Zavod za telekomunikacije

Poslijediplomski studij

za stjecanje doktorata znanosti

Ak.g. 2009./2010.

Konkurentni sustavi

Modeliranje konkurentnosti u objektno orijentiranim i agentskim sustavima

22.1.2010

Sadržaj predavanja

- Općenito o konkurentnosti, procesima i nitima
- Modeliranje konkurentnosti UML-om
  - Dijagram aktivnosti, slijedni dijagram, predlošci, pregledni interakcijski dijagrami, dijagram stanja
- Konkurentnost u Javi
  - Niti, kritični odsječci, zaključavanje, sinkronizacija, koordinacija
- AUML
  - Konkurente interakcije, dinamika između i unutar agenata
- Agentska platforma JADE
  - Konkurentnost u ponašanjima

Zašto konkurentnost?

- Razlozi:
  - poboljšanje performansi u višeprocesorskim sustavima (paralelizam)
  - povećanje aplikacijske propusnosti (U/I operacije mogu blokirati samo jednu nit)
  - povećanje interaktivnosti u aplikacijama (korisnik može koristiti aplikaciju bez obzira na pozadinske operacije)
- Područja uporabe:
  - mrežno programiranje
  - kompleksni proračuni
  - aplikacije s grafičkim sučeljem
  - aplikacije s intenzivnim U/I operacijama

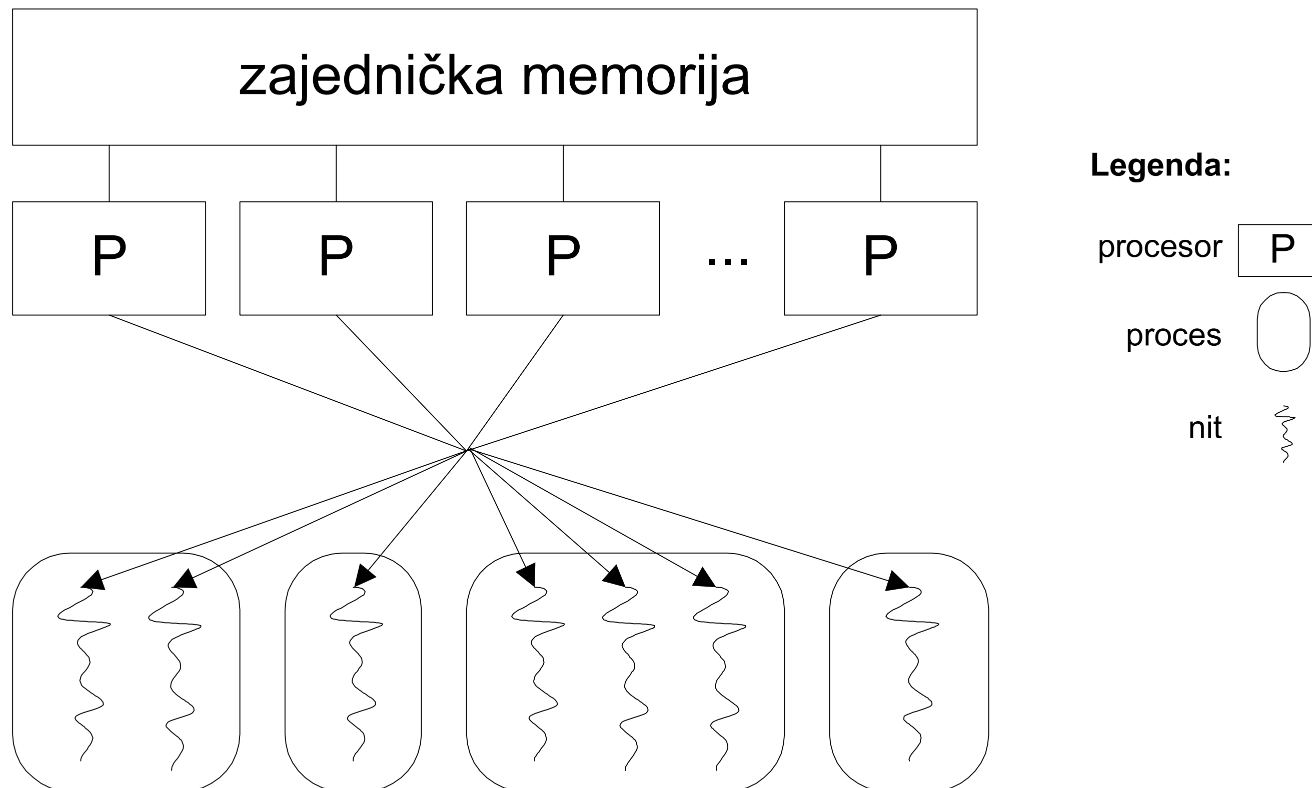
Prednosti konkurentnosti

- reaktivno programiranje
  - odgovor na više istodobnih aktivnosti
    - npr. web preglednik istodobno dohvaća Web stranicu, snima i prikazuje broj primljenih okteta
- raspoloživost
  - objekt služi kao sučelje prema usluzi, na svaki zahtjev stvara se nova konkurentna aktivnost tako da se novi zahtjev prihvaća brže i ne treba čekati da stari završi
- upravljivost
  - aktivnosti mogu biti obustavljene, ponovno pokrenute i zaustavljene od drugih objekata
- aktivni objekti
  - programski objekt modelira stvarni objekt koji ima svoje autonomno ponašanje (pokretanje nove aktivnosti)
- asinkrone poruke
- paralelizam
- zahtijevana konkurentnost (npr. glazba i slika)

Ograničenja konkurentnosti

- sikroniziranost
  - više aktivnosti koje nisu potpuno neovisne, jedna može slati poruke objektima koji su uključeni u druge aktivnosti, potrebna sinkronizacija ("ništa loše neće se nikad dogoditi")
- aktivnost
  - beskonačno čekanje na nastavljanje treba izbjeći ("sve će se nekad dogoditi")
- nedeterminizam
  - svako izvođenje može biti drukčije uz više aktivnosti
- ne treba stvarati konkurentnu aktivnost kad se mora čekati na odgovor
- ne treba stvarati novu konkurentnu aktivnost uz svaki objekt
- konstrukcijsko povećanje (*overhead*)
  - konkurentno povećava zahtjeve za memorijom i vremenom, jednostavne aktivnosti je bolje staviti u objekt i pozivati
- povećanje zbog prebacivanja sadržaja (*context switching*) i raspoređivanja (*scheduling*)
- sinkronizacijsko povećanje

Niti i procesi



- proces – aktivna jedinka koja održava skup resursa
- nit – koristi resurse svojeg procesa

UML

- UML – *Unified Modeling Language*

- grafički jezik za:
  - vizualizaciju,
  - specificiranje,
  - konstrukciju i
  - dokumentiranje

objektno orijentiranih programskih rješenja

Svojstva UML-a

- olakšava komunikaciju sudionika (kupac tj. naručitelj, konzultant, dizajner, programer, …)
- neovisan o programskom jeziku
- neovisan o razvojnom procesu
- posjeduje mehanizme za proširenje i specijalizaciju (prilagodba novim potrebama)
- neiskusni programeri uz UML lakše uče i shvaćaju OO koncepte

UML dijagrami

Sequence Diagrams

Use Case Diagrams

Class Diagrams

Object Diagrams

Collaboration Diagrams

Models

Component Diagrams

Statechart Diagrams

Activity Diagrams

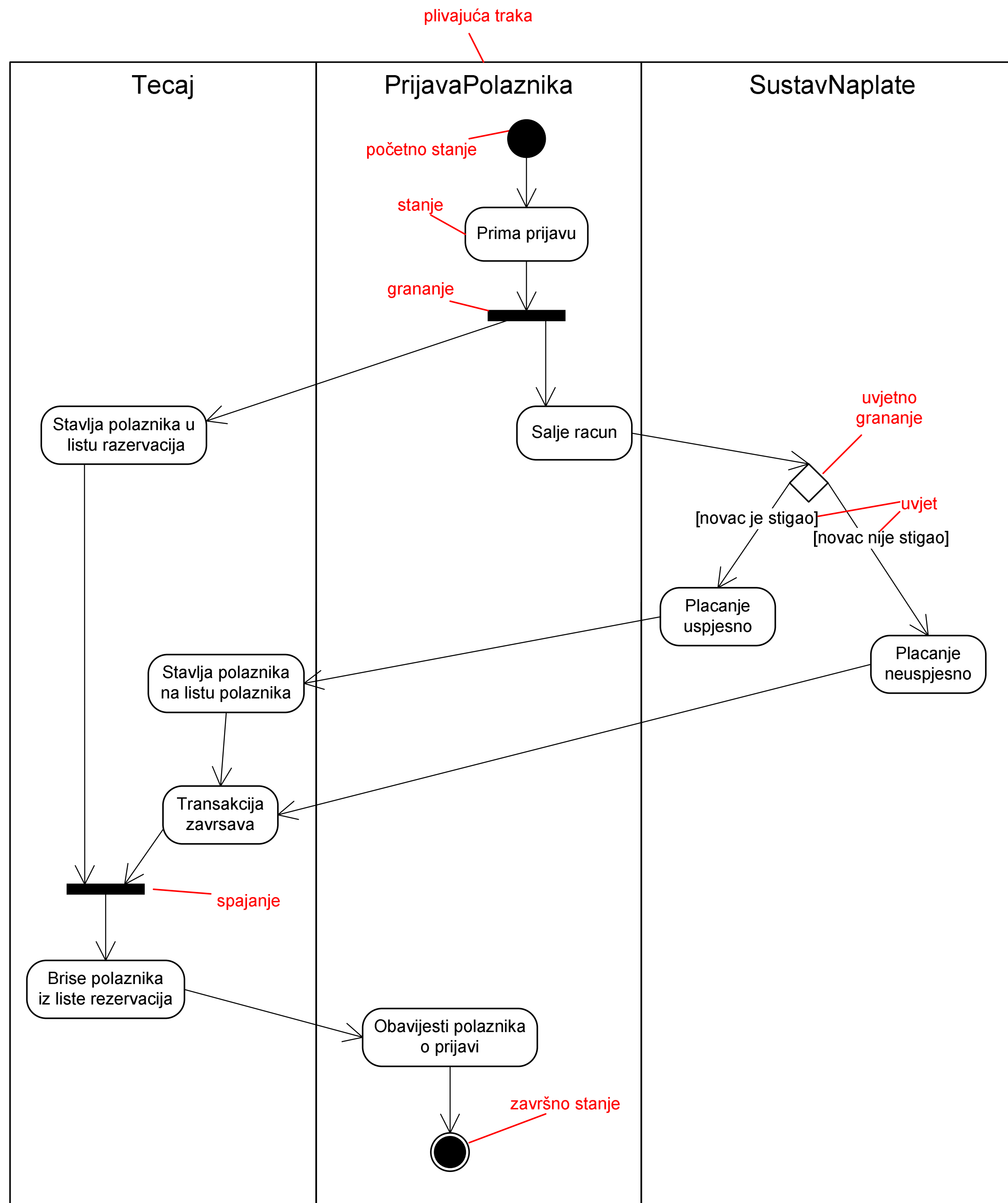
Deployment Diagrams

Dinamički dijagrami

Statički dijagrami

Dijagram aktivnosti (*activity diagram*)

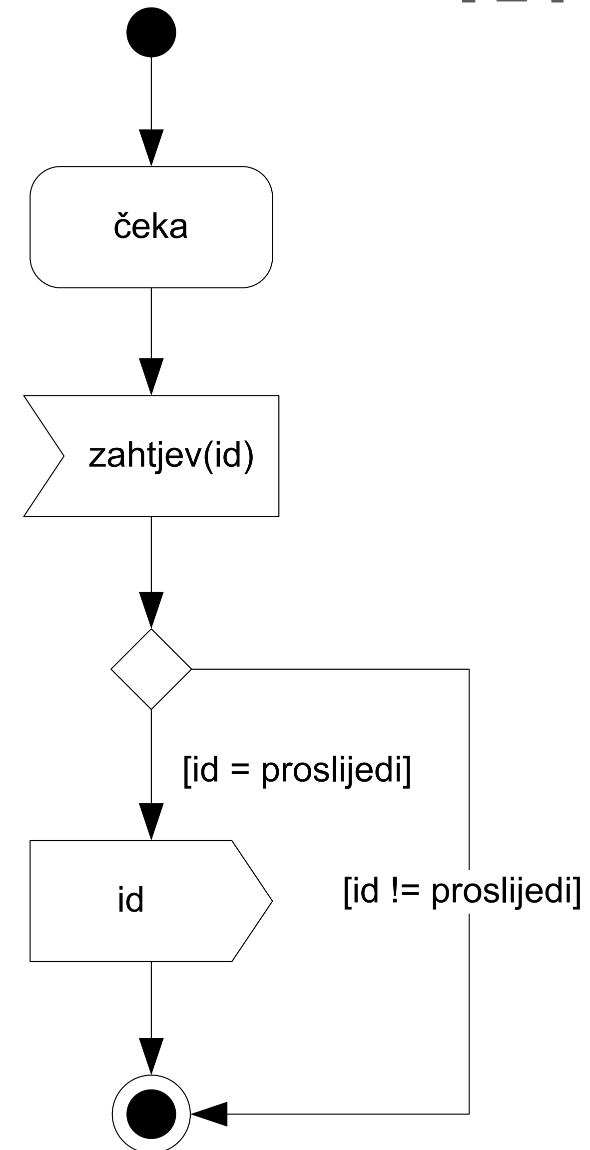
- opisuje redoslijed aktivnosti koje mogu biti slijedne ili paralelne
- nadopunjuje dijagram klasa pokazujući tijek aktivnosti (*workflow*)
- pomaže pri pronalaženju paralelnih aktivnosti i time omogućuje skraćenje vremena procesiranja
- moguće je i pokazati tko izvodi određene akcije pomoću plivajućih traka (*swimlanes*)
- sastoji se od stanja koja se ovdje nazivaju aktivnosti (*activity*), prijelaza, odluka, grananja (*fork*) i spajanja (*join*)



