SCALAGENT Distributed Technologies

Systèmes et applications asynchrones
Middleware à message

Roland Balter
ScalAgent Distributed Technologies
Roland.Balter@scalagent.com

www.scalagent.com

Plan du cours

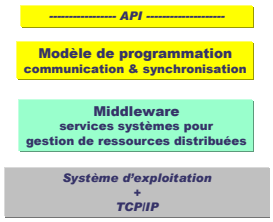
I. Caractérisation des systèmes asynchrones
II. Modèles pour la programmation asynchrone
III. Middleware asynchrone (MOM)
IV. JMS : un exemple de programmation asynchrone
V. JORAM : un exemple de middleware asynchrone
VI. Etude de cas
VII. Conclusion : état des lieux et perspectives

Objectifs du cours

- **Contexte : Module CR - Construction d'applications parallèles et réparties**
  - Introduction aux systèmes répartis
  - Caractéristiques des systèmes répartis
- **Zoom sur les applications asynchrones**
  - Importance croissante du domaine des applications asynchrones
    - *Internet à grande échelle, réseaux sans fil, terminaux mobiles*
  - Présenter les problèmes posés par la conception et la mise en œuvre des systèmes asynchrones
  - Présenter les solutions en usage aujourd'hui (modèle de programmation et middleware) et les illustrer à l'aide d'exemples d'applications
  - Etat des lieux et perspectives du domaine : industrie et recherche

Architecture distribuée

- **Application/système distribué**
  - Ensemble de composants logiciels coopérants
  - Coopération = communication + synchronisation
- **Eléments de choix d'une architecture distribuée**



CORBA, JMS, . .

Synchrone : Client-serveur (RPC, RMI, ORB, etc.)
Asynchrone : messages et événements
Objets mobiles
Objets partagés

Flots d'exécution, mémoire, persistance
Fiabilité, disponibilité, sécurité, scalabilité, . .

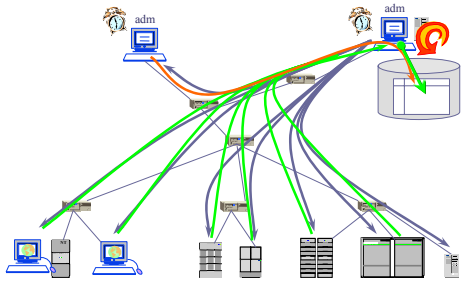
Exemple : supervision de ressources distribuées

- **Cahier des charges**
  - Surveillance de l'état de machines, de systèmes d'exploitation et d'applications dans un environnement distribué.
    - *Flot continuel de données (d'état et d'usage) en provenance de sources hétérogènes sur le réseau*
  - Les éléments du système peuvent apparaître ou disparaître dynamiquement
    - *connexion – déconnexion,*
    - *évolution du système (exemple : nouvelle machine)*
  - Les administrateurs doivent pouvoir accéder à l'information quelle que soit leur localisation

Solution (traditionnelle) client / serveur

- **Interrogation régulière des éléments à surveiller par l'application d'administration et mise à jour d'une base de données centralisée.**
  - Utilisation d'une configuration complexe afin de connaître l'ensemble des éléments à surveiller.
  - Maintien de cette configuration lorsque des machines ou des applications rejoignent, quittent ou se déplacent dans le système.
- **Interrogation par les administrateurs de la base centrale.**

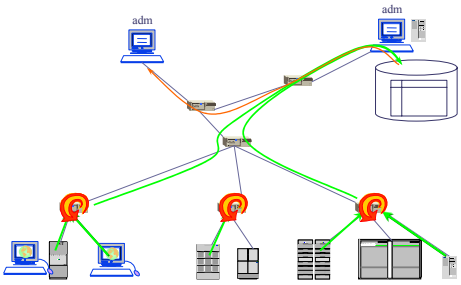
Solution traditionnelle client / serveur



Solution « Messaging »

- **Les différents éléments administrés émettent des messages** :
  - changements d'état et de configuration
  - alertes, statistiques
- **Un ou plusieurs démons reçoivent ces notifications et maintiennent l'état courant du système**
  - suivi des changements de configuration dynamiques
  - émission de messages signalant les changements d'états significatifs ou les mises à jour
- **Inversion des rôles des producteurs et des consommateurs de données**

Solution « Messaging »



Un peu d'histoire

- **Les systèmes asynchrones sont en usage dans le monde de l'Internet depuis longtemps**
  - Le courrier électronique (communication point-à-point)
    - *le producteur envoie un message à un destinataire qu'il connaît*
    - *le message est stocké sur un serveur, le consommateur lit ultérieurement le message lorsqu'il se connecte*
  - Les listes de diffusion (communication multi-points)
    - *Le message est diffusé à tous les éléments de la liste*
  - Les news (Anonymat, Publish/Subscribe)
    - *le consommateur s'abonne à une liste de diffusion*
    - *le producteur publie une information dans un forum*
    - *le consommateur lit le contenu du forum quand il le souhaite*

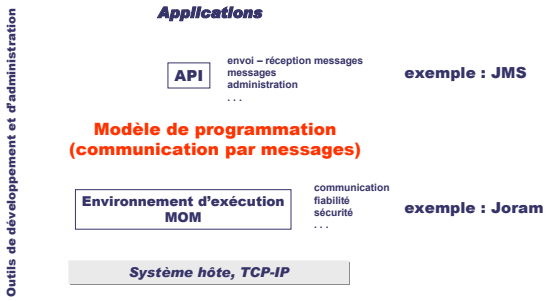
Bilan : les atouts du mode "asynchrone"

- **Systèmes faiblement couplés**
  - Couplage temporel : systèmes autonomes communicants
    - *Communication « spontanée » en mode « push »*
    - *Fonctionnement en mode déconnecté : site absent ou utilisateur mobile*
  - Couplage spatial : systèmes à grande échelle
    - *Fonctionnement en mode partitionné : pannes temporaires de réseau*
    - *Communication « anonyme » : non connaissance des correspondants*
- **Simplicité**
  - Modèle de communication « canonique »
    - *Envoi de message*
    - *Base universelle : TCP-UDP/IP*

Les usages des systèmes asynchrones

- **Supervision**
  - Parc d'équipements distribués
  - Applications distribuées
- **Echange et partage de données**
  - Envoi de documents (EDI)
  - Mise à jour d'un espace de données partagées distribuées
- **Intégration de données**
  - Alimentation d'un datawarehouse/datamart depuis des sources de données hétérogènes autonomes : ETL (Extract – Transfer – Load)
- **Intégration d'application**
  - Intra-entreprise : EAI (communication, routage, workflow)
  - Inter-entreprises : B2B et Web Services (communication, orchestration)
- **Informatique mobile**
  - Communication entre équipements mobiles (souvent déconnectés) et serveurs d'appplication

Systèmes asynchrones : mode d'emploi

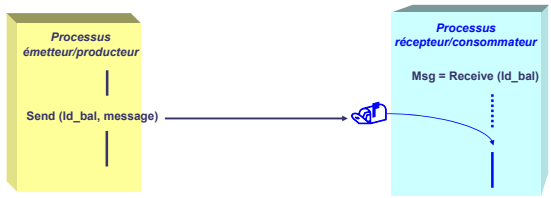


Communication par messages : modèle de programmation

- **Mode de synchronisation**
  - *Communication asynchrone*
    - *émission non bloquante*
    - *réception bloquante (attente jusqu'à arrivée d'un message, ou retour d'erreur)*
- **Mode de désignation**
  - communication directe entre processus (*acteurs, agents, . . .*)
  - communication indirecte entre processus via des objets intermédiaires (*portes, boîtes aux lettres, queues de message, etc.*)
- **Mode de transmission**
  - messages possiblement typés

