**, a tijek procesa nastavlja se prema izlaznoj putanji definiranoj za taj *boundary* događaj.
2. **Non-interrupting boundary event** (neprekidajući *boundary* međudogađaj) - kada se dogodi ovaj *boundary* događaj, **aktivnost na koju je vezan se ne prekida**, već se paralelno izvršava izlazna putanja definirana za taj *boundary* događaj, dok se glavna aktivnost normalno nastavlja izvoditi.

Prema BPMN specifikaciji, zadani tip *boundary eventa* je **interrupting** (prekidajući), a prikazuje se neisprekidanim kružnicama (kao na slici iznad - jednako međudogađaju). Ako želimo definirati **non-interrupting** (neprekidajući) *boundary event*, tada se koristi isprekidana linija za kružnice.

> Slika 19. Interrupting (prekidajući) *boundary event* prikazan neisprekidanim kružnicama (jednako kao klasičan međudogađaj). Non-interrupting (neprekidajući) *boundary event* prikazan isprekidanim kružnicama.

Za početak, pokazat ćemo kada koristiti *boundary event* na procesu POLAGATI ISPIT koji se sastoji od staza PROFESOR i UČENIK.

Primjer: Učenik polaže ispit koji traje određeno vremensko razdoblje (npr. 30 minuta). Slijed aktivnosti u stazi UČENIK uključuje aktivnosti: "Rješavati zadatke" i "Predati završeni ispit" koja završava njegov ispitni proces čak i ako preda zadaću prije isteka vremena. Međutim, ako istekne vrijeme i učenik nije predao zadaću, **aktivnost "Rješavati zadatke" se mora prekinuti**, profesor sakuplja zadaću i ocjenjuje je.

Za boldani dio teksta - prekidanje aktivnosti "Rješavati zadatke" - koristit ćemo *interrupting timer boundary event* vezan za tu aktivnost.

> Slika 20. Primjer modeliranog procesa POLAGATI ISPIT s *interrupting timer boundary eventom* vezanim za aktivnost "Rješavati zadatke".

Do sada bismo ovakve situacije modelirali jednostavnim *timer intermediate catch eventom* koji bismo paralelno granali s aktivnošću "Rješavati zadatke" po principu "što se prije dogodi". Međutim, korištenjem *boundary eventa* jasno naglašavamo da je vremensko ograničenje **izravno povezano s aktivnošću** "Rješavati zadatke" te da se ta aktivnost mora prekinuti kada istekne vrijeme.

> Slika 21. Primjer modeliranog procesa POLAGATI ISPIT s paralelnim *timer intermediate catch eventom* koji nije vezan za aktivnost "Rješavati zadatke". Isto ispravno, ali model je teži za čitanje, a i može doći do zabune jer nije jasno da je vremensko ograničenje izravno povezano s aktivnošću "Rješavati zadatke".

2.1 Vrste interrupting boundary eventa

Slično kao kod standardnih međudogađaja, i *boundary eventi* dolaze u različitim varijantama ovisno o vrsti okidača koji hvataju. Generalni *interrupting boundary event* može se koristiti za hvatanje različitih vrsta eskalacija, signala ili poruka **koje mogu prekinuti aktivnost na koju je vezan**.

U nastavku su prikazane najčešće korištene vrste *interrupting boundary eventa* u kontekstu modeliranja poslovnih procesa:

> `Interrupting Boundary Event` **(prekidajući boundary događaj)** generalni prekidajući međudogađaj koji se može koristiti za hvatanje različitih vrsta eskalacija, signala ili poruka koje mogu prekinuti aktivnost na koju je vezan. Zamislimo ga kao _milestone_ međudogađaj koji se odnosi na aktivnost i kada se dogodi, aktivnost se prekida.

Primjeri imenovanja: "Hrana djelomično gotova", "Dio namirnica nedostaje", "Javljanje osobe s većim prioritetom"...

> `Interrupting Timer Boundary Event` **(prekidajući vremenski boundary događaj)** koristi se za prekidanje aktivnosti nakon određenog vremenskog razdoblja ili u određenom specificiranom vremenskom trenutku. Zamislimo ga kao **alarm koji se postavlja na aktivnost** i kada istekne vrijeme, aktivnost se prekida.

Primjeri imenovanja: "Isteklo vrijeme za rješavanje zadataka", "Rok za dostavu dokumentacije je prošao", "Prošlo je više od 24 sata"...

> `Interrupting Message Boundary Event` **(prekidajući obavještajni boundary event)** koristi se za hvatanje dolaznih poruka koje mogu prekinuti aktivnost. Zamislimo ga kao **signal koji prekida aktivnost** kada stigne određena poruka.

