NSY102
Conception de logiciels Intranet
MOM
Message-Oriented Middleware
en utilisant JMS
Java Message Service

Cnam Paris
jean-michel Douin, douin au cnam point fr
version 7 Avril 2016

Sommaire

- **Objectifs**
  - Quelques patrons issus de la bibliographie
  - Un langage de description
- **JMS une introduction**
  - La spécification JMS
    - Un ensemble d'interfaces
  - Une implémentation « Open » OpenJMS
- **Une démonstration**
  - Un exemple basique

Bibliographie utilisée

- **Tous les Patrons présentés sont extraits de, *Gregor Hohpe and Bobby Woolf***
  - http://www.enterpriseintegrationpatterns.com/eaipatterns.html
  - **http://www.enterpriseintegrationpatterns.com/docs/EnterpriseIntegrationPatterns_HohpeWoolf_ch03.pdf**
  - **http://www.enterpriseintegrationpatterns.com/docs/jaoo_hohpeg_enterpriseintegrationpatterns.pdf**

- **JMS : L'indispensable tutoriel de Sun**
  - **http://java.sun.com/products/jms/tutorial/index.html**
  - **http://www.devx.com/Java/Article/20903/1954?pf=true**
  - **http://marine.edu.ups-tlse.fr/~torguet/cours/m2miage/jms1.1/doc/api/**

- **Joram**
  - http://www;objectweb.org
- **La présentation de l'EPFL, (École polytechnique de Lausannne)**
  - **http://lsrwww.epfl.ch/webdav/site/lsrwww/shared/Enseignement/SysRep07/Slides/JMS.pdf**
- **Architecture réparties en Java, de Annick Fron chez Dunod**
  - **http://www.dunod.com/livre-dunod-9782100511419-architectures-reparties-en-java.html**

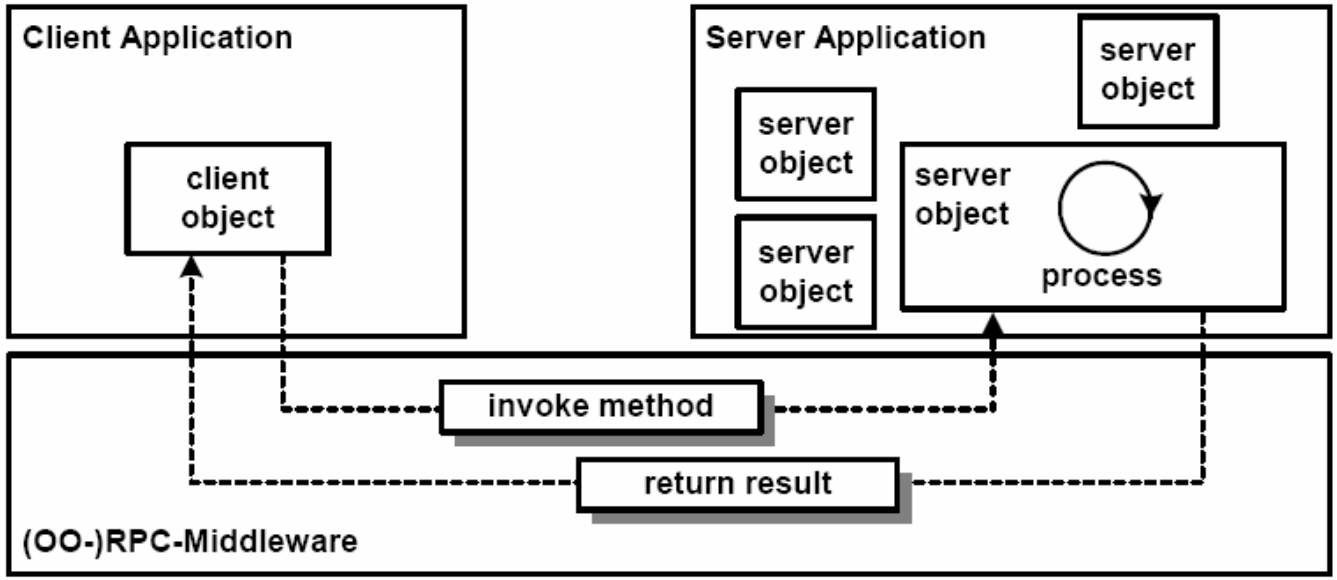
- **Using publish/subscribe messaging to distribute MIDI commands**
  - http://www.ibm.com/developerworks/websphere/library/techarticles/0412_ibbotson/0412_ibbotson.html

Pré-requis

- **Notions des patrons**
  - **Adaptateur,**
  - **Procuration,**
  - **Chaîne de responsabilités/Interceptor*,**
  - **Observateur/observé,**
  - **Fabrique.**

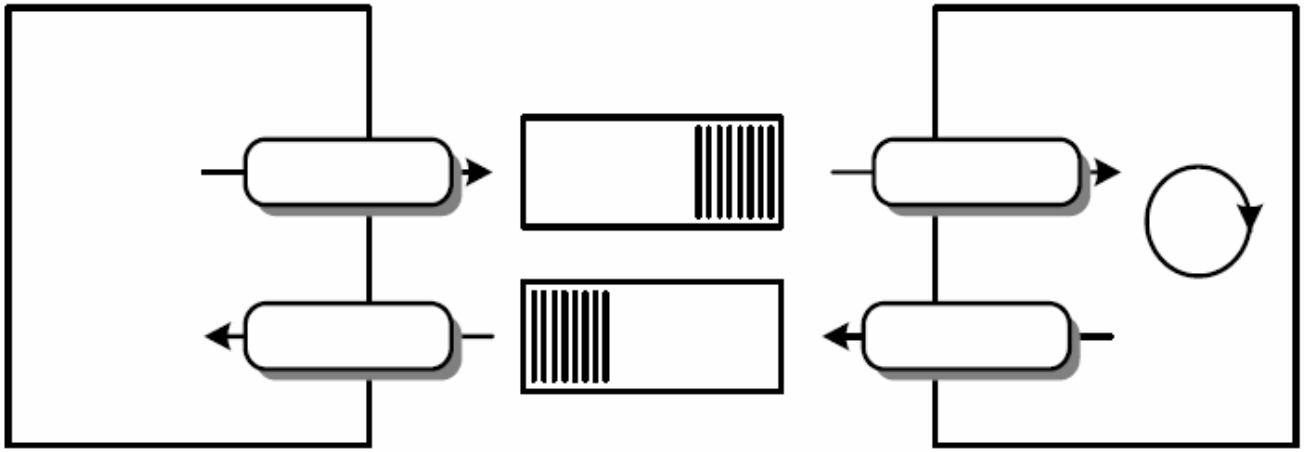
  - *** rappelé annexe**

Style RPC-OO/Messaging, schémas d'Uwe Zdun



- **RPC-OO**
  - java-rmi
  - .NET remoting
  - CORBA

- **Messaging**
  - JMS*
  - IBM WebSphere MQ
  - Microsoft MSMQ



* implementations de Java Message Service

Message-Oriented Middleware Objectifs

- **Envoi et réception de messages**
  - Réceptions synchrone et asynchrone
- **Modèles**
  - Point à point
  - Publish-subscribe
- **Fiabilité de la délivrance des messages**
  - Différents formats de messages
  - Persistance souhaitable
- **Indépendance des canaux de communication / applications**
- **Couplage faible assuré**
  - Les canaux de communications sont indépendants des applications
    - Référence aux canaux plutôt qu'aux adresses de machines
    - Serveur JMS, un courtier

Exemples … bien connus

- Les news d'internet, ou le forum de jfod …
  - **Enregistrement** d'un « client » à un sujet de discussion,
  - Un des « clients » décide de poster un message,
  - Les utilisateurs **à leur initiative** vont chercher l'information,
  - ***Publish-subscribe, mode pull***
- Le mail d'internet
  - Un émetteur **envoie** un message à un destinataire,
  - Le récepteur **reçoit** ce message (sans action particulière),
  - Les messages sont persistants,
  - ***Point à Point, synchrone, asynchrone***
- Les listes de diffusion, logiciels de causerie, (« chat »)
  - **Abonnement** d'un « client » à une liste de diffusion,
  - Un des « clients » décide de poster un message,
  - Tous les abonnés **reçoivent** ce message,
  - ***Publish-subscribe, mode push***

Quelques patrons

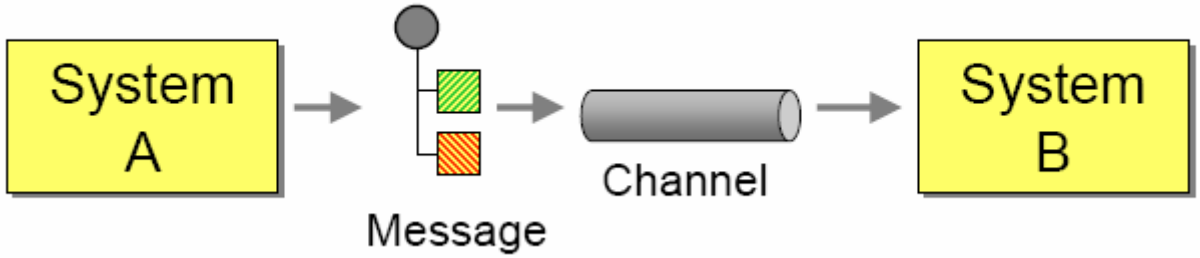
- **Patrons comme langage de description d'une architecture de type MOM**

- **Extraits de** http://www.enterpriseintegrationpatterns.com/eaipatterns.html

Et

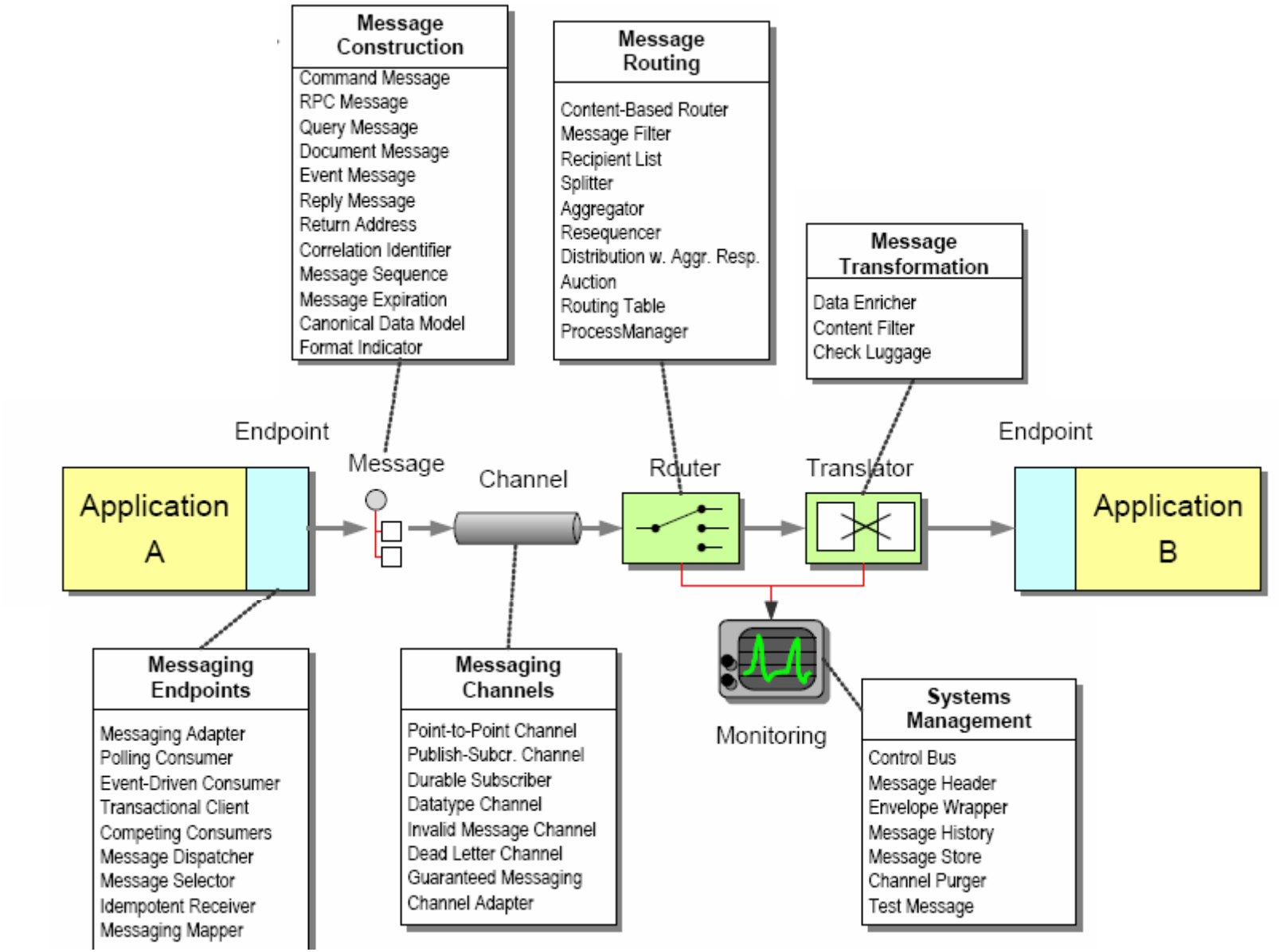
- Mark grand
  - MailBox, Retransmission, Connection Multiplexing, Publish-Subscribe, …

Notation empruntée



http://www.enterpriseintegrationpatterns.com/docs/jaoo_hohpeg_enterpriseintegrationpatterns.pdf

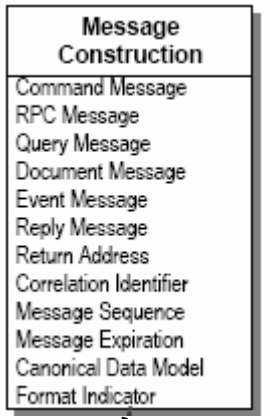
MOM, Les patrons (65) !, selon Gregor Hohpe

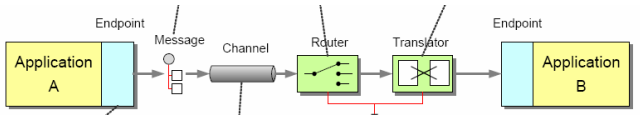


!