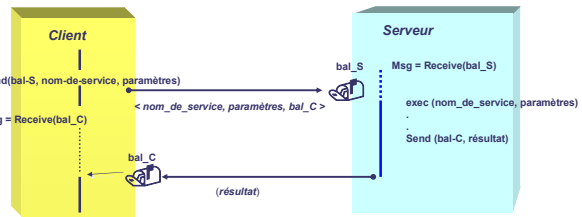
Communication par messages : exemple 1

- **Réalisation d'un échange asynchrone**



Communication par messages : exemple 2

- **Réalisation d'une interaction de type « client-serveur »**



Communication par messages : exemples d'environnements existants

- **Environnement de type "micro-noyau"**
  - mécanisme et primitives de base
  - exemples : Chorus, Mach/OSF-1
- **Environnement "à la unix"**
  - "sockets"
- **Environnement de programmation parallèle**
  - PVM et/ou MPI
- **Environnement d'intégration d'applications (et/ou de données)**
  - middleware à messages: MOM
  - interface de programmation ad hoc
    - *tentative de normalisation via Java JMS*

Les éléments de définition d'un système de messagerie

- **Modèle de programmation**
  - Structure des messages
  - Mode de production des messages : non bloquant
  - Mode de désignation : indirect, groupe, anonyme
  - Mode de communication : point à point, multi-points
  - Mode de consommation des messages : push, pull
- **API**
  - JMS (*Java Messaging Service*)
- **Environnement d'exécution (*run time*)**
  - Architecture : centralisée, partitionnée, répartie
  - Qualité de service : fiabilité, sécurité, scalabilité, etc.

Plan du cours

I. Caractérisation des systèmes asynchrones
II. Modèles pour la programmation asynchrone
III. Middleware asynchrone (MOM)
IV. JMS : un exemple de programmation asynchrone
V. JORAM : un exemple de middleware asynchrone
VI. Etude de cas
VII. Conclusion : état des lieux et perspectives

Format des messages

- **Entête**
  - Information permettant l'identification et l'acheminement du message
    - *Id. unique, destination, priorité, durée de vie, etc.*
- **Attributs**
  - Couples (nom, valeur) utilisables par le système ou l'application pour sélectionner les messages (opération de filtrage)
- **Données définies par l'application**
  - *Texte*
  - *Données structurées (XML)*
  - *Binaire*
  - *Objets (sérialisés)*
  - . . .

Modes de désignation

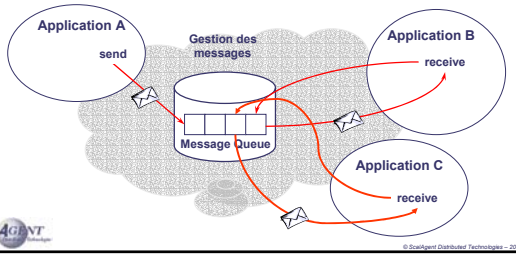
- **Désignation indirecte**
  - Les entités communiquent via un objet intermédiaire : ***destination***
    - *Destination : structure de données réceptacle de messages*
    - *Exemple : Queue (file) de messages*
- **Désignation de groupe**
  - groupe = ensemble de consommateurs identifiés par un nom unique
    - *gestion dynamique du groupe : arrivée/départ de membres*
    - *différentes politiques de service dans le groupe : 1/N, N/N*
- **Désignation anonyme**
  - désignation associative : les destinataires d'un message sont identifiés par des propriétés (attributs du message)
  - Publish/Subscribe (Publication / Abonnement)

Modes de communication

- **4 Relations entre producteur(s) et consommateur(s)**
  - 1 producteur → 1 consommateur
  - 1 producteur → N consommateurs
  - P producteurs → 1 consommateur
  - P producteurs → N consommateurs
- . . ***mais seulement***
- **2 Modèles de communication de base**
  - Point-To-Point : 1 producteur → 1 consommateur
  - Multi-points : 1 producteur → N consommateurs

Modèle Point à point (1/2)

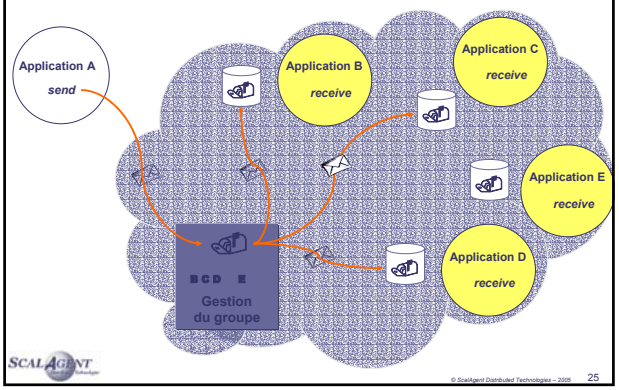
- **Un message émis sur une queue de messages donnée est consommé par une unique application**
  - asynchronisme et fiabilité



Modèle Point à Point (2/2)

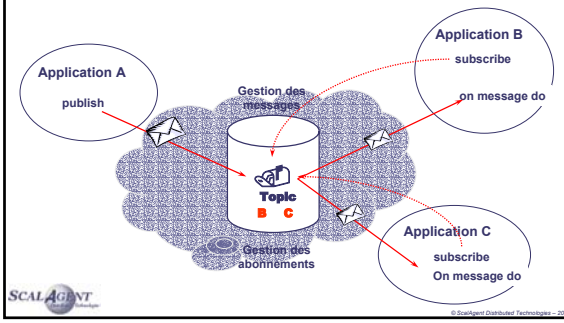
- **Séparation entre destination et consommateur**
  - destination statique : commune à plusieurs producteurs/consommateurs
  - Consommateur unique pour un message donné
- **Indépendance du producteur et du consommateur**
  - Via l'objet « Queue de messages »
  - Rend possible l'évolution de la relation producteur - consommateur
  - Indépendance temporelle : asynchronisme
- **Acquittement du traitement par le consommateur**
  - Acquittement de niveau système
    - *Libération de l'emplacement du message dans la queue*
    - *Fiabilisation de l'opération de consommation*
  - Acquittement de niveau applicatif : à programmer explicitement

Modèle multi-points : communication de groupe (1/2)



Modèle Publish/Subscribe (1/2)

- Un message émis vers un sujet (*Topic*) est délivré à l'ensemble des applications abonnées à ce *Topic*.