Primjeri imenovanja: "Otkaži/storniraj narudžbu", "Hitno zaustavi izvođenje", "Stigao zahtjev za promjenu", "primljeni podaci vanjske analize ..." itd.

> `Interrupting Escalation Boundary Event` **(prekidajući eskalacijski boundary event)** koristi se za hvatanje eskalacija koje mogu prekinuti aktivnost. Eskalacije se obično koriste za označavanje situacija koje zahtijevaju hitnu pažnju ili intervenciju. Zamislimo ga kao **signal upozorenja koji prekida aktivnost** kada se dogodi eskalacija. Ovo je podskup generalnog _interrupting boundary eventa_.

Primjeri imenovanja: "Rizik premašio dopušteni prag", "Kritični resurs nedostupan", "Rezultat nije vjerodostojan", "Projekt kasni"...

> `Interrupting Error Boundary Event` **(prekidajući boundary event greške)** koristi se za hvatanje grešaka koje mogu prekinuti aktivnost. Zamislimo ga kao **signal greške koji prekida aktivnost** kada se dogodi greška. Ovo je podskup generalnog _interrupting boundary eventa_.

Primjeri imenovanja: "Greška u sustavu", "Neuspjela validacija podataka", "Pogrešan unos korisnika"

> Napomena: Postoji još vrsta *interrupting boundary eventa* (npr. *cancel boundary event*, *compensation boundary event*, *signal boundary event*), ali se oni rjeđe koriste u kontekstu modeliranja poslovnih procesa, pa ih nećemo detaljnije razmatrati u ovoj skripti.

Timer i *Message* varijante interrupting *boundary eventa* su nam poznatije budući da se ponašaju jednako kao i njihovi ekvivalenti među standardnim međudogađajima na sekvencijalnom slijedu, međutim sada su vezani izravno na aktivnost i mogu je **prekinuti usred njezina izvođenja**.

Escalation boundary event koristi se za hvatanje raznih eskalacija koje mogu prekinuti aktivnosti.

Primjer: Na modeliranom procesu "Organizacija koncerta u Pulskoj Areni" u rješenjima iz vježbi UPP3, upotrijebljen je *escalation boundary event* vezan za aktivnost "Monitoring prodaje ulaznica" koji hvata eskalaciju "Slaba prodaja!". Međutim, nismo detaljnije razmatrali njegovo značenje. Sada znamo da se radi o *interrupting escalation boundary eventu* koji bi ustvari prekinuo navedenu aktivnost kada se dogodi eskalacija, pa je u tom slučaju točnije koristiti varijantu *non-interrupting escalation boundary event*. Više u nastavku...

U istom poslovnom procesu korišten je i *interrupting timer boundary event* na aktivnosti "Prodaja ulaznica preko weba" koji hvata eskalaciju "Sve do dana koncerta ili rasprodanosti ulaznica!". Ovdje je ispravno korišten *interrupting timer boundary event* jer kada se dogodi eskalacija (pristiže dan koncerta), aktivnost se mora prekinuti. Međutim, za rasprodanost ulaznica je bolje koristiti *interrupting boundary event*.

> Slika 22. Isječak iz procesa "Organizacija koncerta u Pulskoj Areni" s korištenim *boundary eventima*.

- ✅ **Ispravno**: Korišten je *interrupting timer boundary event* na aktivnosti "Prodaja ulaznica preko weba" koji hvata eskalaciju "Sve do dana koncerta ili rasprodanosti ulaznica!". Kada se dogodi eskalacija (pristiže dan koncerta), aktivnost se mora prekinuti.
- ❌ **Manje ispravno**: Korišten je *interrupting escalation boundary event* na aktivnosti "Monitoring prodaje ulaznica" koji hvata eskalaciju "Slaba prodaja!". Bolje bi bilo koristiti varijantu *non-interrupting escalation boundary event* jer se radi o eskalaciji koja ne zahtijeva prekid aktivnosti, već samo dodatnu pažnju.
- ❌ **Manje ispravno**: Za eskalaciju "rasprodanosti ulaznica" bolje je koristiti *interrupting boundary event* umjesto *interrupting timer boundary event* jer rasprodanost nije vremensko ograničenje, već stanje koje može nastupiti u bilo kojem trenutku.

Dakle, moguće je dodati i dva ili više *boundary eventa* na istu aktivnost, ovisno o potrebama modeliranja poslovnog procesa.

> Slika 23. Isječak iz procesa "Organizacija koncerta u Pulskoj Areni" s dva *interrupting boundary eventa* na aktivnosti "Prodaja ulaznica preko weba".

Pogledat ćemo još jedan jednostavni primjer eskalacijskog *boundary eventa*.