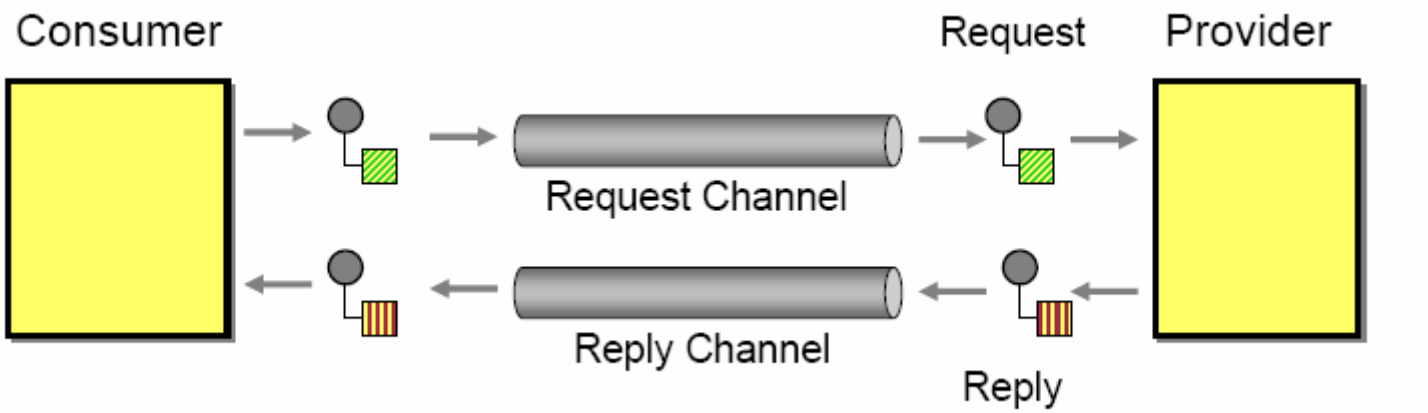
Message construction

- **Request Reply**
- **Return Address**
- **Correlation Identifier**
- **…**



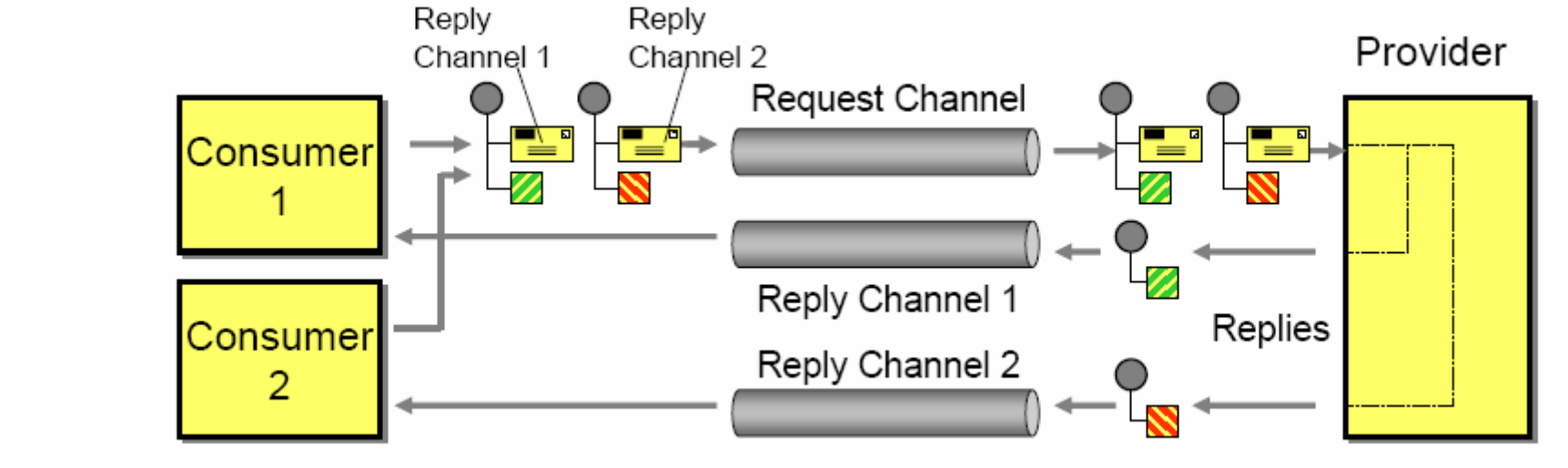


Patron Request-Reply



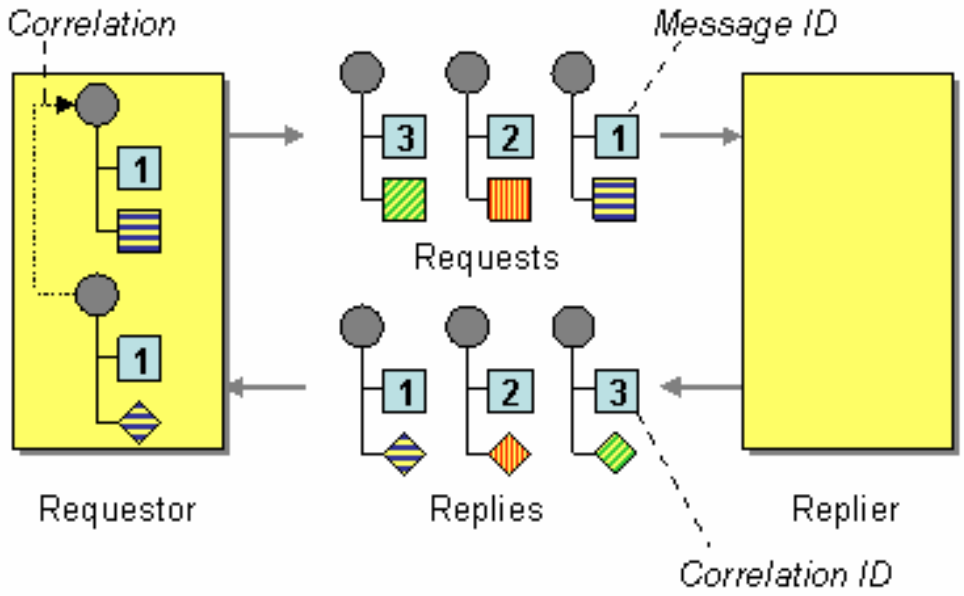
- **Analogue à l'architecture « Client/Serveur »**
- **Canaux de communications**
  - **Point à point**
  - **« Unidirectionnel »**
  - **Séparation des requêtes et de la réponse**

Patron Return Address



- **Plusieurs clients/consommateurs**
- **Identification du consommateur, « receveur »**
  - **Dans le contenu du message émis**

Patron Correlation Identifier



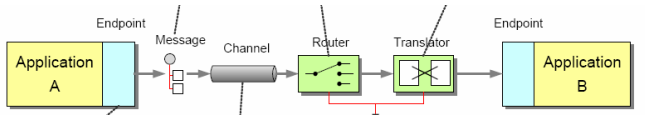
- **Le message contient un identificateur de *Corrélation***
  - Un nombre unique permettant l'identification et le bon acheminement du message
    - Message ID
    - GUID (Globally Unique ID)
    - Business key (e.g. ID de bon de commande, de facture)
  - Producteur copie l'ID dans le message retourné
  - Le client/consommateur peut ainsi corréler la requête et sa réponse

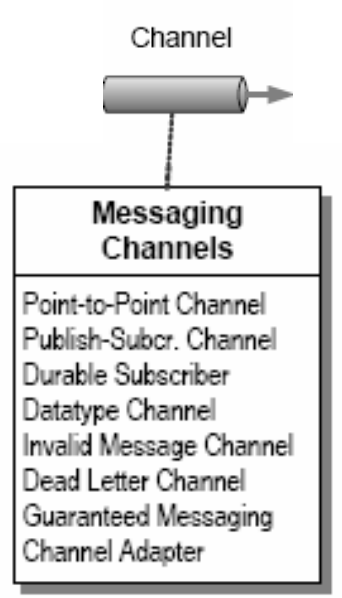
…MessageConstruction, fin de l'extrait

- **Discussion**
  - **Utile/inutile**
- **La suite ici**
  - http://www.enterpriseintegrationpatterns.com/eaipatterns.html

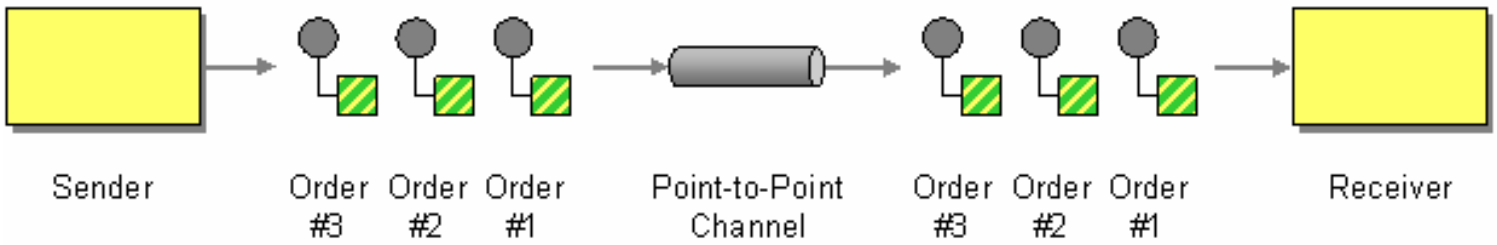
Message channels

- **Point à Point**
- **Publish Subscribe**
  - **Observateur/Observé ?**
- **Durable Subscriber**
- **Guaranteed Delivery**
- **…**



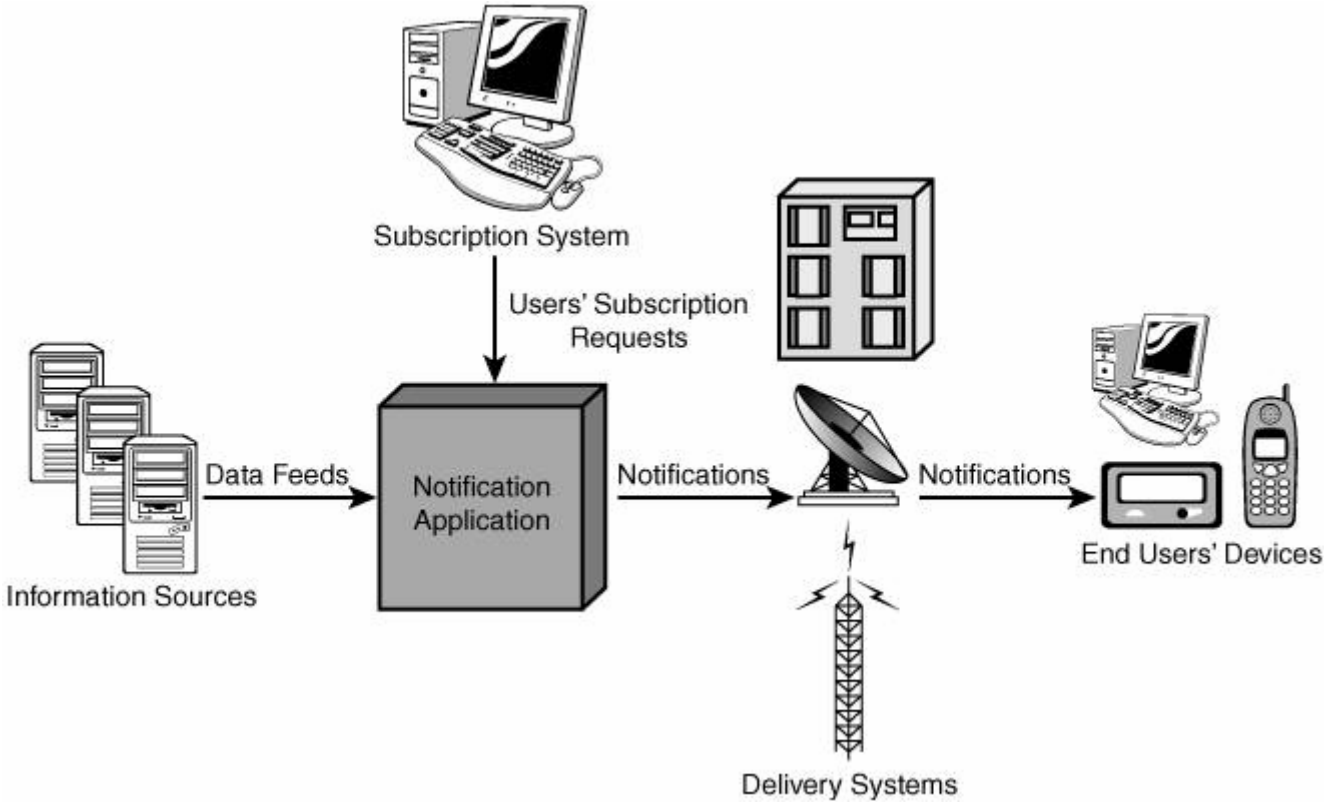


Point à point



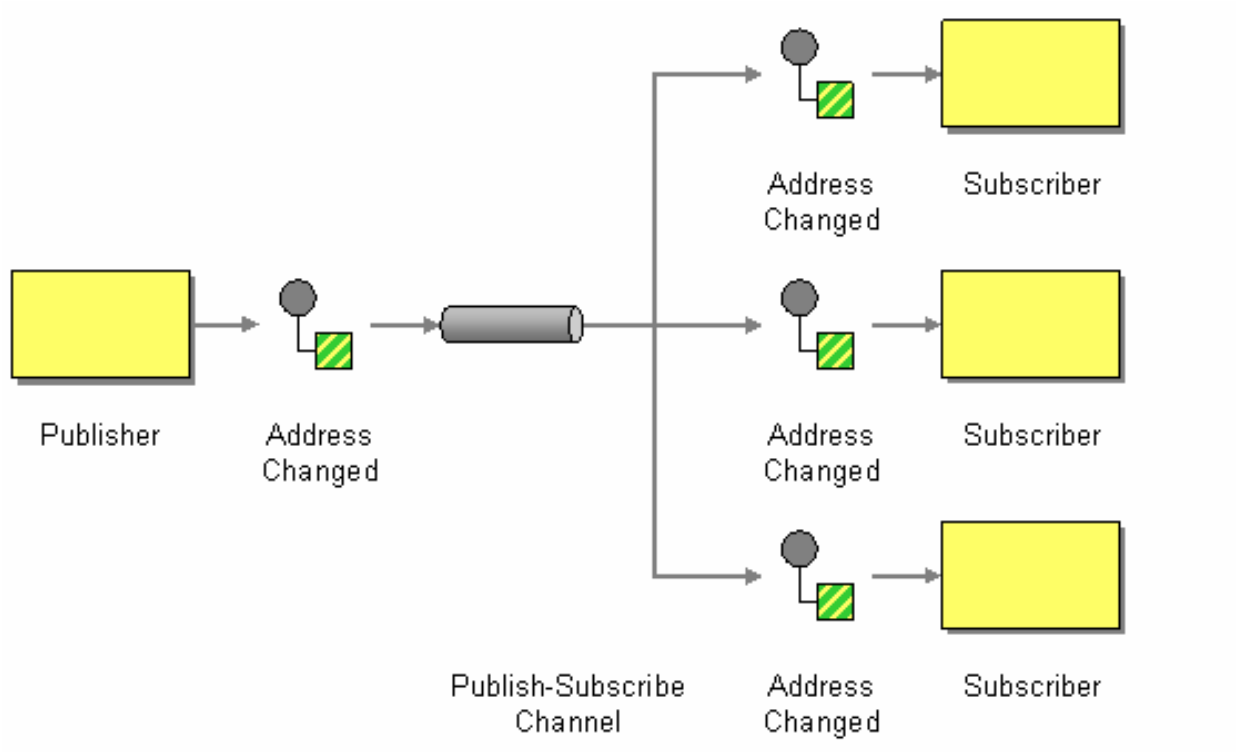
- **Envoi d'un message sur un canal en point à point, assure qu'un seul receveur aura ce message**
  - **La persistance peut être assurée**
    - **Surtout si le récepteur n'est pas en ligne lors de l'émission**
  - **Réception synchrone ou asynchrone**
  - **Exemple déjà vu : le mail**