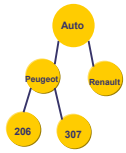


Communication de groupe (2/2)

- **Gestion du groupe**
  - Communication multi-points (1 producteur → N consommateurs)
  - Indépendance vis-à-vis des émetteurs
  - Gestion dynamique
    - *Arrivée / départ de membres*
- **Politiques de gestion des messages**
  - Persistance, durée de vie
  - Historique (forum)
- **Politiques de service des messages**
  - 1 / N : répartition de charge
  - N / N : réplication (tolérance aux pannes)
  - P / N : réplication partielle

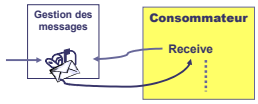
Modèle Publish/Subscribe (2/2)

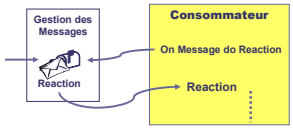
- **Relation producteur - consommateur**
  - Communication multi - points : 1 producteur → N consommateurs
  - Désignation anonyme via le *Topic*
  - Dépendance temporelle
    - *Le message est délivré aux consommateurs « actifs » lors de la production*
- **Critères d'abonnement**
  - Statique : « subject based »
    - *Organisation plate ou hiérarchique des sujets*
  - Dynamique : « content based »
    - *Implantation distribuée délicate*
- **Abonnements temporaires/durables**



Modes de consommation

- « *Pull* » – consommation explicite
  - Les consommateurs programment explicitement l'accès aux messages
  - En cas d'absence de message : attente ou exception
- « *Push* » – consommation implicite
  - Une méthode prédéfinie (*réaction*) est attachée à la production d'un message (*événement*)
  - L'occurence d'un événement entraîne l'exécution de la réaction associée.
  - ➔ *Modèle Evénement / Réaction*





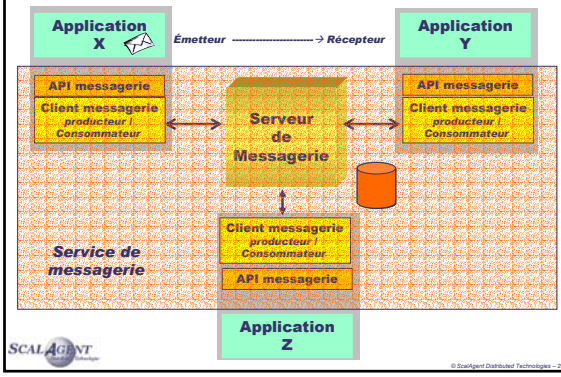
Plan du cours

I. Caractérisation des systèmes asynchrones
II. Modèles pour la programmation asynchrone
III. Middleware asynchrone (MOM)
IV. JMS : un exemple de programmation asynchrone
V. JORAM : un exemple de middleware asynchrone
VI. Etude de cas
VII. Conclusion : état des lieux et perspectives

Environnement d'exécution (run time)

- Architecture logique
  - Service de messagerie (*message server, message provider*)
    - *Gestion des messages (queues, topics)*
    - *Gestion des connexions avec les clients : producteurs, consommateurs*
  - Clients du service : producteurs, consommateurs
- Architecture physique
  - Centralisée, partitionnée, répartie
- Qualité de service
  - *Disponibilité, fiabilité*
  - *Sécurité*
  - *Performance*
  - *scalabilité*

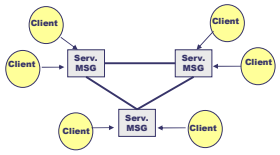
Architecture logique



Architecture physique

- **Serveur centralisée**
  "Hub and Spoke"

- **Serveur distribué**
  "Snowflake"



- **Serveur distribué**
  Bus

Administration

- **Administration du MOM**
  - Configuration, déploiement
  - Monitoring
- **Administration du service de messagerie**
  - Service de désignation : identification des serveurs, émetteurs/récepteurs
  - Service de messagerie : gestion des objets de communication : queues, topics, etc.
    - *Configuration, déploiement*
    - *Monitoring, reconfiguration*
  - Clients : création/destruction, droits d'accès

Qualité de service

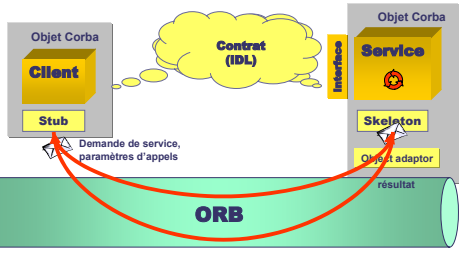
- **Disponibilité et fiabilité**
  - Persistance des messages et Garantie de délivrance
    - *Au plus une fois, au moins une fois, exactement une fois*
- **Performance et scalabilité**
  - Nombre de sites, nombre de messages, taille des messages
    - *Réseau local (Intranet)*
    - *Réseaux hétérogènes à grande échelle (Internet)*
- **Transaction**
- **Sécurité**
- **Répartition de charge**
- **Ordonnancement**

Systèmes asynchrones : technologies alternatives

- **Web asynchrone**
  - HTTP asynchrone ('one way')
    - *Requête sans réponse*
    - *Mêmes propriétés que HTTP (fiabilité, sécurité, etc.)*
- **CORBA**
  - Corba Notification Service
- **Communication asynchrone en Java**
  - JAXM
  - JINI
  - **JMS**

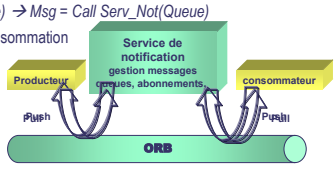
Les systèmes asynchrones dans le monde Corba (1/2)

- **Bus Corba**
  - Modèle de communication synchrone (client-serveur)
  - API : interfaces IDL
  - Middleware : ORB (Object Request Broker)



Les systèmes asynchrones dans le monde Corba (2/2)

- **Service de Notification Corba**
  - Service de notification : courtier de messages
    - *Gestionnaire des messages, queues, abonnements, . .*
    - *Accessible via des interfaces IDL*
  - Production
    - *Send (msg, Queue) → Call Serv_Not(msg, queue)*
  - Consommation
    - *Msg = Receive (Queue) → Msg = Call Serv_Not(Queue)*
  - Modes de production – consommation
    - *Mixte push – pull*
  - QoS
    - ***Garantie délivrance***
    - ***Reprise après panne***
    - ***Persistance***

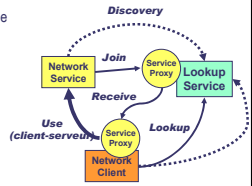


Messagerie Java : JAXM

- **JAXM: Java API for XML Messaging**
  - API Java pour émettre et recevoir des messages SOAP
  - Usage de profils de messagerie définis sur SOAP
  - Support de protocoles de transport variés : HTTP, SMTP, POP, . .
  - Modèles de programmation
    - *Point à point : client-serveur (sans garantie de délivrance)*
    - *Asynchrone : via un service de messagerie (JAXM provider)*
  - Différences avec JMS
    - *Système de messagerie « léger » (version stand-alone sans service de msg.)*
    - *Interopérabilité avec un client SOAP 1.1*
    - *Pas mature – peu d'implémentations disponibles*

JINI

- **La technologie JINI**
  - Modèle de programmation pour construire un système comme une fédération de services et de clients
  - Run time pour ajouter, enlever, localiser et accéder aux services
- ***Discovery, Join & Lookup***
  - ***Discovery*** : localisation d'un service de recherche
  - ***Join*** : enregistrement d'un service
  - ***Lookup*** : recherche d'un service
- ***Distributed events***
  - ***Event generator*** : génère des événements sur changement d'état
  - ***Event listener*** : s'abonne aux événements de l'Event Generator
  - ***Événement*** : identifié par un ID-événement et un ID-source