> *Primjer:* Poslovni proces KUPITI POVLAŠTENU PUTNU KARTU modelira proces prodaje povlaštenih putnih karata u obalnom brodskom prijevozu, što je povlastica stanovnika naših otoka. Pravo na povlašten prijevoz ostvaruje se prilikom kupnje putne karte, a dokazuje se pokazom elektroničke otočne iskaznice (eOtls) koju je izdalo ovlašteno tijelo državne uprave. Radi brže kupnje karata putnik daje eOtls službeniku pri ulasku na brod, koji očitava na čitaču beskontaktnih "pametnih" kartica. Očitavanjem se automatski poziva web servis koji na osnovi upita u središnju bazu podataka vraća službeniku

> podatak možel li se prema toj eOtls izdati povlaštena putna karta. Službenik na ulazu u brod identificira putnika prema slici na eOtls-u pa može uočiti ako je eventualno riječ o zloupotrebi prava (npr. ako kartu kupuje osoba koja nije nedvojbeno slična licu s fotografije na eOtls-u), ali nema vremena za dodatnu verifikaciju. On će putniku izdati povlaštenu putnu kartu (jer nije ovlašten da mu uskrati pravo na putovanje), ali će zadržati eOtls i poslati je kontroloru, koji će provesti dodatnu identifikaciju te eventualno oduzeti povlaštenu putnu kartu ako se utvrdi zloupotreba.

> Slika 24. Primjer modeliranog procesa KUPITI POVLAŠTENU PUTNU KARTU s *non-interrupting escalation boundary eventom* vezanim za aktivnost "Identificirati putnika prema eOtls-u" kojim se hvata eskalacija "blagajnik izrazio sumnju" i pokreće se slijed aktivnosti kontrolora za dodatnu verifikaciju.

Pokazat ćemo još jedan primjera korištenja interrupting message *boundary eventa* i interrupting error *boundary eventa* na već poznatom procesu OBRADA NARUDŽBE.

> *Primjer:* Poslovni proces OBRADA NARUDŽBE je tipičan proces promptnog ispunjavanja narudžbe za isporuku proizvoda iz područnog skladišta. Osnovni je tok procesa vrlo jednostavan: nakon što kupac pošalje narudžbu, skladište je prima, priprema robu za isporuku i šalje je kupcu. Međutim, tijekom procesa mogu se pojaviti različite iznimne situacije koje zahtijevaju posebnu obradu. Na primjer, kupac može poslati zahtjev za otkazivanjem/storniranjem narudžbe prije nego što je roba isporučena. U tom slučaju, skladište mora prekinuti proces obrade narudžbe i izvršiti otkazivanje. Također, može doći do greške u sustavu tijekom zaprimanja narudžbe ili utvrđivanja uplate, što također zahtijeva prekid procesa i pokretanje postupka za rješavanje greške.

U ovom primjeru koristimo tri *interrupting boundary eventa*:

1. **Interrupting Error Boundary Event** vezan za aktivnost "Upisivanje narudžbe u ERP" - hvata grešku "Greška u sustavu prilikom upisa narudžbe" i pokreće slijed aktivnosti za rješavanje greške.
2. **Interrupting Escalation Boundary Event** vezan za aktivnost "Provjeriti zalihe robe" unutar potprocesa skladišta PAKIRANJE I OTPREMA ROBE - hvata eskalaciju "Nema robe!" te završava potproces bez otpremanja robe.
3. **Interrupting Message Boundary Event** vezan za **potproces** skladišta PAKIRANJE I OTPREMA ROBE - hvata poruku koja dolazi od kupca "zahtjev za otkazivanje/storniranje narudžbe" i prekida potproces te pokreće slijed aktivnosti za otkazivanje narudžbe.

> **Važno je napomenuti**: *Boundary evente* možemo koristiti i na potprocesima budući da su oni ništa drugo nego složene aktivnosti koje sadrže vlastiti tok procesa unutar sebe, a koji ima smisla modelirati za naš poslovni proces. Eskalacijom ovakvog događaja, prekidamo cijeli potproces, što je često korisno u praksi. Dodatno, unutar aktivnosti samog potprocesa možemo koristiti boundary evente na pojedinačnim aktivnostima unutar potprocesa, čime se ne prekida cijeli potproces, već samo pojedinačne aktivnosti unutar njega (npr. nema robe unutar aktivnosti "Provjeriti zalihe robe").

> Slika 25. Primjer modeliranog procesa OBRADA NARUDŽBE s korištenim različitim *interrupting boundary eventima* na aktivnostima, ali i na potprocesu.