FER

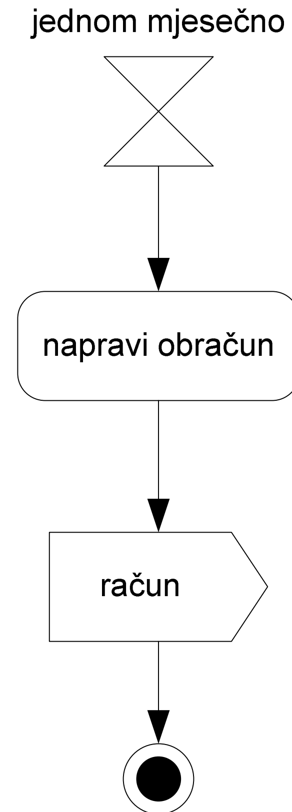
Slanje i primanje poruka u dijagramu aktivnosti

- tijek:
  - prvo se primi zahtjev
  - ona imamo odluku
  - ako je zahtjev za proslijediti onda se on proslijedi
- oznake preuzete iz jezika SDL



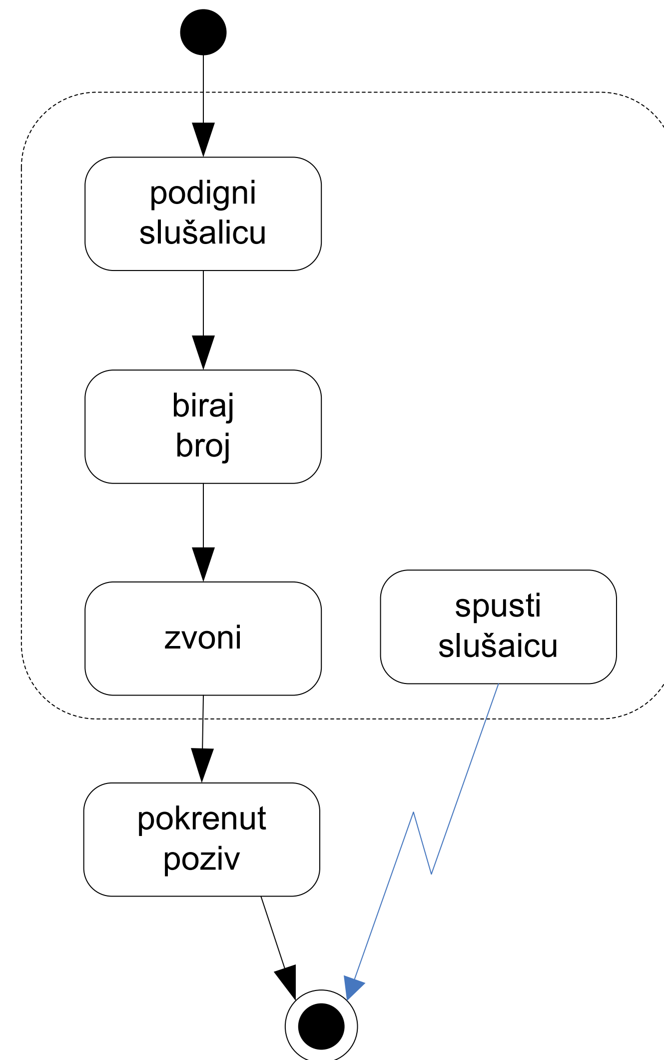
Dijagram aktivnosti s vremenskim elementima

- početni znak predstavlja akciju koja se dogodi u nekom vremenskom trenutku
- vrijeme se opisuje tekstualno uz oznaku
- u vremenski znak se može i ulaziti
  - npr. nakon ulaska se čeka neko vrijeme



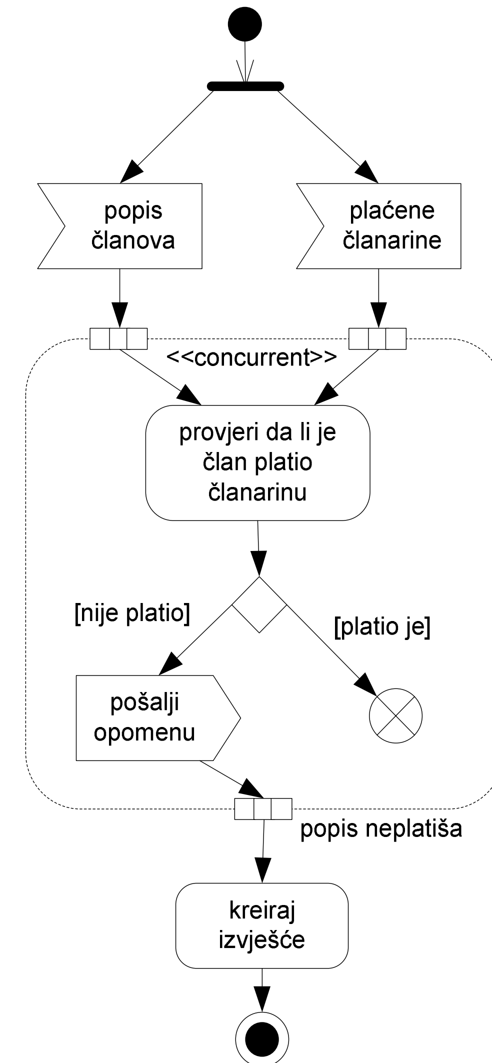
Prekinuto izvođenje u dijagramu aktivnosti

- strelica u obliku groma prikazuje prekid aktivnosti
- isprekidano zaokruženje prikazuje regiju iz koje se izlazi prilikom prekida aktivnosti



Područje ekspanzije u dijagramu aktivnosti

- područje ekspanzije može predstavljati konkurentne aktivnosti koje se izvode
- ulazi i izlazi imaju više podataka koji se mogu konkurentno ili paralelno obrađivati



Slijedni dijagram (*sequence diagram*)

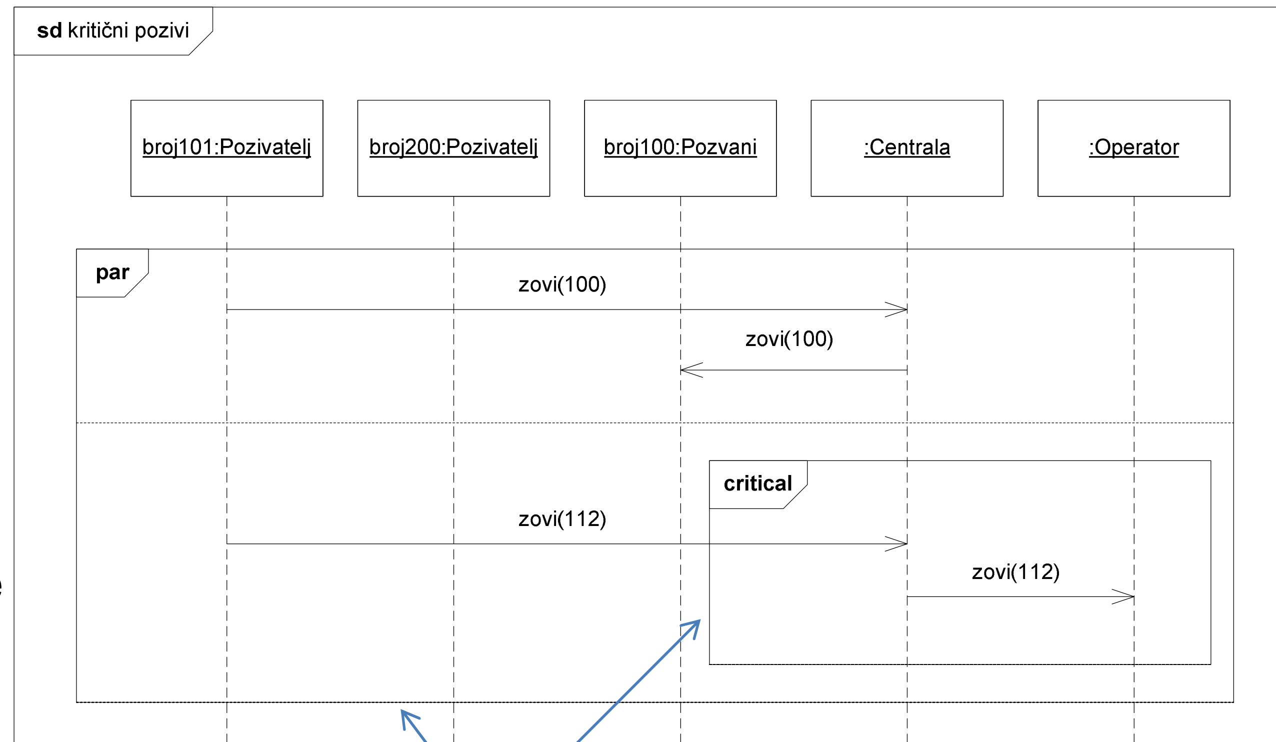
- opisuje dinamičku suradnju između objekata
- prikazuje **vremenski slijed poruka** koje objekti razmjenjuju (najčešće se pokazuje primjer u kojem se vide sve poruke koje objekti razmjenjuju)
- pokazuje vrijeme života objekta (kreiranje i uništavanje)
- objekti se prikazuju u formatu:

```
naziv_objekta:naziv_klase
```

Primjer slijednog dijagrama s aktivnim objektima

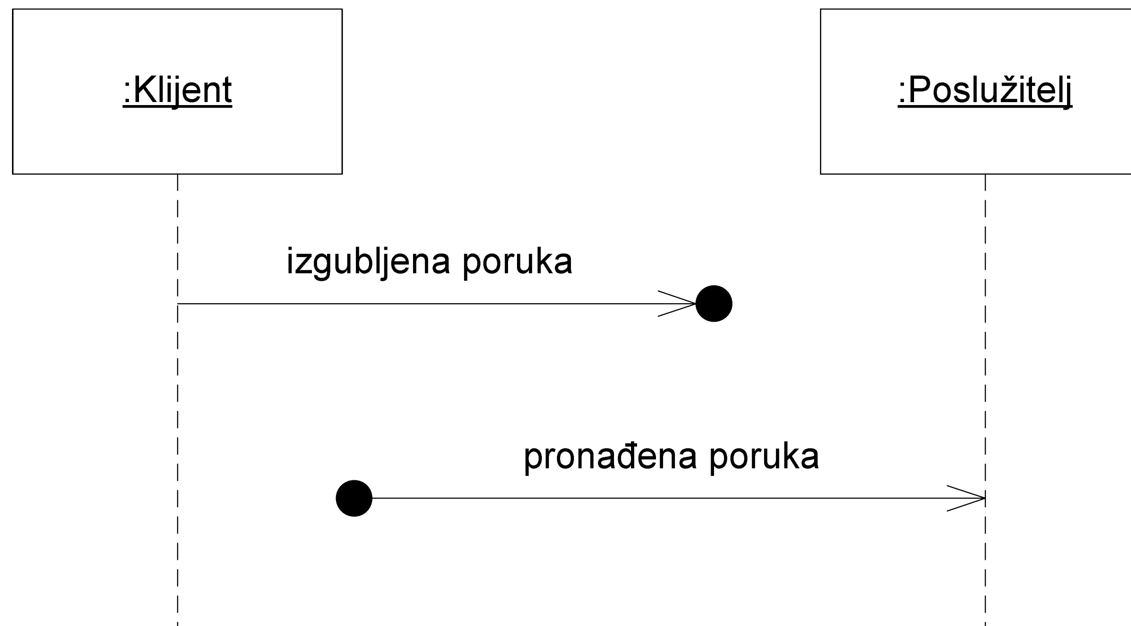
Slijedni dijagram s fragmentima

- fragmenti se u slijednom dijagramu mogu koristiti za različite stvari:
  - **alt** – alternativa (treba koristiti ograničenja)
  - **opt** – opcija (kao if)
  - **break** – kao obrada izvanrednih situacija
  - **par** – paralelno izvođenje
  - **seq** – slaba sekvencijalnost koja se ne mora uvijek poštivati
  - **strict** – jaka sekvencijalnost
  - **critical** – atomično izvođenje
  - **loop** – petlja (potrebno je koristiti ograničenja)