Plan du cours

I. Caractérisation des systèmes asynchrones
II. Modèles pour la programmation asynchrone
III. Middleware asynchrone (MOM)
IV. JMS : un exemple de programmation asynchrone
V. JORAM : un exemple de middleware asynchrone
VI. Etude de cas
VII. Conclusion : état des lieux et perspectives

Présentation de JMS

- JMS en bref : objectifs et limitations
- Les concepts de JMS
- JMS en mode point à point
- JMS en mode Publish – Subscribe
- Qualité de service (QoS)

JMS en bref (1/2)

- **Spécification d'un système de messagerie Java**
  - API Java entre une application et un bus à messages
  - Modèles de communication
    - *Point à point : message queuing*
    - *Multi points : publish – subscribe*
  - Divers types de données échangées : binaire, objets, texte, XML, . .
- **JMS ne définit pas le mode de fonctionnement du bus**
- **Les applications JMS sont indépendantes d'un bus (e.g. objectif de portabilité)**
  - … mais l'interopérabilité entre des plates-formes JMS hétérogènes nécessite la définition d'une passerelle entre les bus à messages

JMS en bref (2/2)

- **La spécification JMS n'est pas complète**
  - e.g. déploiement et administration sont spécifiques d'un produit donné
  - Les produits offrent des fonctions additionnelles : "topics" hiérarchiques, . .
- **Le support de JMS est un élément stratégique de la spécification J2EE 1.4**
- **La dernière spécification JMS 1.1. unifie la manipulation**
  - des "*queues*" : communication point-à-point
  - et des "*topics*" : communication multi-points (Publish-Subscribe)
    - *API simplifiée*
    - *Ressources réduites*

Les concepts de JMS

Éléments d'architecture

Serveur JMS
(JMS Provider)

Client JMS

Le fournisseur d'une solution JMS

Objets administrés

ConnectionFactory

Destination
(Queue & Topic)

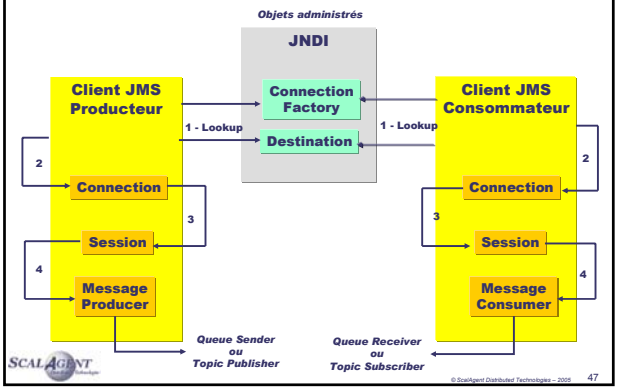
L'administrateur (administration JMS)

Objets de communication

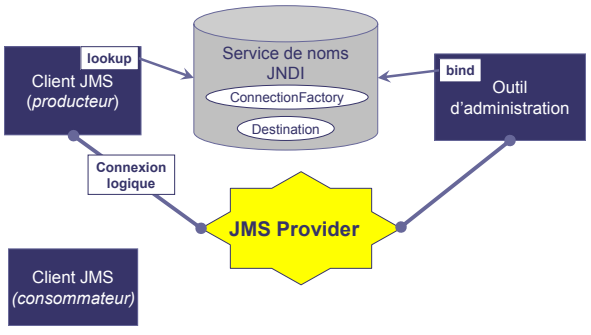
Connexions
Sessions
Producteur/consommateur
Messages

Le programmeur d'application (API JMS)

Objets de communication et diagramme de séquence



Architecture d'une application JMS



« Messaging Domains »

- **Point-to-Point**
- **Publish/Subscribe**
- **JMS 1.1 : unification des domaines**
  - Réduit et simplifie l'API (à terme)
  - Permet l'utilisation de Queues et Topics dans une même session (transaction)

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

Les objets JMS

- **Objets administrés**
  - *ConnectionFactory* : point d'accès à un serveur MOM
  - *Destination* : *Queue* ou *Topic*
- **Objet *Connection***
  - Crée à partir d'un objet *ConnectionFactory*
  - Authentifie le client et encapsule la liaison avec le JMS provider
  - Gère les *Sessions*
- **Objet *Session***
  - Créé à partir d'un objet *Connection*
  - Fournit un contexte (transactionnel) mono-threadé de production/consommation de messages
  - Gère les acquittements de messages et les transactions

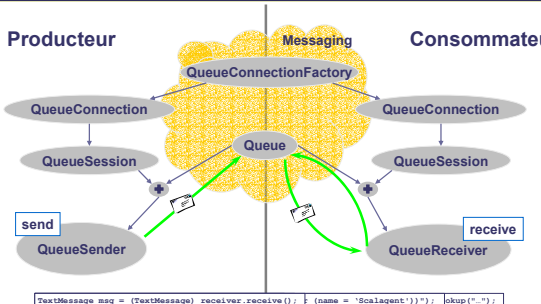
Les objets JMS

- **MessageProducer**
  - Fabriqué par la session → *QueueSender*, *TopicPublisher*
  - Permet l'émission de message → méthodes *Send*, *Publish*
- **MessageConsumer**
  - Fabriqué par la session → *QueueReceiver*, *TopicSubscriber*
  - Permet la réception de message
    - Synchrone → méthodes *Receive { (), (timeout)} ReceiveNoWait ()*
    - Asynchrone → objet *MessageListener* à l'écoute des messages
      exécution de la méthode *onMessage* de l'objet
  - Permet le filtrage des messages
    - *Syntaxe proche d'une condition SQL appliquée sur les champs de l'en-tête et des propriétés*

Le message JMS

- **Entête**
  - JMSMessageId, JMSDestination, JMSDeliveryMode, JMSExpiration, JMSPriority, etc.
- **Propriétés**
  - Couple <nom, valeur>
- **Corps**
  - TextMessage, MapMessage
  - StreamMessage, ObjectMessage
  - BytesMessage

JMS - "Point-to-Point"



JMS - "Point-to-Point"

```
QueueConnectionFactory connectionFactory = (QueueConnectionFactory) messaging.lookup("…");
Queue queue = (Queue) messaging.lookup("…");
QueueConnection connection = connectionFactory.createQueueConnection();
connection.start();
QueueSession session = connection.createQueueSession(…);
```

```
QueueSender sender =
session.createSender(queue);
String selector = new String("(name = 'ObjectWeb') or (name = 'Scalagent')");
QueueReceiver receiver = session.createReceiver(queue, selector);
```

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

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


JMS - "Publish/Subscribe"

```
TopicConnectionFactory connectionFactory = (TopicConnectionFactory) messaging.lookup("…");
Topic topic = (Topic) messaging.lookup("/A/x");
TopicConnection connection = connectionFactory.createTopicConnection();
connection.start();
TopicSession session = connection.createTopicSession(false, Session.CLIENT_ACKNOWLEDGE);
```

