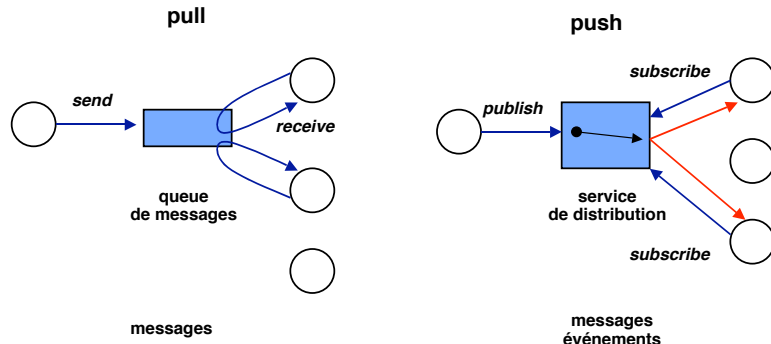


## Rappel sur bus à messages

### ■ Deux modes de fonctionnement



## Trois exemples de mise en œuvre des messages et événements

### ■ JMS (Java Message Service)

- ◆ Interface Java standard pour communication par message
- ◆ Ne définit pas d'implémentation
  - ❖ Peut servir à interfacier un système existant
    - ▲ IBM MQ Series, Novell, Oracle, Sybase, Tibco
  - ❖ Des implémentations spécifiques peuvent être réalisées
  - ❖ Supporte point à point et *publish/subscribe*

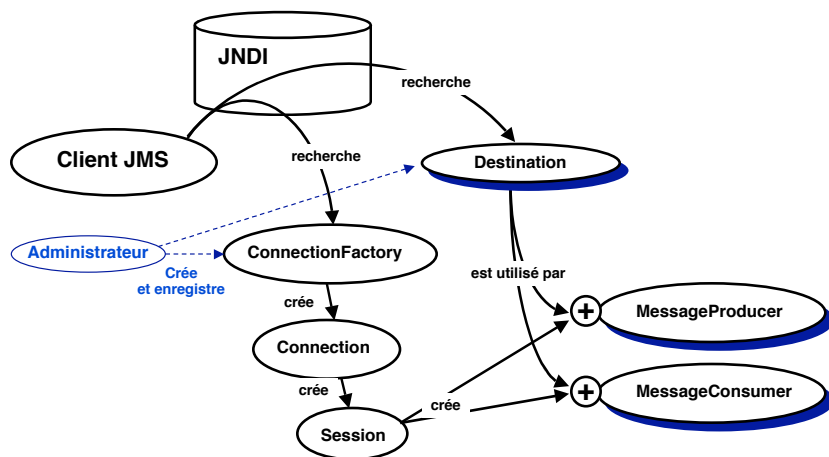
### ■ Event Service de CORBA

- ◆ L'un des *Object Services* définis par l'OMG
- ◆ Définit un ensemble de classes permettant de réaliser un service d'événements
- ◆ Supporte les modes *push* et *pull*

### ■ Jini (Sun)

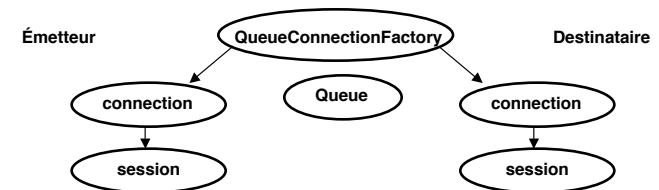
- ◆ Service de coordination à base d'objets partagés (Java)
- ◆ Utilisé pour des environnements dynamiquement variables (découverte de ressources)

## Architecture générale de JMS



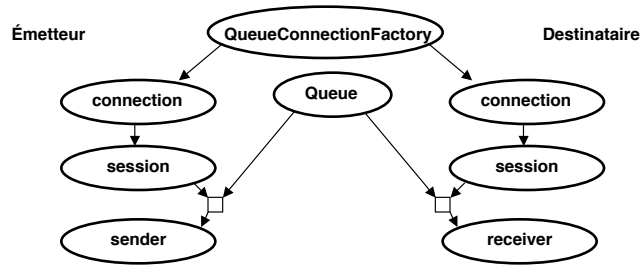
## JMS "Point-to-Point"

