Modèle à messages

Daniel Hagimont

IRIT/ENSEEIHT
2 rue Charles Camichel - BP 7122
31071 TOULOUSE CEDEX 7

Daniel.Hagimont@enseeiht.fr
http://hagimont.perso.enseeiht.fr

Modèle à messages Introduction

- Modèle client-serveur
  - Appels synchrones
  - Bon pour des composants fortement couplés
  - Désignation explicite du destinataire
  - Connexion 1-1
- Modèle à message
  - Communications asynchrones
  - Désignation anonyme (ex : annonce sur un newsgroup)
  - Connexion 1-N

Modèle à messages Introduction

- Exemple d'application
  - Surveillance des équipements d'un réseau
  - Évolution des équipements et de l'état des équipements
- Solution client-serveur
  - Interrogation régulière
- Solution messages
  - Émission de notifications des changements
  - Les administrateurs s'abonnent à ces notifications

Intergiciels à messages … utilisé tous les jours

- Les forums électroniques (News)
  - technologie pull
  - le consommateur s 'abonne à un forum
  - le producteur publie une information dans un forum
  - le consommateur va lire le contenu du forum quand il le souhaite
- Le courrier électronique
  - technologie push
  - listes de diffusion (multicast - publish/subscribe)
  - le consommateur s 'abonne à une liste de diffusion
  - le producteur envoie un message à tous les abonnées de la liste
  - le consommateur reçoit les messages sans avoir à faire d'action

- Asynchrone
- Anonyme
- 1-N

Intergiciels à messages
Principes directeurs

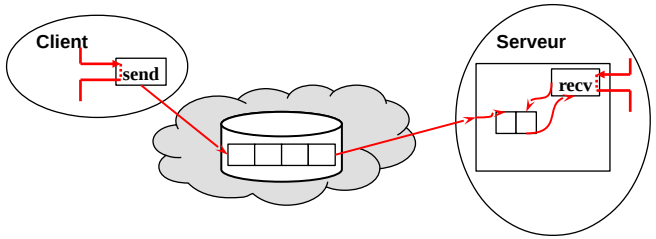
- Message Passing (communication par messages)
- Message Queuing (communication par file de message)
- Publish/Subscribe (communication par abonnements)
- Events (communication événementielle)

Intergiciels à messages
Message passing

- Communication par message
  - dans une architecture classique : sockets
  - dans un environnement de programmation parallèle : PVM, MPI
  - Dans d'autres environnements : portes (ex : Mach)

Intergiciels à messages
Message Queuing

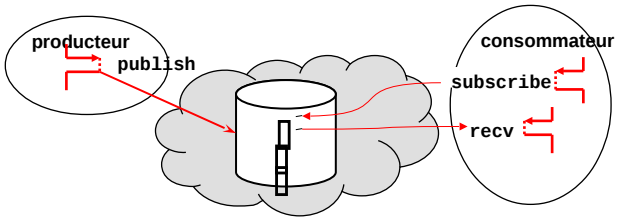
- Queue de messages
  - persistantes ⇒ asynchronisme et fiabilité



- Indépendance de l'émetteur et du destinataire
  - Le destinataire n'est pas forcément actif

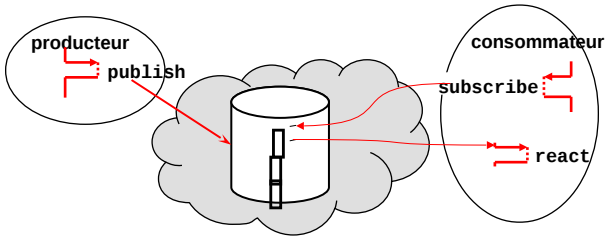
Intergiciels à messages
Publish/Subscribe

- Désignation anonyme
  - L'émetteur envoie un message
    - Basé sur un sujet (subject-based)
    - Basé sur un contenu (content-based)
  - Le récepteur s'abonne (à un sujet ou un contenu)
- Communication 1-N
  - Plusieurs récepteurs peuvent s'abonner



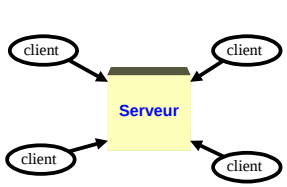
Intergiciels à messages
Events

- Concepts de base : événements, réactions (traitements associés à l'occurrence d'un événement)
- Principe d'attachement : association dynamique entre un type d'événement et une réaction

