



Tehnologii Java

Curs -

Cristian Frăsinaru

`acf@infoiasi.ro`

Facultatea de Informatică

Universitatea "Al. I. Cuza" Iași



Comunicarea prin mesaje

Cuprins

- Introducere
- Arhitectura sistemelor bazate pe mesaje
- Componentele JMS
- Modele de comunicare
- Modelul de programare JMS API
- Exemplu



Introducere



Ce reprezintă mesageria ?

Mesageria = Formă de comunicare bazată pe mesaje între componente sau aplicații (clienți). Caracteristici:

- **asincronă**
- **loosely-coupled** (spre deosebire de RMI care este *tightly-coupled*)
- clienții comunică prin intermediul unor *agenți*

JMS



JMS = Java Message Service

- API Java pentru crearea trimiterea, primirea, citirea de mesaje în mod *sigur, relaxat și asincron*.
- Scopul inițial: uniformizarea accesului la *messaging-oriented middleware (MOM)*
- Starea actuală: integrată în arhitectura Java EE
- Implementări: GlassFish, JBoss, IBM Websphere, BEA WebLogic, ActiveMQ (Apache), OpenJMS (Sun), etc.





Arhitectura sistemelor bazate pe mesaje



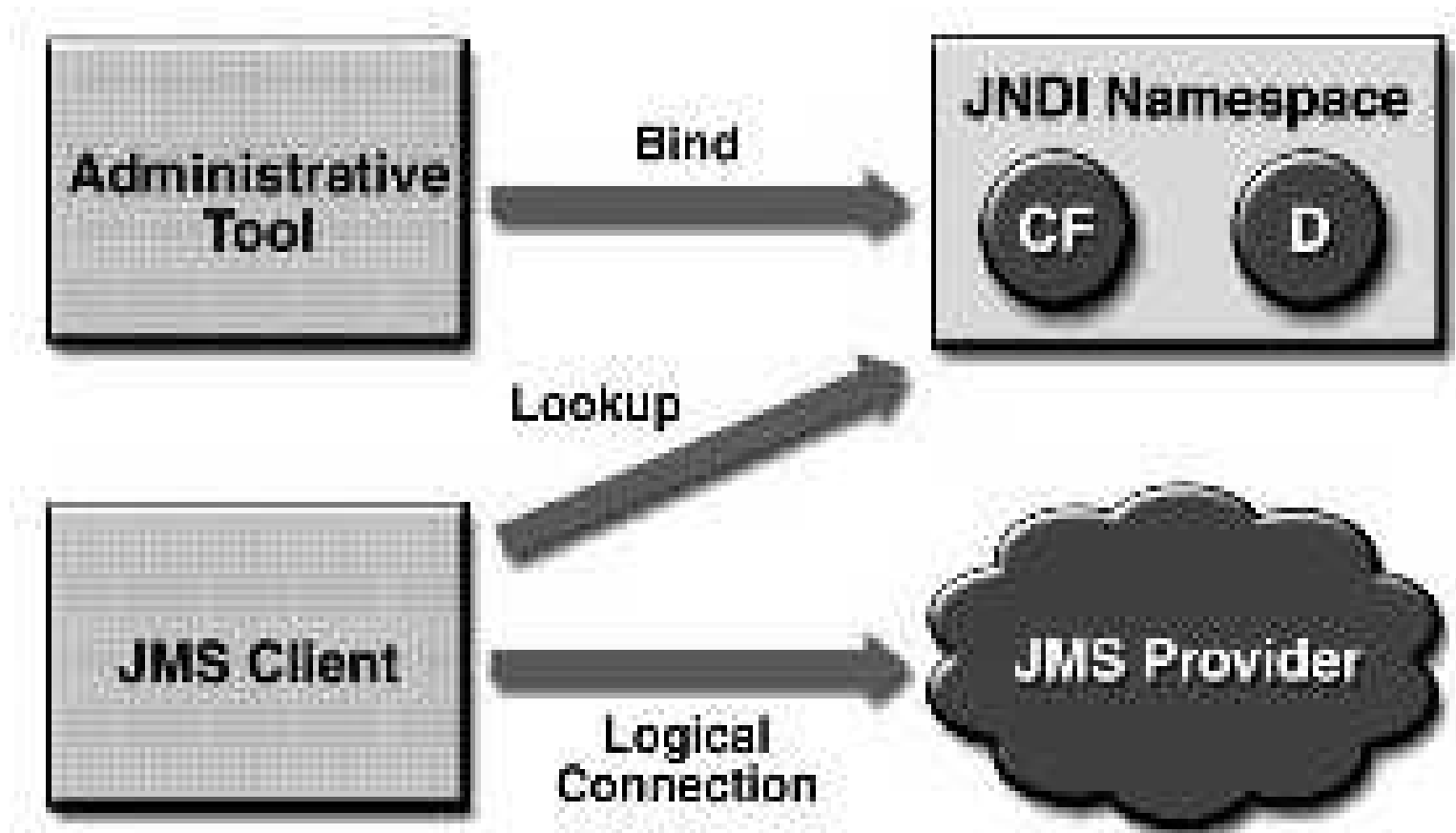
Conceptele de bază

- **Mesaj**
 - **tip:** comanda, transfer de date, eveniment, etc.
 - **adresare**
 - **formatare, secvențializare**
 - **durata de viață**
- **Canal de comunicație**
- **Capetele comunicării**
- **Livrare în mai mulți pași:** *Pipes & Filters*
- **Rutare:** *MessageRouter*
- **Transformări:** *MessageTransformer*

Componentele unui sistem JMS

- **Furnizor** - sistem de mesagerie care implementează specificațiile Java EE
- **Clienți** - aplicații sau componente care produc sau consumă mesaje
- **Mesaje** - obiectele de transport a informației între clienți
- **Administratori** - obiecte cu rol administrativ preconfigurate în sistem ce pot fi utilizate de clienți:
 - Destinații (destination factory)
 - Conexiuni (connection factory)

Imagine de ansamblu



Modele de comunicare

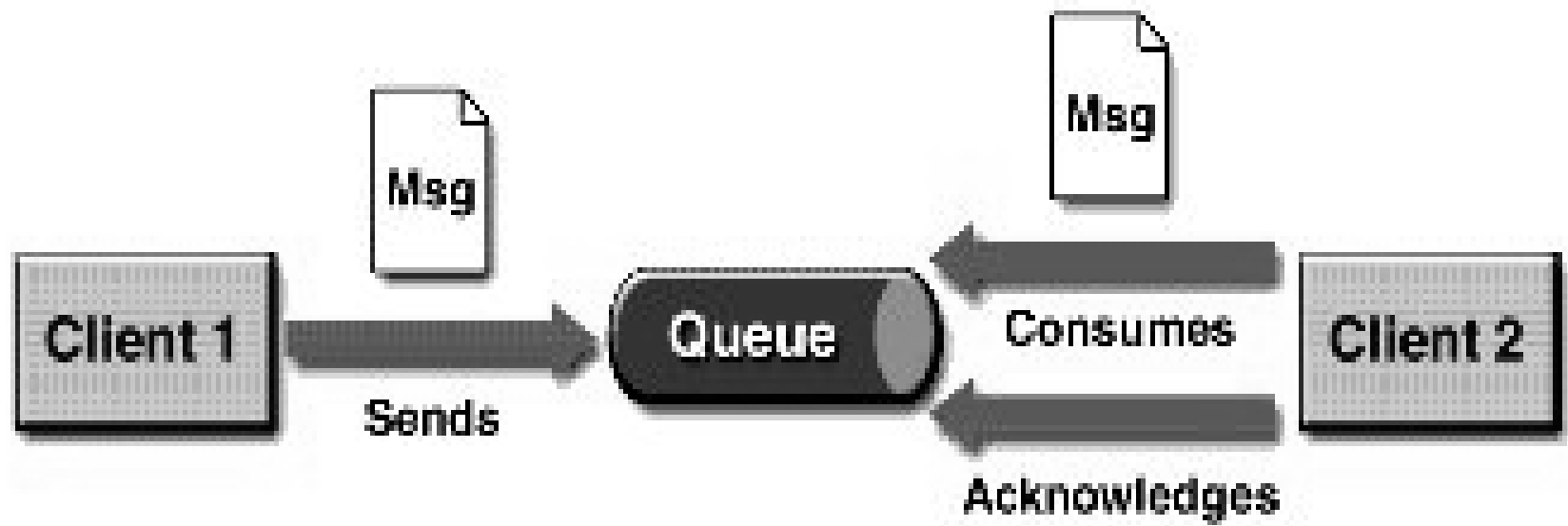
● Point-to-Point (Queuing)

- Producător - *queue* - Consumator
- Un mesaj are un singur consumator
- Consumatorul confirmă primirea mesajelor