Publish/Subscribe ou Observateur/Observé



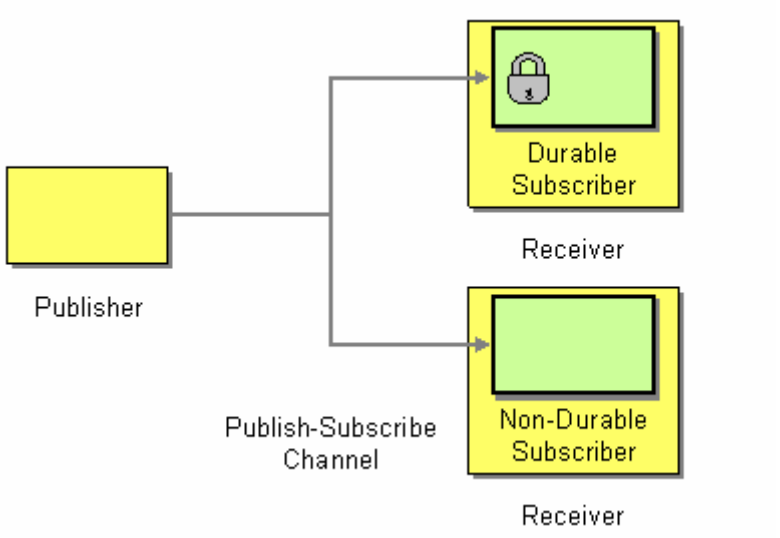
- **http://book.itzero.com/read/MICROSOFT/0603/Sams.Microsoft.SQL.Server.2005.Notification.Services.Feb.2006_html/0672327791/ch01lev1sec2.html**

Publish/Subscribe



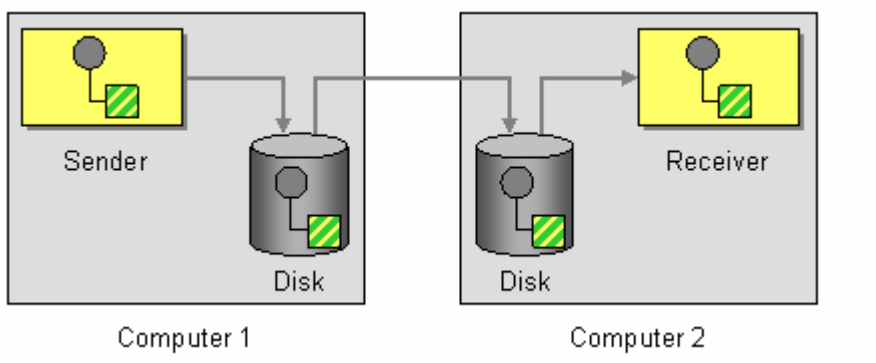
- **Observateur/Observé « sélectif »**
  - **Souscription d'un sujet/thème (topic)**
  - **Publication entraîne le réveil des souscripteurs**
    - **mode push**
    - Si l'abonné est absent ?
    - Sélection de la souscription ?
  - **En mode pull, les abonnés devront s'enquérir des notifications**

Durable Subscriber



- ***Durable Subscriber***
- **Sauvegarde/Persistance du message jusqu'à ce que le souscripteur soit informé de la « publication »**

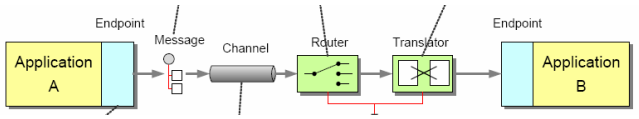
Reliable, Guaranteed Delivery

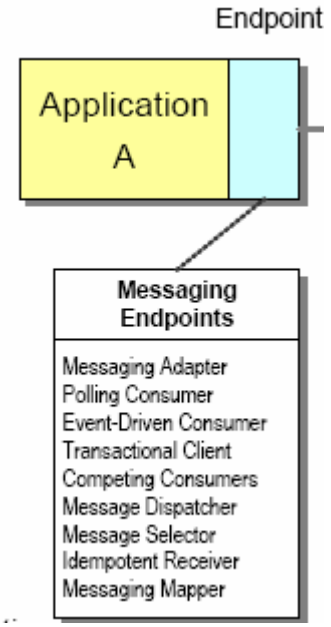


- **Pertes possibles, évitées des messages**
  - **la persistance de ceux-ci, sur chaque poste**

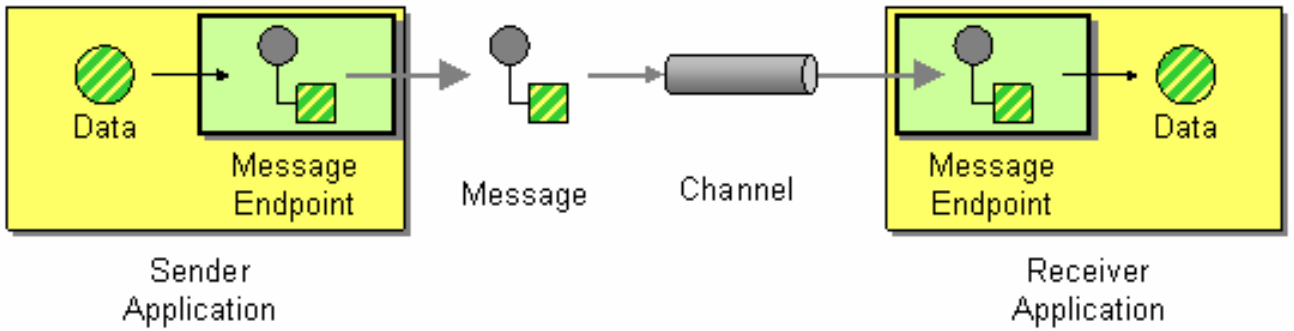
Message endpoints

- **Messaging Adapter**
- **Patron Polling Consumer**
- **Patron Transactional Client**
- **…**



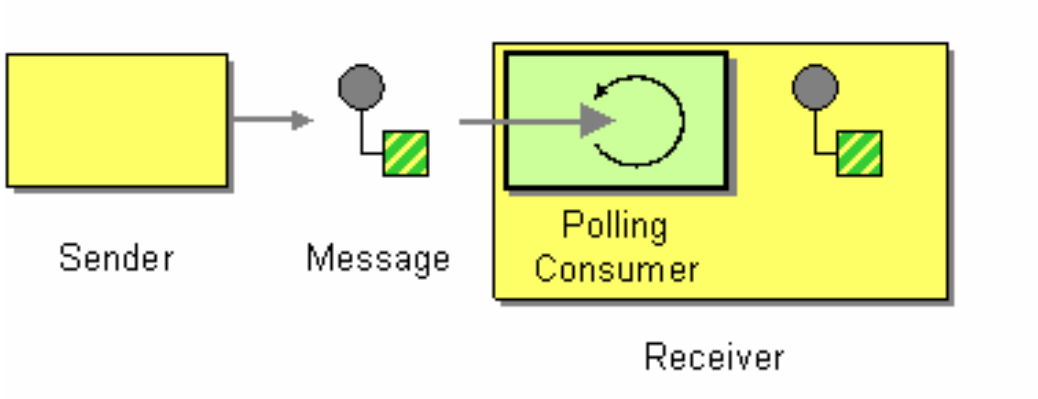


Messaging Adapter



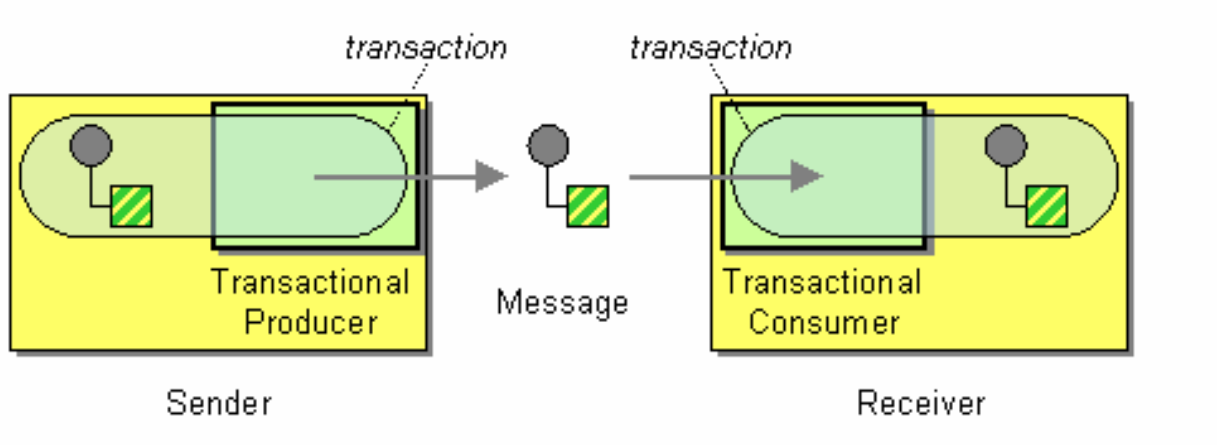
- **Pattern Adaptateur / Wrapper**
  - **Mise en forme, au bon format du contenu du message**

Patron Polling Consumer



- **Récepteur synchrone,**
  - **Le récepteur est bloqué jusqu'à ce que le message soit reçu**

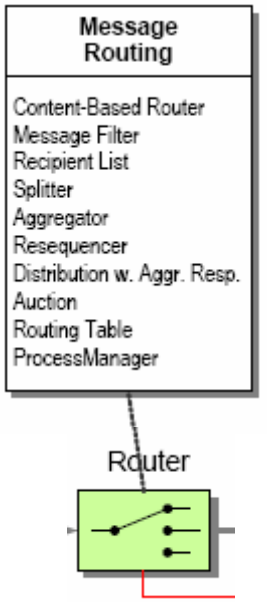
Patron Transactional Client

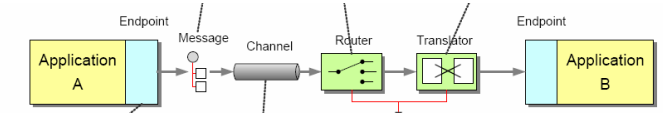


- **Opération atomique**
- **commit, rollback**
- **La session est achevée lors du commit**
  - **Plusieurs Receiver/Sender**

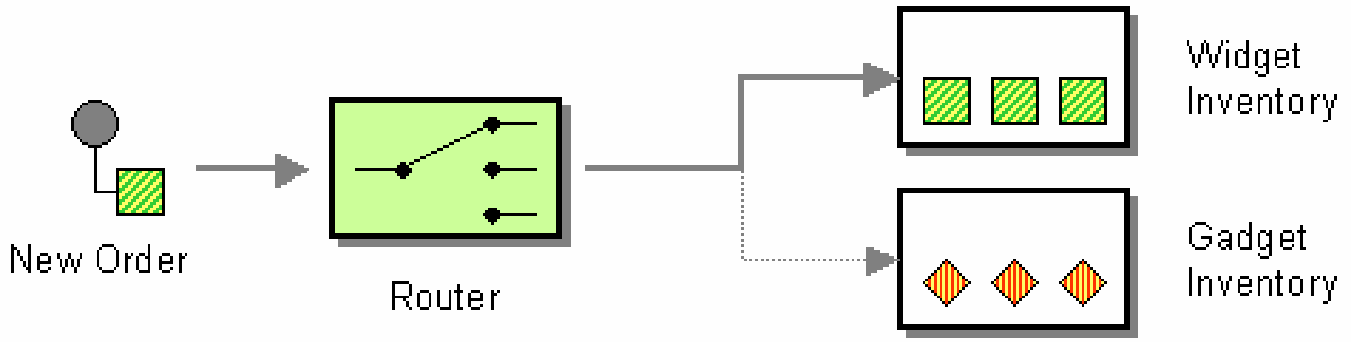
Message Routing

- **Content Based Router**
- **Dynamic Router**
- **Recipient List**
- **Splitter/Agregator**
- **….**



