Diplomski studij

Informacijska i komunikacijska tehnologija:

Telekomunikacije i informatika

Računarstvo:

Programsko inženjerstvo i informacijski sustavi

Računarska znanost

Ak.god. 2009./2010.

Raspodijeljeni sustavi

3.

Procesi i komunikacija u raspodijeljenom sustavu (2. dio)

Sadržaj predavanja

- Komunikacija porukama
- Model objavi-pretplati
- Java Message Service (JMS)

Komunikacija porukama

- Procesi/objekti komuniciraju razmjenjujući poruke.
- U komunikaciji sudjeluju izvor (pošiljatelj poruke) i odredište.
- Izvor šalje poruku, poruka se pohranjuje u rep koji je pridijeljen odredištu.
- Odredište čita poruku iz repa.
- Poruke sadrže podatke, važna je adresa odredišnog repa.
- Adresiranje se izvodi najčešće na nivou sustava, svaki rep ima jedinstven identifikator u sustavu.

Izvođenje komunikacije porukama (1)

1 izvor : 1 odredište

Izvođenje komunikacije porukama (2)

- `put` – dodaj poruku u rep
- `get` – pročitaj poruku iz repa, primatelj je blokiran ako je rep prazan
- `poll` – provjeri postoje li poruke u repu i pročitaj prvu poruku ako takva postoji, primatelj nije blokiran

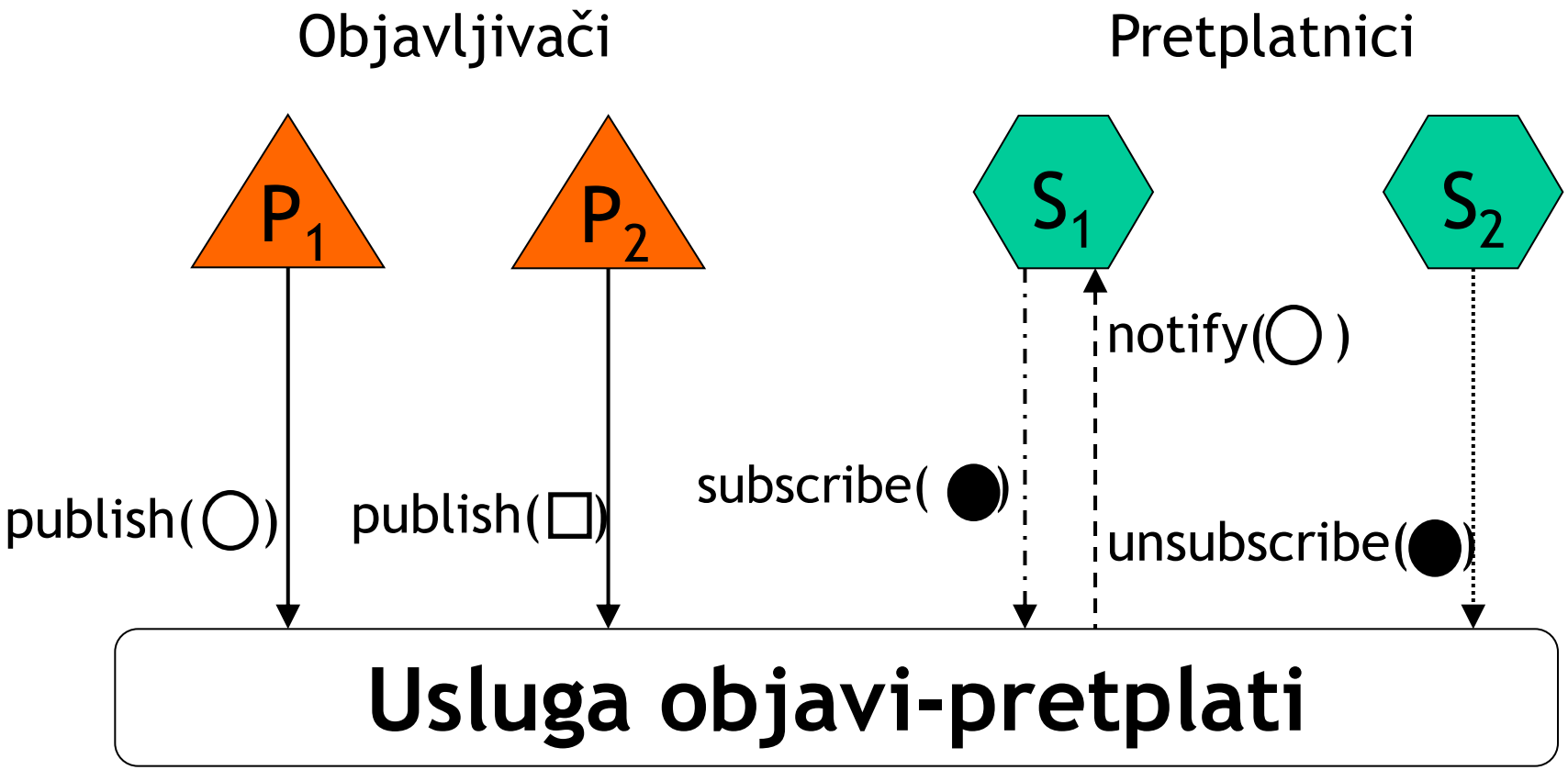
Karakteristike komunikacije porukama

- vremenska neovisnost
  - primatelji i pošiljatelji ne moraju istovremeno biti aktivni, poruka se sprema u rep
- pošiljatelj mora znati identifikator odredišta, tj. njegovog repa
- komunikacija je persistentna
- asinkrona komunikacija
  - pošiljatelj šalje poruku i nastavlja procesiranje neovisno o odgovoru od strane primatelja
- *pull* pokretanje komunikacije
  - primatelj provjerava postoji li poruka u repu

Sadržaj predavanja

- Komunikacija porukama
- Model objavi-pretplati
- Java Message Service (JMS)

Interakcija objavi-pretplati



Osnovni pojmovi (1)

- objavljivači (*publishers*)
- pretplatnici (*subscribers*)
- usluga objavi-pretplati:
  - sustav za obradu događaja (*event service* – ES)
- objavljivači i pretplatnici komuniciraju razmjenjujući sljedeće entitete preko usluge objavi-pretplati
  - obavijesti (*notifications*) – definiraju objavljivači,
  - pretplate (*subscriptions*) – definiraju pretplatnici i
  - odjave pretplata (*unsubscriptions*) – definiraju pretplatnici.