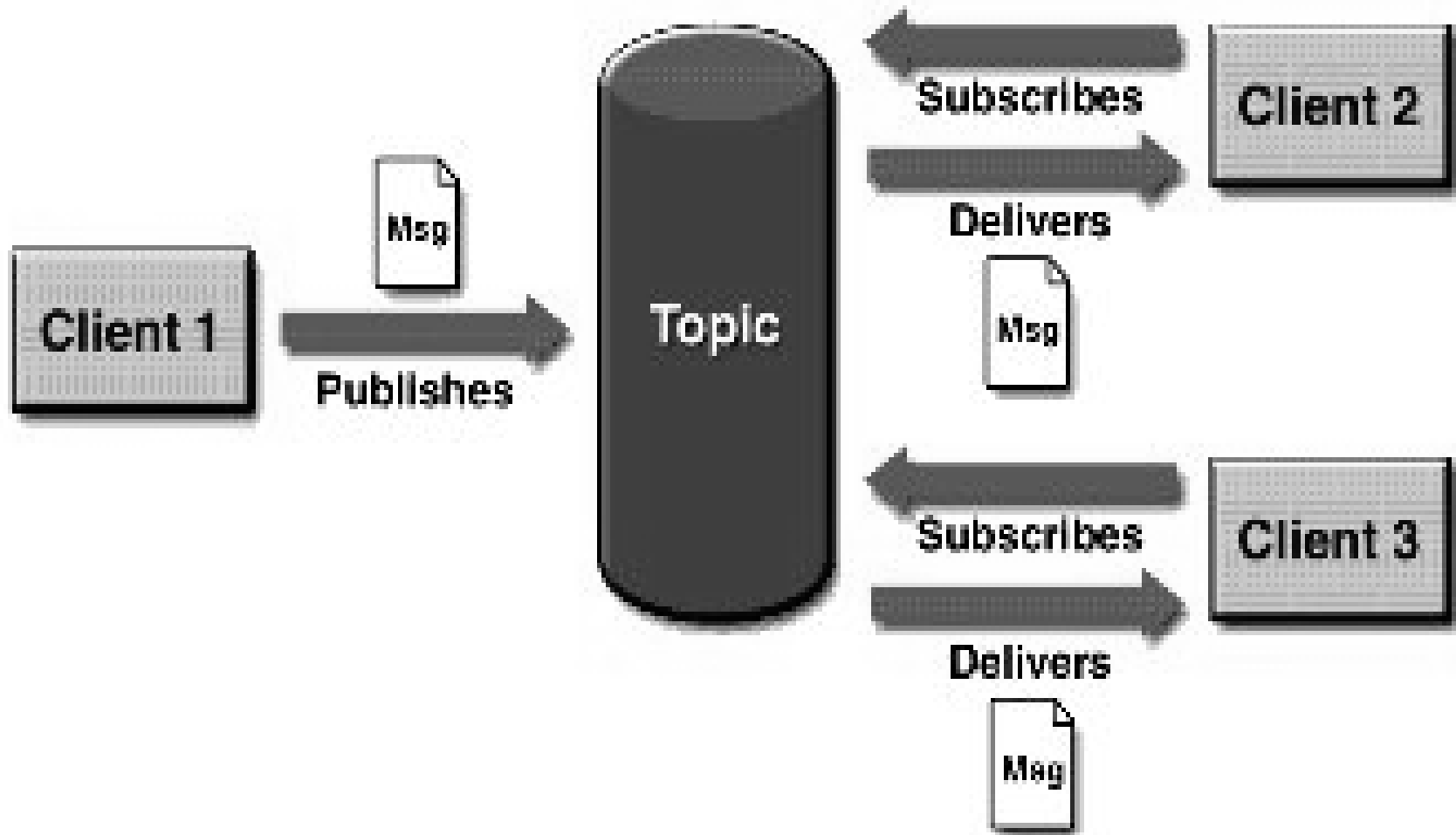
● Publish/Subscribe

- Producător - *topic* - Consumator
- Un mesaj poate avea mai mulți consumatori

Modelul *Point-to-Point*



Modelul *Publish/Subscribe*





"Consumarea" mesajelor

Conceptual mesageria este **asincronă** = nu există o dependență temporală între producerea și consumarea unui mesaj. JMS definește însă următorii termeni legați de consumul unui mesaj:

- *Consum sincron (blocant)* - clientul apelează explicit metoda `receive`
- *Consum asincron (ne-blocant)* - clientul înregistrează un *message listener* și definește codul metodei `onMessage`