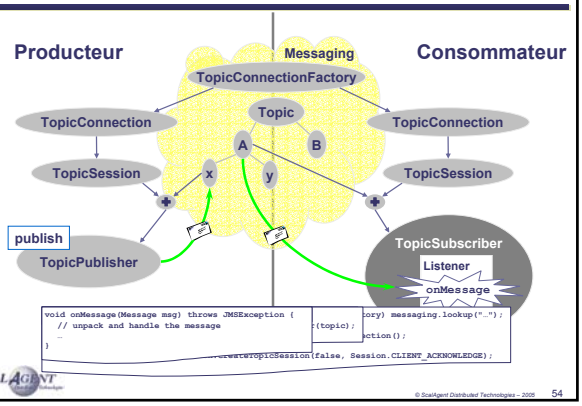
```
TopicPublisher publisher = session.createPublisher(topic);
```

```
Topic topic = (Topic) messaging.lookup("/A");
TopicSubscriber subscriber = session.createSubscriber(topic);
Subscriber.setMessageListener(listener);
```

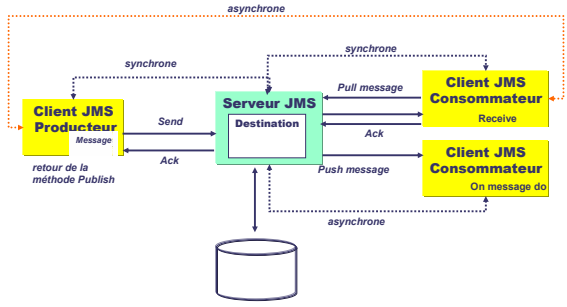
```
publisher.publish(msg);
```

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


JMS - "Publish/Subscribe"

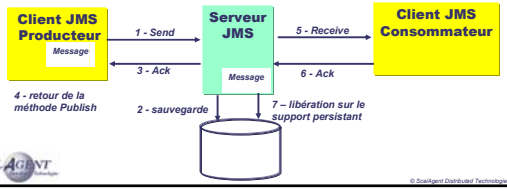


Production et consommation des messages



Qualité de service

- **Abonnements durables (*durable subscriptions*)**
  - ID unique associé à un abonnement durable
  - Si l'abonné n'est pas actif, le serveur JMS stocke le message sur un support persistant jusqu'à l'activation (ou jusqu'à expiration du TTL)
  - Par défaut, les abonnements ne sont pas durables
- **Messages persistants**



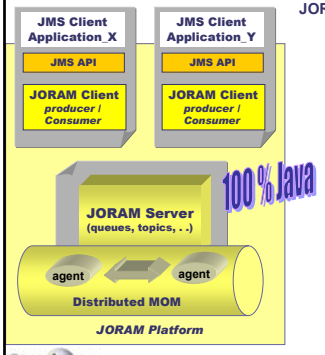
Plan du cours

I. Caractérisation des systèmes asynchrones
II. Modèles pour la programmation asynchrone
III. Middleware asynchrone (MOM)
IV. JMS : un exemple de programmation asynchrone
V. JORAM : un exemple de middleware asynchrone
VI. Etude de cas
VII. Conclusion : état des lieux et perspectives

JORAM

- **Implantation open source de la spécification JMS**
  - API JMS : communication asynchrone entre applications Java
  - Bus à messages distribué
- **Disponible sur ObjectWeb**
  - http://joram.objectweb.org
  - Développé initialement par une équipe mixte Bull – Université – INRIA
  - Développement et support par *ScalAgent Distributed Technologies*
- **Usage double**
  - Service de messagerie autonome pour applications Java (J2EE à J2ME)
  - Composant de messagerie asynchrone intégré dans un serveur d'application J2EE (JonAS, Jboss, autres)

JORAM : une plate-forme JMS

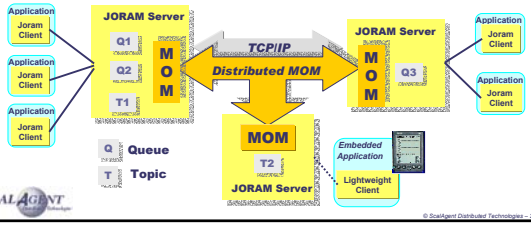


JORAM est une plate-forme JMS open source

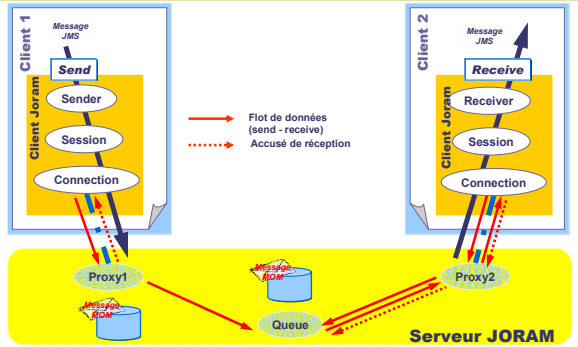
- conforme à la specification JMS
- qui fournit les deux modes de communication point-à-point et Publish/Subscribe
- qui s'appuie sur un MOM fiable, construit à l'aide d'une technologie d'agents distribués

JORAM : architecture logique

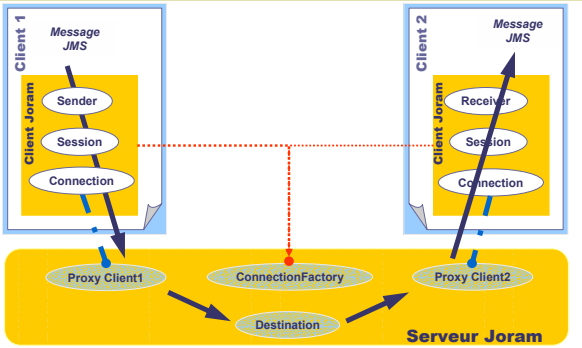
- **Architecture distribuée (type "snowflake")**
  - Serveurs de messagerie interconnectés sur Internet
- **Architecture configurable**
  - Serveurs et clients associés
  - Objets de communication (*queues, topics*)
  - Protocoles de communication, sécurité, persistance, etc.