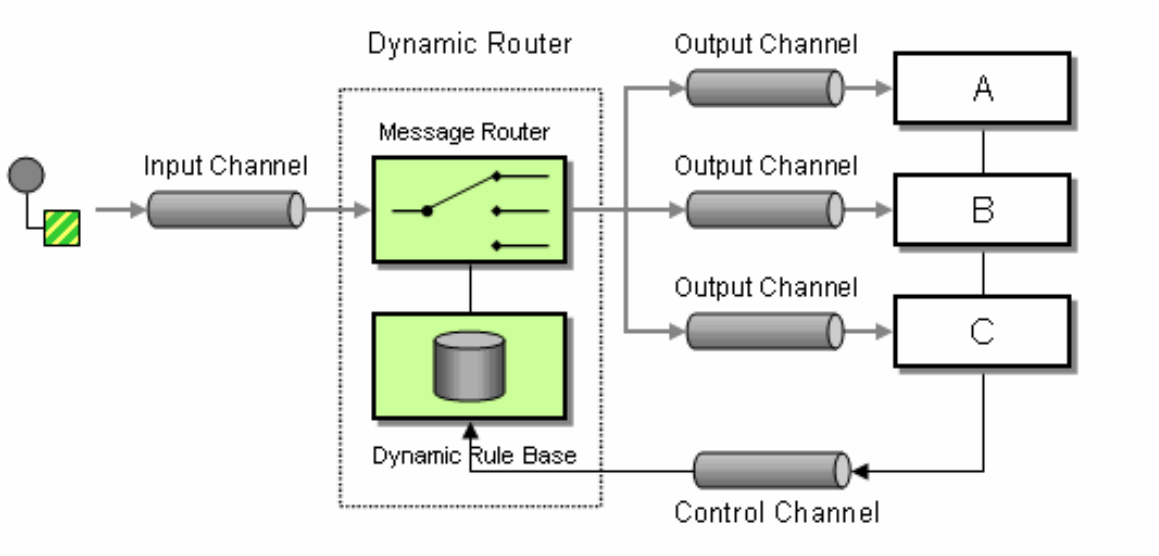


Patron Content Based Router



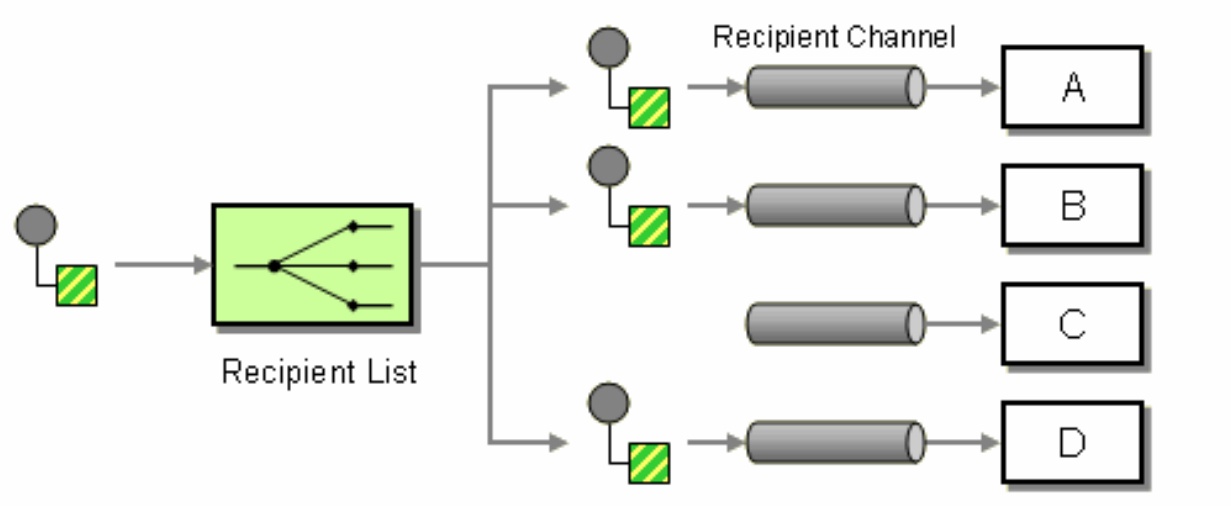
- **Selon le contenu du message, une sélection du canal de communication est effectuée**

Dynamic Router



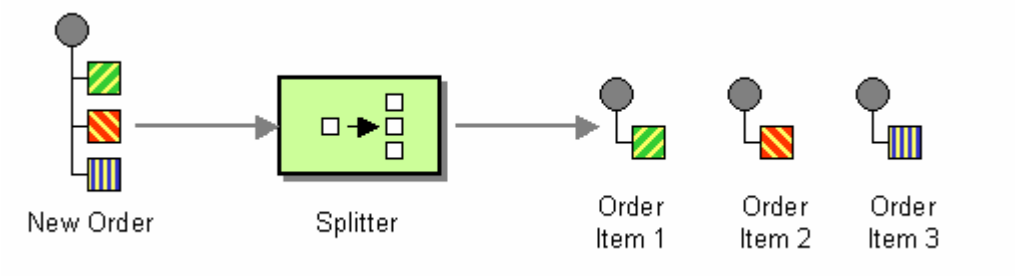
- **En cours d'exécution, selon le contenu du message et certaines conditions/règles la sélection du canal est effectuée**

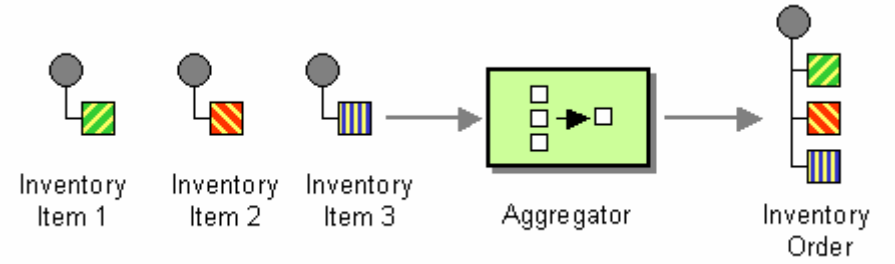
Patron Recipient List



- **Un Canal par destinataire**
- **Envoie une copie du message sur chaque canal**
  - **Store and forward**

Splitter/Agregator

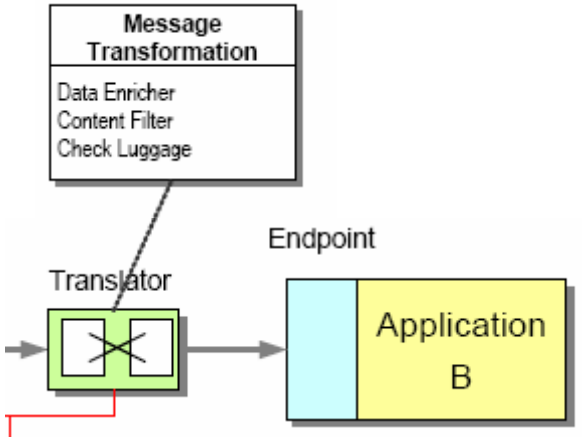


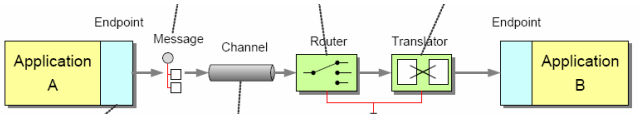


- **Découpage et Assemblage du message**

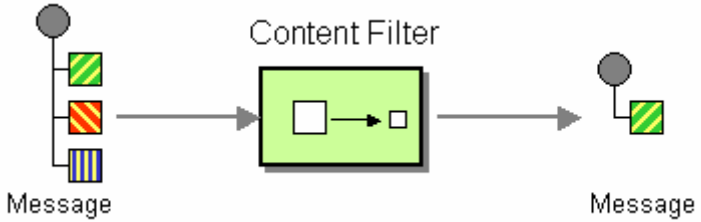
Message transformation

- **ContentFilter**
- **Claim Check**



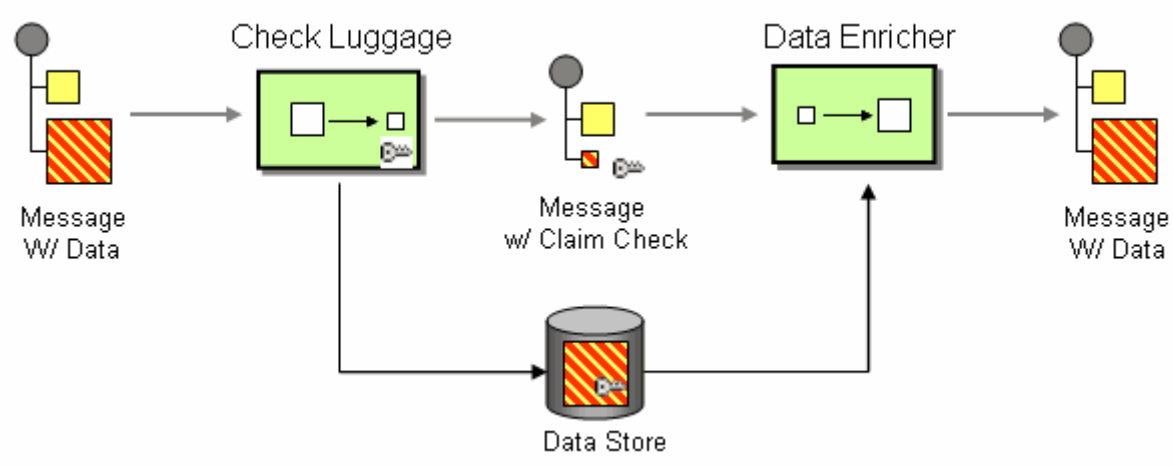


Patron Content Filter



- **Filtrage des informations « essentielles »**

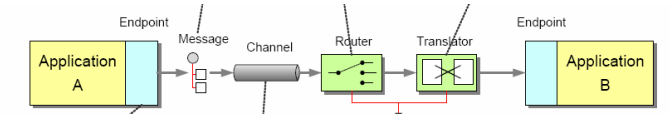
Patron Claim Check

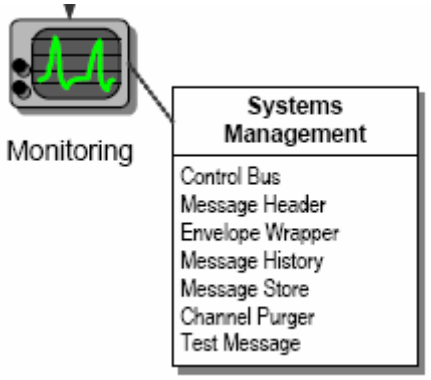


- **Persistance du message**
- **Reconstitution de celui-ci**

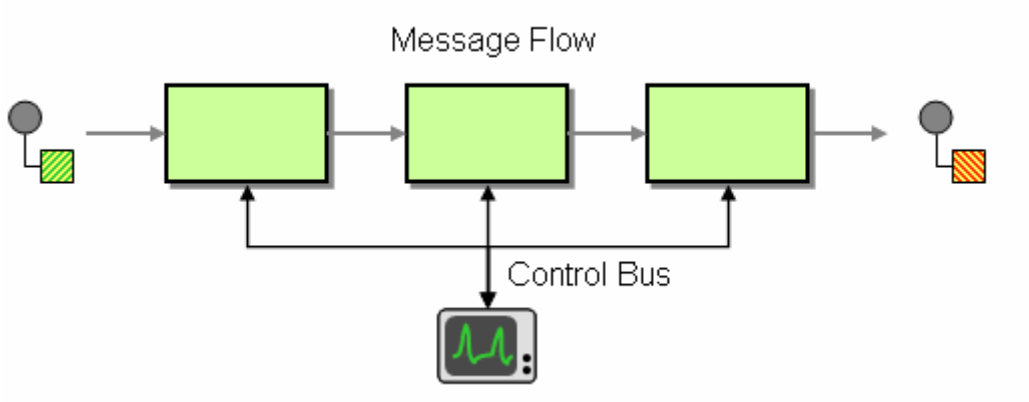
System Management

- **Supervision et contrôle**
  - **ControlBus**
  - **Detour**
- **Mesures du trafic …**
  - **Message History**
- **Test et mise au point**
  - **Test Message**