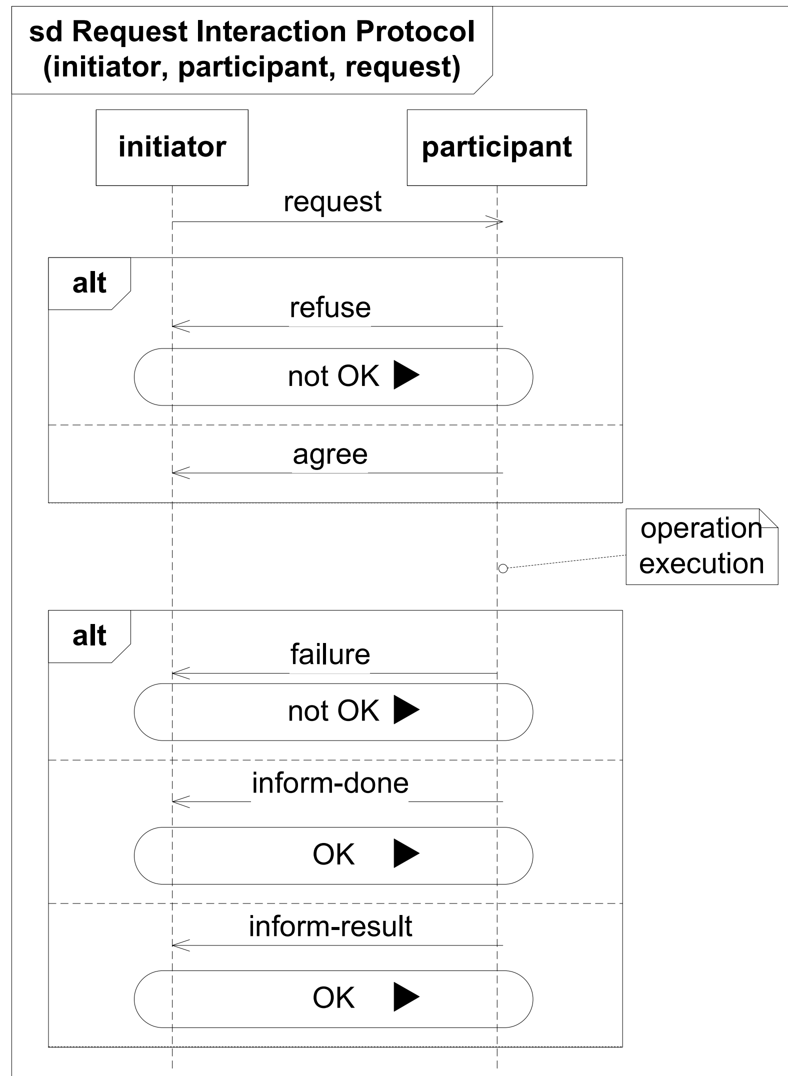


Vremenske oznake u slijednom dijagramu

Izgubljene i nađene poruke u slijednom dijagramu

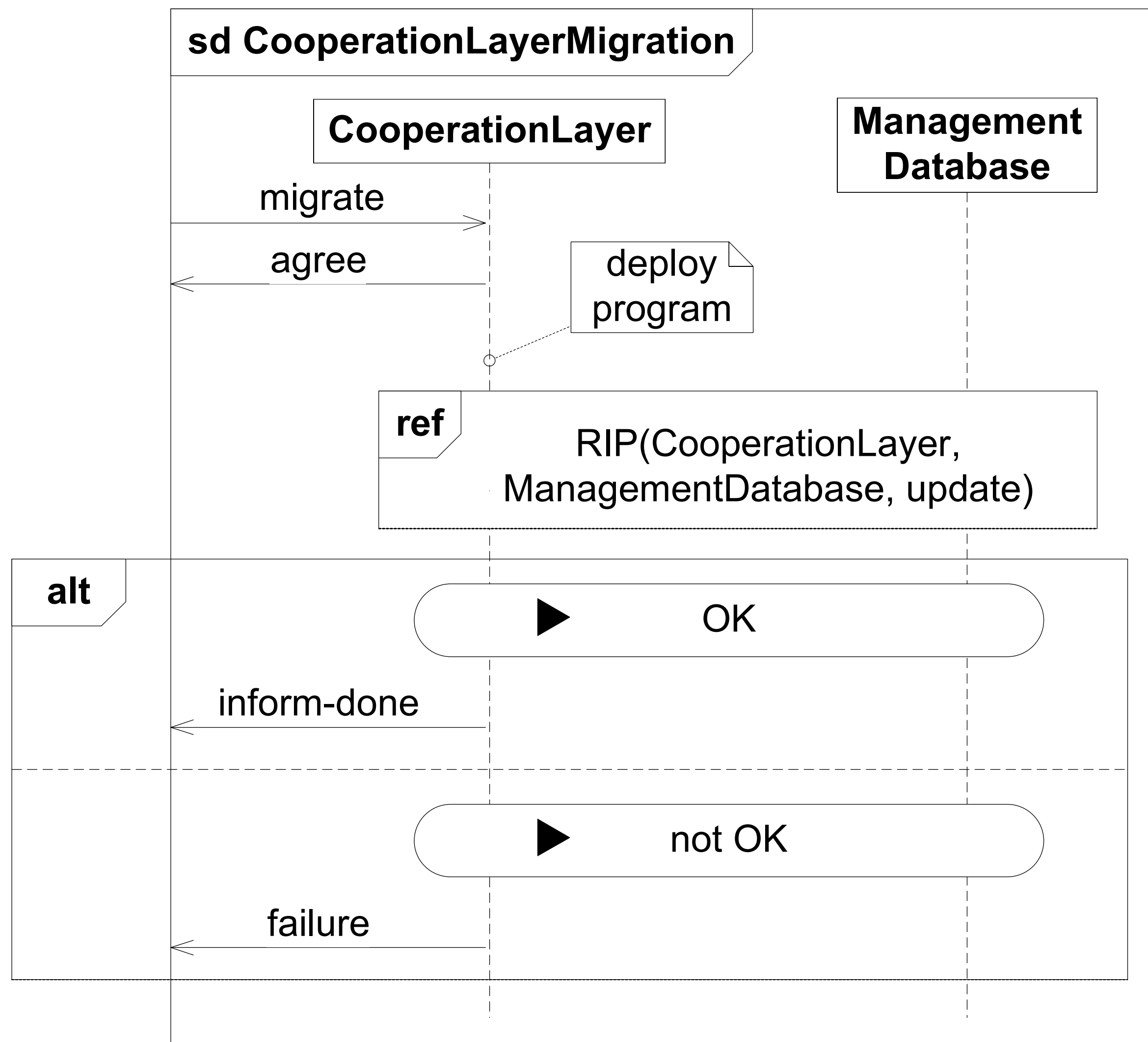


Predlošci (engl. *template*)

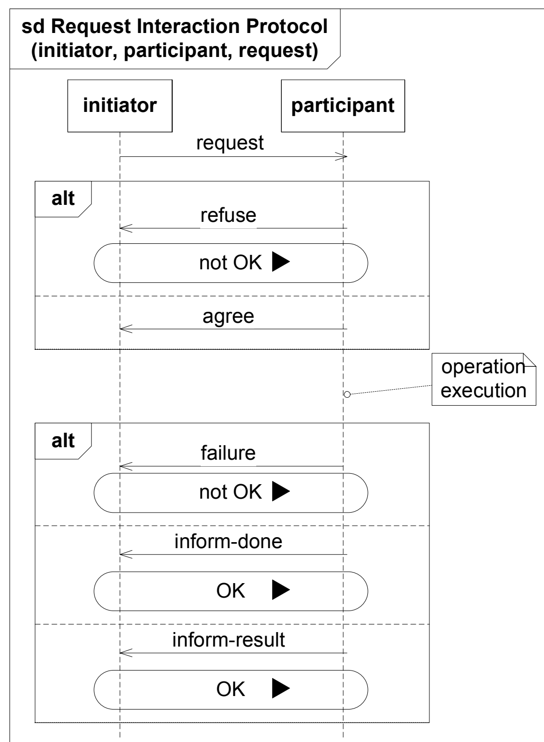


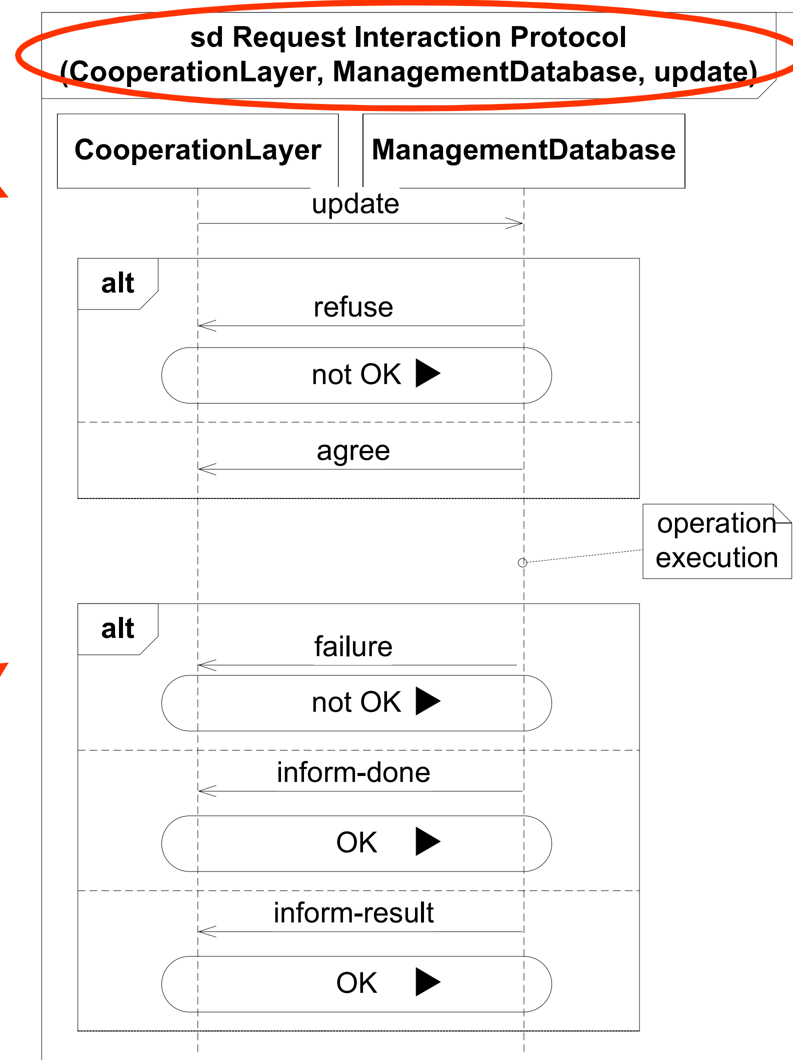
- protokol se tretira kao jedna cjelina
- konceptualna povezanost slijednih interakcija
- protokol kao uzorak koji se može prilagoditi sličnim problemima
- parametri predstavljaju komunikacijske činove koji se mogu promijeniti u toku instanciranja predloška

Primjena dijagrama predloška (1)



Primjena dijagrama predloška (2)





Pregledni interakcijski dijagram

- pregled toka gdje svaki čvor može biti interakcijski dijagram

sd Migration

ref
RIP(multioperation agent,
ref CooperationLayerMigration, migrate)

OK

not OK

sd

multioperation agent

multioperation agent 2

OK report

sd

multioperation agent

multioperation agent 2

error report

Dijagram stanja (*statechart diagram*)

- opisuje dinamičko ponašanje jednog entiteta sustava
- dijagram stanja se stvara samo za entitete sa značajnim dinamičkim ponašanjem
- prikazuje sva moguća stanja entiteta te prijelaze iz jednog stanja u drugo
- stanje entiteta
  - okolnost u kojoj se entitet nalazi kada zadovoljava određene uvjete, izvodi akcije ili čeka događaj
  - aktivnosti koje se obavljaju u stanju opisujemo:
    - `do/aktivnost` (izvodi se za vrijeme dok je entitet u tom stanju),
    - `entry/aktivnost` (izvodi se pri ulasku u stanje)
    - `exit/aktivnost` (izvodi se pri izlasku iz stanja)
  - posebna stanja:
    - početno stanje – svaki dijagram mora imati barem jedno i u tom stanju se entitet nalazi kada se kreira
    - krajnje stanje – svaki objekt može imati više njih

Dijagram stanja (*statechart diagram*)