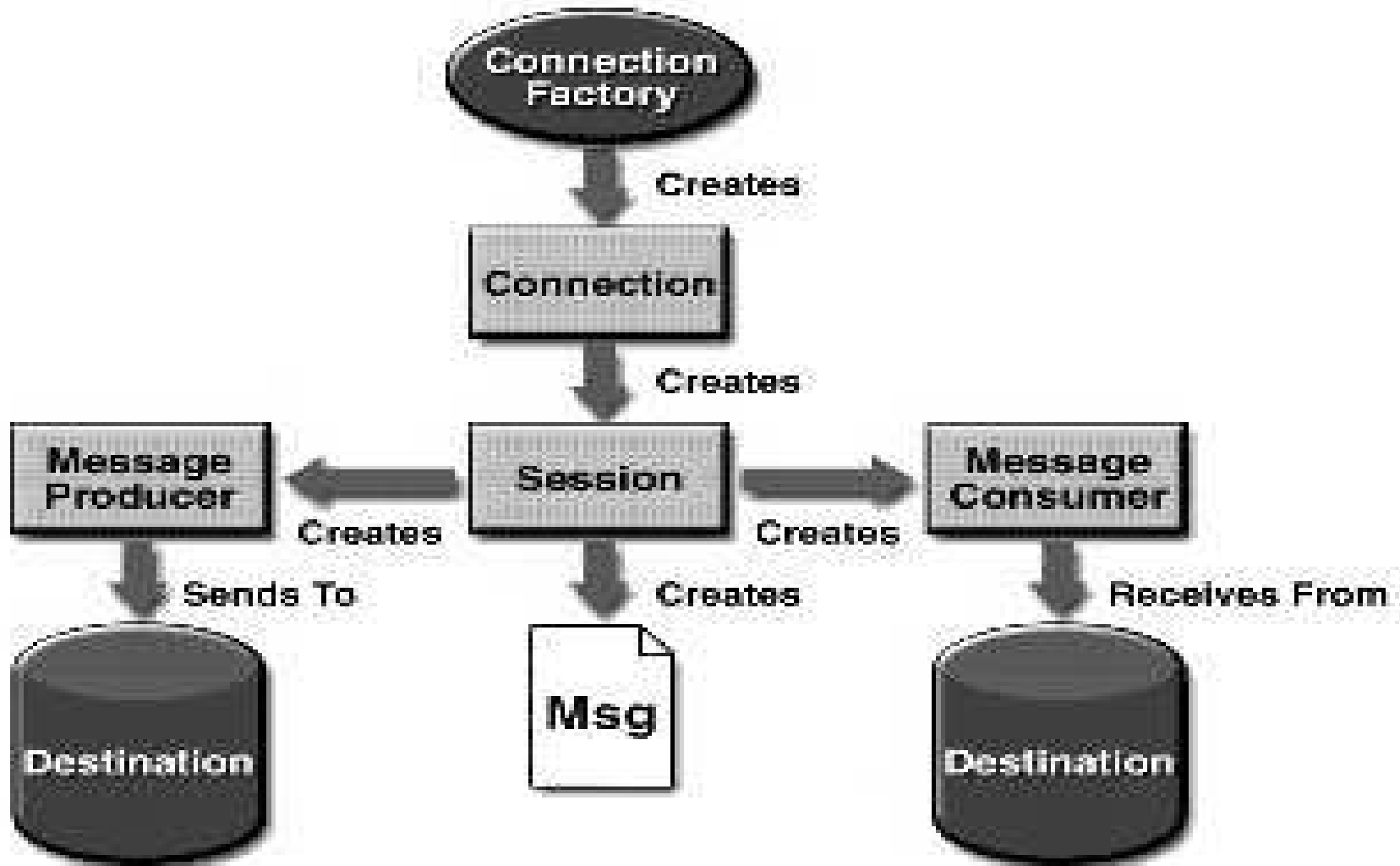




Modelul de programare



Imagine de ansamblu



Obiecte administrative

- Pot fi de unul din tipurile:
 - **ConnectionFactory**
 - **DestinationFactory**
- Sunt create uzual de administratorul sistemului prin instrumente specifice serverului de aplicații.
- Sunt accesate prin JNDI

```
Context ctx = new InitialContext();
ConnectionFactory connectionFactory = (ConnectionFactory)
    ctx.lookup("MyConnectionFactory");
Destination myDest = (Destination) ctx.lookup("MyTopic");
Queue myQueue = (Queue) ctx.lookup("MyQueue");
```

Conexiuni



Conexiune =

- abstract: modul de comunicare cu furnizorul JMS
- concret: (poate fi) socket TCP/IP legat de un serviciu de tip daemon

```
Connection connection = connectionFactory.createConnection();  
...  
    connection.start();  
    ...  
    connection.stop();  
...  
connection.close();
```



Sesiuni

Sesiune = Context pentru producerea și consumarea mesajelor ce rulează într-un fir propriu de execuție.