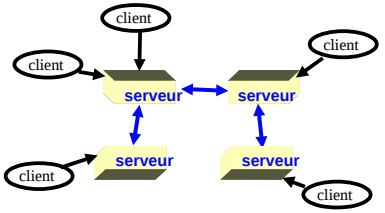


- Valable pour du Message Passing, du Message Queuing, ou du Publish/Subscribe

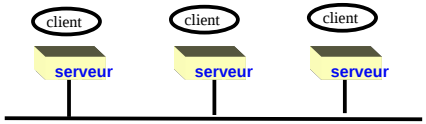
Intergiciels à messages
Implantations



Serveur centralisé
(Hub & spoke)



Serveur réparti
(Snow flake)



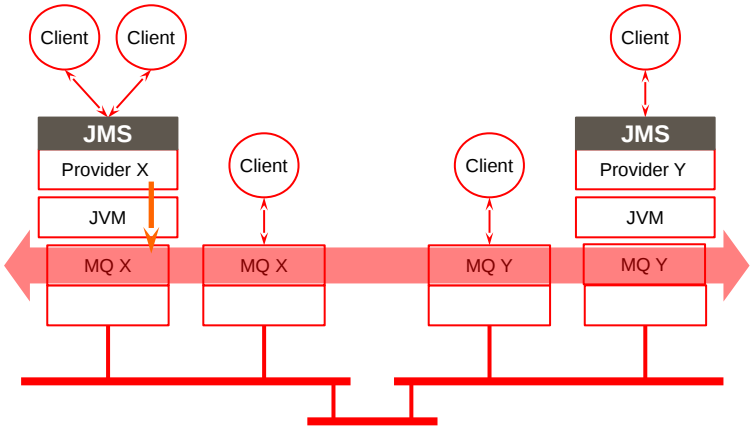
Bus logiciel

Java Message Service

- JMS : API Java d'accès uniforme aux systèmes de messagerie
  - IBM, Oracle,
  - Novell, Sybase, Tibco

  - Message Queue
  - Publish/Subscribe
  - Evénements

JMS : une interface
(portabilité, pas Interopérabilité)



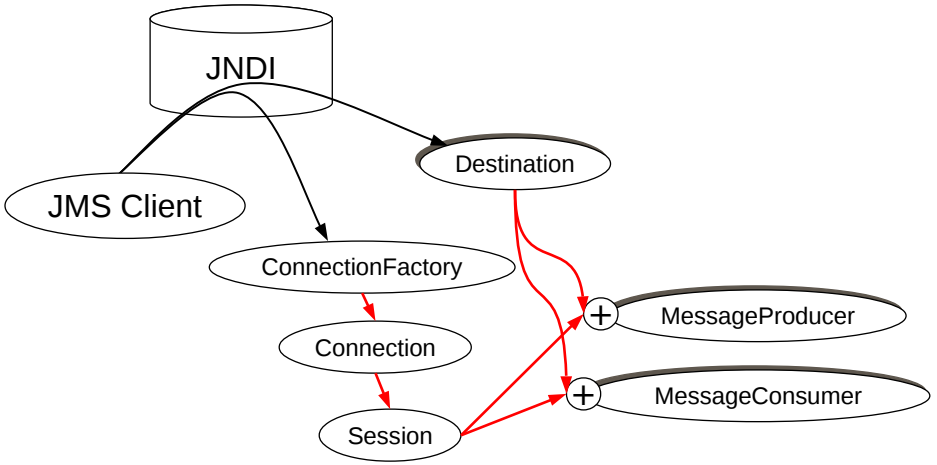
Interopérabilité : AMQP (Advanced Message Queuing Protocol)

Interface JMS

- *ConnectionFactory* : objet pour créer une connexion avec un serveur JMS
- *Connection* : une connexion active avec un serveur JMS
- *Destination* : une destination ou une source
- *Session* : un contexte mono-thread pour émettre ou recevoir
- *MessageProducer* : un objet pour émettre dans une session
- *MessageConsummer* : un objet pour recevoir dans une session

- Les implantations dépendent des fournisseurs ...