- prijelazi
  - predstavljaju promjenu stanja iz trenutnog u novo (novo stanje može biti to isto stanje)
  - inicirani su događajima i uvjetima, a opisuju se kao

    `događaj[uvjet]/akcija`
  - da bi se prijelaz dogodio potrebno je da se dogodi događaj i da bude ispunjen uvjet
  - smatra se da je vrijeme za izvođenje prijelaza 0 i da se ne može prekinuti

Primjer dijagrama stanja

Stanje

do / AkcijaUStanju
entry / UlaznaAkcija
exit / IzlaznaAkcija

dodajPolaznika [brojPolaznika < 10]/brojPolaznika++

prijelazi

Otvoreno

[brojPolaznika = 10]

Zavoreno

dodajPolaznika / brojPolaznika = 1

otkaz

otkaz

tecajZavrsio

Inicijalizacija

Otkazano

početno stanje

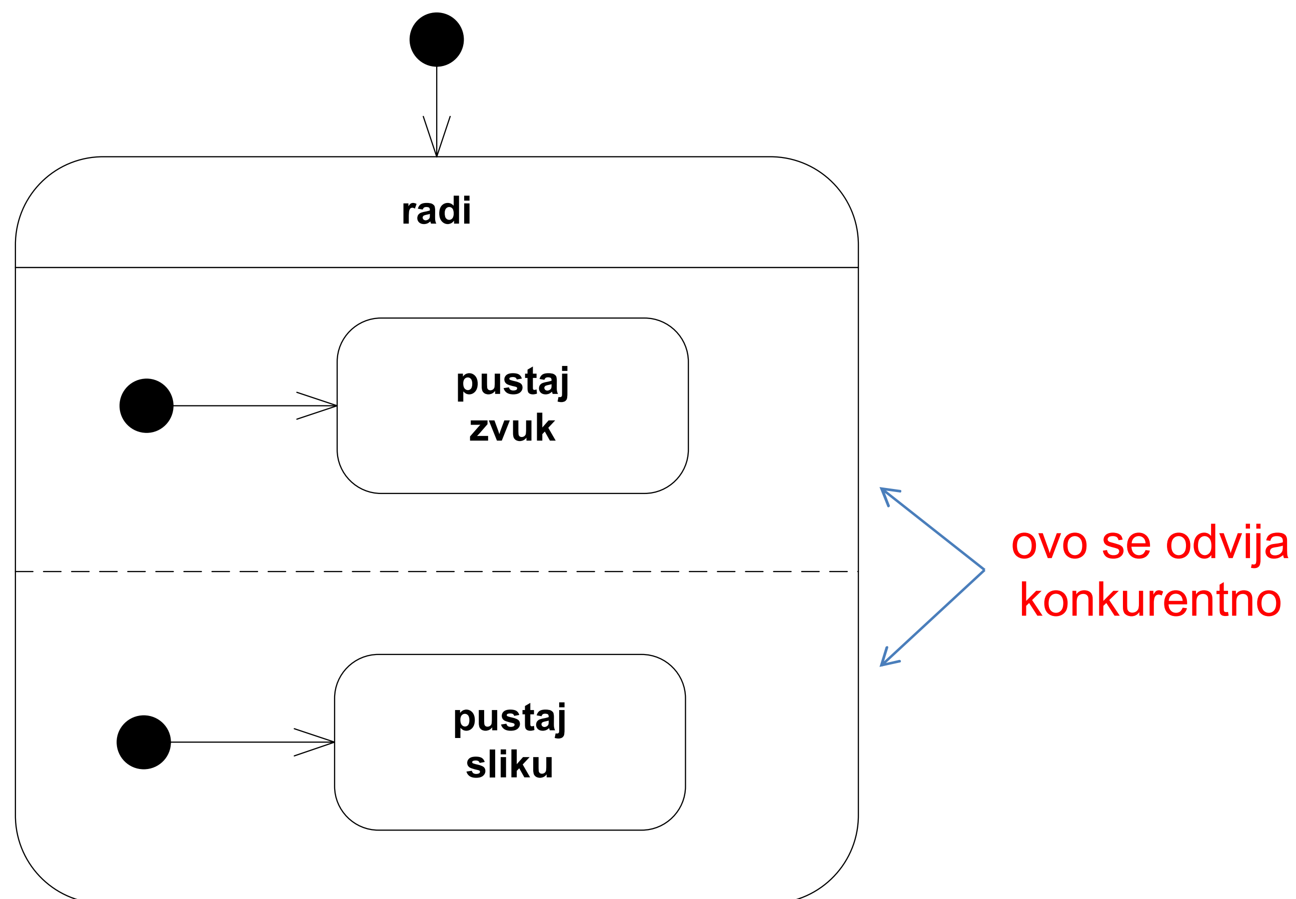
obična stanja

krajnje stanje

Kompozitna stanja u dijagramu stanja

delete/RemoveDayDigit
1,2/SetDay
initial
4..9/SetDay
3/SetDay
day_1_2
day_3
day_4_to_9
0..9/AddDayDigit
0,1/AddDayDigit
delete
/RemoveDayDigit
delete
/RemoveDayDigit
day_10_20
day_30
H
dot/PushState
delete/PopState
month
initial
delete/RemoveMonthDigit
1/SetMonth
2[day<=29]/SetMonth
3,5,7,8/SetMonth
4,6,9[day!=31]/SetMonth
month_3_5
month_4_6

Konkurentne aktivnosti u dijagramu stanja



Modeliranje protokola dijagramom stanja

Modeliranje protokola slijednim dijagramom

klijent

poslužitelj

alt

zahtjev

potvrda

odgovor

[gubitak potvrde]

zahtjev

potvrda

otkaži

Konkurentnost u objektno orijentiranim jezicima

- Pitanja koja se postavljaju:
  - Kako implementirati?
    - Da li konkurentnost ugraditi u jezik ili u biblioteke?
    - Da li ugraditi mehanizme kako što su dijeljenje vremena (*time sharing*) ili se osloniti na mehanizme koje podržava operacijski sustav?
    - Kolika je najmanja jedinica koja ima ekskluzivni pristup objektu?
- Koje oblike konkurentnosti ugraditi?
  - dijeljena memorija
  - višezadaćnost (*multitasking*)
  - mrežno programiranje
  - raspodijeljeno procesiranje
  - aplikacije u realnom vremenu
- Kako komunicirati?
  - sinkrono ili asinkrono

Konkurentnost u Javi

- zasniva se na nitima/dretvama
- najmanja jedinica ekskluzivnog pristupa objektu je sinkronizirani blok:
  - blok naredbi
  - cijela metoda
- sinkrono izvođenje uz čekanje u repu na ekskluzivni pristup objektu
- dijeljenje resursa je riješeno pomoću izvođenja sinkroniziranih blokova
- dio konkurentnosti je ugrađen u jezik, a dio u biblioteke
- implementacija ovisi o operacijskom sustavu i izvedbi JVM-a
  - *time slicing*
  - niti u Javi ekvivalentne nitima na operacijskom sustavu

Primjer niti u Javi

```
public class FirstThreadExtends
    extends Thread {

    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println("broj " + i);
        }
        System.out.println("Gotovo brojenje");
    }

    public static void main(String[] args) {
        new FirstThreadExtends().start();
        new FirstThreadExtends().start();
    }
}
```

```
broj 0
broj 1
broj 0
broj 2
broj 1
broj 3
broj 2
broj 4
broj 3
Gotovo brojenje
broj 4
Gotovo brojenje
```

Stanja niti

alive

new

yield()

- sleep()
- suspend() / resume()
- wait() / notify()
- kritični odsječak

start()

New thread

Runnable

Not runnable

[završila metoda run()]

stop()

dead

Nitna sigurnost

- Objekt čije se vrijednosti atributa nalaze u ispravnom stanju bez obzira na broj niti koje koriste taj objekt je nitno siguran.
- Ako nije nitno siguran:
  - Javlja se neočekivan rad kada više niti istovremeno spremaju podatake u neki atribut

Primjer nitne nesigurnosti (1)

```
public class Point {
    private int x,y;

    public Point() {
    }

    public void setPoint(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public String toString() {
        return "(" + x + "," + y + ")";
    }
}
```

Primjer nitne nesigurnosti (2)

```
public class ThreadSetter extends Thread {
    private Point point;
    int start;

    public ThreadSetter(Point p, int s) {
        point = p;
        start = s;
    }

    public void run() {
        for (int i = start; i < 5 + start; i++) {
            point.setPoint(i,i);
            String str = point.toString();
            System.out.println("nit" + start + "  " + str);
        }
    }
}
```

Primjer nitne nesigurnosti (3)

```
public class Main {
    public static void main(String[] args) {
        Point p = new Point();
        new ThreadSetter(p, 0).start();
        new ThreadSetter(p, 10).start();
    }
}
```

nit0 (0,0)
nit10 (10,10)
nit0 (1,1)
nit10 (11,11)
nit10 (12,12)
nit0 (2,2)
nit0 (3,3)
nit0 (4,4)
nit10 (4,13)
nit10 (14,14)

Kako se to moglo dogoditi?

nit0:ThreadSetter

p:Point

nit10:ThreadSetter

set(13,13)

x=13

set(4,4)

x=4
y=4

toString()

return "(4,4)"

y=13

toString()

return "(4,13)"

Rješenje: zaključavanje kritičnih odsječaka

- dijelovi koda koji pristupaju istom objektu iz različitih niti nazivaju se kritični odsječci
- u Javi kritični odsječak može biti dio koda ili metoda
- kritični odsječak se označava ključnom riječi `synchronized`

```
synchronized (obj) {
  // kritični odsječak
     ...
}
```

```
public synchronized void metoda()
{
  // kritični odsječak
     ...
}
```

- kada neka nit izvodi kod unutar kritičnog odsječka onda je taj objekt zaključan i ključ ima ta nit

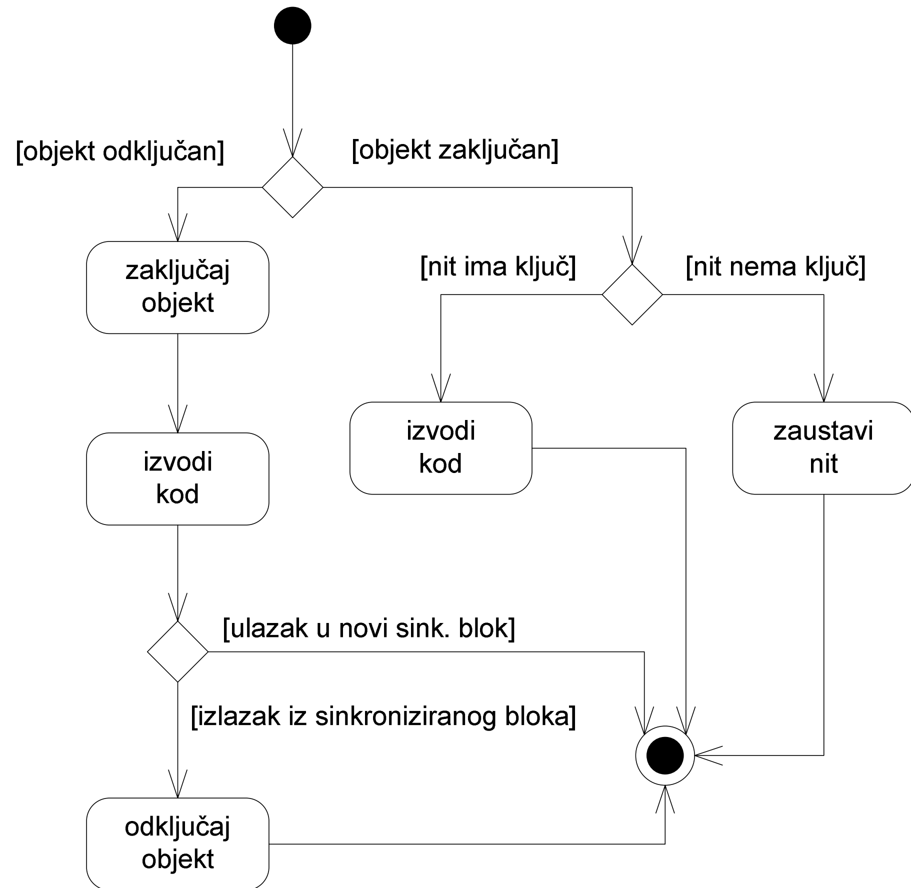
Algoritam zaključavanja i otključavanja

```
public class Point {
    private int x,y;

    public Point() {
    }

    public synchronized void
     setPoint(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    public synchronized String
     toString()
    {
        return "(" + x + "," + y + ")";
    }
}
```



Rad objekta sa zaključavanjem

```
public class Point {
    private int x,y;

    public Point() {
    }

    public synchronized void setPoint(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public synchronized String toString() {
        return "(" + x + "," + y + ")";
    }
}
```

nit0 (0,0)
nit10 (10,10)
nit0 (1,1)
nit10 (1,1)
nit10 (12,12)
nit0 (2,2)
nit0 (3,3)
nit0 (4,4)
nit10 (13,13)
nit10 (14,14)