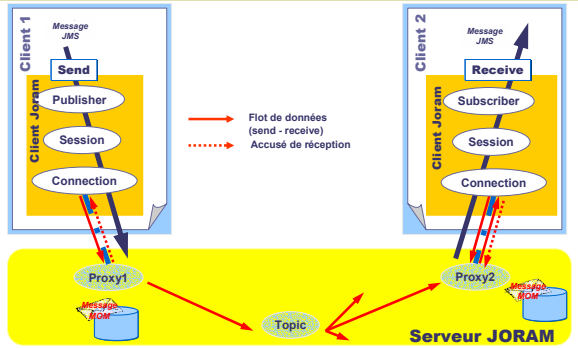
JORAM – Point To Point



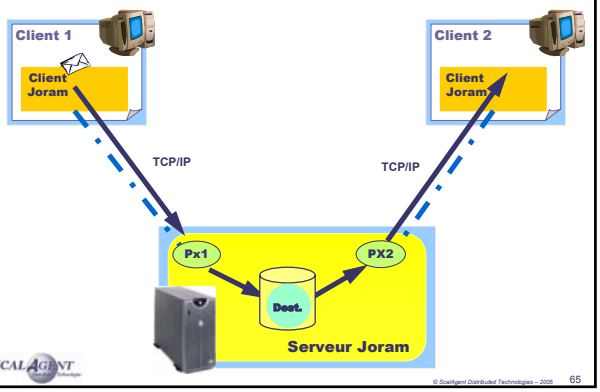
JORAM – Architecture logique



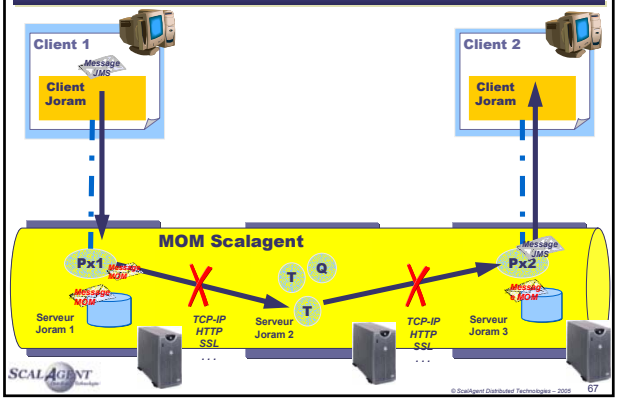
JORAM – Publish/Subscribe



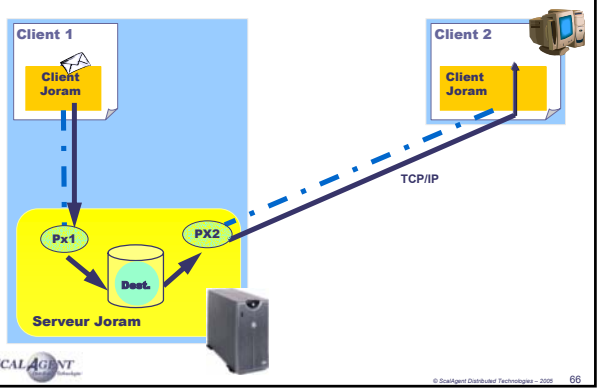
JORAM – Architecture centralisée



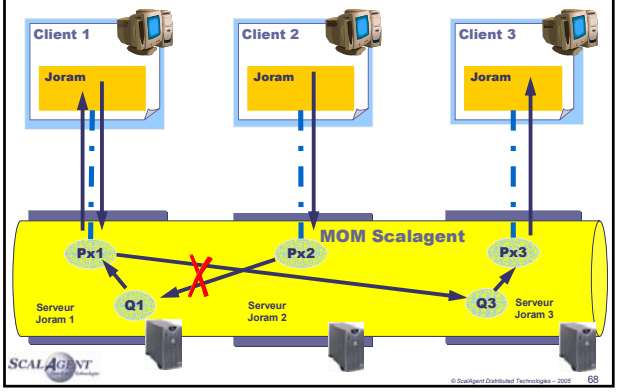
JORAM – Architecture distribuée



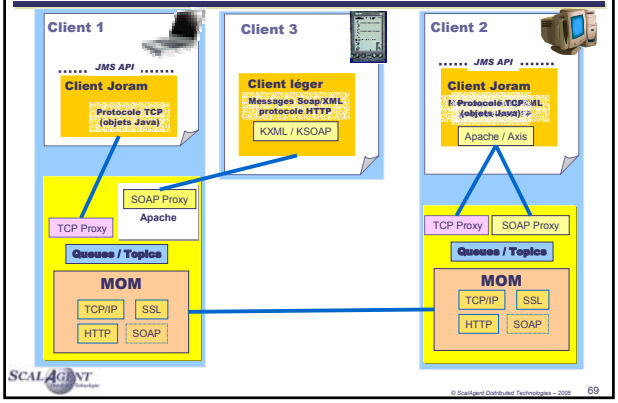
JORAM – Serveur co-localisé



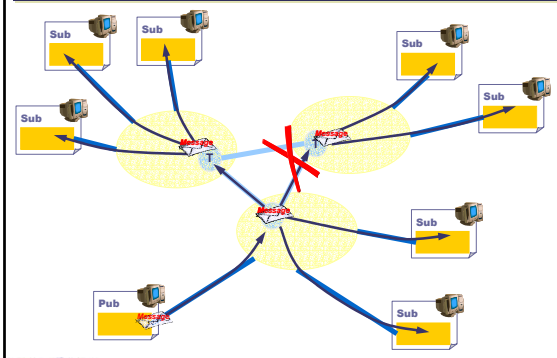
JORAM – Architecture distribuée



JORAM : protocoles de communication



Joram – Topic « clusterisé »

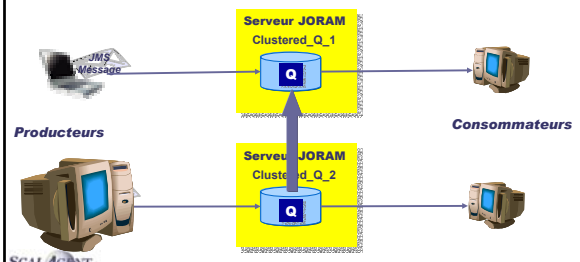


JORAM V4.x – fonctions avancées

- **Répartition de charge**
  - Topic "clustérisé"
  - Queue "clustérisée"
- **Haute disponibilité**
  - Serveur Joram HA
    - *Queues et topics répliqués*
- **Connectivité**
  - Passerelle JMS
  - Passerelles Mail et FTP
  - Client "léger" - KJoram
  - Support de JCA 1.5
- **Annuaire JNDI distribué**
- **Administration**
  - Support de l'API JMX
- **Sécurité**
  - Liens sécurisés par protocole SSL (Server to Server, Client to Server)
  - Gestion des pare-feux
- **Joram "embarqué"**
  - "bundle" JORAM pour passerelle OSGi
  - Serveur embarqué
    → communication peer-to-peer

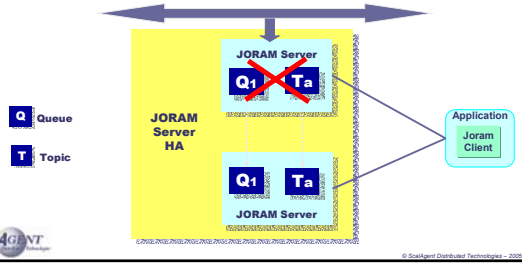
JORAM – Clustered Queue

- **"Clustered" Queue = ensemble de queues coopérantes**
  - Indépendance des queues (les accès locaux sont privilégiés)
  - Répartition de charge lorsque le nombre de messages reçus/demandés excède une limite
  - La sémantique des Queues est respectée (i.e. un seul consommateur)

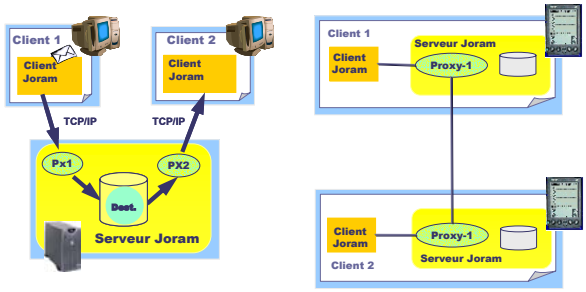


Haute disponibilité : JORAM HA

- Serveur JORAM HA
  - Architecture physique en grappe (*cluster*)
  - Réplication (Queues et Topics) et redondance active



Communication peer-to-peer serveur JORAM « embarqué »

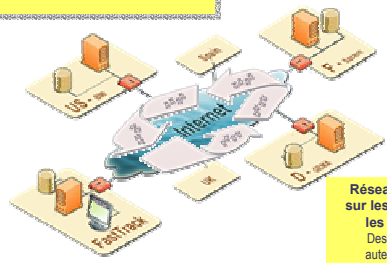


Plan du cours

I. Caractérisation des systèmes asynchrones
II. Modèles pour la programmation asynchrone
III. Middleware asynchrone (MOM)
IV. JMS : un exemple de programmation asynchrone
V. JORAM : un exemple de middleware asynchrone
VI. Etude de cas
VII. Conclusion : état des lieux et perspectives

Etude de cas : FastTrack

FastTrack est un GIE regroupant des sociétés de gestion de droits d'auteurs du monde entier



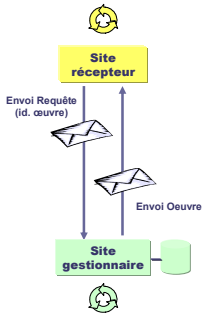
Réseau d'échange mondial sur les droits d'auteurs pour les oeuvres musicales
Description des oeuvres et des auteurs, utilisation des oeuvres, gestion des droits, ...