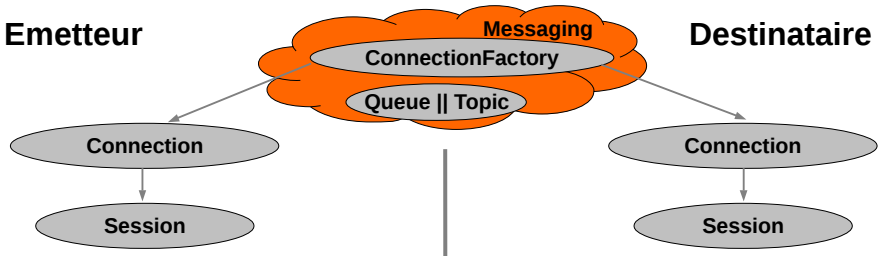
JMS - Architecture



Interfaces PTP et P/S

	Point-To-Point	Publish/Subscribe
ConnectionFactory	*QueueConnectionFactory*	*TopicConnectionFactory*
Connection	*QueueConnection*	*TopicConnection*
Destination	*Queue*	*Topic*
Session	*QueueSession*	*TopicSession*
MessageProducer	*QueueSender*	*TopicPublisher*
MessageConsumer	*QueueReceiver*	*TopicSubscriber*

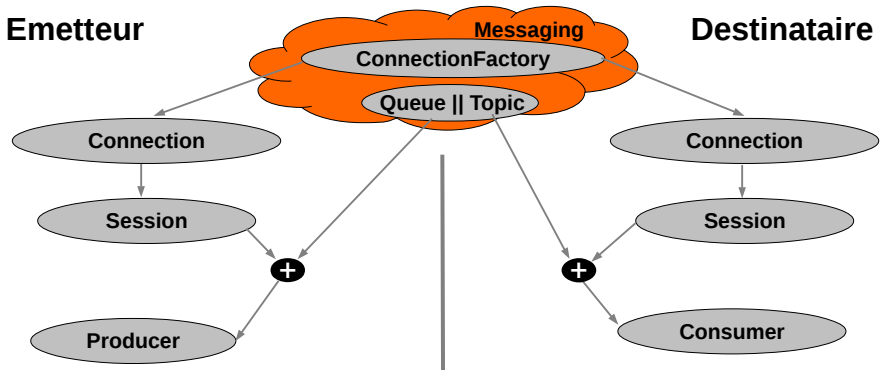
JMS - initialization




```
ConnectionFactory connectionFactory =
        new ActiveMQConnectionFactory(ActiveMQConnection.DEFAULT_BROKER_URL);
Connection connection = connectionFactory.createConnection();
connection.start();
Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);

Destination destination = session.createQueue("myQueue");
Destination destination = session.createTopic("MyTopic");
```

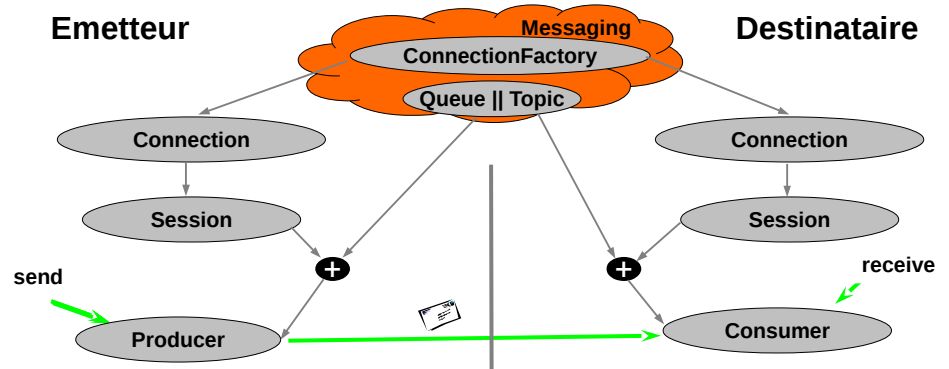
JMS – producer / consumer




```
MessageProducer producer =
      session.createProducer(destination);
```

```
MessageConsumer consumer =
     session.createConsumer(destination);
```