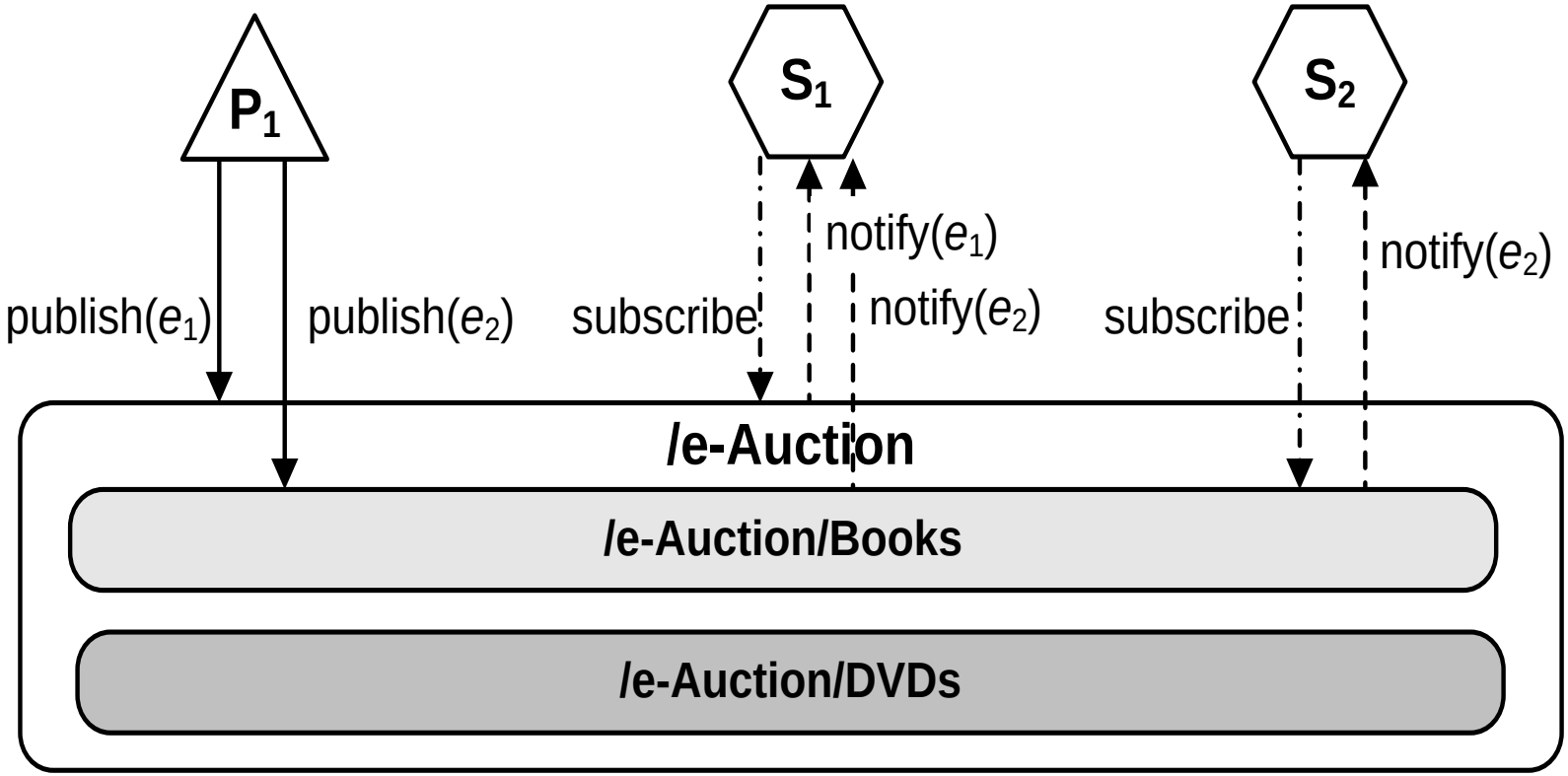
Osnovni pojmovi (2)

- objavljivači (*publishers*)
  - definiraju obavijesti
- pretplatnici (*subscribers*)
  - pretplatama i odjavama pretplata izražavaju namjeru primanja određenog skupa obavijesti
- usluga objavi-pretplati
  - obrađuje i pohranjuje primljene obavijesti/pretplate/odjave pretplata
  - isporučuje obavijesti pretplatnicima prema njihovim aktivnim pretplatama

Vrste pretplate

- Pretplata na kanal
  - tematsko grupiranje obavijesti (npr. vrijeme)
  - hijerarhijski odnos kanala (npr. vrijeme u Europi, Hrvatskoj, Zagrebu)
  - kanal – logička veza između izvora i odredišta
- Pretplata na sadržaj
  - pretplata se definira ovisno o svojstvima i sadržaju obavijesti (skup atributa i vrijednosti)