Zašto to tako radi?

nit0:ThreadSetter

p:Point

nit10:ThreadSetter

set(11,11)

x=11
y=11

set(1,1)

x=1
y=1

toString()

return "(1,1)"

toString()

return "(1,1)"

Problem sinkronizacije

- kriva sinkronizacija
  - nema nikakve sinkronizacije
  - krivo postavljeni uvjeti sinkronizacije koji dovode do zastoja (*deadlock*)
- osiguravanje pravednog sustava
  - svaka nit dobije dovoljno pristupa resursima tako da sve niti podjednako napreduju u svojim poslovima, nema stagnacije niti i zastoja (*deadlocks*)
  - stagnacija (*starvation*) –čekanje na na procesor i ne može dalje napredovati
    - nit ima niži prioritet ne može se izvršavati jer se niti s višim prioritetom izvode
  - zastoj (*deadlock*) – trajno čekanje na uvjet tj. sve niti čekaju
    - dvije ili više niti čekaju na uvjet koji se ne može zadovoljiti

Mehanizmi koordinacije (1)

- ***Counting semaphor***
  - Dijkstra *counting semaphor*
    - postoji nekoliko resursa
    - svaka nit treba jedan resurs
    - više niti se natječe za resurse
  - poseban slučaj: mutex – semafor s jednim resursom
- **Rep (*queue*)**
  - blokirajući rep
    - nakon popunjavanja kapaciteta stavljanje blokira nit
    - kod uzimanja iz praznog repa blokira se nit
  - rep s kašnjenjem
    - kod stavljanja se može navesti vrijeme koliko se element neće moći izvaditi iz repa
    - kod uzimanja se može definirati koliko se maksimalno čeka element
  - rep s prioritetima
    - elementi se u repu sortiraju
  - sinkronizirani rep
    - zapravo nije rep
    - prilikom stavljanja nit se blokira dok druga nit ne uzme podatak iz repa

Mehanizmi koordinacije (2)

- **Izmjenjivač podataka (*exchanger*)**
  - dvije niti žele zamijeniti podatke
  - kada jedna pozove metodu za izmjenu podataka onda se ona blokira
  - kada druga nit pozove metodu za izmjenu podataka onda se njoj vrati objekt prve niti, a prvoj niti objekt druge niti
- **Prepreka (*barrier*)**
  - prepreka je sinkronizacijska točka u kojoj se sinkronizira izvođenje više niti
  - svaka nit koja se treba sinkronizirati pozove metodu u prepreci
  - kada specificiran broj niti pozove metodu u prepreci onda se sve niti oslobađaju
  - broj niti se specificira prilikom konstruiranja prepreke

Mehanizmi koordinacije (3)

- **Uzorak zaključavanja (*lock pattern*)**
  - općenitija implementacija mehanizma zaključavanja
  - fleksibilno strukturiranje zaključavanja
- **Ponovno iskorištavanje niti**
  - strategije korištenja niti:
    - jedna nit za svaki zahtjev (mrežni poslužitelj)
      - trošenje resursa za pokretanje i zaustavljanje niti
      - preopterećenje poslužitelja za velik broj zahtjeva
    - jedna nit za sve zahtjeve (GUI)
      - usko grlo
  - ponovno iskorištavanje niti iz "bazena" niti

Agenti

- entitet koji predstavlja korisnika u nekoj okolini (mreža) i autonomno obavlja poslove umjesto korisnika
- svojstva:
  - autonomnost,
  - inteligencija,
  - pokretljivost,
  - reaktivnost,
  - proaktivnost,
  - komunikativnost i
  - društvenost.

AUML – *Agent Unified Modeling Language*

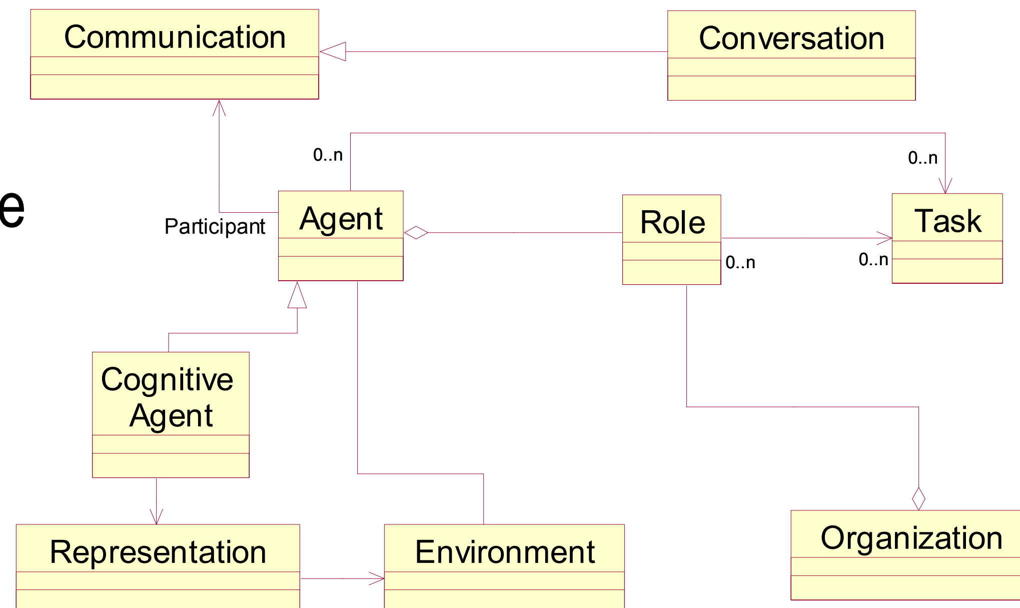
- Ideja je prikazati agente uz pomoć standarda za specifikaciju i vizualizaciju objektno-orijentiranih programskih rješenja → UML-a
- Treba prikazati agentski-orijentirane koncepte:
  - agent,
  - interakcijski protokol i
  - ontologija
- AUML je prijedlog proširenja UML-a za verziju 2.0

AUML – *Agent Unified Modeling Language*

- FIPA Modeling Technical Committee (Modeling TC)
- AUML – zajednička semantika, meta-model, apstraktna sintaksa za agentsko temeljene metodologije
- temelji se na UML 2.0
- plan – 12/2004 – završni prijedlog integracije metoda
- izvori:
  - UML 2.0
  - AOR
  - PASSI
  - MESSAGE
  - Tropos (uključujući *i** i GRL)
  - ADELFE
  - Gaia
  - BRIC
  - Styx
  - Prometheus
  - MADkit
  - OPM

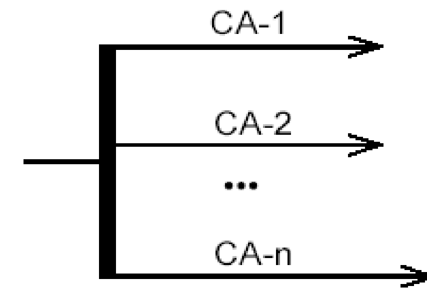
Višeagentski sustavi

- model višeagentskog sustava treba sadržavati:
  - skup agenata
  - odgovornosti svakog agenta
  - interakcijski protokoli
  - definicije poruka i njihovog sadržaja
  - semantiku svake poruke

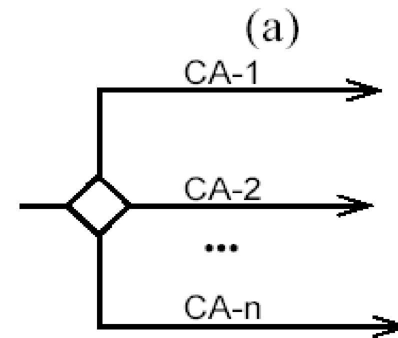


Konkurentne interakcije

a) sve će se, od CA-1 do CA-n, poslati istovremeno (konkurentno)

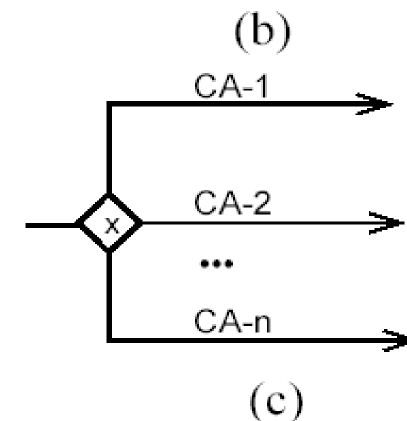


b) odluka (*decision box*) će odlučiti koliko će se CA poslati (nijednu ili više njih)

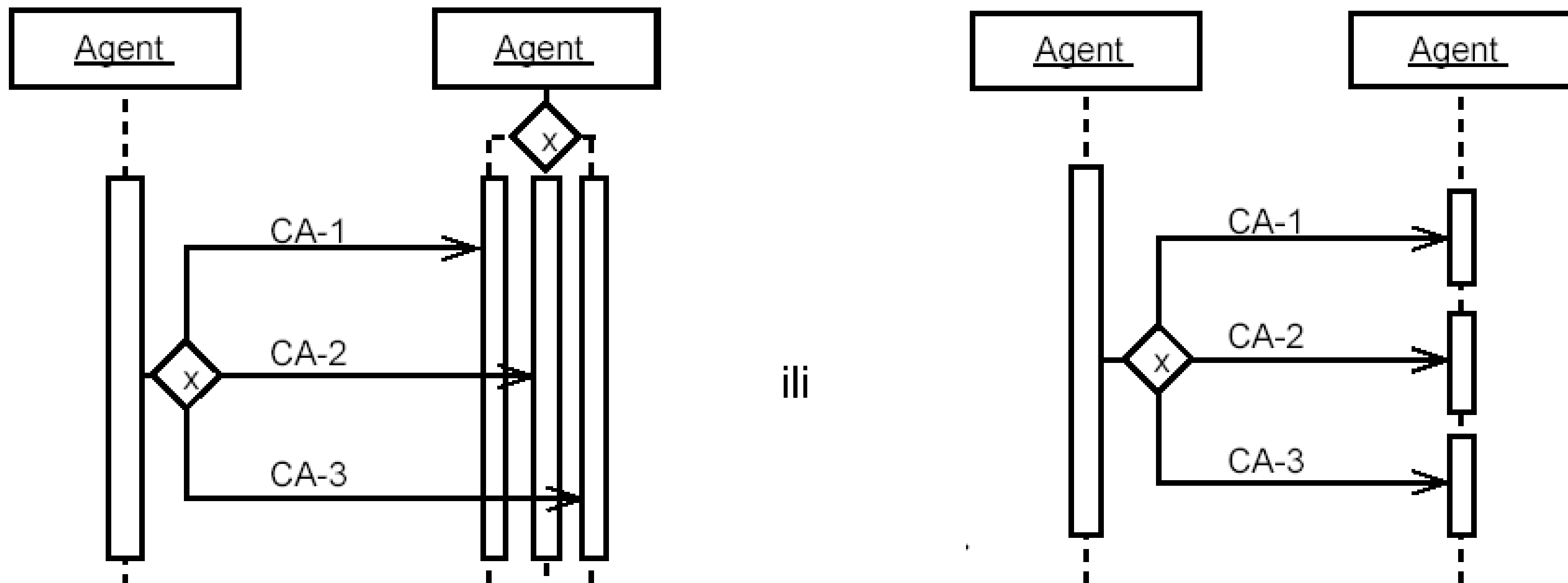


c) ekskluzivni ili (*exclusive or*) slanje, točno jedan CA će biti poslan

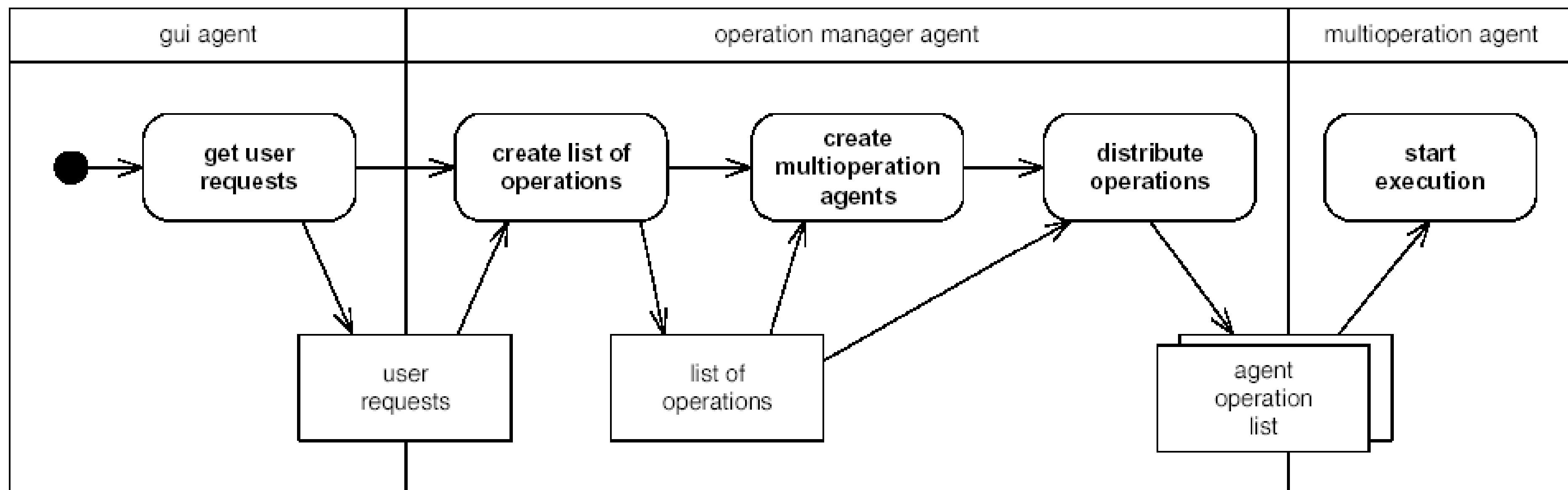
- može se prikazati fragmentom označenim s alternative