Opérations préalables :  
 création de *Queue* et *QueueConnectionFactory* par l'administrateur  
 enregistrement dans *messaging* sous JNDI



```
QueueConnectionFactory connectionFactory =
    (QueueConnectionFactory) messaging.lookup("nom_fabrique");
Queue queue =
    (Queue) messaging.lookup("nom_queue.");
QueueConnectionFactory connection = connectionFactory.createQueueConnection();
QueueSession session = connection.createQueueSession();
```

## JMS "Point-to-Point"



### Chez l'émetteur

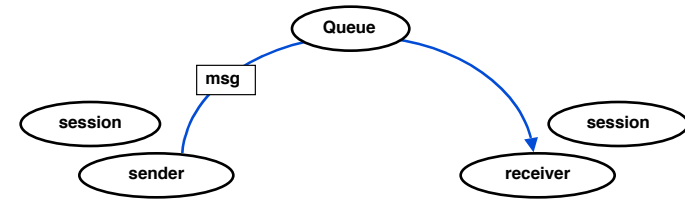
```
QueueSender sender =
    session.CreateSender (queue) ;
```

### Chez le destinataire

```
String selector =
    new String ("(name = 'Bull') or (name = 'IBM')");
QueueReceiver receiver =
    session.CreateReceiver (queue, selector) ;
```

Réception sélective des messages dont l'attribut *name* contient une chaîne donnée

## JMS "Point-to-Point"



### Chez l'émetteur

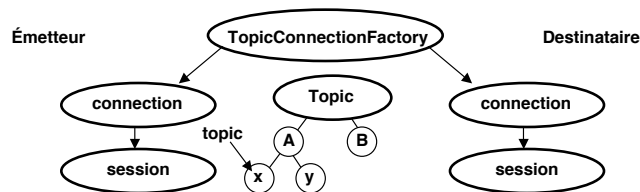
```
TextMessage msg =
    session.createTextMessage ;
msg.setText ("le texte du message") ;
Sender.send (msg) ;
```

### Chez le destinataire

```
TextMessage msg = (TextMessage)
    receiver.receive () ;
```

## JMS "Publish-Subscribe"

Opérations préalables :  
création de *Topic* et *TopicConnectionFactory* par l'administrateur  
enregistrement dans *messaging* sous JNDI



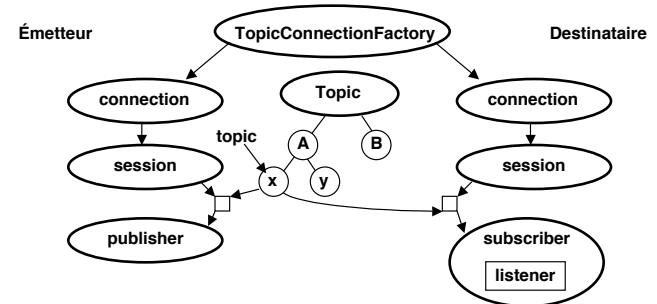
```
TopicConnectionFactory connectionFactory =
    (TopicConnectionFactory) messaging.lookup ("nom_fabrique") ;
Topic topic =
    (Topic) messaging.lookup ("/A/x") ;
```

```
TopicConnection connection = connectionFactory.createTopicConnection () ;
```

```
TopicSession session = connection.createTopicSession (false, Session.CLIENT_ACKNOWLEDGE) ;
```

pas en mode transaction
acquitté par client

## JMS "Publish-Subscribe"



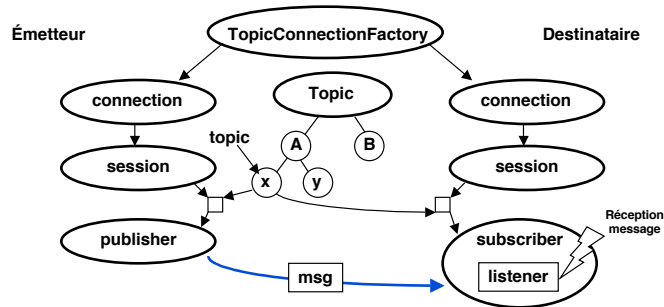
### Chez l'émetteur

```
TopicPublisher publisher =
    session.CreatePublisher (topic) ;
```