Pretplata na kanal



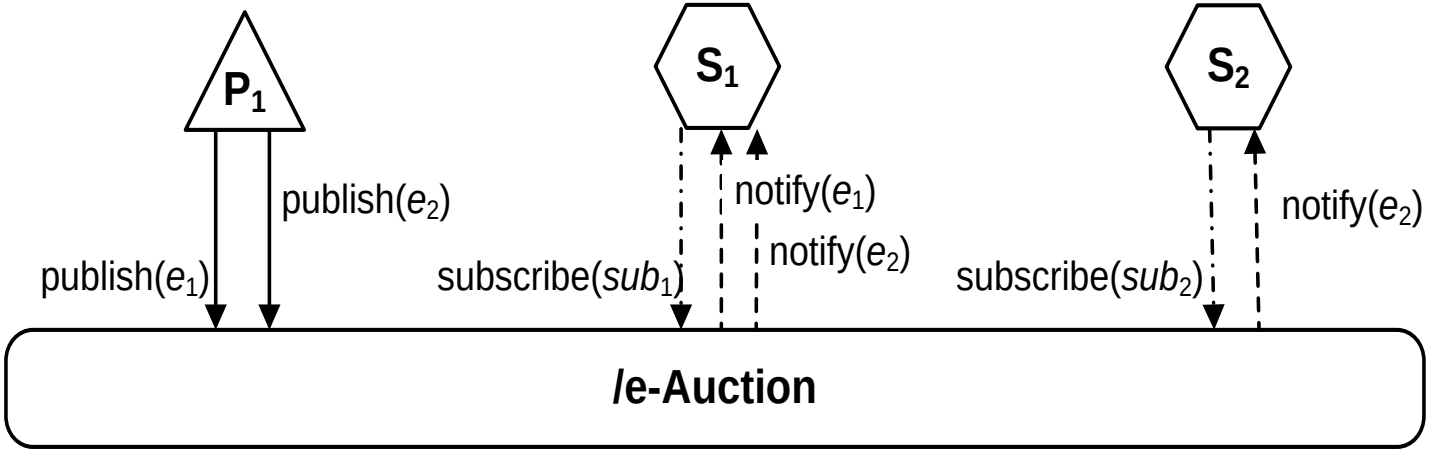
Pretplata na sadržaj

e_1 = (category = “books”
& author = “D. Adams”
& title = “The Hitchhiker's Guide through the Galaxy”
& price = 9.99 EUR)

e_2 = (category = “books”
& author = “J.R.R. Tolkien”
& title = “The Lord of the Rings”
& price = 19.99 EUR)

sub_1 = (category == “books”
& price < 20 EUR)

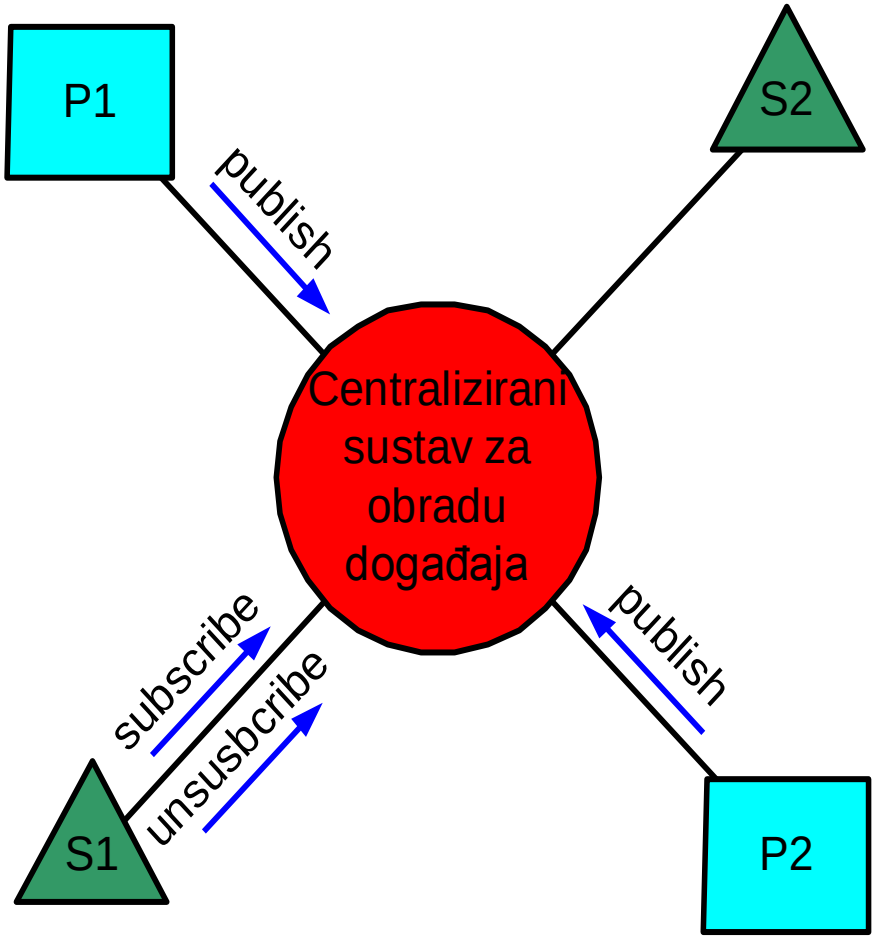
sub_2 = (category == “books” &
author == ”J.R.R. Tolkien”
& price < 20 EUR)