2.2 Vrste non-interrupting boundary eventa

Slično kao kod interrupting *boundary eventa*, i *non-interrupting boundary eventi* dolaze u različitim varijantama ovisno o vrsti okidača koji hvataju. Generalni *non-interrupting boundary event* može se koristiti za hvatanje različitih vrsta eskalacija, signala ili poruka koje **ne prekidaju aktivnost na koju je vezan**.

Non-interrupting varijante **prikazujemo isprekidanim kružnicama**.

Pokazat ćemo samo najčešće korištene vrste *non-interrupting boundary eventa* u kontekstu modeliranja poslovnih procesa:

> `Non-Interrupting Message Boundary Event` **(neprekidajući obavještajni boundary događaj)** koristi se za hvatanje dolaznih poruka koje ne prekidaju aktivnost. Zamislimo ga kao **signal koji paralelno pokreće dodatnu obradu** kada stigne određena poruka, dok se glavna aktivnost normalno nastavlja izvoditi.

Primjeri imenovanja: "Obavijest o promjeni zahtjeva", "Dodatni podaci primljeni", "Po primitku zahtjeva s izmjenama", "Stigla dopunska dokumentacija", "Primljen upit o stanju narudžbe"...

> `Non-Interrupting Timer Boundary Event` **(neprekidajući vremenski boundary događaj)** koristi se za pokretanje dodatne obrade nakon određenog vremenskog razdoblja ili u određenom specificiranom vremenskom trenutku, bez prekidanja aktivnosti. Zamislimo ga kao **alarm koji paralelno pokreće dodatnu obradu** kada istekne vrijeme, dok se glavna aktivnost normalno nastavlja izvoditi.

Primjeri imenovanja: "Podsjetnik nakon 24 sata", "Provjera statusa nakon tjedan dana", "Tri dana poslije primitka narudžbe"...

> `Non-Interrupting Escalation Boundary Event` **(neprekidajući eskalacijski boundary događaj)** koristi se za hvatanje eskalacija koje ne prekidaju aktivnost. Eskalacije se obično koriste za označavanje situacija koje zahtijevaju hitnu pažnju ili intervenciju. Zamislimo ga kao **signal upozorenja koji paralelno pokreće dodatnu obradu** kada se dogodi eskalacija, dok se glavna aktivnost normalno nastavlja izvoditi.

Primjeri imenovanja: "Utvrđen je određeni rizik", "Sporedni resurs nedostupan", "Dokumentacija kasni", "Projekt kasni", "Evidentirana slaba prodaja ulaznica"...

> Napomena: Postoji još mnogo vrsta *non-interrupting boundary eventa* (npr. *non-interrupting signal boundary event*), ali se oni rjeđe koriste u kontekstu modeliranja poslovnih procesa, pa ih nećemo detaljnije razmatrati u ovoj skripti.

Pogledat ćemo primjer korištenja *non-interrupting escalation boundary eventa* na već poznatom procesu ORGANIZACIJA KONCERTA U PULSKOJ ARENI. Rekli smo da želimo koristiti ovaj tip *boundary eventa* na aktivnosti "Monitoring prodaje ulaznica" jer se radi o eskalaciji koja ne zahtijeva prekid aktivnosti, već samo dodatnu pažnju od strane marketinškog tima.

> Slika 26. Primjer modeliranog procesa ORGANIZACIJA KONCERTA U PULSKOJ ARENI s korištenim *non-interrupting escalation boundary eventom* vezanim za aktivnost "Monitoring prodaje ulaznica" kojim se hvata eskalacija "Evidentirana slaba prodaja ulaznica" i pokreće se slijed aktivnosti marketinškog tima za poboljšanje prodaje ulaznica,

> dok se glavna aktivnost normalno nastavlja izvoditi.

Sve preostale *non-interrupting boundary evente* možemo koristiti na isti način, samo moramo paziti da ne prekidaju glavnu aktivnost na koju su vezani.

Primjerice, možemo nadograditi proces OBRADA NARUDŽBE s *non-interrupting message boundary eventom* vezanim za potproces skladišta PAKIRANJE I OTPREMA ROBE koji hvata poruku "upit o statusu narudžbe" od kupca i pokreće slijed aktivnosti za informiranje kupca o statusu narudžbe, dok se glavni potproces normalno nastavlja izvoditi.

> Slika 27. Primjer modeliranog procesa OBRADA NARUDŽBE s korištenim *non-interrupting message boundary eventom* vezanim za potproces skladišta PAKIRANJE I OTPREMA ROBE kojim se hvata poruka "upit o statusu narudžbe" od kupca i pokreće slijed aktivnosti za informiranje kupca o statusu narudžbe, dok se potproces normalno nastavlja izvoditi.