### Chez le destinataire

```
TopicSubscriber subscriber =
    session.CreateSubscriber (topic) ;
Subscriber.setMessageListener(listener)
```

## JMS "Publish-Subscribe"



Chez l'émetteur

```
publisher.publish (msg);
```

Chez le destinataire : programme du listener

```
public class listener implements javax.jms.MessageListener {
    void onMessage (Message msg) throws JMSEException {
        // déballer et traiter le message
        ...
    }
}
```

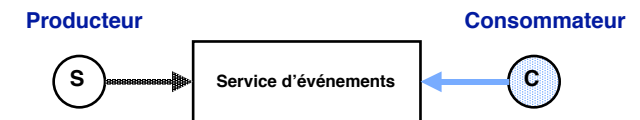
## Service d'événements CORBA

- Définit un ensemble de classes permettant la communication par événements
- Deux modes : *push* et *pull*

*push* (initiative émetteur, ou producteur d'événements)



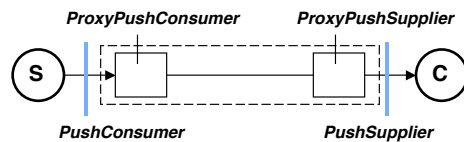
*pull* (initiative récepteur, ou consommateur d'événements)



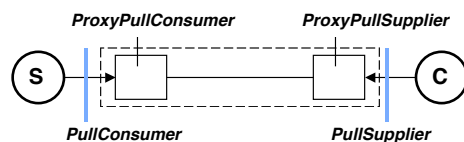
## Événements CORBA : principes de base

- Utilisation du principe du mandataire (*proxy*)
  - ◆ Chaque correspondant s'adresse au *proxy* de l'autre (des autres)
  - ◆ Le service d'événements fournit un mécanisme pour la création des *proxy*

Mode *push*



Mode *pull*



## Le module *EventComm*

- ◆ Fournit les 4 interfaces de base du service d'événements

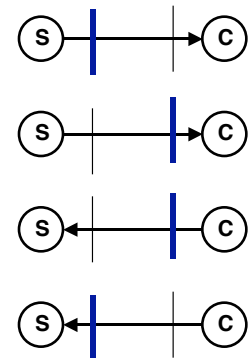
```
module CosEventComm {
    exception Disconnected {};

    interface PushConsumer {
        void push (in any data) raises (Disconnected);
        void disconnect_push_consumer ();
    };

    interface PushSupplier {
        void disconnect_push_supplier ();
    };

    interface PullSupplier {
        any pull () raises (Disconnected);
        any try_pull ();
        void disconnect_pull_supplier ();
    };

    interface PullConsumer {
        void disconnect_pull_consumer ();
    };
};
```



## Gestion des participants : *EventChannel*

### ■ Fonctions

- ◆ Gestion de la communication entre producteurs multiples et consommateurs multiples
- ◆ Ces participants sont connectés à travers un canal d'événements (*EventChannel*)



- ◆ Fonctionne selon le principe de la fabrique (*factory*)
- ◆ Connexion d'un nouveau participant = fabrication et fourniture du *proxy* correspondant à sa fonction (S ou C, *push* ou *pull*)
- ◆ L'interface *EventChannel* fournit les primitives de gestion des canaux d'événements