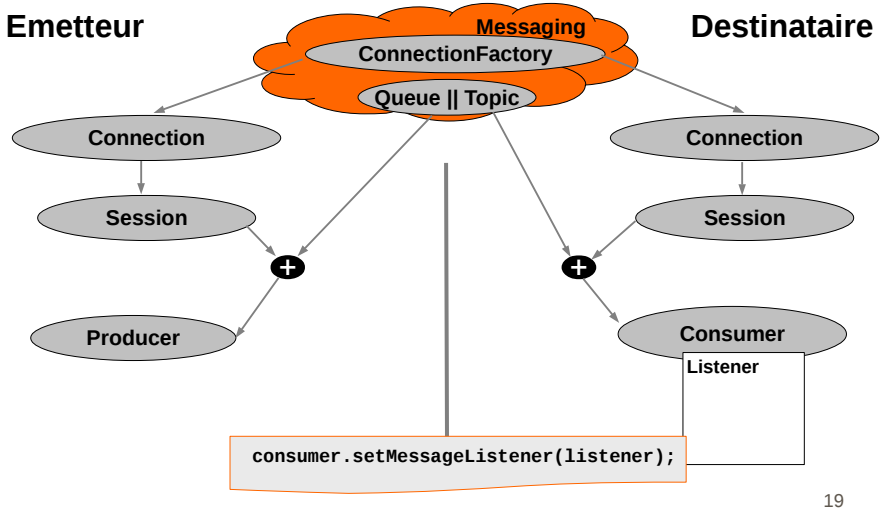

JMS - communication




```
TextMessage msg =
    session.createTextMessage("...");
sender.send(msg);
```

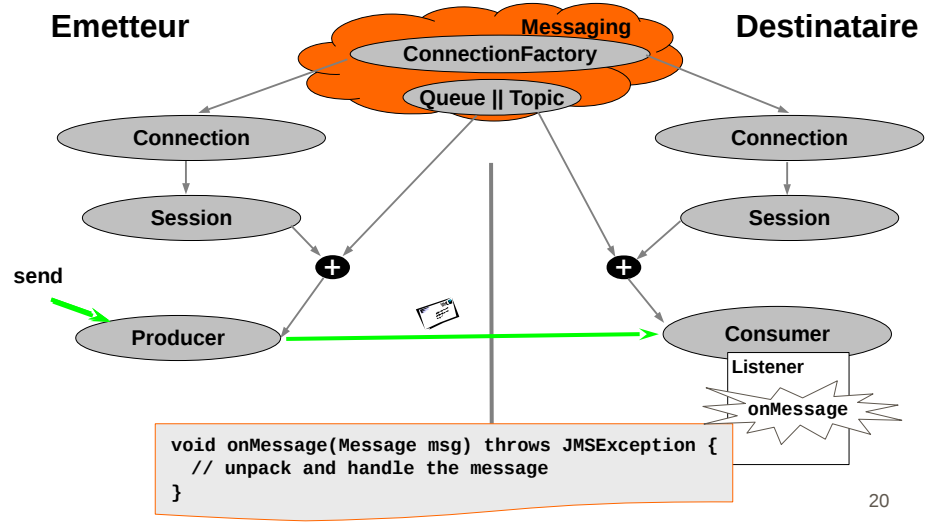
```
TextMessage m =
   (TextMessage)consumer.receive();
```


JMS - Listener




```
consumer.setMessageListener(listener);
```


JMS - Listener




```
void onMessage(Message msg) throws JMSException {
  // unpack and handle the message
}
```

JMS – les messages

- TextMessage (une chaîne de caractères)

```
String data;
TextMessage message = session.createTextMessage();
message.setText(data);
```

```
String data;
data = message.getText();
```

- BytesMessage (tableau de bytes)

```
byte[] data;
BytesMessage message = session.createByteMessage();
message.writeBytes(data);
```

```
byte[] data;
int length;
length = message.readBytes(data);
```


JMS – les messages

- MapMessage (une suite de paire nom-valeur)
  - une valeur est un type primitif

```
MapMessage message = session.createMapMessage();

message.setString("Name", "…");
message.setDouble("Value", doubleValue);
message.setLong("Time", longValue);
```

```
String name = message.getString("Name");
double value = message.getDouble("Value");
long time = message.getLong("Time");
```


JMS – les messages

- StreamMessage (série de valeurs)
  - une valeur est un type primitif
  - La lecture doit respecter l'ordre de l'écriture

```
StreamMessage message = session.createStreamMessage();

message.writeString("…");
message.writeDouble(doubleValue);
message.writeLong(longValue);
```

```
String name = message.readString();
double value = message.readDouble();
long time = message.readLong();
```


JMS – les messages

- ObjectMessage (objet sérialisé)

```
ObjectMessage message = session.createObjectMessage();

message.setObject(obj);
```

```
obj = message.getObject();
```

Conclusions

- Communication par messages
  - Modèle de programmation simple…
  - possédant de nombreuses extensions, variantes, …
    - bus logiciel à messages, modèles d’acteurs, plates-formes à agents, systèmes multi-agents, …
  - très utilisé pour relier entre eux des « outils », existants, indépendants, …
- Attention… la simplicité n'est qu’apparente
  - propagation et récupération des erreurs
  - outils de développement