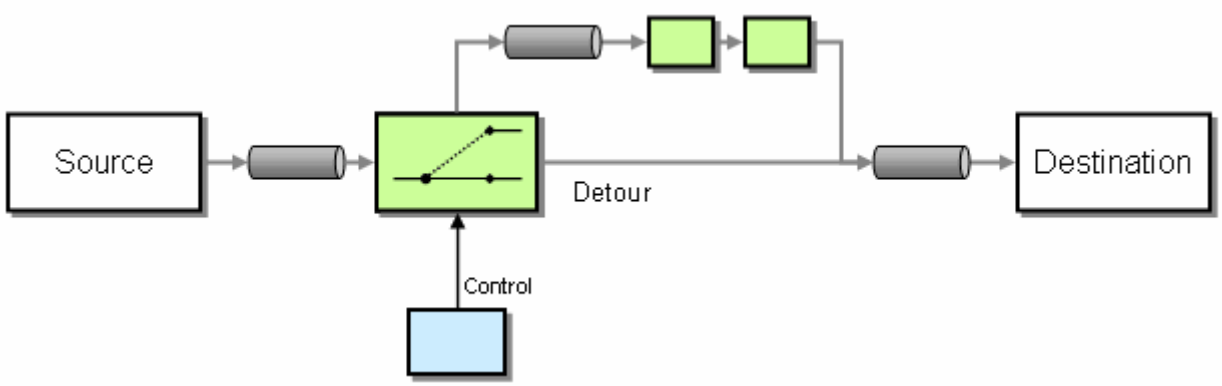


Patron ControlBus



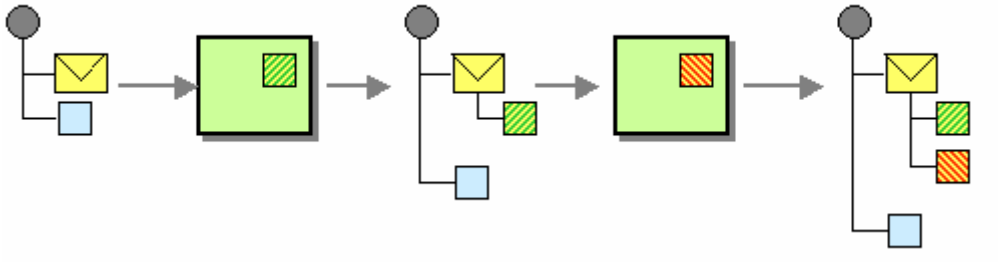
- **Outil de supervision**

Patron Detour



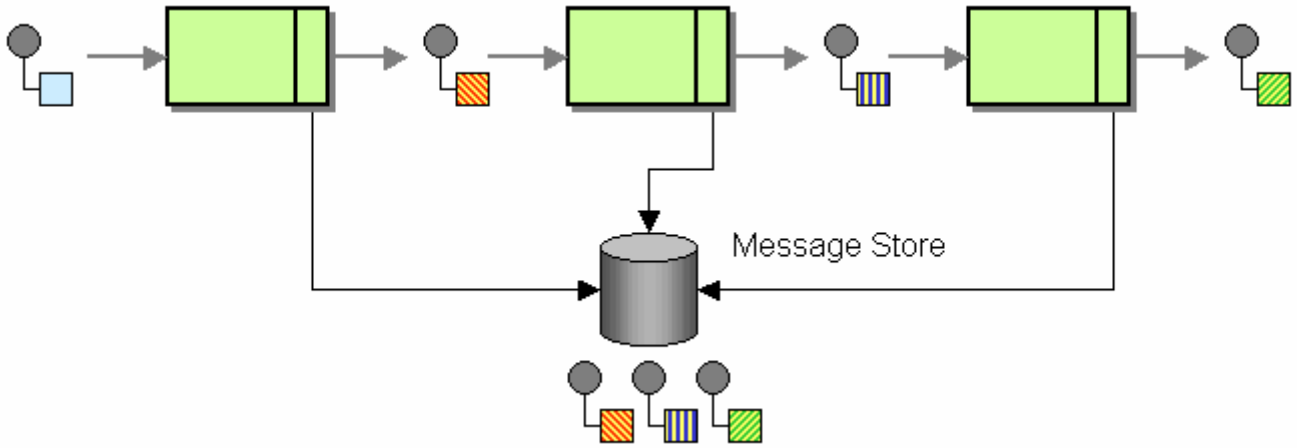
- **Débogage …**

Message History



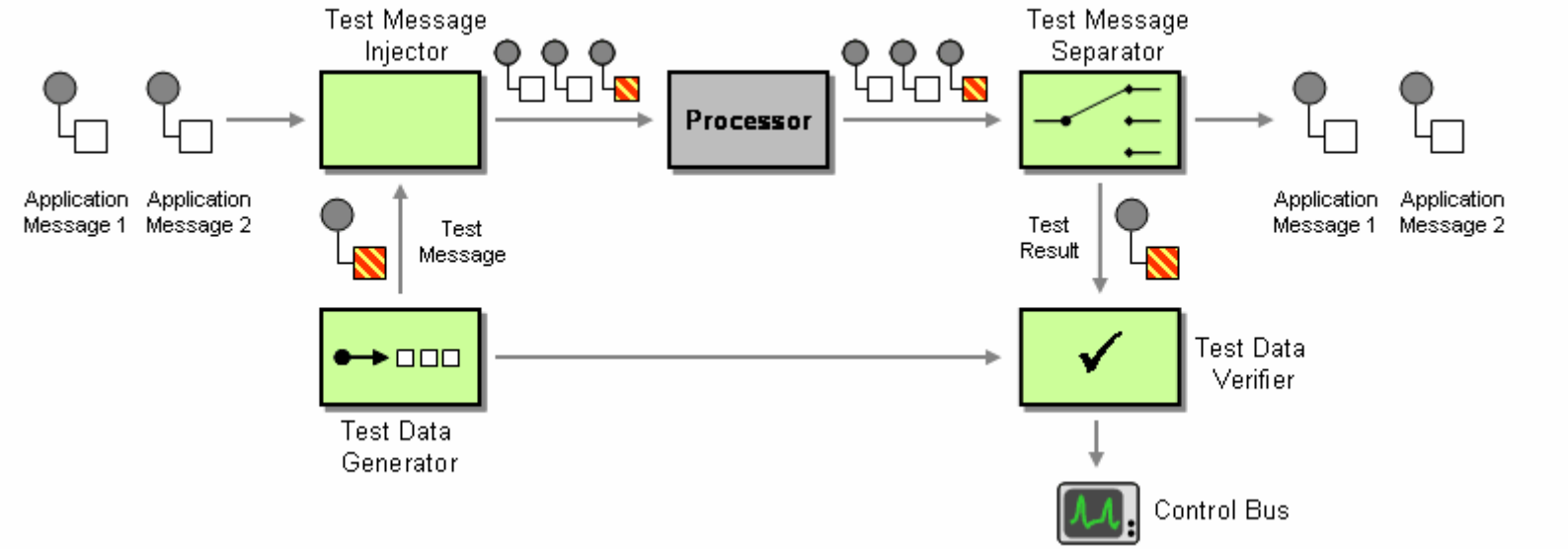
- **Historique des opérations effectuées**

Patron Message store



- **Duplication du message**
- **Persistance**

Patron TestMessage



- **Techniques de test …**

Langage de description, à base de patrons

- **MOM, possède les mêmes patrons !**
  - Message-oriented middleware (MOM)
    - IBM WebSphere MQ
    - Microsoft MSMQ
    - Java Message Service (JMS) Implementations
  - EAI Suites
    - TIBCO, WebMethods, SeeBeyond, Vitria
  - Asynchronous Web services
    - Sun’s Java API for XML Messaging (JAXM)
    - Microsoft’s Web Services Extensions (WSE)
  - http://www.middleware.org/mom/basicmom.html

Conclusion intermédiaire

- **Transparents extraits de**
  - http://www.enterpriseintegrationpatterns.com/eaipatterns.html

- **À lire donc**

JMS, Java Message Service

- **N'est qu'une spécification**
  - http://java.sun.com/products/jms/docs.html
- **Soit en java**
  - Un ensemble d'interfaces que tout fournisseur est tenu de respecter
  - Une série de tests de « conformité » http://jmscts.sourceforge.net/
- **Comment ?**
  - « Patron fabrique »
    - Proposés par les fournisseurs JMS
    - Exemple :

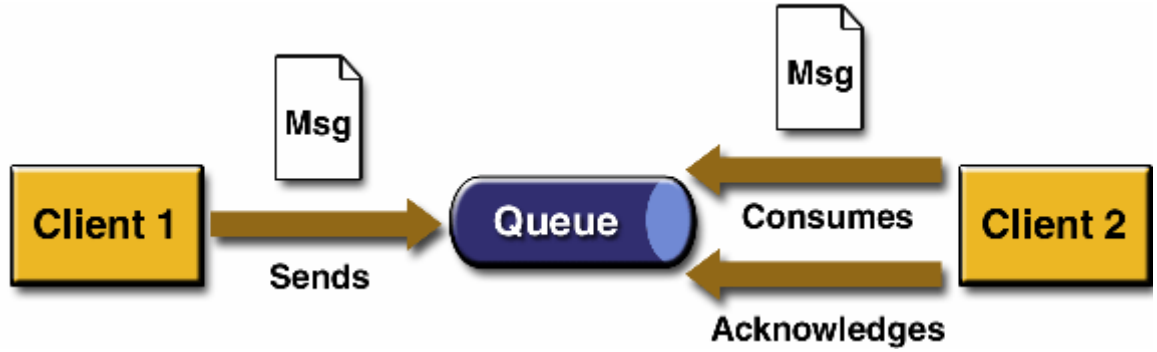
```
ConnectionFactory fabrique = contexte.lookup("ConnectionFactory");
Connection connexion = fabrique.createConnection();
```

Modèles

- **Point à Point**
  - **Queue**
- **Publish Subscibe**
  - **Topic**

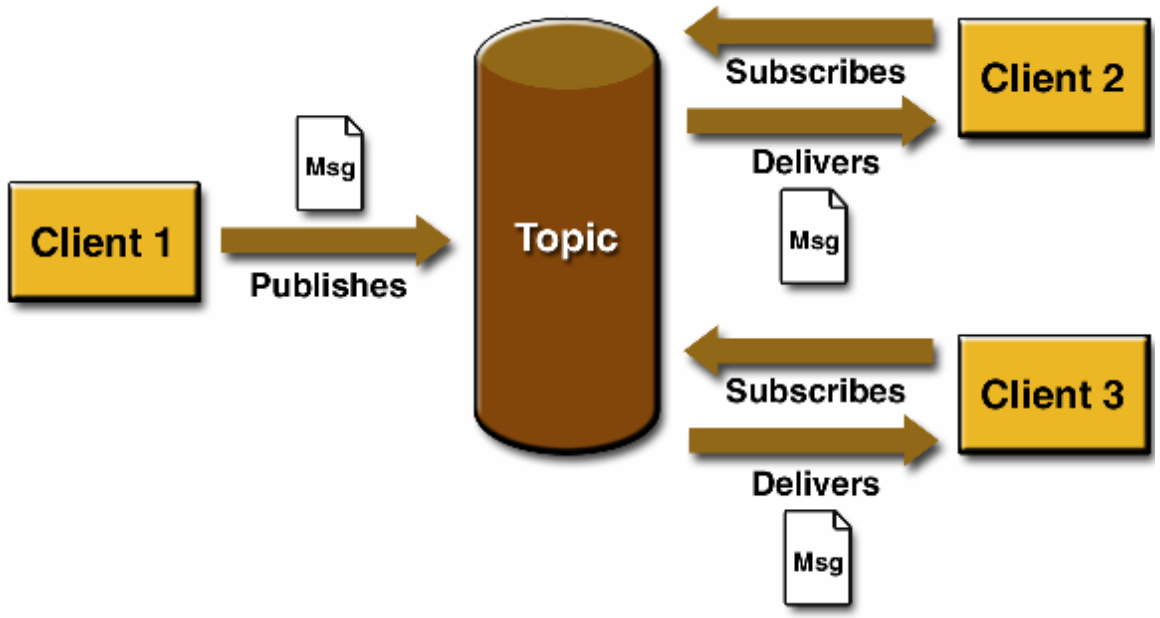
JMS Queue

Queue - Point à Point



- Schéma extrait du tutorial
- Le courtier/serveur JMS gère les files, les communications, la persistance

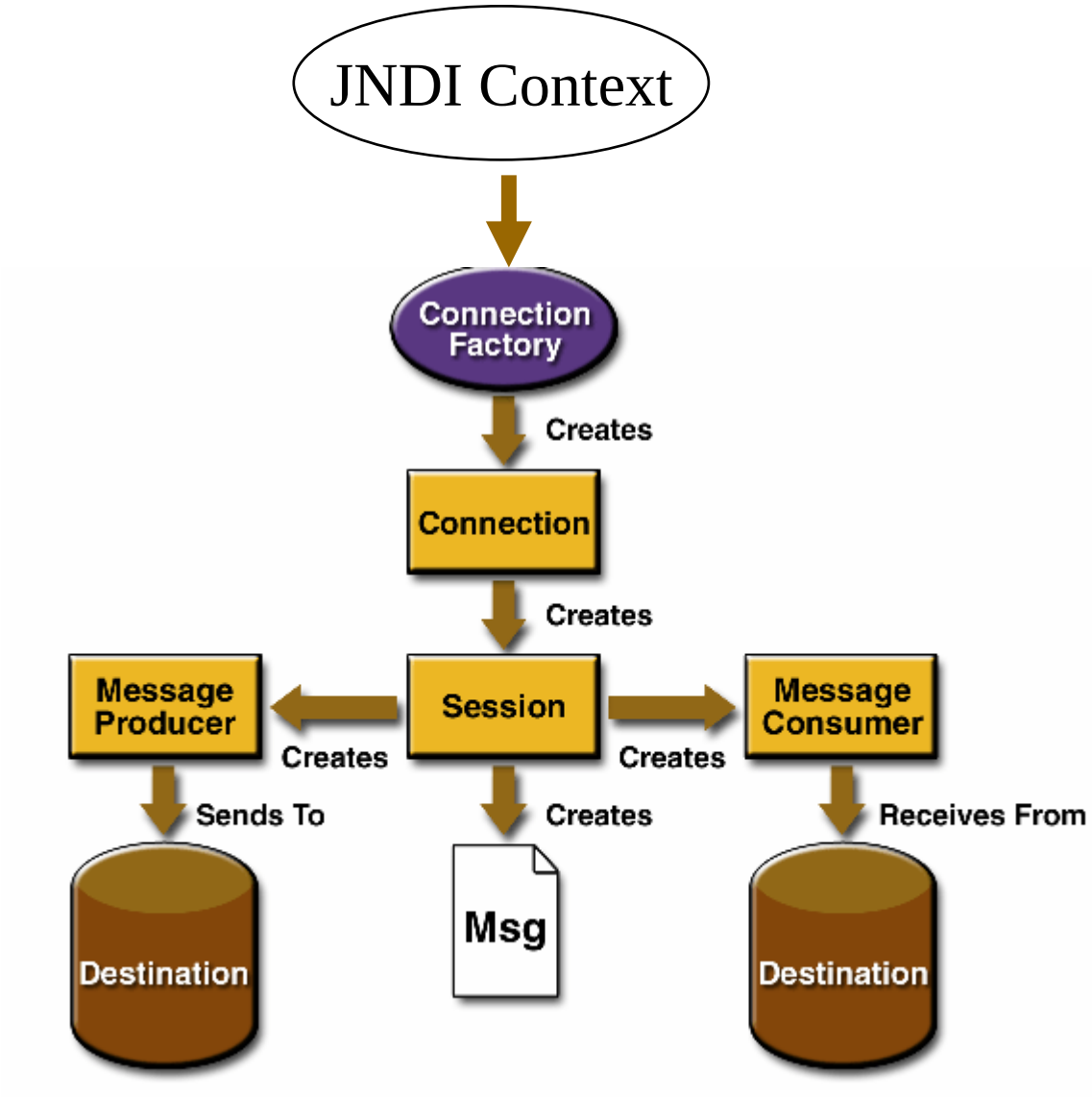
JMS Topic



Topic - Publish-Subscribe

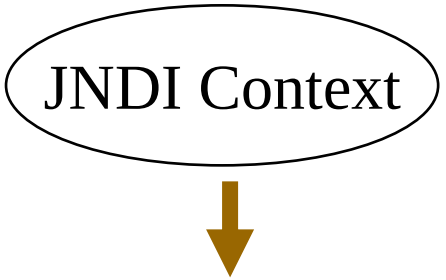
- **Schéma extrait du tutorial**

La session au centre



Context

- **Recherche de l'annuaire, i***ci OpenJMS*




```
Hashtable<String,String> props;
props = new Hashtable<String,String>();

props.put(Context.INITIAL_CONTEXT_FACTORY,
                  "org.exolab.jms.jndi.InitialContextFactory");
props.put(Context.PROVIDER_URL, "tcp://localhost:3035/");
Context contexte = new InitialContext(props);
```

Voir javax.naming.Context

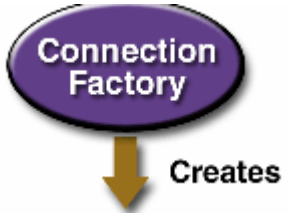
Ou bien en ligne de commande

```
java … -Djava.naming.factory.initial=org.exolab.jms.jndi.InitialContextFactory
 -Djava.naming.provider.url=tcp://localhost:3035/  …
```

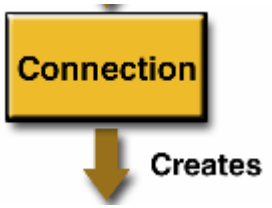
Voir le fichier jndi.properties

Connection

```
ConnectionFactory fabrique;
fabrique = (ConnectionFactory)
  contexte.lookup("ConnectionFactory");
```




```
Connection connexion = fabrique.createConnection();
connexion.start();
```



- *It encapsulates an open connection with a JMS provider. It typically represents an open TCP/IP socket between a client and the service provider software.*
- *Its creation is where client authentication takes place.*
- *It can specify a unique client identifier.*
- *It provides a ConnectionMetaData object.*
- *It supports an optional ExceptionListener object.*