## Le module *EventChannelAdmin*

```
#include "EventComm.idl"
module CosEventChannelAdmin {
  exception AlreadyConnected {};
  exception TypeError {};

  interface ProxyPushConsumer ...
  ...
  interface ProxyPushSupplier ...
  ...
  interface ProxyPullConsumer ...
  ...
  interface ProxyPullSupplier ...
  ...

  interface ConsumerAdmin {
    ProxyPushSupplier obtain_push_supplier ();
    ProxyPullSupplier obtain_pull_supplier (); };

  interface SupplierAdmin {
    ProxyPushConsumer obtain_push_consumer ();
    ProxyPullConsumer obtain_pull_consumer (); };

  interface EventChannel {
    ConsumerAdmin for_consumers (); // ajouter nouveau consommateur
    SupplierAdmin for_suppliers (); // ajouter nouveau producteur
    void destroy (); // détruire le canal
  };
};
```

Les interfaces des *proxy*  
(détails plus loin)

L'interface de gestion  
des consommateurs

L'interface de gestion  
des producteurs

L'interface de gestion  
du canal d'événements

## Interfaces des *proxy*

```
module CosEventChannelAdmin {
  exception AlreadyConnected {};
  exception TypeError {};

  interface ProxyPushConsumer : CosEventComm ::PushConsumer {
    void connect push_supplier
      (in CosEventComm::PushSupplier push_supplier)
      raises AlreadyConnected; }

  interface ProxyPushSupplier : CosEventComm ::PushSupplier {
    void connect push_consumer
      (in CosEventComm::PushConsumer push_consumer)
      raises AlreadyConnected, TypeError; }

  interface ProxyPullConsumer : CosEventComm ::PullConsumer {
    void connect pull_supplier
      (in CosEventComm::PullSupplier pull_supplier)
      raises AlreadyConnected, TypeError; }

  interface ProxyPullSupplier : CosEventComm ::PullSupplier {
    void connect pull_consumer
      (in CosEventComm::PullConsumer pull_consumer)
      raises AlreadyConnected; }

  ...
};
```

## Étapes du développement d'une application

- ◆ Obtenir un canal d'événements (soit *channel*)
  - ❖ La gestion des canaux n'est pas du ressort du service d'événements
  - ❖ On utilise une fabrique de canaux et un serveur de noms
- ◆ Côté producteur (par ex. mode *push*) [très schématique]

```
supplier_admin = channel->for_suppliers();
proxy_push_consumer = supplier_admin->obtain_push_consumer();
proxy_push_consumer->connect_push_supplier(...);
```

Initialiser une structure de "callback" (appel en retour)  
pour activer le consommateur lors de l'arrivée du message

```
proxy_push_consumer->push (message);
```
- ◆ Côté consommateur

```
consumer_admin = channel->for_consumers();
proxy_push_supplier = consumer_admin->obtain_push_supplier();
proxy_push_supplier->connect_push_consumer (...);
```

Lors du "callback" signalant l'arrivée d'un message, extraire et traiter le message reçu

## Jini : un service de coordination à objets partagés

### ■ Motivations

- ◆ Développement d'applications réparties dans lesquelles l'environnement change dynamiquement par ajout ou suppression d'éléments (objets mobiles, appareils divers, etc.) et où l'application doit s'adapter dynamiquement à ces changements

### ■ Objectif

- ◆ Fournir un service assurant la coordination d'un ensemble de processus à travers un espace d'information partagé
- ◆ En particulier, permettre l'intégration et la découverte dynamiques de ressources dans un environnement réparti
- ◆ Implémentation : intégré à Java, mais les principes peuvent être réutilisés dans un environnement non Java
  - ❖ Repose sur un modèle de coordination par espace d'objets partagés (JavaSpaces)