3. Predlošci tokova rada

Poslovni procesi s kojima se susrećemo izgledaju nam međusobno vrlo različiti: čini se da svaki od njih ima svoje specifične ciljeve, da se provodi u drugom okruženju i da raspolaže drugim resursima. Premda je to točno, dublja studija ipak otkriva da u logičkoj strukturi modela procesa ima mnogo više sličnosti nego što se to čini u prvom trenutku. Ta se sličnost može utvrditi na dvjema (možemo reći **makro** i **mikro**) razinama.

Sličnost na razini poslovne domene (tzv. makrorazina)

U dosadašnjim primjerima razmatrali smo modele koji bi se mogli primijeniti u više različitih organizacija. Tako se npr. roba široke potrošnje sa svakog veleprodajnog skladišta distribuira prema modelu koji je sličan onom koji smo spomenuli na početku vježbi (narudžba, otprema, dostava). Iako se detalji mogu razlikovati, osnovni tok poslovnog procesa je isti, odnosno aktivnosti se provode prema općoj shemi: PRIHVATITI NARUDŽBU → PROVJERITI MOGUĆNOST ISPORUKE → IZUZETI ROBU SA SKLADIŠTA → OTPREMITI ROBU KUPCU → IZRADITI RAČUN.

Ako prepoznamo tipske procese u više uspješnih organizacija u određenom poslovnom području, moći ćemo izabrati one koji najbolje odgovaraju našem poslovanju (*eng. best practice*) ili ih optimizirati i prilagoditi svojim specifičnim potrebama. Takva tipizacija procesa vodi nas do tzv. **referentnih poslovnih procesa** (obično ih nude proizvođači sustava ERP).

Sličnost na razini aktivnosti koje čine proces (tzv. mikrorazina)

U dosadašnjim smo primjerima vidjeli da se svaki poslovni proces sastoji od niza objekata toka koji su međusobno povezani slijednim (*eng. sequential flow*) ili informacijskim vezama (*eng. message flow*). Već letimična analiza pokazuje da se u različitim procesima često ponavljaju odnosi između objekata toka, kao na primjer:

- **slijed** (AKTIVNOST A → slijedna veza → AKTIVNOST B → slijedna veza → AKTIVNOST C...)
- **izbor** (AKTIVNOST A nakon čega slijedi AKTIVNOST B ili AKTIVNOST C ili AKTIVNOST D...)
- **paralelno izvođenje** dvaju ili više aktivnosti itd.

Za navedene tipične oblike odnosa između objekata toka uobičajen je naziv **predlošci tokova rada** (*eng. workflow patterns*).

Predložaka za upravljanje tokom rada ima jako puno, a moguće ih je podijeliti u nekoliko kategorija. U nastavku će, kroz potpoglavlja, biti prikazani neki od najčešće korištenih predložaka tokova rada.

3.1 Osnovni predlošci za upravljanje slijedom

U ovoj grupi je ukupno **pet predložaka o upravljanju slijedom izvođenja aktivnosti**. Gotovo sve ste ih već nesvjesno koristili u dosadašnjim primjerima modeliranja procesa. Ovdje ćemo ih još jednom navesti teorijski i ukratko objasniti.

Koristit ćemo sljedeće oznake za predloške:

- `WCP` (*Workflow Control Pattern*) - kratica za definiranje predloška

- `A` - aktivnost
- `P` - polje
- `O` - entitet na informacijskog vezi
- `S` - skretnica

WCP-1 Slijed (eng. Sequence)

Neka aktivnost (npr. `A2`) može započeti ako je završena aktivnost koja joj prethodi (npr. `A1`).

> Slika 28. Primjer predloška WCP-1: Slijed između dvije aktivnosti

Ipak, treba podsjetiti na to kako aktivnosti modelirati kada ih izvode različiti sudionici, u različitim poljima. Koristimo `Message Flow` za komunikaciju između polja te odgovarajuće **međudogađaje**:

> Slika 29. Primjer predloška WCP-1: Slijed između dvije aktivnosti u različitim poljima

WCP-2 Paralelno dijeljenje (eng. Parallel Split)

Nakon neke aktivnosti, proces se dijeli u više paralelnih grana. To znači da nakon završetka `A1` mogu započeti aktivnosti `A2` i `A3` i `A4` te se obavljati istodobno, a iza svake od njih može slijediti neka druga aktivnost.

Mogući početak istovremene aktivnosti ne implicira njihov istovremeni završetak!

Od jedne značke (*eng. token*) koja ulazi u paralelnu skretnicu `S1`, uvijek se stvara (bez provjere uvjeta) onoliko kopija koliko ima izlaznih grana i svaka od tih kopija značke dalje se kreće po jednoj od paralelnih grana.