Responsabile cu:

- Crearea obiectelor de tip producător, consumator, mesaj
- Serializarea apelurilor `onMessage` din cadrul obiectelor *message listener*
- Crearea de *tranzacții* (unități atomice de lucru)

```
boolean transacted = false;  
Session session = connection.createSession(transacted,  
    Session.AUTO_ACKNOWLEDGE);
```

Producători de mesaje

● Crearea producătorului

```
MessageProducer producer = session.createProducer(dest);  
MessageProducer producer = session.createProducer(queue);  
MessageProducer producer = session.createProducer(topic);  
MessageProducer anon_prod = session.createProducer(null);
```

● Crearea mesajului

● Trimiterea mesajului

```
producer.send(message);  
anon_prod.send(queue, message);
```

Consumatori de mesaje

● Crearea producătorului

```
MessageConsumer consumer = session.createConsumer(dest);  
MessageConsumer consumer = session.createConsumer(queue);  
MessageConsumer consumer = session.createConsumer(topic);
```

● Consumarea mesajului (cu blocare)

```
Message m = consumer.receive();
```

```
Message m = consumer.receive(1000);  
// time out after a second
```

Comunicarea prin evenimente

Obiectele de tip `MessageListener` oferă o modalitate **neblocantă** de consumarea mesajelor.

```
consumer.setMessageListener(  
    new MessageListener() {  
        public void onMessage(Message message) {  
            // A aparut un mesaj  
            // Cod specific de tratare a mesajului  
        }  
    });
```

Mesaje

Un *mesaj* compus din:

- **Antet** - Conține informații necesare pentru **identificare și rutare**; aceleași câmpuri indiferent de tipul mesajului;
- **Proprietăți** - Valori specifice aplicațiilor ce pot fi utilizate la **filtrarea** automata a mesajelor;
messageSelector: "JMSType = 'car' AND color = 'blue' AND weight > 2500"
- **Conținut**

Tipuri de mesaje

- **Stream**
- **Map**
- **Text**
- **Object**
- **Bytes**

```
TextMessage message = session.createTextMessage();  
message.setText(msg_text);  
producer.send(message);
```

```
Message m = consumer.receive();  
TextMessage message = (TextMessage) m;  
System.out.println(message.getText());
```

Configurări simple

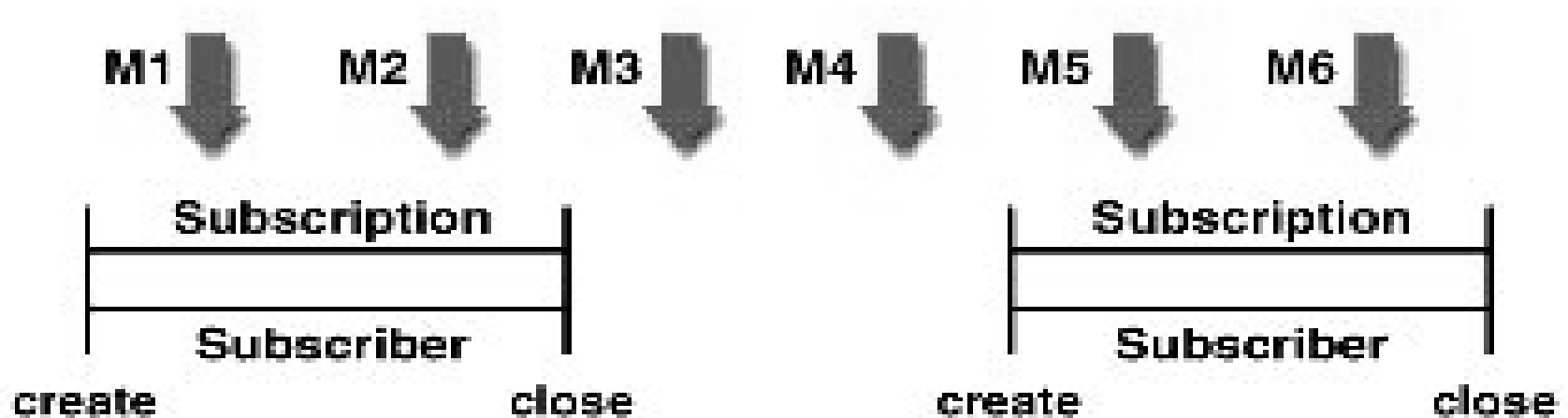
- Controlul mecanismului de **confirmare** a primirii mesajelor
AUTO, CLIENT, DUPS_OK_ACKNOWLEDGE
- Setarea **nivelurilor de prioritate** 0 – 9
- Specificarea **persistenței** mesajelor
PERSISTENT, NOT_PERSISTENT
- Specificarea **duratei de viață** a mesajelor; mesajele pot **expira**
- Crearea unor **destinații temporare**