Session

```
Session session;
session = connexion.createSession(false,
                                  Session.AUTO_ACKNOWLEDGE);
```

- *It is a factory for its message producers and consumers.*
- *It supplies provider-optimized message factories.*
- *It is a factory for TemporaryTopics and TemporaryQueues.*
- *It provides a way to create Queue or Topic objects for those clients that need to dynamically manipulate provider-specific destination names.*
- *It supports a single series of transactions that combine work spanning its producers and consumers into atomic units.*
- *It defines a serial order for the messages it consumes and the messages it produces.*
- *It retains messages it consumes until they have been acknowledged.*
- *It serializes execution of message listeners registered with its message consumers.*
- *It is a factory for QueueBrowsers.*

Destination (Queue par exemple)

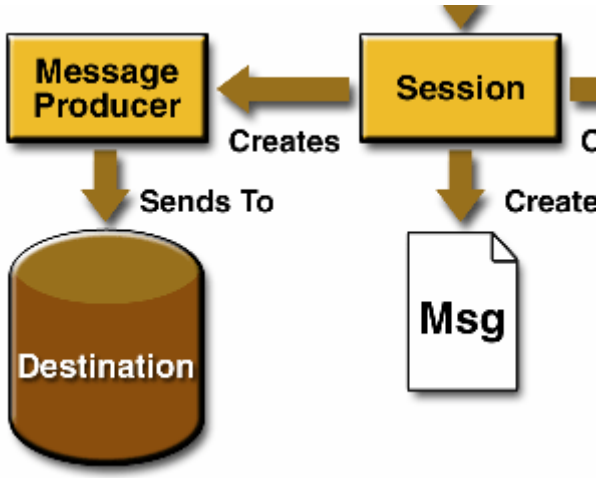
```
Destination dest;
dest = (Destination) contexte.lookup("queue1");
```

Destination est un Objet JMS

- **Queue / Topic**

MessageProducer

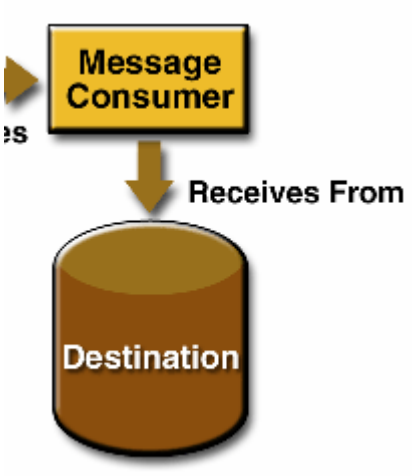
- `MessageProducer prod = session.createProducer(dest);`



- `TextMessage message;`
- `message = session.createTextMessage("hello");`
- `prod.send(message);`

MessageConsumer

- **TextMessage message ;**
- **Message = (TextMessage)receiver.receive();**
- **System.out.println(message.getText());**



Réception asynchrone : MessageListener

```
public class Consumer implements MessageListener {


  public void onMessage(Message message){
     TextMessage message ;
      Message = (TextMessage)receiver.receive();
      System.out.println(message.getText());
  }


}
```

Graphe de classes

Interface « parent »	*Point-à-point*	*Publish/Subscribe*
Destination	**Queue**	**Topic**
ConnectionFactory	**QueueConnectionFactory**	**TopicConnectionFactory**
Connection	**QueueConnection**	**TopicConnection**
Session	**QueueSession**	**TopicSession**
MessageProducer	**QueueSender**	**TopicPublisher**
MessageConsumer	**QueueReceiver**	**TopicSubscriber**

```
Queue                 extends Destination
QueueConnectionFactory extends ConnectionFactory
QueueConnection       extends Connection
QueueSession          extends Session
QueueSender           extends MessageProducer
QueueReceiver         extends MessageConsumer
```

idem pour Topic

Fabriques encore …

- `ConnectionFactory fabrique = (ConnectionFactory) contexte.lookup("ConnectionFactory");`

- `QueueConnectionFactory fabrique = (QueueConnectionFactory) contexte.lookup("JmsQueueConnectionFactory");`

- `TopicConnectionFactory fabrique = (TopicConnectionFactory) contexte.lookup("JmsTopicConnectionFactory");`

Point à Point Queue

- **En résumé**
  - **Persistance des messages**
  - **Réception synchrone ou asynchrone**

Queue : un exemple extrait de
http://fiehnlab.ucdavis.edu/staff/wohlgemuth/java/jms-1

```
// JNDI contexte
InitialContext ctx = new InitialContext(props);


QueueConnectionFactory qcf = (QueueConnectionFactory)
  ctx.lookup("JmsQueueConnectionFactory");
QueueConnection qc = qcf.createQueueConnection();
Queue queue = (Queue) ctx.lookup("queue1");

QueueSession qs = qc.createQueueSession(false,
                                        Session.AUTO_ACKNOWLEDGE);


QueueReceiver receiver = qs.createReceiver(queue);
qc.start();
```

Publish-Subscribe

- **Même schéma**
  - **De programme,**
  - **Concept identique au Canal de communication**

- **Mais les messages publiés ne sont pas persistants**
  - **Les messages publiés ne sont pas connus du souscripteur avant sa connexion**

Extrait de programme, Publisher

```
// contexte : JNDI
contexte = new InitialContext(props);

TopicConnectionFactory fabrique = (TopicConnectionFactory)
 contexte.lookup("JmsTopicConnectionFactory");

TopicConnection connexion = fabrique.createTopicConnection();

TopicSession session = connexion.createTopicSession(false,
                                    Session.AUTO_ACKNOWLEDGE);

Topic topic = (Topic) contexte.lookup(topicName);

TopicPublisher sender = session.createPublisher(topic);

connexion.start();
```

Publisher, suite : envoi du message

```
TextMessage message = session.createTextMessage("test");
sender.publish(message);
```

Avec, voir transparent précédent

```
TopicPublisher sender = session.createPublisher(topic);
```

Un extrait : le Subscriber

```
 // contexte JNDI
contexte = new InitialContext(props);

TopicConnectionFactory fabrique = … idem Publisher
TopicSession session = … idem Publisher
Topic topic = … idem Publisher

  in = session.createSubscriber(topic);
  in.setMessageListener(this);

  connexion.start();
}

  public void onMessage(Message message) {
     // traitement du message
  }
```

DurableSubscriber

- **Persistance jusqu'à ce que le souscripteur se réveille …**

Même architecture

```
• TopicSubscriber subscriber;
• subscriber = session.createDurableSubscriber(
•                   topic, subscriptionName);
```

C'est tout

Souscription avec filtrage

- **Abonnement conditionnel, sélectif**
- **Sélection des messages sur**
  - **La valeur de leurs attributs**
- **Syntaxe sous-ensemble de SQL**
  - **NOT, AND, OR, IS NULL …**
  - **BETWEEN ..**
  - **LIKE …**

http://openjms.sourceforge.net/modules/openjms/apidocs/org/exolab/jms/selector/Selector.html

Souscription avec filtrage (2)

- **Même architecture**

```
• TopicSubscriber subscriber;
• subscriber = session.createDurableSubscriber(
•                  topic, subscriptionName
•                  " valeur >10 AND nom LIKE ‘urgent%’ "  ,
•                  true);
```

  - Le booléen true inhibe les messages publiés au sein de la même connexion

- *À vérifier avec OpenJMS …*

Message

• TextMessage	String	getText(), setText()
• MapMessage	<Clé,Valeur>	setString(), getString()
• BytesMessage	flot de byte	writeBytes(), readBytes()
• StreamMessage	flot type primitifs	writeString(), …
• ObjectMessage	Objet sérialisé	getObject(), setObject()

Détail sur les accusés de réception(ACK)

- **Mode `Session`.AUTO_ACKNOWLEDGE**
  - **ACK, envoyé automatiquement**
  - **À la fin de la méthode receive (synchone)**
  - **Ou à la fin de la méthode onMessage (asynchrone)**

- **Mode `Session`.CLIENT_ACKNOWLEDGE**
  - **ACK, effectué par le récepteur**
  - **Appel de la méthode message.acknowledge()**

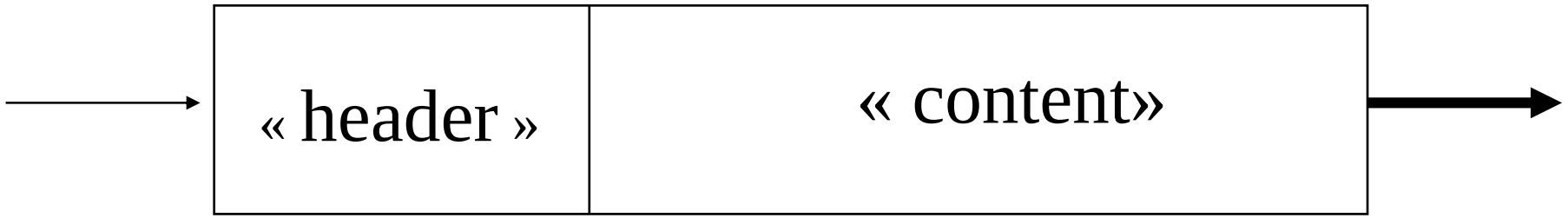
Persistance, fiabilité, défaillance

- **Le serveur JMS s'arrête …**
  - **À la remise en état, tous les messages (persistants) sont de nouveau envoyés …**

  -> cela engendre donc qu'un même message peut être reçu plusieurs fois …

  - **Mode `Session.`AUTO_ACKNOWLEDGE**
    - **Une table des ID des messages est conservée par le client, un test de la présence du message dans cette table évite la duplication**
  - **Mode `Session.`DUPS_OK_ACKNOWLEDGE**
    - **Lorsque la réception multiple ne pose pas de problèmes**
  - **Mode `Session`.CLIENT_ACKNOWLEDGE**
    - **Une exception est levée (ExceptionListener), c'est au client d'en tenir compte**

Message, quelques méthodes



Un Message

- **Affectation des champs du « header »**

- **JMSMessageID**
- **JMSCorrelationID**
- **JMSExpiration**
- **JMSReplyTo**
- **JMSDestination**
- **…**

Message suite

- **JMSMessageID:**
  - **L'identificateur unique du message**

- **JMSCorrelationID:**
  - **Identification du message pour le client**
  - **Lie une réponse à une requête,**
  - **Le client vérifiera que ce nombre correspond à l'identificateur envoyé**

Message suite, quelques méthodes

- **JMSExpiration:**
  - **Durée de vie d'un message**
  - **0 : infini**

Message

- **JMSReplyTo:**
  - **Précisé par le client afin de répondre au message envoyé**
- **JMSDestination:**
  - **Afin de connaître le destinataire**

Patron Request Reply + Correlation ID

- **Message msg =**

- *// (« header ») affectation du champ reply-to*
- producer = session.CreateProducer(msg.getJMSReplyTo());

- *// (« content »)*
- reply = session.createTextMessage("reply");

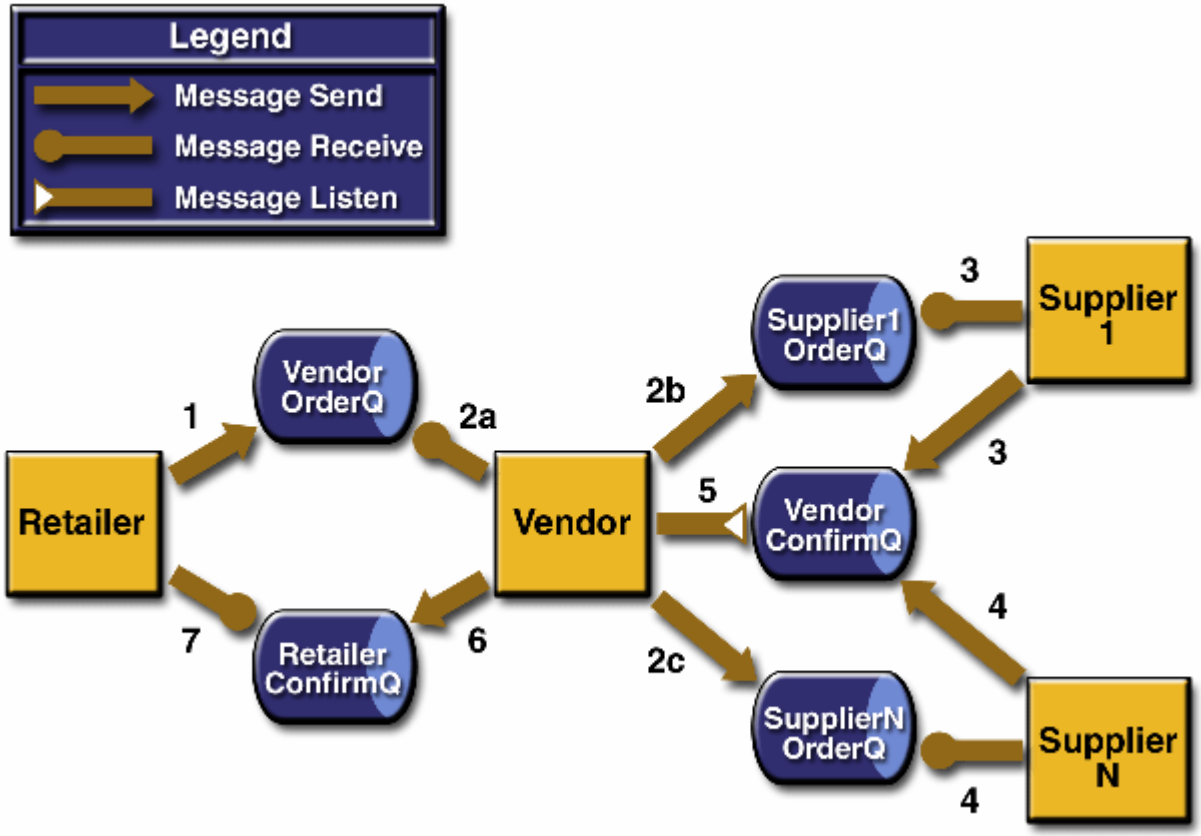
- *// (« header ») Correl-ID == Message-ID*
- reply.setJMSCorrelationID(msg.getJMSMessageID());

- producer.send(reply);

Un exemple, à suivre

- **http://www.enterpriseintegrationpatterns.com/ObserverJmsExample.html**
- **http://www.enterpriseintegrationpatterns.com/index.html**

Transaction & session



- **Transfert de compte, débit-crédit,**
- **commit/rollback**
- **Voir http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JMS6.html**