> Slika 30. Primjer predloška WCP-2: Paralelno dijeljenje

WCP-3 Sinkronizacija (eng. Synchronization)

Neka aktivnost može početi ako su prije nje završene aktivnosti na svim paralelnim granama (mogu biti dvije ili više). To znači da aktivnost `A5` može započeti tek nakon što su završene aktivnosti `A2`, `A3` i `A4`. U **paralelnoj skretnici spajanja S2** (*eng. AND Merge*) sve se ulazne značke uvijek spajaju, bez provjere uvjeta, **u jednu izlaznu**.

> Slika 31. Primjer predloška WCP-3: Sinkronizacija

WCP-4 Ekskluzivni izbor (eng. Exclusive Choice)

Nakon neke aktivnosti proces će se nastaviti **samo u jednoj** od više mogućih grana. To znači da će nakon `A1` biti izvedena aktivnost `A2` ili `A3` ili `A4` (odnosno slijed kojem su one na početku). Značka koja ulazi u ekskluzivnu XOR skretnicu `S1` ne dijeli se, već nastavlja jednim od putova koji udovoljava uvjetu što se ispituje prije te skretnice.

> Slika 32. Primjer predloška WCP-4: Ekskluzivni izbor

WCP-5 Jednostavno spajanje (eng. Simple Merge)

Neka aktivnost može početi čim je izvedena neka od aktivnosti koje su se izvodile u dva ili više paralelnih sljedova. To znači da aktivnost `A5` može započeti kad završe ili `A2` ili `A3` ili `A4` (odnosno slijed kojem su one bile na kraju).. Aktivnost `A5` će pokrenuti ona značka koju je ekskluzivna skretnica `S1` uputila na neki od sljedova, a koje je prošla kroz ekskluzivnu skretnicu spajanja `S2`.

> Slika 33. Primjer predloška WCP-5: Jednostavno spajanje

3.2 Predlošci za grananje, sinkronizaciju i iteraciju

U ovoj grupi su predlošci koji se koriste za grananje i sinkronizaciju toka izvođenja aktivnosti. Uobičajeno se koriste u situacijama kada je potrebno izvršiti nekoliko aktivnosti istovremeno ili kada se proces nastavlja samo ako su završene sve aktivnosti koje su se izvodile u paralelnim granama.

WCP-6 Višestruki izbor (eng. Multiple Choice)

Nakon neke aktivnosti proces se može nastaviti u jednoj, dvjema ili u više mogućih grana, **ali najmanje u jednoj**. To znači da poslije `A1` može biti izvedena bilo koja aktivnost, ili bilo koje dvije aktivnosti ili sve tri aktivnosti od mogućih `A2`, `A3` i `A4`.

> Slika 34. Primjer predloška WCP-6: Višestruki izbor

WCP-7 Strukturno sinkronizirano spajanje (eng. Structured Synchronizing Merge)

Neka aktivnost može početi ako su izvedene sve aktivnosti koje su se izvodile u dvama ili u više paralelnih sljedova stvorenih ranije u procesu. To znači da `A5` može započeti kad je završila jedna ili više aktivnosti od mogućih `A2`, `A3` i `A4` koje su pokrenule kopije značaka stvorene u inkluzivnoj skretnici grananja (`S1`). Drugim riječima, u `S2` se sinkroniziraju (ili spajaju) kopije onih značaka koje su prije toga stvorene u `S1`. Bez obzira na to koliko je kopija značaka ušlo u izlaznu skretnicu `S2`, izaći će samo jedna.

U poslovnom smislu to znači da će se procesna instanca, koja je obrađena u `A1`, moći obraditi u `A5` nakon što je provedena barem jedna ili više aktivnosti iz skupa `A2`, `A3` i `A4`.

> Slika 35. Primjer predloška WCP-7: Strukturno sinkronizirano spajanje

WCP-8 Nesimetrično sinkronizirano spajanje (eng. Acyclic Synchronizing Merge)

Neka aktivnost može početi ako su izvedene sve aktivnosti na dva ili više paralelnih sljedova, stvorenih ranije u procesu na inkluzivnoj skretnici `S1` ali se odluka o tome što treba spajati odnosi na temelju **međudogađaja** koji prethode ekskluzivnoj skretnici spajanja `S2`.

Nesimetrično spajanje riješeno je kombinacijom inkluzivne skretnice `S1` (koja će stvoriti jednu, dvije ili tri značke na bilo kojem od tri slijeda) te uvjetovanih događaja na sva tri slijeda ispred konvergentne ekskluzivne skretnice `S2`. Ti će uvjetovani događaji dopustiti izvođenje aktivnosti "Pokrenuti marketinške akcije" kad završe one od prethodnih aktivnosti koje se moraju uskladiti.