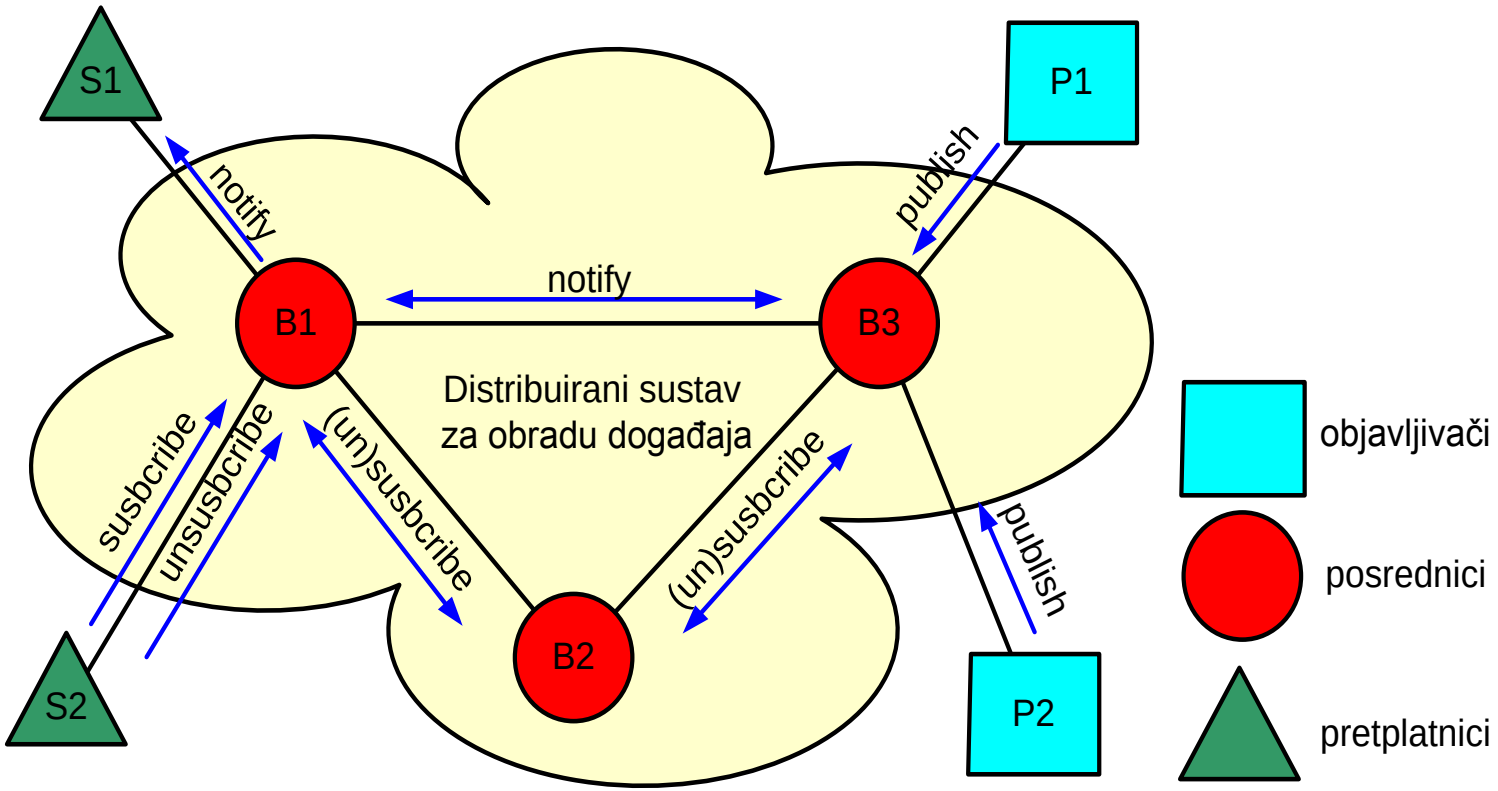
Arhitektura usluge objavi-pretplati

- Centralizirana
  - svi objavljivači i pretplatnici razmjenjuju obavijesti i definiraju pretplate preko jednog poslužitelja posrednika
  - poslužitelj pohranjuje sve pretplate i prosljeđuje obavijesti
- Distribuirana
  - skup poslužitelja, svaki je poslužitelj zadužen za objavljivače i pretplatnike u svojoj domeni
  - algoritmi za usmjeravanje informacija o pretplatama i usmjeravanje objava