Configurări avansate

- Configurarea modelului Publisher/Subscriber
 - subscripții **non-durabile**
 - subscripții **durabile**
- Folosirea **tranzacțiilor** (unități atomice de lucru)
 - `commit` - toate mesajele produse sunt trimise, toate mesajele consumate sunt confirmate
 - `rollback` - toate mesajele produse sunt distruse, toate mesajele consumate sunt retrimise

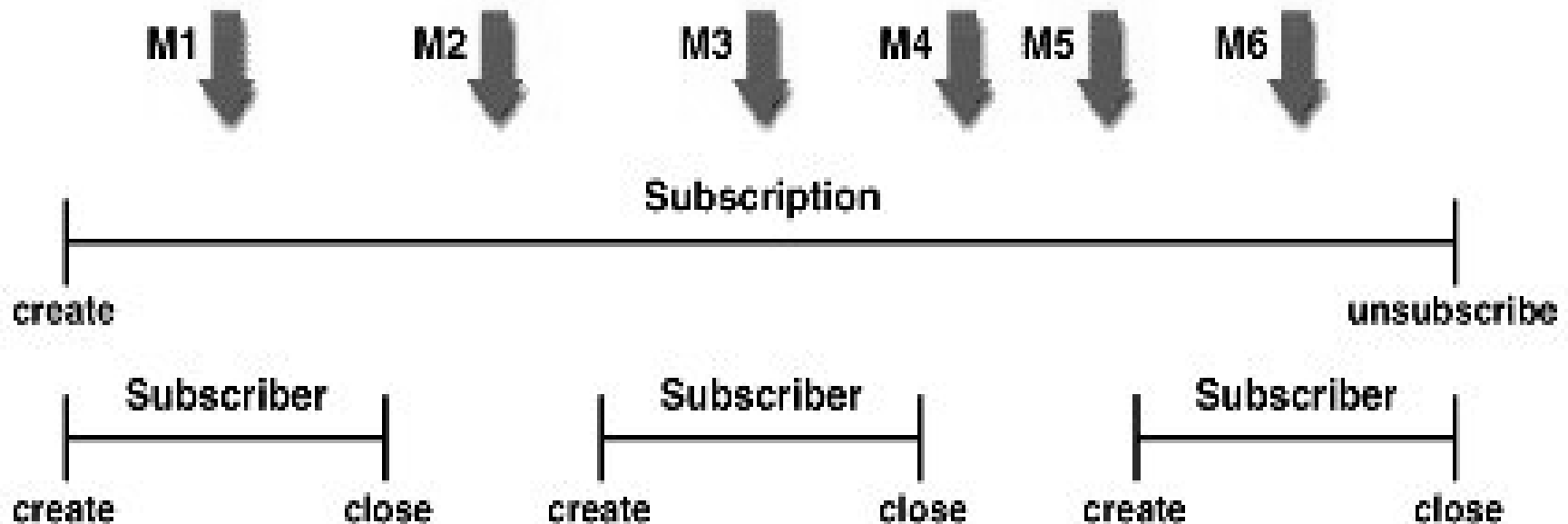
Subscripții non-durabile

Mesajele publicate în perioada în care consumatorul nu este activ sunt pierdute.



Subscripții durabile

Mesajele publicate în perioada în care consumatorul nu este activ sunt memorate într-o listă și trimise la reactivarea acestuia.



Dependency Injection

```
public class MyServlet extends HttpServlet {  
  
    @Resource(mappedName="jms/ConnectionFactory")  
    private static ConnectionFactory connectionFactory;  
  
    @Resource(mappedName="jms/Queue")  
    private static Queue queue;  
  
    @Resource(mappedName="jms/Topic")  
    private static Topic topic;  
  
    ...  
}
```