Motivations et Objectifs

- **Motivations**
  - Coopération entre sociétés de gestion des droits d'auteurs
    - *Echange de documentation sur les œuvres*
    - *Traitement des droits d'auteurs*
  - Automatisation des procédures
    - *Diminuer (éliminer) les échanges "physiques"*
    - *Accélérer le traitement des droits d'auteurs*
    - *Arbitrer les conflits*
- **Objectif**
  - Fournir un système d'échange d'information sur les œuvres
    - *Commun à l'ensemble des partenaires*
    - *Qui respecte l'indépendance et l'autonomie des partenaires (choix de fournisseur, procédures internes, exploitation, etc.)*

Spécifications fonctionnelles

- **Application de consultation des œuvres**
  - Serveur Web (HTTP, SOAP, XML)
- **Application d'échange des œuvres**
  - Echanges interactifs
    - *Mode **Push** : œuvre demandée par un utilisateur extérieur*
    - *Mode **Pull** : œuvre envoyée par un utilisateur local à des sites distants*
  - Echanges batch
    - *Envoi de mises à jour sur un ensemble d'œuvres*
- **Futurs usages**
  - *Collection et traitement des droits d'auteurs*
  - *Liaison automatisée avec les procédures internes*
  - *Croissance de la charge → dimensionnement*



Cahier des charges

- **Couplage faible**
  - Couverture à grande échelle (planète) sur Internet
  - Organisations et modes de fonctionnement indépendants
  - Solution non intrusive (respecte l'autonomie des participants)
- **Solution scalable et configurable**
  - Gros volumes de données et communications multi - points
  - Échanges asymétriques (ex. US goulot d'étranglement)
- **Solution évolutive**
  - Croissance rapide de la communauté d'utilisateurs
  - Nouveaux usages
- **Mise en oeuvre de la solution**
  - Maîtrise des coûts -→ solution « logiciel libre »

Dimensionnement

- **La description des œuvres**
  - >15 millions aujourd'hui,
  - En augmentation rapide : musique électronique, arrivée de nouveaux membres
  - Documentation sur une œuvre : de 5 KB à 10 KB
- **Les échanges entre sociétés**
  - Interactifs : une (ou plusieurs) œuvre(s)
  - Batch (mise à jour) : un grand nombre d'œuvre simultanément
    - → *n MB*
  - Charge interactive (exemple : Sacem)
    - → *20 utilisateurs concurrents (mais pics de charge)*
  - Echanges asymétriques
    - *60% des œuvres US utilisées en France*
    - *90% des œuvres US utilisées en Allemagne*
    - *Situation très hétérogène à l'intérieur de l'Europe*

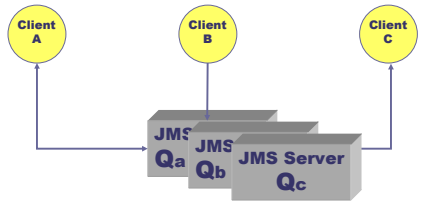
Choix d'architecture

- **Architecture centralisée**
  - Serveur JMS unique
- **Architecture asymétrique**
  - Un serveur par site/organisation
  - Serveurs indépendants (pas de communication entre eux)
  - Fonctionnement privilégiant les clients « locaux »
- **Architecture distribuée**
  - Un serveur par site/organisation (par groupe de sites)
  - Serveurs interconnectés
- **Architectures hybrides**
  - Fondées sur un placement optimal des *Queues* et *Topics*

Critères d'évaluation

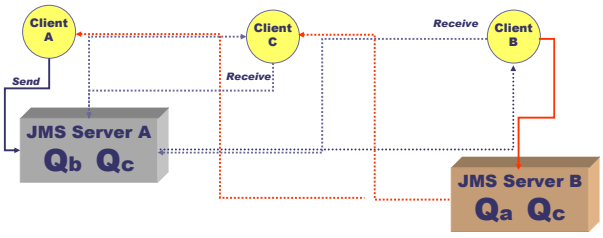
- **Autonomie**
  - Préserver l'autonomie et la disponibilité des SI locaux malgré les pannes
- **Fiabilité**
  - Garantie de délivrance des messages
- **Performance**
  - Nb. et volume des messages,
  - Dimensionnement des ressources systèmes
- **Scalabilité**
  - Capacité de croissance des charges dans les évolutions du système
- **Exploitation**
  - Déploiement et administration
  - Monitoring et reporting
- **Flexibilité - évolutivité**
  - Évolution du nombre d'organisations
  - Évolution des usages
- **Coût de la solution**
  - Coûts de développement/maintenance
  - Coûts de déploiement (hardw; & softw.)
  - Coûts d'exploitation
- **Autres. . .**

Architecture centralisée



- ➔ Administration/Exploitation simple et efficace
- ✗ Faible disponibilité : panne du serveur, panne réseau vers le serveur
- ✗ Faible scalabilité : toutes ressources concentrées sur le même serveur
  - Réponse possible : architecture en grappe (cluster) ➔ répartition de charge mais « clustering » ne signifie pas ici « haute disponibilité »

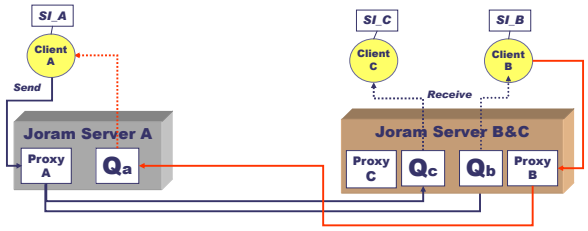
Architecture asymétrique



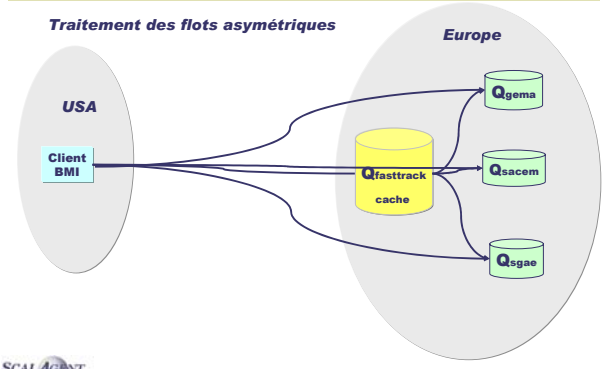
- ➔ Autonomie : les clients locaux sont privilégiés
- ➔ Administration: systèmes JNDI locaux (pas de connexion entre serveurs)
- ✗ Performance : 2 échanges à grande distance pour une lecture
- ✗ Scalabilité : la complexité de la gestion des queues croit comme le carré du nombre de sites