Centralizirana arhitektura



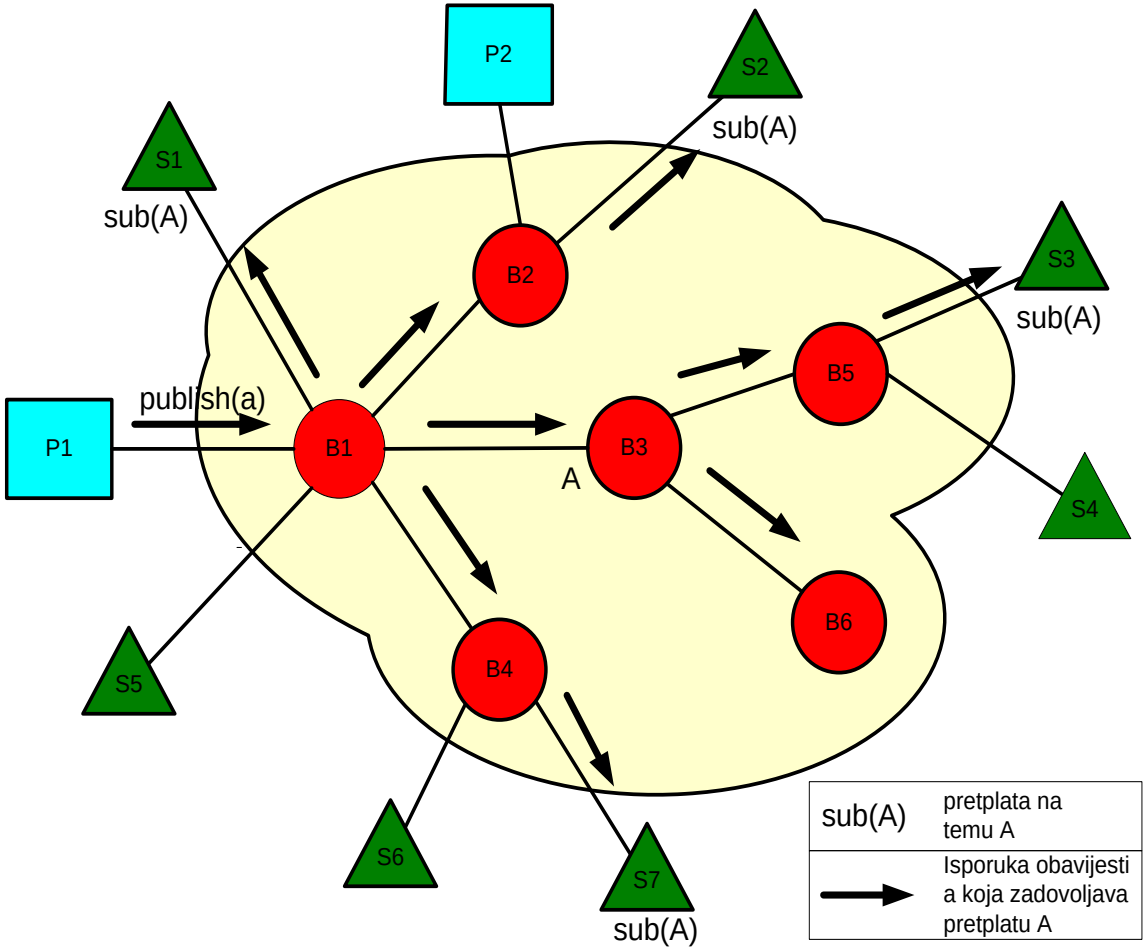
Distribuirana arhitektura



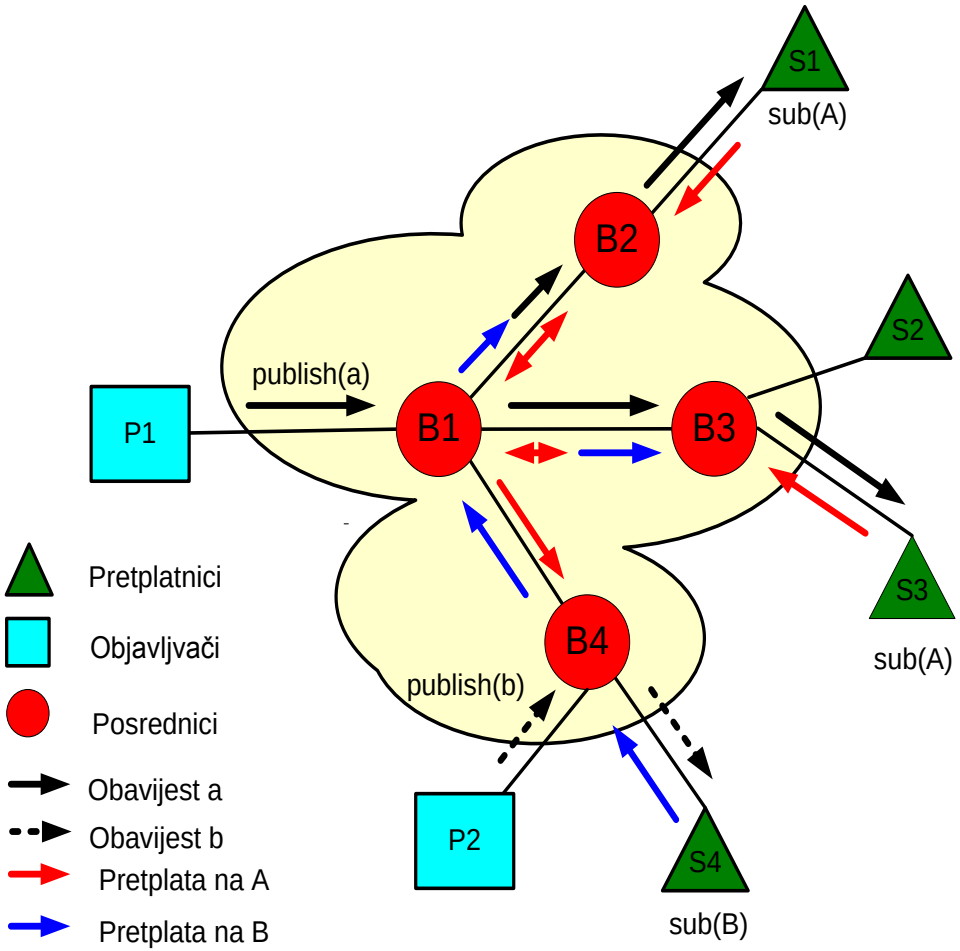
Osnovna načela usmjeravanja

- preplavljivanje
  - svaka primjena poruka (obavijest, pretplata ili odjava pretplate) prosljeđuje se svim susjedima osim onome od koga je poruka primljena
  - posrednik posjeduje tablicu usmjeravanja koja sadrži informacije o svim susjednim posrednicima i lokalnim pretplatnicima
- filtriranje poruka
  - filtriranje poruka se izvodi usporedbom obavijesti s aktivnim pretplatama koje definiraju svojstva obavijesti za koje je pretplatnik zainteresiran
  - osnovni cilj je isporuka samo onih obavijesti koje pretplatnika zanimanju
  - omogućuje i smanjenje prometa u mreži posrednika zbog sprječavanja širenja obavijesti “nezainteresiranim” posrednicima