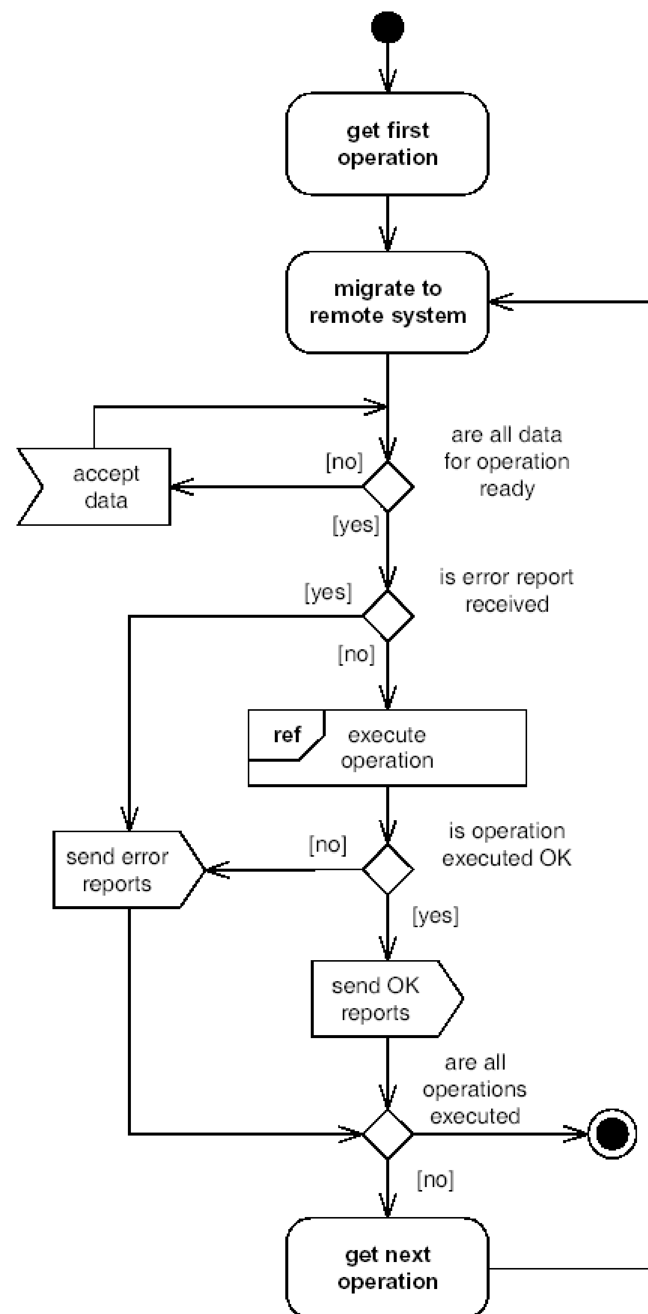
Konkurentne interakcije – primjer



Modeliranje dinamike između agenata – dijagram aktivnosti



Modeliranje dinamike unutar agenta – dijagram aktivnosti



Modeliranje dinamike unutar agenta – dijagram aktivnosti – promjena uloga

«attribute» providerRole: Role

Order Handler

Receive Order

[order accepted]

Fill Order

Ship Order

Close Order

Invoice Handler

Send Invoice

Accept Payment

Invoice

«external»

Customer

Make Payment

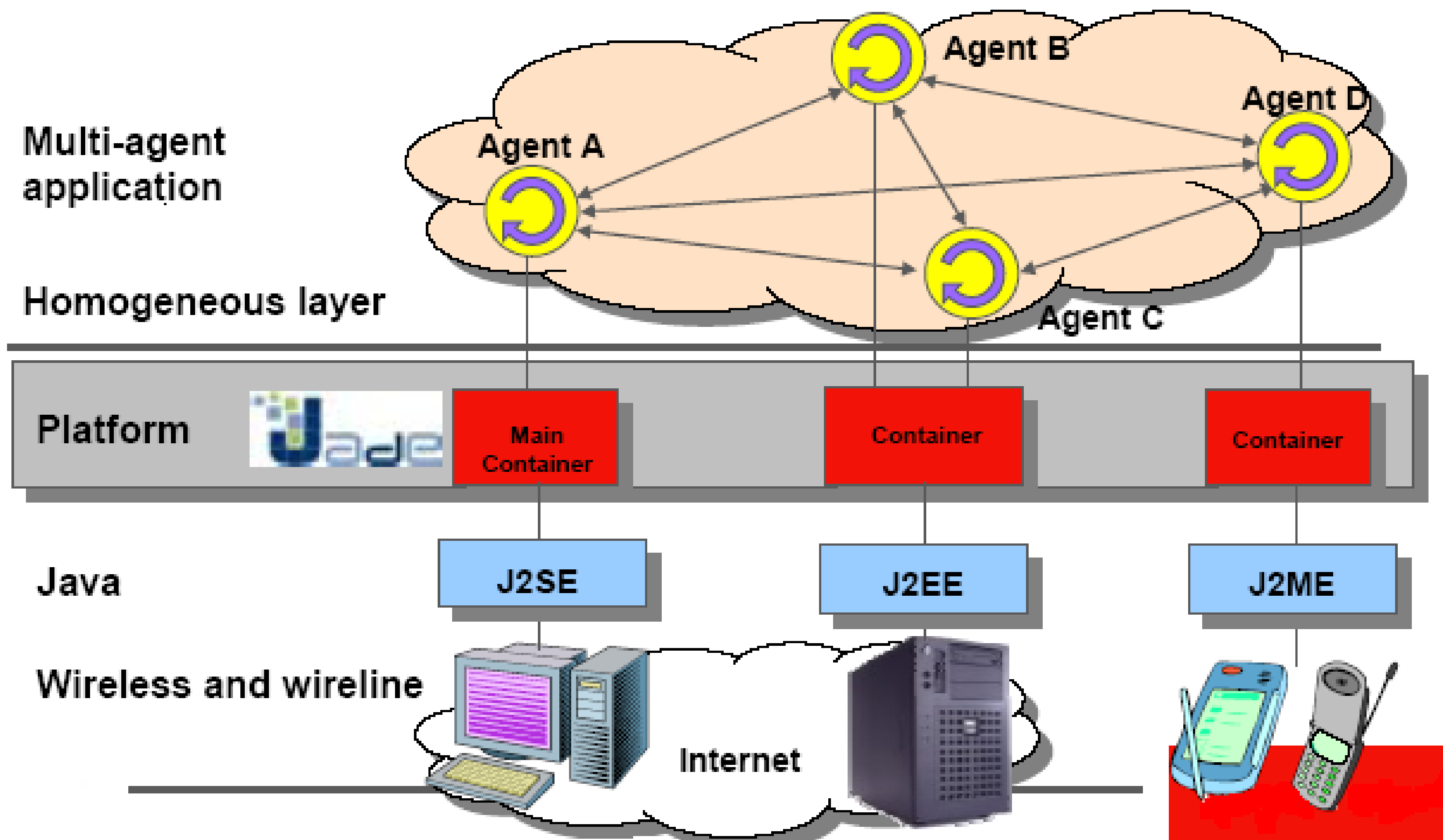
Agentska platforma JADE (1)

- Open Source (LGPL licenca)
- Koncepti platforme i kontejnera
- Intenzivan razvoj
- Greške se brzo otklanjaju
- Ugrađeni novi sigurnosni mehanizmi (JAAS)
- Asinkrona komunikacija porukama
- Transport porukama: IIOP, HTTP, JMS, e-mail

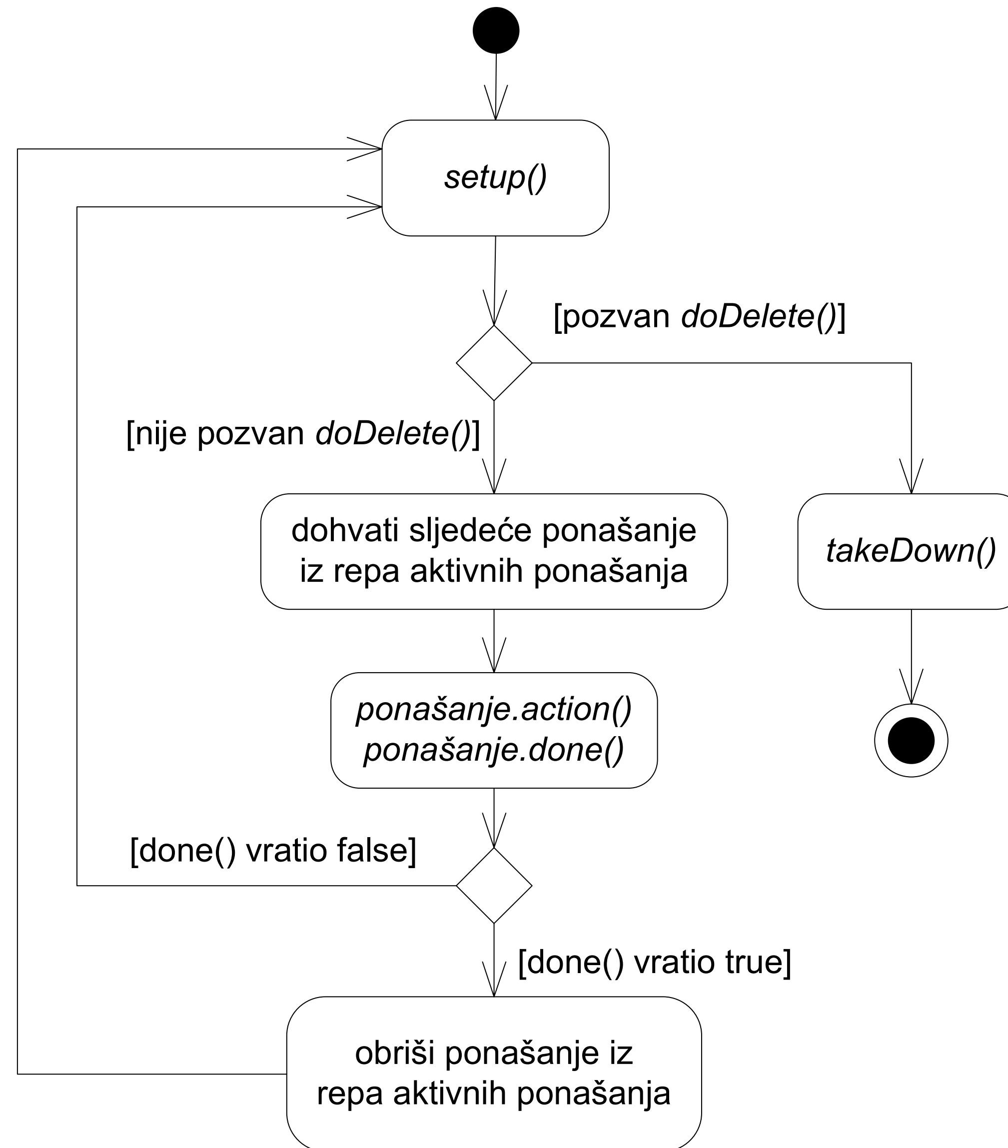
Agentska platforma JADE (2)

- U potpunosti u skladu s FIPA standardima
- Podržava J2EE, J2ME (MIDP 1.0 CLDC i CDC profili) i Personal Java
- Integracija s Web aplikacijama
- Integracija s Web uslugama (WSIG)
- Integracija s razvojnom okolinom Eclipse
- Mnoštvo dodataka:
  - BDI agenti (JEDEX)
  - Ekspertni sustavi (JESS)
  - Ontologije