## JavaSpaces : coordination par objets partagés

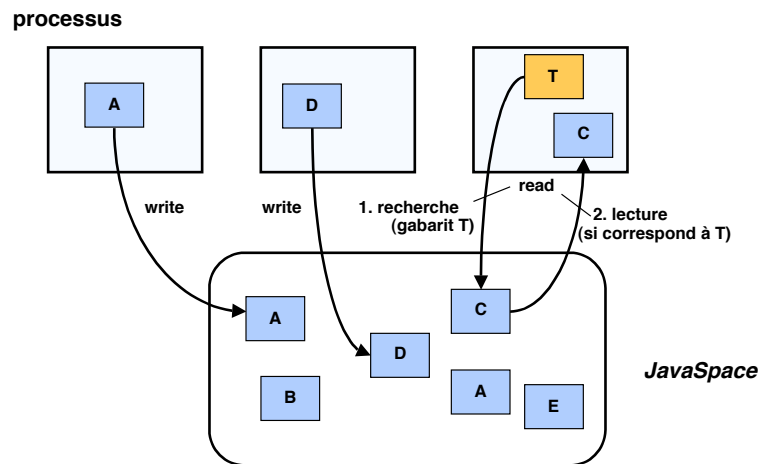
### ■ Espace de tuples

- ◆ Modèle inspiré de Linda
- ◆ Un *JavaSpace* = un espace partagé contenant des **tuples** (ensembles de références typées à des objets Java)
- ◆ Représentation d'un tuple : ensemble de références sérialisées

### ■ Opérations sur les tuples

- ◆ Écrire (*write*) : insérer un tuple dans un *JavaSpace*
- ◆ Lire : rechercher dans un *JavaSpace* un tuple filtré par un gabarit (*template*)
- ◆ Gabarit : modèle pour un tuple, contient des références typées, peut contenir des champs NULL (qui sont conformes à toute référence)
  - ❖ Opération bloquante pendant la recherche
  - ❖ Deux variantes
    - ▲ *read* : le tuple lu reste dans le *JavaSpace*
    - ▲ *take* : le tuple lu est extrait du *JavaSpace*

## Opérations sur un *JavaSpace*



## Architecture de Jini

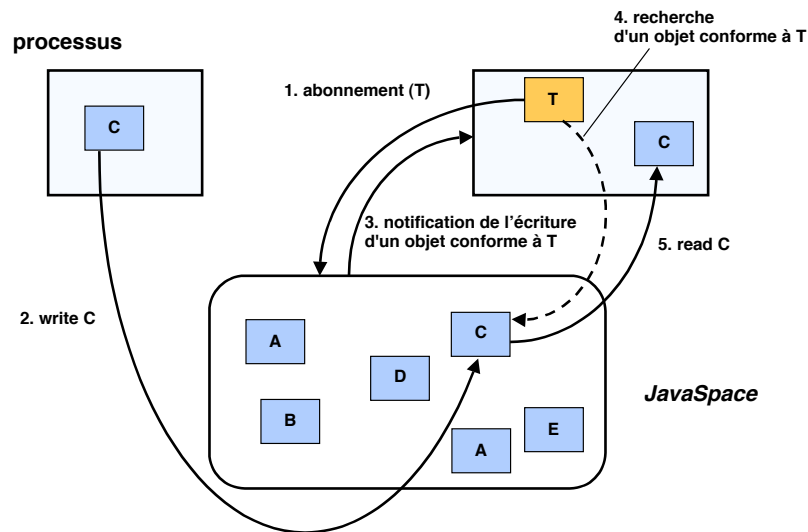
### ■ 3 niveaux

- ◆ Bas : infrastructure de communication (Java RMI), localisation de services
- ◆ Moyen : service d'événements ; notifications, réalisation de transactions, autres
- ◆ Haut : clients et services (sécurité, transactions, etc.)