Preplavljivanje obavijestima



Preplavljivanje pretplatama



Za obavijest na temu	Šalji prema
A	B2,B3
B	B4

Tablica usmjeravanja posrednika *B*1

Obilježja modela objavi-pretplati

- vremenska neovisnost
  - objavljivači i pretplatnici ne moraju istovremeno biti aktivni, posrednik pohranjuje poruku
- objavljivač ne mora znati identifikator pretplatnika (anonimnost), o tome se brine posrednik
- komunikacija je perzistentna
- asinkrona komunikacija
  - objavljivač šalje poruku i nastavlja procesiranje neovisno o odgovoru od strane odredišta
- pokretanje komunikacije na načelu *push*
  - objavljivač šalje poruku posredniku koji je prosljeđuje pretplatnicima bez prethodnog eksplicitnog zahtjeva

Obilježja modela objavi-pretplati (2)

- personalizacija primljenog sadržaja
  - filtriranje objavljenih poruka prema pretplatama
- proširivost sustava
  - dodavanje novog objavljivača ili pretplatnika ne utječe na ostale strane u komunikaciji
- skalabilnost
  - distribuirana arhitektura

Sadržaj predavanja

- Java RMI
- Komunikacija porukama
- Model objavi-pretplati
- Java Message Service (JMS)

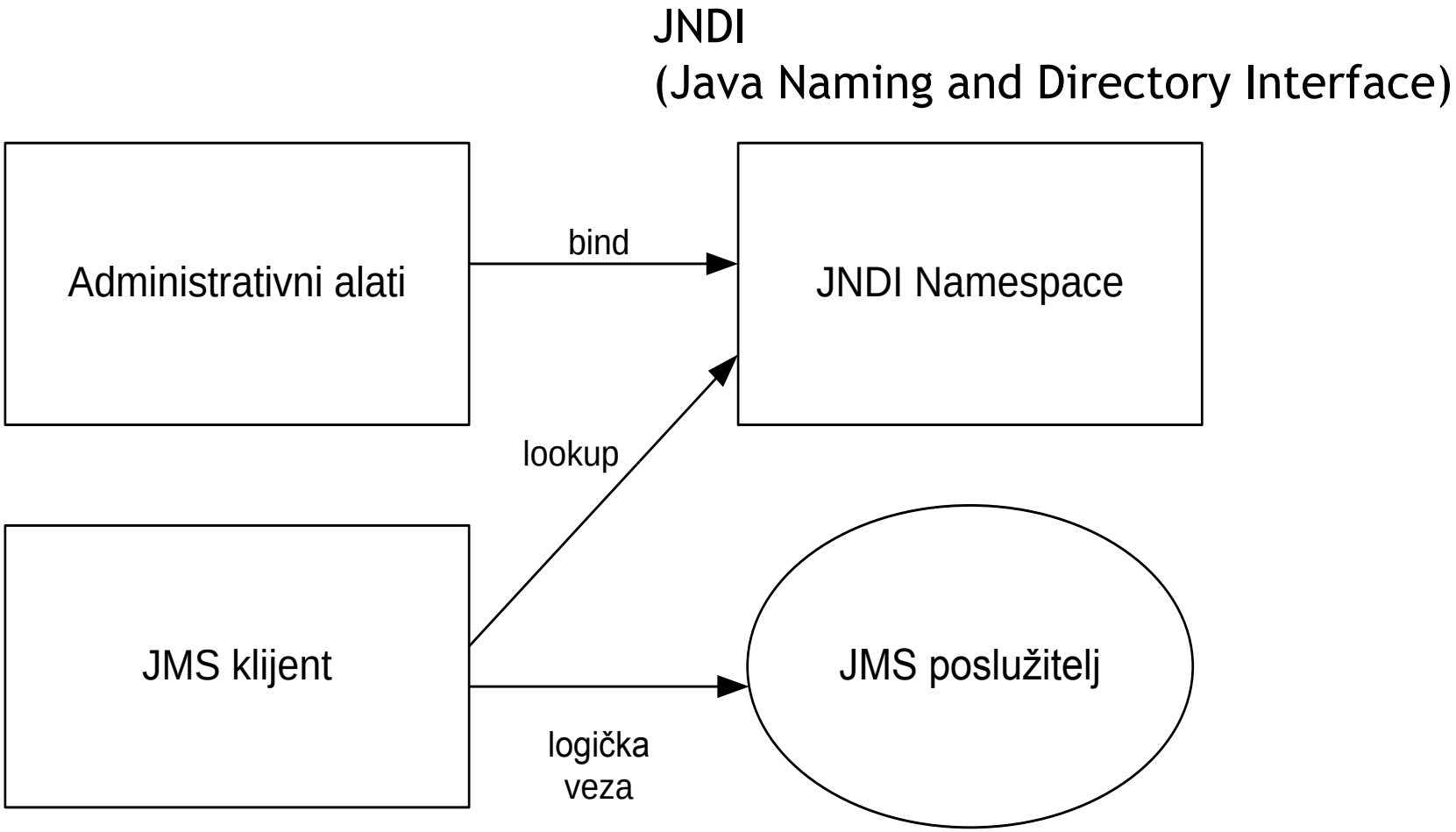
Java Message Service

- Sunova specifikacija za komunikaciju porukama i komunikaciju na načelu objavi-pretplati.
- JMS API definira skup sučelja i pripadajuću semantiku koja omogućuje programima pisanim u Javi komunikaciju razmjenom poruka i na načelu objavi-pretplati.