JADE – platforma

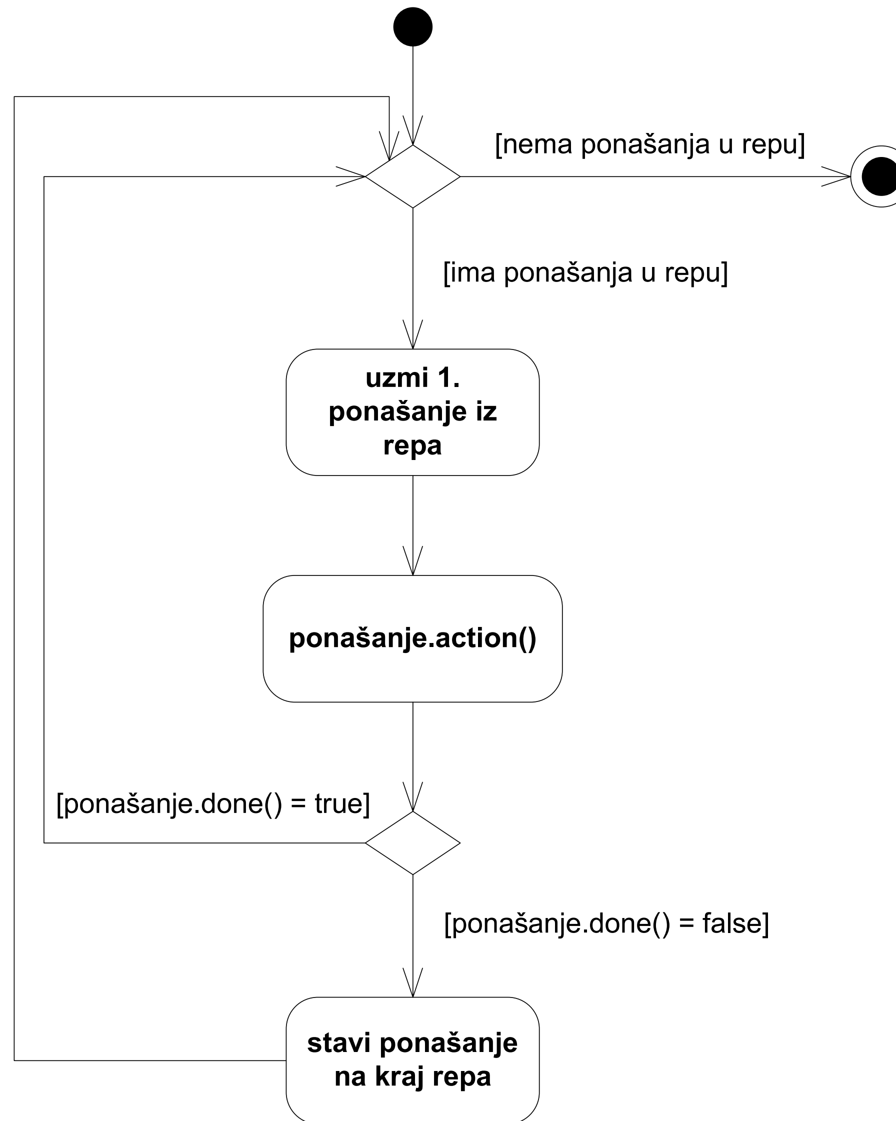


Živorni ciklus agenta



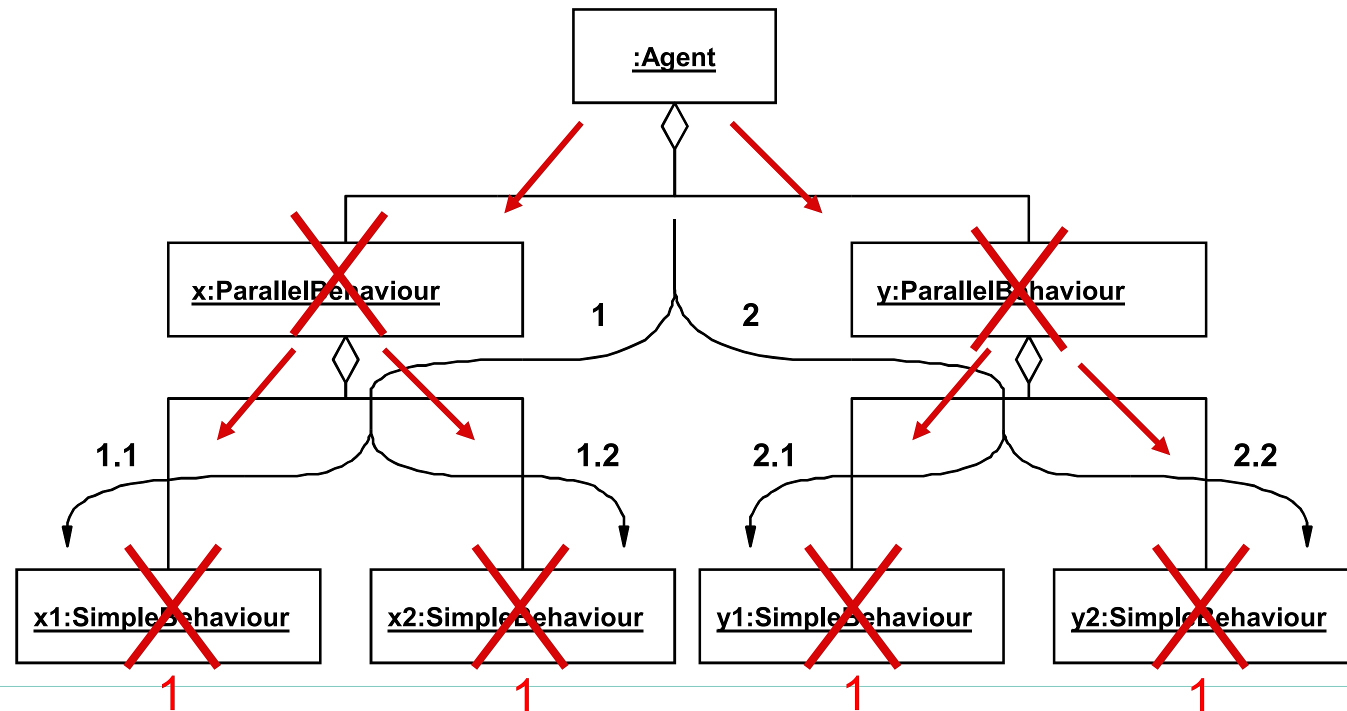
Ponašanja (Behaviours)

Algoritam izvođenja paralelnog ponašanja

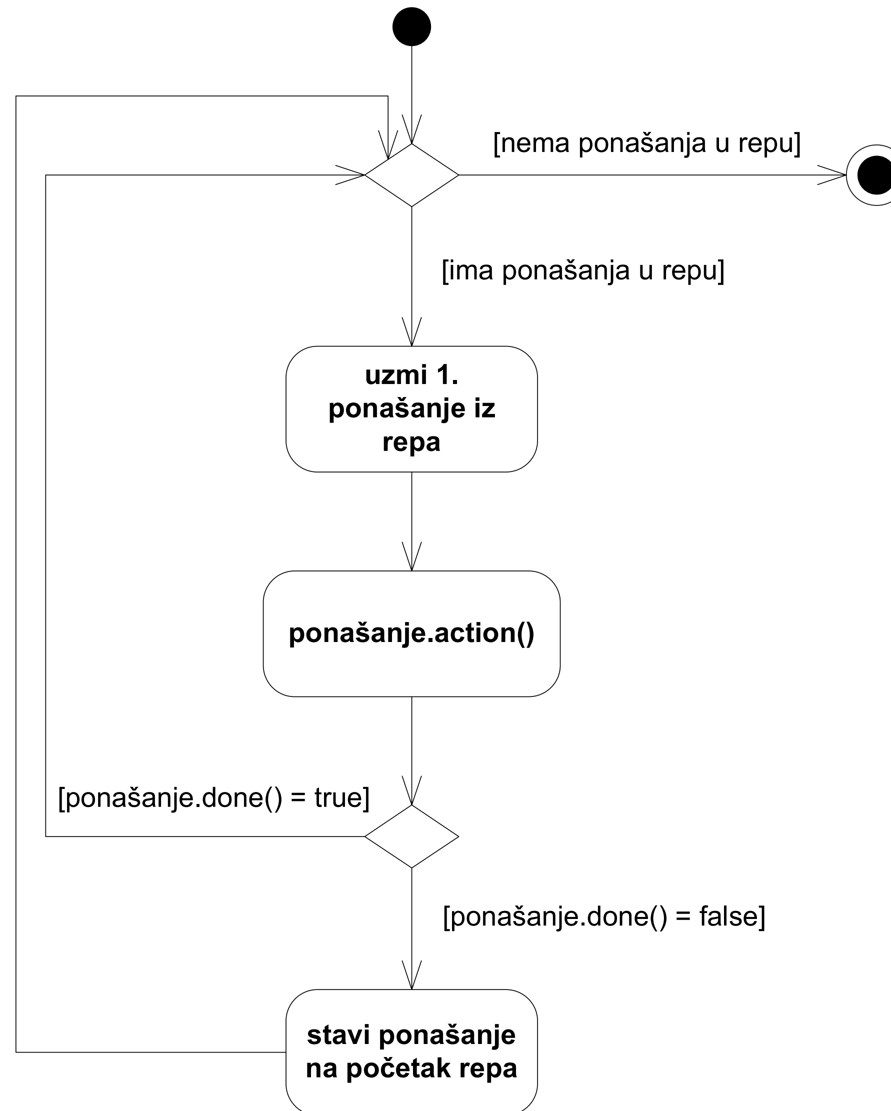


Konkurentno izvođenje paralelnog ponašanja

- Paralelno ponašanje s dugim potponašanjima:
  1. X, X1
  2. Y, Y1
  3. X, X2
  4. Y, Y2
  5. X, X1
  6. Y, Y1
  7. X, X2
  8. Y, Y2