> Slika 36. Primjer predloška WCP-8 na primjeru procesa organizacije marketinške kampanje: Nesimetrično sinkronizirano spajanje

Izvođenje aktivnosti u složenom poslovnom procesu slično je izvođenju procedura u složenom programu. Simboli BPMN-a omogućuju prikaz takvih struktura kao što su `GOTO`, `WHILE...DO`, `REPEAT...UNTIL` u programiranju.

Međutim, u poslovnoj praksi česte su i druge strukture, nepoznate u strukturnom programiranju, koje opisuju ponavljanje odnosno **iteracije** pojedinačne aktivnosti ili grupe aktivnosti.

WCP-9 Proizvoljno ponavljanje (eng. Arbitrary Cycles)

Ovaj predložak opisuje točku u procesu nakon koje se može ponoviti jedna ili više aktivnosti. Općenito, unaprijed se ne zna treba li uopće nešto ponavljati i ako treba - koliko puta, već je to specifično za svaku instancu procesa pa se stoga to naziva još i **nestrukturiranom petljom**.

Tipičan primjer za ovaj predložak može se pronaći u zdravstvu, a prikazan je na sljedećem primjeru:

> Slika 37. Primjer predloška WCP-9: Proizvoljno ponavljanje

Primarna zdravstvena zaštita kod nas funkcionira tako da pacijent najprije odlazi na pregled svom osobnom liječniku. Osobni liječnik nakon pregleda odlučuje koje su dodatne pretrage potrebne te za njih izdaje uputnice. Laboratorij ili specijalist će "Obaviti pretragu i izdati nalaz" te zadržati uputnicu (radi obračuna usluge), a pacijent (ako ima još uputnica) će otići na sljedeću pretragu. Osobni će liječnik "Odrediti način liječenja" na temelju nalaza u provedenim pretragama. Općenito se ne zna koliko laboratorijskih pretraga treba napraviti, već će se napraviti onoliko pretraga koliko je potrebno točno određenom pacijentu i primjerenoj njegovoj bolesti.

Ovo je zanimljiv primjer proizvoljnog ponavljanja gdje XOR skretnica spajanja **prethodi** XOR skretnici grananja.

3.3 Predlošci za okidače

U ovom potpoglavlju prikazat ćemo nekoliko predložaka koji se koriste za modeliranje okidača u poslovnim procesima. **Okidači** su događaji koji pokreću izvođenje procesa, a mogu biti izazvani **vremenski**, **porukom** ili **signalom**.

WCP-10 Prolazni okidač (eng. Transient Trigger)

Predložak opisuje proces u kojem izvođenje jedne aktivnosti ovisi o nekom vanjskom poticaju ili drugom procesu (odnosno, vanjski poticaj "okida" aktivnost).

Okidač koji to omogućuje zovemo prolaznim jer nestaje ako u osnovnom procesu već ne čeka instanca koja bi se mogla pokrenuti. Prolazni okidač zapravo je običan prijamni međudogađaj (npr. `Message Intermediate Catch Event`) koji se koristi za hvatanje signala ili poruka od drugih procesa.

Primjer opisuje rad noćnog čuvara u nadziranom objektu. Čuvar će se poslije dolaska (u 20 sati) smjestiti u kontrolnu sobu i "Uključiti nadzorni sustav" koji se sastoji od kamera i senzora kretanja. Ako senzor registrira pokret u objektu, on će "Poslati signal" u kontrolnu sobu. Ako je čuvar u sobi, on će "Provjeriti ishodište alarma". Ako pak čuvara nema, poslani signal neće biti iskorišten i propast će (zato ga zovemo prolaznim).

> Slika 38. Primjer predloška WCP-10: Prolazni okidač

WCP-11 Stalni okidač (eng. Persistent Trigger)

Izvođenje aktivnosti u ovom predlošku također ovisi o nekom vanjskom poticaju ili drugom poslovnom procesu (vanjski poticaj okida aktivnost). Okidač djeluje stalno i aktivan je sve dok na njega dolaze instance procesa, a modelira se također kao prijamni događaj (npr. `Message Intermediate Catch Event`) u osnovnom procesu.

Ishodište iz kojeg dolazi poticaj i ovdje se modelira kao predajni međudogađaj (npr. `Message Intermediate Throw Event`) koji šalje ciljanu obavijest određenom okidaču.

Stalni okidač djeluje slično kao prolazni, a **razlika je u tome što se vanjski poticaj ne gubi ako u osnovnom procesu trenutno nema instance koja bi na njega čekala**. Sljedeći primjer pokazuje izradu prozora. Prozor se izrađuje u tri faze: a) prozorski okvir od drva ili profilirane plastike, b) izolacijska kutija od dvije ili tri staklene ploče između kojih je inertni plin, a razmak održavaju letvice s brtvom i c) ugradnja izolacijske kutije u pripremljeni prozorski okvir.