### ■ Service d'événements (analogue à *publish-subscribe*)

- ◆ Abonnement : le processus client qui s'abonne indique un objet veilleur (*listener*) qui doit être appelé (via RMI) si l'événement arrive
- ◆ L'abonnement a toujours une durée limitée (doit être renouvelé au delà)
- ◆ Cas particulier d'événement : écriture d'un tuple, correspondant à un gabarit spécifié, dans un *JavaSpace*

## Événements



## Problèmes d'implémentation

### ■ Implémentation répartie d'un *JavaSpace*

- ◆ Le *JavaSpace* est dupliqué sur toutes les machines
  - ❖ Accès efficace (local)
  - ❖ Mais il faut garder la cohérence, cher si modifications fréquentes
  - ❖ Nécessite service de diffusion (fiable) - cf plus loin dans le cours, tolérance aux fautes
- ◆ Le *JavaSpace* n'est pas dupliqué
  - ❖ Accès distant nécessaire hors du site de résidence
  - ❖ Gestion simple, mais problèmes d'efficacité
- ◆ Solutions mixtes : duplication partielle

## Localisation de services

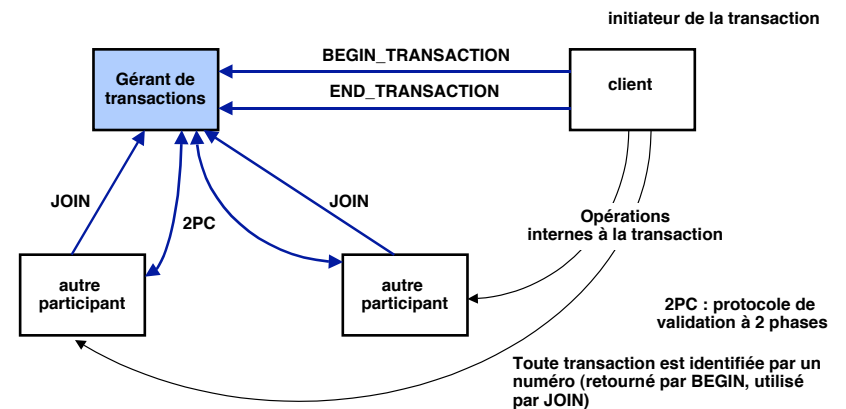
### ■ Une application Jini peut utiliser divers services

#### ■ Les services doivent être localisés

- ◆ La localisation utilise un principe analogue aux *JavaSpaces*, mais plus efficace : recherche par gabarit
- ◆ Chaque service s'enregistre avec des couples (attribut, valeur)
- ◆ La recherche se fait sur un ou plusieurs attributs (cf pages jaunes)
- ◆ Plusieurs services de localisation peuvent coexister
- ◆ Les services de localisation sont eux-mêmes localisés
  - ❖ Au moyen d'une recherche par diffusion
  - ❖ En se signalant eux-mêmes
  - ❖ Noter qu'il n'y a pas d'adresse fixe pour un service primitif de localisation

## Synchronisation

### ■ Outil de base : transactions



## Références

---

Modèles à bus de messages (M. Riveill, A. Freyssinet)

<http://www.essi.fr/~riveill/cours/grecoinfo/ch3-bus-de-message.pdf>

**Systèmes de communication par queues de messages**

MQSeries d'IBM : <http://www.software.ibm.com/ts/mqseries>

MessageQ de BEA : <http://www.beasys.com/products/mq/mqdata.htm>

FalconMQ : <http://www.level8.com>

**Bus logiciels**

AAA : [http://www.newcastle.research.ec.org/c3ds/trs/papers\\_pdf/c11agent\\_infra.pdf](http://www.newcastle.research.ec.org/c3ds/trs/papers_pdf/c11agent_infra.pdf)

Ambrosia : <http://www.openhorizon.com>

JMS : <http://www.java.com/jms>

iBus : <http://www.softwired.ch/ibus>

Tibco : <http://www.rv.tibco.com>

**Service d'événements de CORBA (avec exemples d'utilisation)**

D. C. Schmidt, S. Vinoski. OMG Event Object Service, SIGS, vol. 9, 2, Feb. 1997

<http://www.cs.wustl.edu/~schmidt/C++-report-col9.ps.gz>

D. C. Schmidt. On Overview of OMG CORBA Event Services (flips)

<http://www.cs.wustl.edu/~schmidt/coss4.ps.gz>

**Jini**

J. Waldo, The Jini Specifications, 2nd ed. Prentice Hall, 2000

Ressources sur Jini : <http://www.jini.org>