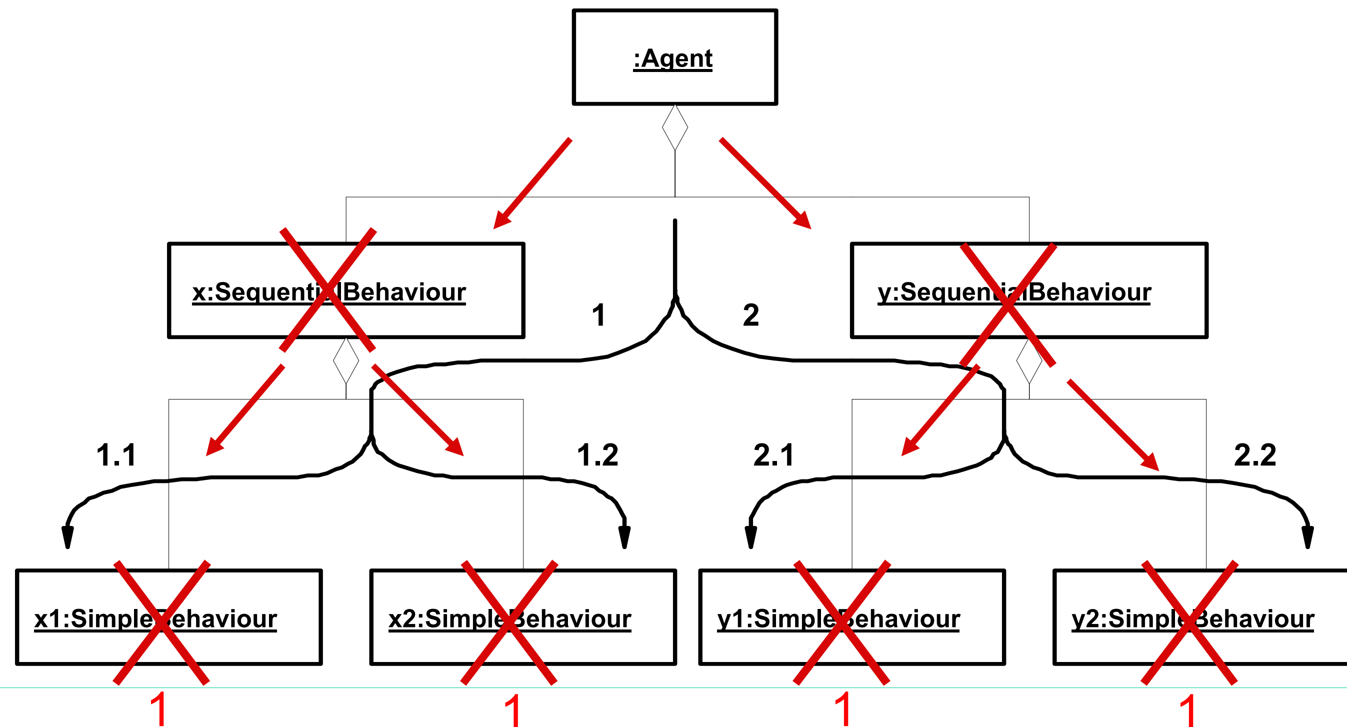


Algoritam izvođenja slijednog ponašanja

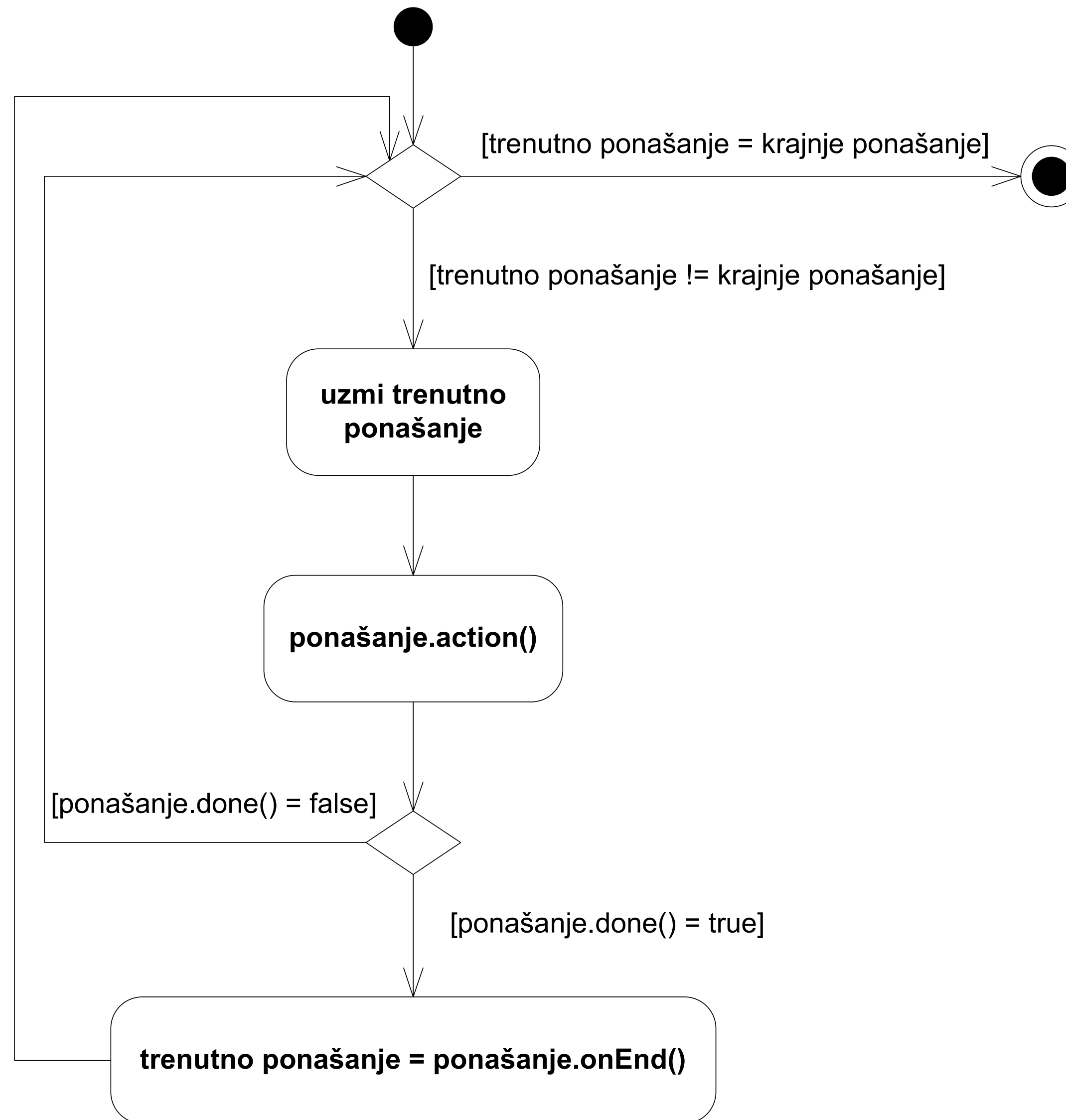


Konkurentno izvođenje slijednog ponašanja

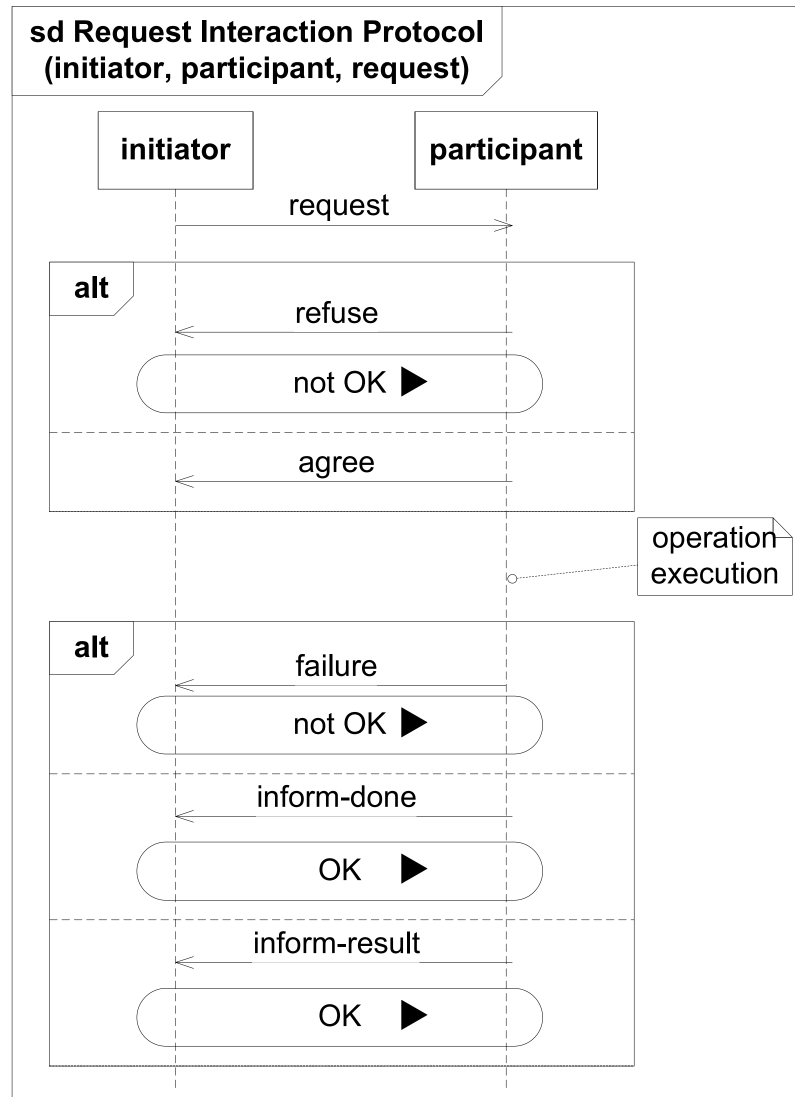
- Slijedno ponašanje s dugim potponašanjima:
  1. X, X1
  2. Y, Y1
  3. X, X1
  4. Y, Y1
  5. X, X2
  6. Y, Y2
  7. X, X2
  8. Y, Y2



Algoritam izvođenja ponašanja FSM (*finite state machine*)



Implementacija protokola RIP (1)



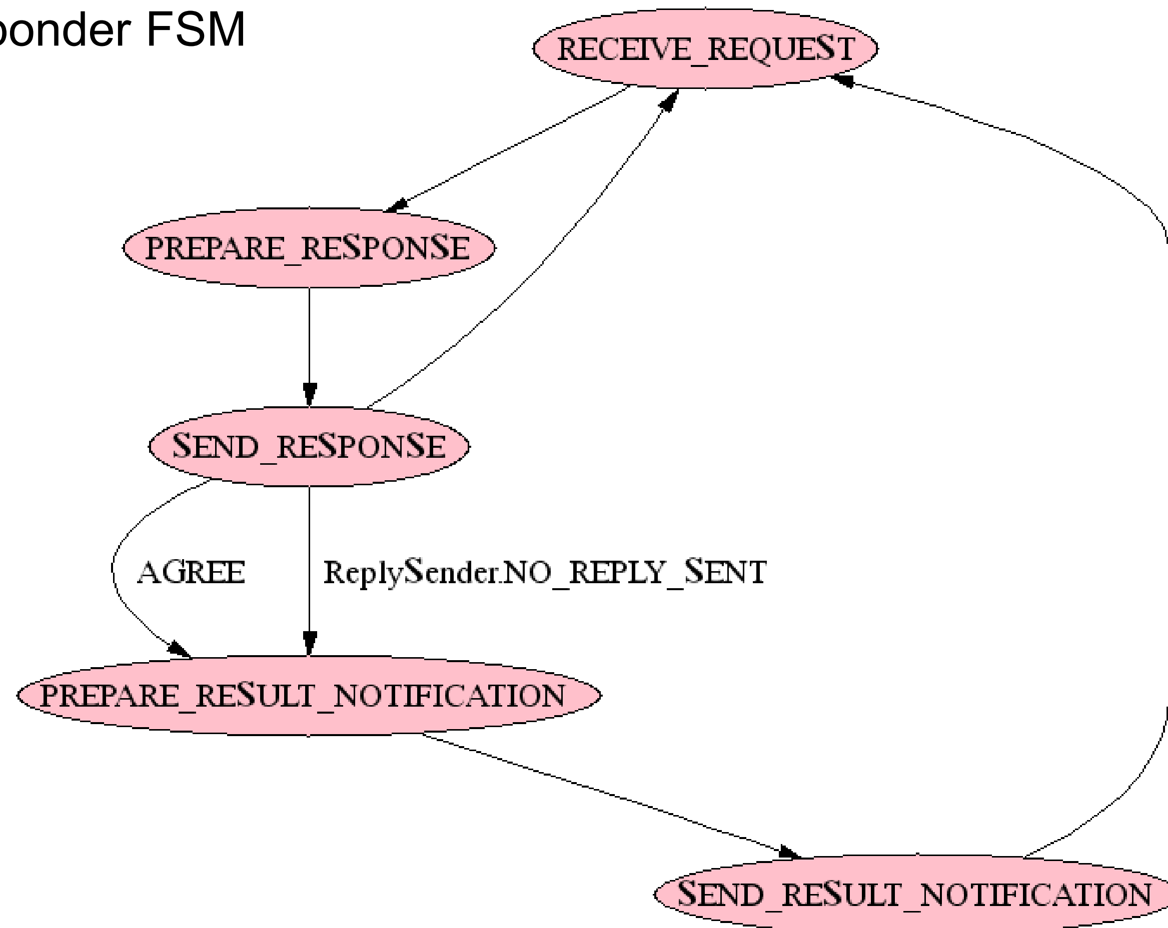
Implementacija protokola RIP (2)

AchieverREInitiator FSM

PREPARE_INITIATIONS
SEND_INITIATIONS
0
RECEIVE_REPLY
DUMMY_FINAL
CHECK_IN_SEQ
NOT_UNDERSTOOD
REFUSE
INFORM
FAILURE
AGREE
MsgReceiver.TIMEOUT
MsgReceiver.INTERRUPT
toBeReset
HANDLE_NOT_UNDERSTOOD
HANDLE_REFUSE
HANDLE_INFORM
HANDLE_FAILURE
HANDLE_POSITIVE_RESPONSE
HANDLE_OUT_OF_SEQ
toBeReset
CHECK_SESSIONS
ALL_RESPONSES_RECEIVED
HANDLE_ALL_RESPONSES
ALL_RESULT_NOTIFICATIONS_RECEIVED
CHECK_AGAIN
0
HANDLE_ALL_RESULT_NOTIFICATIONS

Implementacija protokola RIP (3)

AcheiverREResponder FSM



Literatura (1)

1. J.Odell H.Van, Dyke Parunak, B.Bauer “Extending UML for Agents”
2. F.Bergenti, A.Poggi “Exploiting UML in the Design of Multi-Agent System”
3. B.Bauer, J.Odell “Agent UML :A Formalism for Specifying Multiagent Interaction”
4. E.F. Lima, P.D. Machado, J.C. Figueiredo, F. R. Sampaio “Implementing Mobile Agent Design Patterns in the JADE framework”, Special Issue on JADE of the TILAB Journal EXP, 2003
5. http://www.auml.org

Literatura (2)

6. B.Bauer, J.Odell, “UML 2.0 and agents: how to build agent-based systems with the new UML standard”, Journal of Engineering Applications of Artificial Intelligence Volume 18, Issue 2 , March 2005, Pages 141-157.
7. Grady Booch, James Rumbaugh, Ivar Jacobson: The Unified Modeling Language User Guide, Addison-Wesley, 1998.
8. http://www.omg.org/uml/
9. Giovanni Rimassa, “Runtime Support for Distributed Multi-Agent Systems”, Ph. D. Thesis, University of Parma, January 2003.
10. Mario Kušek, “Koordinacija pokretnih agenata za daljinske operacije s programskim sustavom”, Ph. D. Thesis, FER, University of Zagreb, 2005.