Transaction un schéma

```
session = connexion.createTopicSession(true,
                                       Session.AUTO_ACKNOWLEDGE);

   void onMessage(Message msg){
   try{
     // un traitement, susceptible de lever une exception
     m2 = …;
     publisher.publish(m2);
     session.commit(); // acquittement des messages
    }catch(Exception e){
     session.rollback(); // annulation des messages
    }

assert session.getTransacted() == Session.SESSION_TRANSACTED;
```

Transaction comme Session

- **JMS propose (seulement) les transactions au sein d'une même session**
- **Transaction distribuée avec les XASession**
  - **Optionnel pour les fournisseurs de JMS**
- **Passerelle vers JTA (Java Transaction API)**
  - **XASession**
  - **XAQueueSession,**
  - **XATopicSession**

JMS et JMX

- Supervision et « management » à distance
  - Voir http://lmi92.cnam.fr/NSY102/tp_nsy102_en_salle/tp_03_jms_jmx/

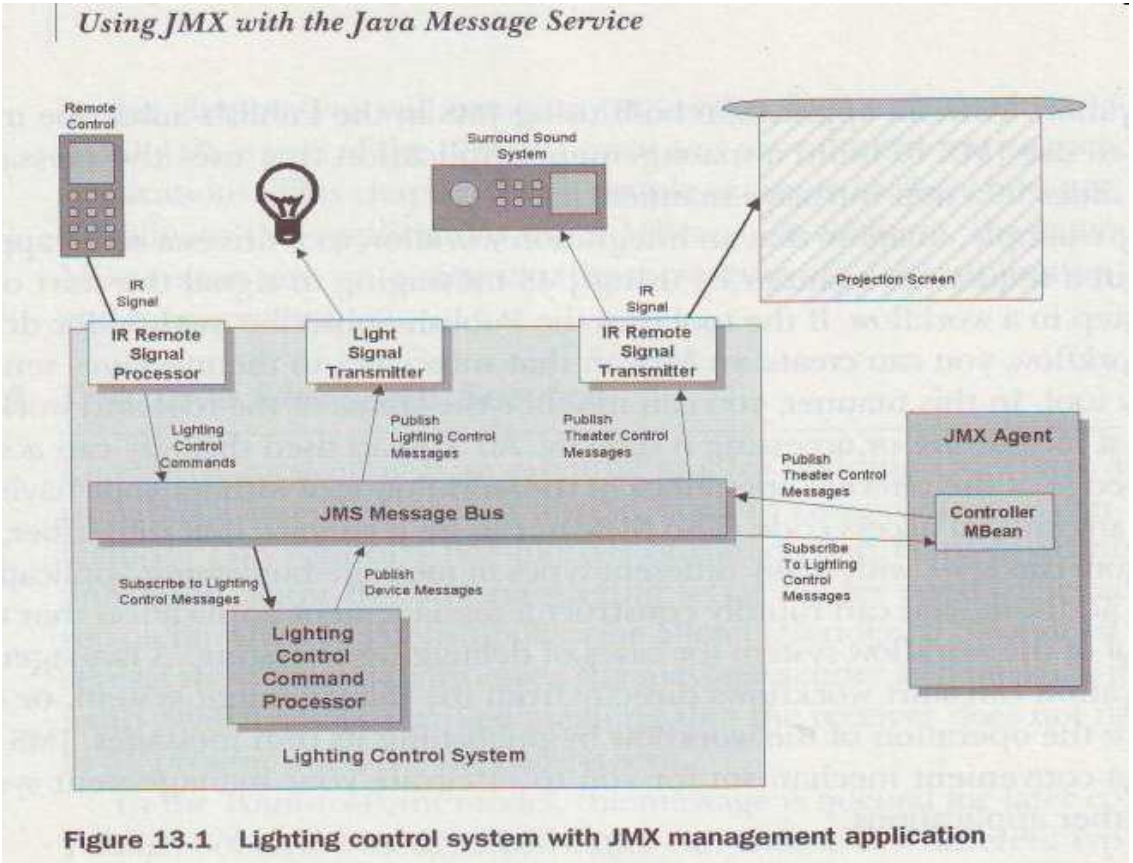


Figure 13.1 Lighting control system with JMX management application

  - figure extraite de ce livre http://www.manning.com/sullins/

Conclusion

- **JMS1.1 se trouve en J2EE1.4**

- **À suivre**
  - **Supervision, JMX**
  - **Transactions**
    - **Mise en oeuvre**
    - **Distribuées JTA**

Annexe

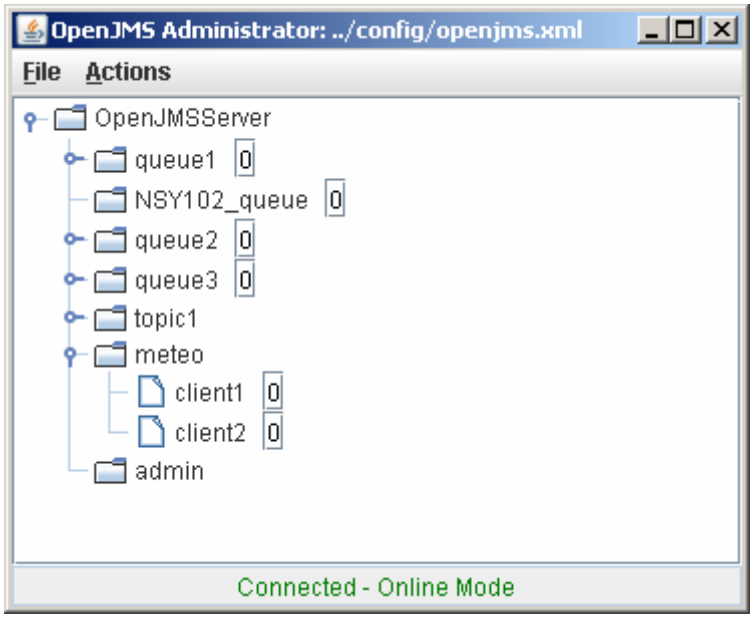
- **http://openjms.sourceforge.net/**
  - Point-to-Point and publish-subscribe messaging models
  - Guaranteed delivery of messages
  - Synchronous and asynchronous message delivery
  - Persistence using JDBC

  - Local transactions
  - Message filtering using SQL92-like selectors
  - Authentication
  - Administration GUI
  - XML-based configuration files
  - In-memory and database garbage collection
  - Automatic client disconnection detection
  - Applet support
  - Integrates with Servlet containers such as Jakarta Tomcat
  - Support for TCP, RMI, HTTP and SSL protocol stacks
  - Support for large numbers of destinations and subscribers

OpenJMS

- 1) D:\openjms-0.7.7-beta-1\bin>set JAVA_HOME=C:\jdk1.6.0
- 2) D:\openjms-0.7.7-beta-1\bin>start startup

- D:\openjms-0.7.7-beta-1\bin>start admin



- Exemples fournis,
  - D:\openjms-0.7.7-beta-1\examples\basic>build
  - D:\openjms-0.7.7-beta-1\examples\basic>start run Sender queue1 100
  - D:\openjms-0.7.7-beta-1\examples\basic>start run Receiver queue1 100
  - D:\openjms-0.7.7-beta-1\examples\basic>start run DurableSuscriber meteo

OpenJMS + hsqldb, /config/

- **Par défaut derby est utilisée**

```
<DatabaseConfiguration>
   <RdbmsDatabaseConfiguration
     driver="org.apache.derby.jdbc.EmbeddedDriver"
     url="jdbc:derby:openjmsdb;create=true"
     user="openjms"
     password="openjms"/>
 </DatabaseConfiguration>
```

- **hsqldb http://hsqldb.org/**

```
<DatabaseConfiguration>
   <RdbmsDatabaseConfiguration
     driver="org.hsqldb.jdbcDriver"
     url="jdbc:hsqldb:http://localhost:770/OPENJMSDB"
     user="sa"
     password=""/>
 </DatabaseConfiguration>
```

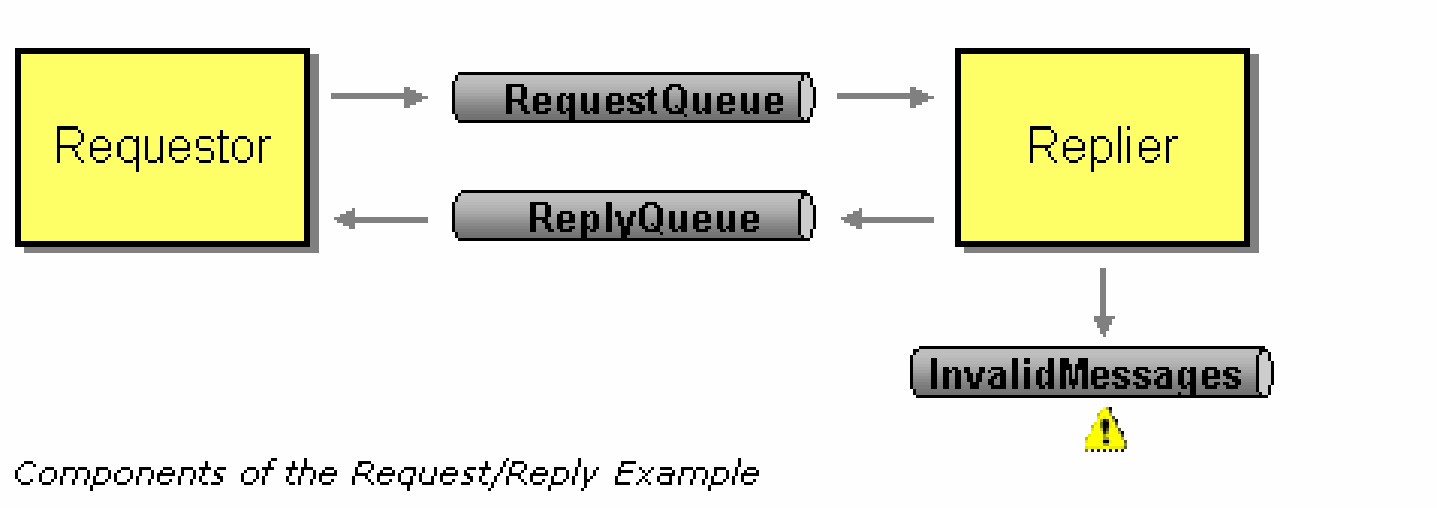
Avec en préalable

```
1) java -cp ../lib/hsqldb1.8.jar org.hsqldb.WebServer -port 770 -silent true -database.0
file:OPENJMSDB -dbname.0 OPENJMSDB
2) dbtool -create -config D:\openjms-0.7.7-beta-1\config\openjms.xml
```

Alternative à OpenJMS

- **http://activemq.apache.org/index.html**

Annexe Exemple extrait de http://www.enterpriseintegrationpatterns.com



Components of the Request/Reply Example

- Extrait de http://www.enterpriseintegrationpatterns.com/RequestReplyJmsExample.html

```
Context context = // JNDI context
ConnectionFactory factory = // …
connection.start();

Requestor requestor = Requestor.newRequestor(connection,"request","reply","invalid");
Replier replier = Replier.newReplier(connection,"request","invalid");



requestor.send();
requestor.receiveSync();
```

Requestor initialize

```
    private Destination replyQueue;
    private MessageProducer requestProducer;
    private MessageConsumer replyConsumer;
    private MessageProducer invalidProducer;

....

Destination requestQueue = JndiUtil.getDestination(requestQueueName);
replyQueue = JndiUtil.getDestination(replyQueueName);
Destination invalidQueue = JndiUtil.getDestination(invalidQueueName);

requestProducer = session.createProducer(requestQueue);
replyConsumer = session.createConsumer(replyQueue);
invalidProducer = session.createProducer(invalidQueue);


}
```

Replier initialize

```
....

Destination requestQueue = JndiUtil.getDestination(requestQueueName);
Destination invalidQueue = JndiUtil.getDestination(invalidQueueName);

MessageConsumer requestConsumer = session.createConsumer(requestQueue);

MessageListener listener = this;
requestConsumer.setMessageListener(listener);

invalidProducer = session.createProducer(invalidQueue);
```

Requestor send

```
public void send() throws JMSException {
        TextMessage requestMessage = session.createTextMessage();
        requestMessage.setText("Hello world.");
        requestMessage.setJMSReplyTo(replyQueue);
        requestProducer.send(requestMessage);

        System.out.println("Sent request");
        System.out.println("\tTime:       " + System.currentTimeMillis() + " ms");
        System.out.println("\tMessage ID: " + requestMessage.getJMSMessageID());
        System.out.println("\tCorrel. ID: " + requestMessage.getJMSCorrelationID());
        System.out.println("\tReply to:   " + requestMessage.getJMSReplyTo());
        System.out.println("\tContents:   " + requestMessage.getText());
    }
}
```

Replier, onMessage

```
public void onMessage(Message message) {
        try {
                if ((message instanceof TextMessage) && (message.getJMSReplyTo() != null)) {
                        TextMessage requestMessage = (TextMessage) message;
                        System.out.println("Received request");
                        System.out.println("\tTime:       " + System.currentTimeMillis() + " ms");
                        System.out.println("\tMessage ID: " + requestMessage.getJMSMessageID());
                        System.out.println("\tCorrel. ID: " + requestMessage.getJMSCorrelationID());
                        System.out.println("\tReply to:   " + requestMessage.getJMSReplyTo());
                        System.out.println("\tContents:   " + requestMessage.getText());

                        String contents = requestMessage.getText();
                        Destination replyDestination = message.getJMSReplyTo();
                        MessageProducer replyProducer = session.createProducer(replyDestination);
                        TextMessage replyMessage = session.createTextMessage();
                        replyMessage.setText(contents);
                        replyMessage.setJMSCorrelationID(requestMessage.getJMSMessageID());
                        replyProducer.send(replyMessage);

                        System.out.println("Sent reply");
                        System.out.println("\tTime:       " + System.currentTimeMillis() + " ms");
                        System.out.println("\tMessage ID: " + replyMessage.getJMSMessageID());
                        System.out.println("\tCorrel. ID: " + replyMessage.getJMSCorrelationID());
                        System.out.println("\tReply to:   " + replyMessage.getJMSReplyTo());
                        System.out.println("\tContents:   " + replyMessage.getText());
                } else { // envoi sur invalidProducer
```