Zbog različitih tehnologija u fazama a) i b) ti se poslovi organiziraju u dvije radionice. Prvi ćemo odjel nazvati IZRADA I MONTAŽA PROIZVODA, a drugi je staklarska radionica IZRADA IZOLACIJSKE KUTIJE. Ovdje smatramo da prvi odjel vodi posao (među kojima su kontakti s kupcima), a drugi da je kooperant (*outsourced*).

Budući da prvi odjel primi narudžbu, on će "Izraditi nacrt proizvoda" i kopiju poslati staklarskoj radionici te nastaviti s aktivnošću "Izraditi prozorski okvir". Staklarska će radionica prema nacrtu "Izraditi izolacijsku kutiju" i poslati je vodećem odjelu koji, nakon primitka gotove izolacijske kutije, može "Ugraditi izo-kutiju u okvir". Dakle, prijamni događaj "Primljena izolacijska kutija" **okidač** je za ovu aktivnosti. Upravo se u ovom detalju vidi bitna razlika između prolaznog i stalnog okidača: vanjski poticaj (ovdje je to tok "Gotova izolacija kutija") neće se izgubiti ako u okidaču "Primljena izolacijska kutija" još nema odgovarajuće instance procesa (odnosno gotovoga prozorskog okvira) već će se iskoristiti (ovdje to znači ugraditi) kad naiđe ta instanca (odnosno kad prozorski okvir bude gotov).

Analizom modela može se utvrditi da su u procesu zapravo **dva stalna okidača**.

Prvi smo već naveli i on je modeliran eksplicitno. Međutim, drugi okidač modeliran je implicitno i određen svojstvom prijamne aktivnosti "Izraditi ukupni račun za prozor". U tu aktivnost ulazi poruka (entitet) - "Račun za staklarske radove" iz emitirajućeg međudogađaja "Poslan račun za staklarske radove" i pokreće se ("okida") njezino izvođenje.

> Slika 39. Primjer predloška WCP-11: Stalni okidač

Zadaci za Vježbu 4

ServisPlus d.o.o. - Popravak kućanskih uređaja

Vlasnik apartmana koji iznajmljuje smještaj turistima koristi usluge servisne tvrtke ServisPlus d.o.o. za održavanje i popravak kućanskih uređaja u apartmanima. Proces započinje u trenutku kada vlasnik apartmana prijavi kvar na uređaju (npr. perilica rublja, hladnjak ili klima-uređaj) putem web obrasca ili telefonskim pozivom.

Nakon zaprimanja prijave, djelatnik servisnog centra evidentira zahtjev i provjerava osnovne informacije o apartmanu, vrsti uređaja i prirodi kvara. Ako su podaci nepotpuni, vlasnik apartmana se kontaktira radi dopune informacija. Jednom kad je zahtjev ispravno evidentiran, dodjeljuje se serviser te se s vlasnikom apartmana dogovara termin dolaska, uzimajući u obzir boravak gostiju u apartmanu.

Na dogovoreni dan serviser dolazi u apartman i započinje popravak uređaja. Tijekom izvođenja popravka moguće je da se utvrdi kako je potreban dodatni rezervni dio koji trenutačno nije dostupan, zbog čega se popravak privremeno obustavlja do nabave rezervnog dijela u suradnji s vanjskim dobavljačem. U međuvremenu, gosti u apartmanu mogu izraziti nezadovoljstvo zbog neispravnog uređaja ili ometanja boravka, što vlasnik apartmana komunicira servisnom centru.

Popravak je planiran da traje određeno vrijeme. Ako se pokaže da će popravak potrajati dulje od predviđenog, vlasnik apartmana mora biti pravovremeno obaviješten kako bi mogao reagirati prema gostima (npr. ponuditi alternativno rješenje ili financijsku kompenzaciju). Gosti ili vlasnik apartmana mogu u bilo kojem trenutku poslati upit o statusu popravka, na koji servisni centar odgovara bez prekidanja samog izvođenja servisa.

U iznimnim situacijama, gosti mogu zahtijevati hitno rješavanje problema ili zaprijetiti negativnom recenzijom, što vlasnik apartmana eskalira servisnom centru radi ubrzavanja postupka ili promjene prioriteta. Također, prije dolaska servisera, vlasnik apartmana može otkazati zahtjev ako se kvar riješi na drugi način ili ako gosti naposlijetku napuste apartman.

Nakon što je popravak uspješno završen, serviser potvrđuje izvršenu uslugu, a vlasniku apartmana se automatski dostavlja račun i potvrda o obavljenom popravku. Time se proces završava.