Arhitektura JMS-a (1)

- JMS poslužitelj
  - sustav za razmjenu poruka koji implementira JMS sučelja i nudi administrativne i kontrolne usluge
- Klijent
  - bilo koji objekt, proces ili aplikacija koja stvara ili konzumira poruke
- Poruka (*message*)
  - objekt koji se sastoji od zaglavlja koje prenosi identifikacijske i adresne informacije i tijela koje prenosi podatke
- Odredište (*destination*)
  - objekt koji sadrži informacije o odredištu poruke

Arhitektura JMS-a (2)

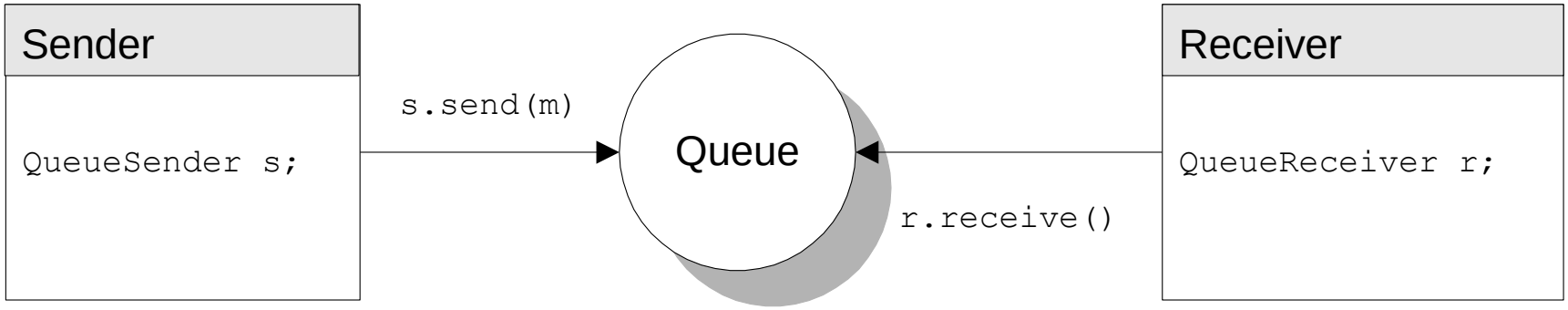


Modeli JMS-a

JMS implementira sljedeće modele za komunikaciju porukama i obavijestima

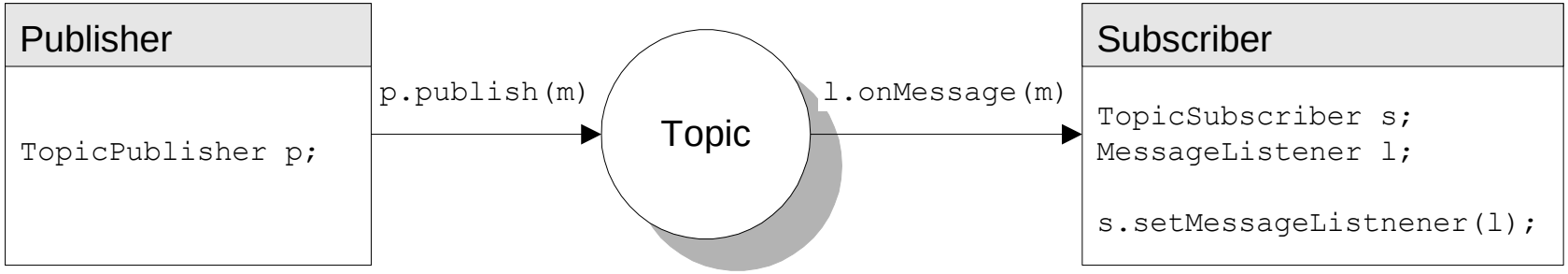
- *Point-to-point*
  - komunikacija porukama, jedna poruka za jedno odredište
- *Publish/subscribe*
  - objavi-pretplati, jedna poruka za skup zainteresiranih pretplatnika

Point-to-point



1. Klijent `s` koji šalje poruku `m` poziva `s.send(m)`. Poruka se sprema u rep.
2. Klijent koji prihvaća poruku mora provjeriti da li u repu postoji poruka. Poziva `r.recieve()`.
3. Poruka se briše iz repa i šalje klijentu.

Publish/subscribe



1. Tijekom inicijalizacije pretplatnik registrira instancu klase koja implementira sučelje `MessageListener` pozivajući `s.setMessageListener(l)`. `Topic` pamti sve pretplate.
2. Izvor objavljuje poruku `m` sa `p.publish(m)`.
3. `Topic` isporučuje poruku pretplatniku pozivajući `l.onMessage(m)`.

Sučelja JMS-a (1)

Nad-sučelje	Point-to-point	Publish/subscribe
Destination	Queue	Topic
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection

- **Destination**
  - administrirani objekt
  - predstavlja odredište - identitet ili adresu repa/teme.
- **ConnectionFactory**
  - administrirani objekt koji sadrži konfiguracijske parametre
  - klijenti ga koriste za stvaranje objekta *Connection*.
- **Connection**
  - predstavlja aktivnu konekciju prema JMS poslužitelju
  - klijenti ga koriste za stvaranje sesije (*Session*).

Sučelja JMS-a (2)

Nad-sučelje	Point-to-point	Publish/subscribe
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver	TopicSubscriber

- **Session**
  - jednostruka nit u kojoj se primaju odnosno šalju poruke
  - klijenti koriste sesiju da stvore jedan ili više *MessageProducer* ili *MessageConsumer* objekata
- **MessageProducer**
  - objekt za slanje poruka odredištu
- **MessageConsumer**
  - objekt za primanje poruka koje su poslane odredištu

Poruke JMS-a

- zaglavlje
  - skup definiranih polja koja sadrže vrijednosti koje identificiraju i usmjeravaju poruku
- svojstva poruke
  - opcionalni parovi ime-vrijednost, a vrijednost može biti *boolean*, *byte*, *short*, *int*, *long*, *float*, *double* ili *String*
- tijelo poruke
  - *TextMessage* sadrži *java.lang.String*. (npr. za slanje XML dokumenata)
  - *StreamMessage* za niz Javinih primitiva.
  - *MapMessage* kada tijelo sadrži skup parova ime-vrijednost.
  - *ObjectMessage* sadrži Java objekt.
  - *ByteMessage* za tijelo koje sadrži niz neinterpretiranih *byte*-ova.