Requestor receiveSync

```
public void receiveSync() throws JMSException {
    Message msg = replyConsumer.receive();
    if (msg instanceof TextMessage) {
      TextMessage replyMessage = (TextMessage) msg;
      System.out.println("Received reply ");
      System.out.println("\tTime:       " + System.currentTimeMillis() + " ms");
      System.out.println("\tMessage ID: " + replyMessage.getJMSMessageID());
      System.out.println("\tCorrel. ID: " + replyMessage.getJMSCorrelationID());
      System.out.println("\tReply to:   " + replyMessage.getJMSReplyTo());
      System.out.println("\tContents:   " + replyMessage.getText());
    } else { // envoi sur invalidProducer
```

Traces d'exécution avec OpenJMS

```
Sent request
        Time:        1178543930125 ms
        Message ID: ID:b1e766d0-c286-1004-8273-d01f0f76bf02
        Correl. ID: null
        Reply to:   reply-true
        Contents:   Hello world.
Received request
        Time:        1178543930250 ms
        Message ID: ID:b1e766d0-c286-1004-8273-d01f0f76bf02
        Correl. ID: null
        Reply to:   reply-true
        Contents:   Hello world.
Sent reply
        Time:        1178543930312 ms
        Message ID: ID:b1e80310-c286-1004-8273-d01f0f76bf02
        Correl. ID: ID:b1e766d0-c286-1004-8273-d01f0f76bf02
        Reply to:   null
        Contents:   Hello world.
Received reply
        Time:        1178543930468 ms
        Message ID: ID:b1e80310-c286-1004-8273-d01f0f76bf02
        Correl. ID: ID:b1e766d0-c286-1004-8273-d01f0f76bf02
        Reply to:   null
        Contents:   Hello world.
```

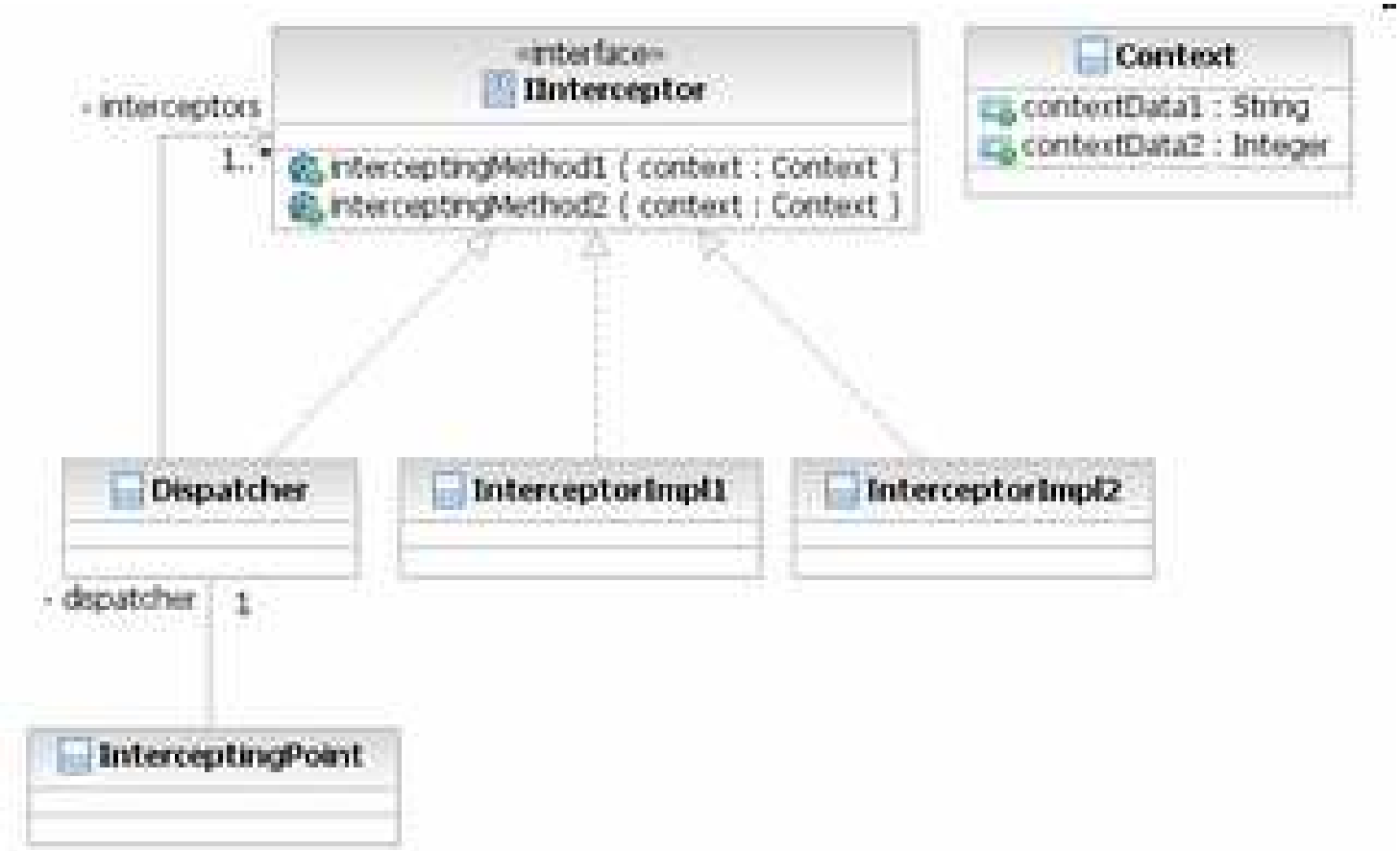
- **Notez l'affectation de Correl ID, lors de la réponse**

JMS et Chat

- **Extrait de http://forums.devx.com/showthread.php?t=137396**

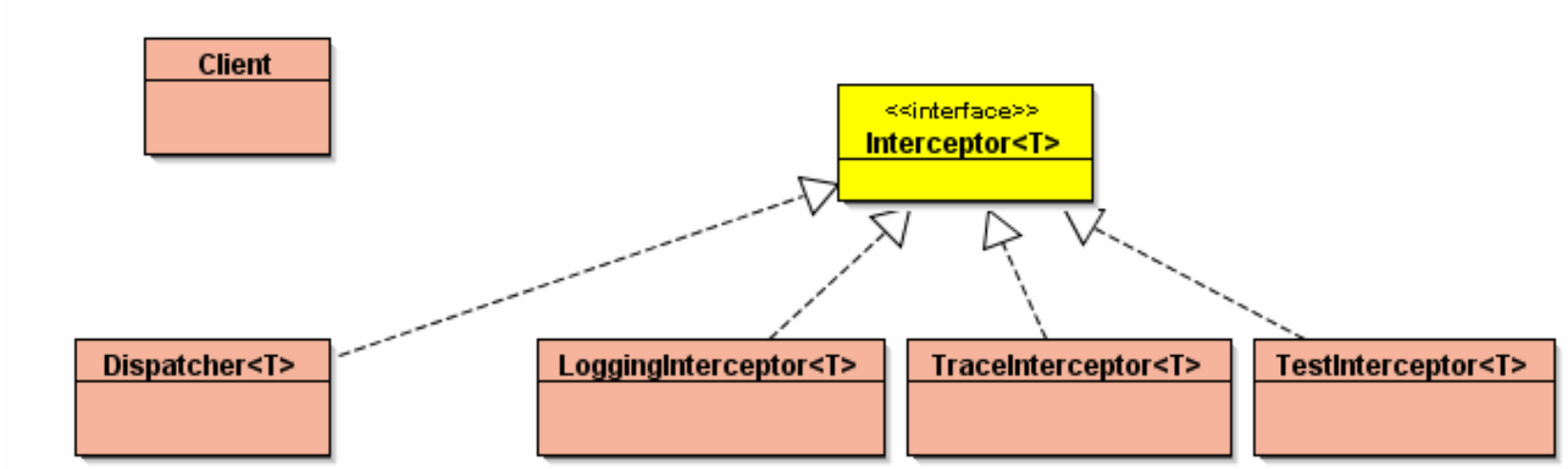
- **Un bon exemple de mise en oeuvre**

Interceptor : une variante « Chain of Responsability »



http://bosy.dailydev.org/2007/04/interceptor-design-pattern.html
http://longbeach.developpez.com/tutoriels/EJB3/Interceptors/
http://homepage.mac.com/cewcew/edu/cmsi688/InterceptorFilters.ppt.

Une implémentation légère



Interceptor<T> et une implémentation

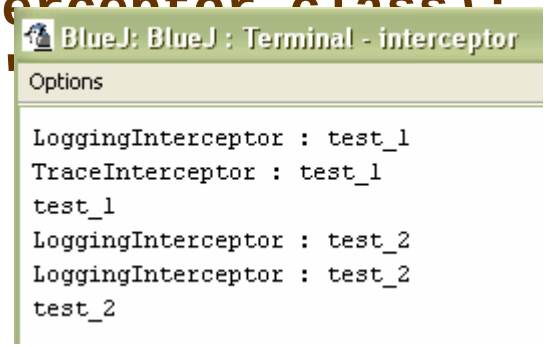
```
public interface Interceptor<T>{

   public T invoke(T in) throws Exception;
}


public class TraceInterceptor<T> implements Interceptor<T>{

  public T invoke(T in) throws Exception{
    System.out.println(this.getClass().getName() + " : " + in);
    return in;
  }
}
```

Dispatcher<T>

```
public class Dispatcher<T> implements Interceptor<T>{
  private Interceptor<T> interceptors[];
  public Dispatcher(final Interceptor<T>... interceptors){
    this.interceptors = interceptors;
  }

//  public Dispatcher(final Class<? extends Interceptor<T>>... interceptors) throws
   InstantiationException, IllegalAccessException{ syntaxe préférée mais Client ne se compile pas
    public Dispatcher(final Class<? extends Interceptor>... interceptors)
                      throws InstantiationException, IllegalAccessException{
    int i = 0;
    this.interceptors = new Interceptor[interceptors.length];
    for(Class<? extends Interceptor> interceptor : interceptors){
      this.interceptors[i] = interceptor.newInstance();
      i++;
    }
  }

  public T invoke(final T in){
    T out = in;
    for(Interceptor<T> interceptor : interceptors){
      try{
        out = interceptor.invoke(out);
      }catch(Exception e){}
    }
    return out;
  }
}
```

Le Client

```
public class Client{

  public static void main(String[] args) throws Exception{
    Dispatcher<String> dispatcher1 =
     new Dispatcher<String>(new
  LoggingInterceptor<String>(),
                           new
  TraceInterceptor<String>());
    System.out.println(dispatcher1.invoke("test_1"));

    Dispatcher<String> dispatcher2 =
         new Dispatcher<String>(LoggingInterceptor.class,
                                LoggingInterceptor.class);
    System.out.println(dispatcher2.invoke("
  }
}
```



Issu de la bibliographie JMX + JMS

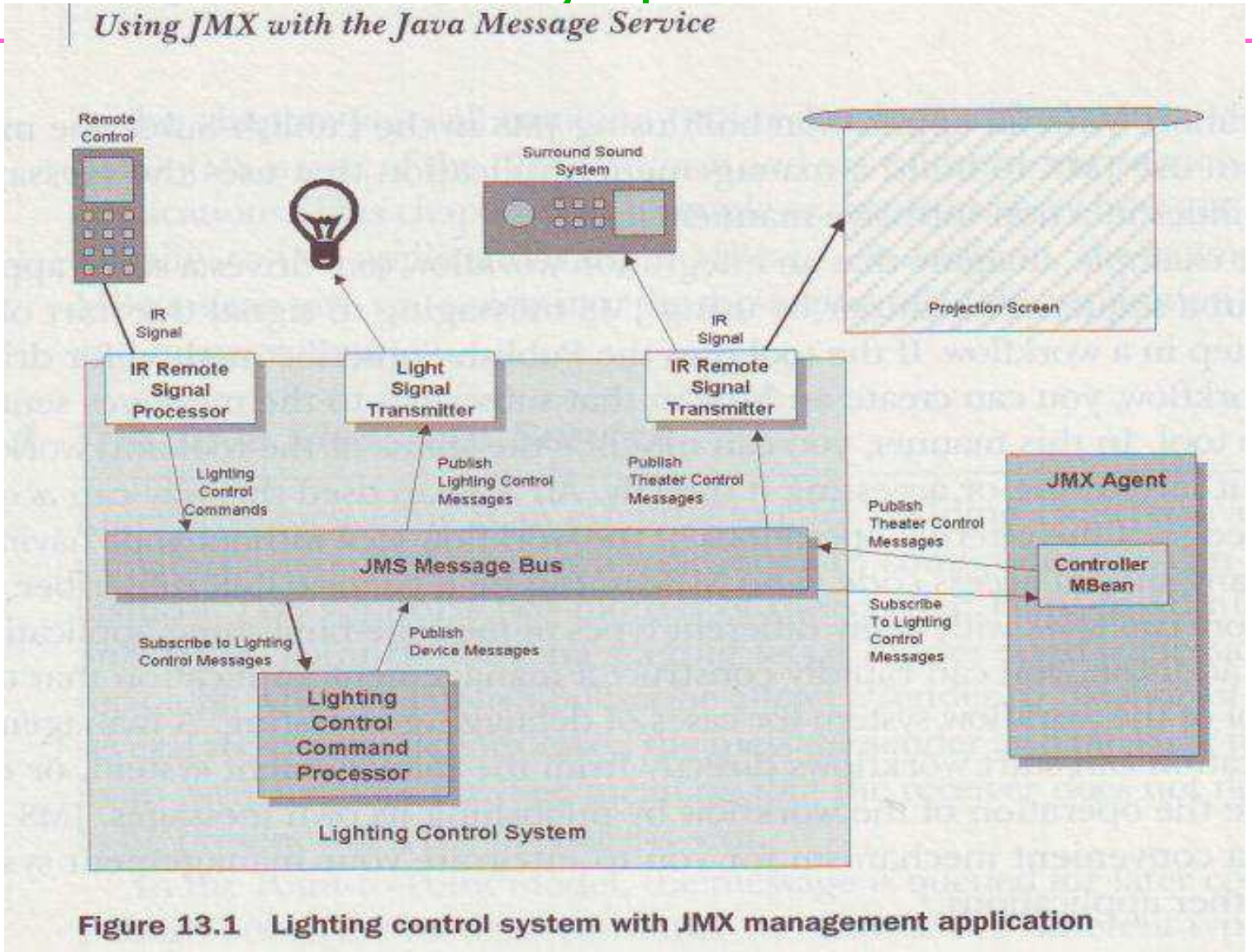


Figure 13.1 Lighting control system with JMX management application

- http://jfod.cnam.fr/NSY102/tp_nsy102_en_salle/tp_03_jms_jmx/index.html
- http://jfod.cnam.fr/NSY102/tp_nsy102_en_salle/tp_03_jms_jmx_une_correction/