Architecture distribuée

- ➔ Haute disponibilité des clients et scalabilité
- ➔ Un serveur peut être partagé par plusieurs organisations
- ➔ Solution reconfigurable
- ✗ Administration plus complexe



Evolution : architecture hybride



Synthèse

	Architecture centralisée	Architecture asymétrique	Architecture distribuée
Fiabilité	OK	OK	OK
Disponibilité	Poor	High for sender Poor for receiver	High
Administration	Easy	Medium	Complex
Scalabilité	Poor	Poor	High
Flexibilité	Easy	Complex	Easy
Coût	Low	Medium	Medium

Plan du cours

I. Caractérisation des systèmes asynchrones
II. Modèles pour la programmation asynchrone
III. Middleware asynchrone (MOM)
IV. JMS : un exemple de programmation asynchrone
V. JORAM : un exemple de middleware asynchrone
VI. Etude de cas
VII. Conclusion : état des lieux et perspectives

Systèmes asynchrones : synthèse

- **Modèle de programmation**
  - Le modèle point à point est stable
  - Nombreux travaux pour faire évoluer le modèle multi-points (publish/subscribe)
  - JMS est le seul effort de normalisation au niveau API
- **Middleware**
  - Pas d'architecture de référence
  - Interopérabilité à travers des passerelles ad hoc (pas d'équivalent du protocole IIOP de Corba)
- **Paramètres d'efficacité**
  - L'architecture distribuée évite les goulots d'étranglement et permet le passage à l'échelle
    - *répartition de charge et réduction de la bande passante*
  - Les réceptacles de messages (queues et topics) doivent être proches des consommateurs
  - Configuration des paramètres de QoS pour éviter les coûts inutiles
    - *persistance, sécurité*
- **Fiabilité et disponibilité**
  - Fonction "*store and forward*"
  - Garantie de délivrance des messages

Systèmes à grande échelle

- **Volumétrie**
  - Nb de sites (émetteurs, consommateurs, serveurs)
  - Volume et taille des messages échangés
    - *Congestion du réseau,*
    - *Capacité de stockage des serveurs (proxy, queues)*
    - *Impact du mode déconnecté : stockage et flots de données à la reprise*
- **Réduction du volume des échanges (Publish/Subscribe)**
  - Utilisation de IP Multicast pour les réseaux locaux
  - Regroupement des messages dans les réseaux à grande distance
    - *par l'administrateur : utilisation de « topics clustérisés »*
    - *directement par le middleware : algorithmes de multiplexage/routage*
  - Elimination des échanges inutiles
    - *Rapprocher les filtres des sources d'information*
    - *Purge des messages ayant perdu leur intérêt pour le consommateur*
      - Paramètre « durée de vie » (positionné par l'émetteur)
      - Critères sémantiques fixés par le consommateur

Systèmes asynchrones : perspectives

- **Systèmes à grande échelle (Internet)**
  - Performance : latence, congestion
  - Montée en charge (scalabilité)
  - Flexibilité : adaptation à des configurations variées
  - Disponibilité et fiabilité : gestion des pannes temporaires
  - Sécurité
- **Publish-Subscribe par le contenu**
  - Gestion des abonnements et des événements
  - Conséquences sur l'architecture du système
  - Passage à l'échelle
- **Exemple : GRYPHON**
  - Projet de recherche (IBM T.J. Watson Center) sur ces thèmes
  - http://www.research.ibm.com/gryphon/research.html

Publish / Subscribe « étendu »

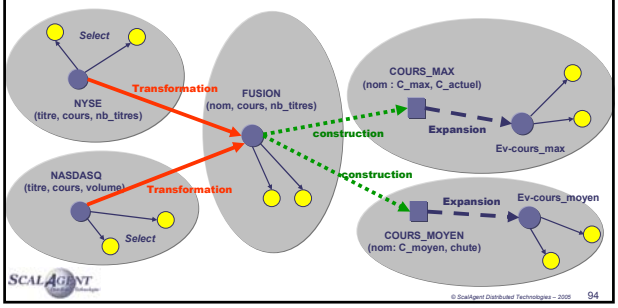
- **Objectif**
  - Rendre l'association entre abonnement et publication plus **dynamique**
  - État des lieux
    - *« topic » hiérarchisés*
    - *Sélecteur sur la partie < en-tête + propriétés> du message*
  - Besoins
    - *Sélectionner des messages en fonction du contenu (content-based PS, data-centric PS)*
    - *créer des événements sémantiquement plus riches par combinaison d'événements élémentaires*
    - *Interpréter des séquences d'événements*
- **Problèmes à résoudre**
  - Gestion de données : expression et résolution de requêtes complexes
  - Mise en œuvre dans un environnement distribué (à grande échelle)

Publish / Subscribe « par le contenu »

- **Expression du filtrage**
  - Chaînes de caractères : requêtes SQL sur des tuples < nom_propriété,valeur >
    - *Extension du mécanisme de filtrage de JMS*
  - Objets « modèles » : l'abonnement fournit un objet de référence pour la sélection
    - *inspiré du modèle de JavaSpaces*
  - Code exécutable : l'abonnement fournit un programme de filtrage
- **Mise en œuvre du filtrage**
  - Diffusion des événements et filtrage local
    - *Utilisation possible de IP Multicast sur réseaux locaux*
  - Propagation des filtres dans le réseau de serveurs

Evénements sémantiques

- **Enrichir l'espace des événements: événements composés**
- **Utiliser les propriétés des événements pour optimiser la gestion des flots de données**



Bibliographie (1/2)

- **ObjectWeb JORAM**
  - http://joram.objectweb.org
  - http://www.scalagent.com
  - Référence article
- **IEEE Distributed Systems OnLine,** http://dsonline.computer.org/middleware/index.htm
  - Message-oriented middleware
  - Event-based middleware
- **Publish/Subscribe**
  - Eugster et al. « The Many Faces of Publish/Subscribe » ACM Computing Surveys, 35(2), 2003
  - Gryphon (2003) : The Gryphon System, http://www.research.ibm.com/gryphon
- **Les agents distribués**
  - http://dsonline.computer.org/agents/index.htm
  - http://www.agentcities.org

Bibiliographie (2/2)

- **IBM WebSphere MQ**
  - http://www.ibm.com/software/integration/mqfamily
- **Microsoft Message Queue Server (MSMQ)**
  - http://www.microsoft.com/windows2000/technologies/communications/msmq
- **BEA MessageQ**
  - http://www.bea.com/content/products/more/messageq
- **TIBCO Rendezvous**
  - http://www.tibco.com/solutions/products/active_enterprise/rv
- **FioranoMQ 5**
  - http://www.fiorano.com/products/fmq/overview.htm
- **Softwired iBus//MessageBus**
  - http://www.softwired-inc.com/products/products.html
- **Sun Java Message Service (JMS)**
  - http://java.sun.com/products/jms
- **Progress Sonic MQ**
  - http://www.sonicsoftware.com/products/enterprise_messaging