Kvaliteta usluge

- Vezana uz garanciju isporuke poruke
  - najviše jednom (*at-most-once*) – ne postoje mehanizmi koji osiguravaju isporuku poruke u slučaju ispada
  - barem jednom (*at-least-once*) – postoje mehanizmi koji će u slučaju ispada ponoviti operaciju, moguće je da će primatelj primiti poruku više puta
  - sigurno jednom (*exactly once*) – primatelj će primiti poruku samo jednom
- JMS podržava perzistentne i neperzistentne poruke
- posebni JMS pretplatnici (*durable subscriber*) se mogu odjaviti iz sustava i ponovo prijaviti u sustav, primiti će sve perzistentne poruke objavljene u međuvremenu

Queue Sender (1)

1. **Perform a JNDI lookup of the ConnectionFactory and Queue**:

```
/* Create a JNDI API InitialContext object if none exists yet. */
   Context jndiContext = null;
   try {
     jndiContext = new InitialContext();
   } catch (NamingException e) {
     System.out.println("Could not create JNDI API " + "context: " +
   e.toString());
     System.exit(1);
   }

/* Look up connection factory and destination. If either does not
   exist, exit. */
   QueueConnectionFactory connectionFactory = null;
   Queue queue = null;
   try {
     connectionFactory = (QueueConnectionFactory)
      jndiContext.lookup("jms/QueueConnectionFactory");
     queue = (Queue) jndiContext.lookup("queue");
     }
   } catch (Exception e) {
     System.out.println("JNDI API lookup failed: " + e.toString());
     e.printStackTrace();
     System.exit(1);
   }
```

Queue Sender (2)

2. Create a Connection and a Session:

```
QueueConnection connection =
  connectionFactory.createQueueConnection();
QueueSession session =
   connection.createQueueSession(false,
   Session.AUTO_ACKNOWLEDGE);
```

3. Create a QueueSender and a TextMessage:

```
QueueSender sender =
   session.createSender(queue);
TextMessage message = session.createTextMessage();
```

Queue Sender (3)

4. Send one or more messages to the queue:

```
for (int i = 0; i < NUM_MSGS; i++) {
  message.setText("This is message " + (i + 1));
  System.out.println("Sending message: " +
    message.getText());
  sender.send(message);
}
```

5. Send an empty control message to indicate the end of the message stream. Sending an empty message of no specified type is a convenient way to indicate to the consumer that the final message has arrived.

```
sender.send(session.createMessage());
```

Queue Sender (4)

6. Close the connection in a finally block, automatically closing the session and QueueSender:

```
} finally {
  if (connection != null) {
    try {
      connection.close();
    } catch (JMSException e) {}
  }
}
```

Queue Receiver (1)

1. Performs a JNDI lookup of the ConnectionFactory and Queue.
2. Creates a Connection and a Session.
3. Creates a QueueReceiver:

```
QueueReceiver receiver =
   session.createReceiver(queue);
```

4. Starts the connection, causing message delivery to begin:

```
connection.start();
```

Queue Receiver (2)

5. Receives the messages sent to the destination until the end-of-message-stream control message is received:

```
while (true) {
     Message m = receiver.receive();
     if (m != null) {
       if (m instanceof TextMessage) {
         message = (TextMessage) m;
         System.out.println("Reading message: " +
           message.getText());
       } else {
         break;
       }
     }
   }
```

- Since the control message is not a TextMessage, the receiving program terminates the while loop and stops receiving messages after the control message arrives.

6. Closes the connection in a finally block, automatically closing the session and QueueReceiver.

TopicPublisher

1. Perform a JNDI lookup of the TopicConnectionFactory and Topic.
2. Create a TopicConnection and a TopicSession.
3. Create a TopicPublisher and a TextMessage.
4. Send one or more messages to the topic.
5. Send an empty control message to indicate the end of the message stream.
6. Close the connection in a finally block, automatically closing the session and TopicPublisher.

TopicSubscriber (1)

1. Perform a JNDI lookup of the TopicConnectionFactory and Topic.
2. Create a TopicConnection and a TopicSession.
3. Create a TopicSubscriber.
4. Create an instance of the TextListener class and registers it as the message listener for the TopicSubscriber:

```
listener = new TextListener();
subscriber.setMessageListener(listener);
```

5. Start the connection, causing message delivery to begin.

TopicSubscriber (2)

6. Listen for the messages published to the topic, stopping when the user types the character q or Q:

```
System.out.println("To end program, type Q or q, " +
  "then <return>");
inputStreamReader = new InputStreamReader(System.in);
while (!((answer == 'q') || (answer == 'Q'))) {
  try {
    answer = (char) inputStreamReader.read();
  } catch (IOException e) {
    System.out.println("I/O exception: "
      + e.toString());
  }
}
```

7. Close the connection, which automatically closes the session and TopicSubscriber.

Message Listener

1. When a message arrives, the onMessage method is called automatically.
2. The onMessage method converts the incoming message to a TextMessage and displays its content. If the message is not a text message, it reports this fact:

```
public void onMessage(Message message) {
  TextMessage msg = null;

  try {
    if (message instanceof TextMessage) {
      msg = (TextMessage) message;
      System.out.println("Reading message: " +
        msg.getText());
    } else {
      System.out.println("Message is not a " +
        "TextMessage");
    }
  } catch (JMSException e) {
    System.out.println("JMSException in onMessage(): " +
      e.toString());
  } catch (Throwable t) {
    System.out.println("Exception in onMessage():" +
      t.getMessage());
  }
}
```

Literatura - JMS

- J2EE, **Java Message Service (JMS)**
  http://java.sun.com/products/jms/
- **Java Message Service Tutorial**
  http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html
  (Chapter 33: The Java Message Service API)