Architecture des Applications IoT

OLIVIER FLAUZAC

le cnam
Grand Est

Architecture globale des applications

Conception d’une application

Intégration de bout en bout des éléments :

- Depuis l’objet connecté
- En passant par le stockage et le traitement
- Jusqu’au site web / application de visualisation

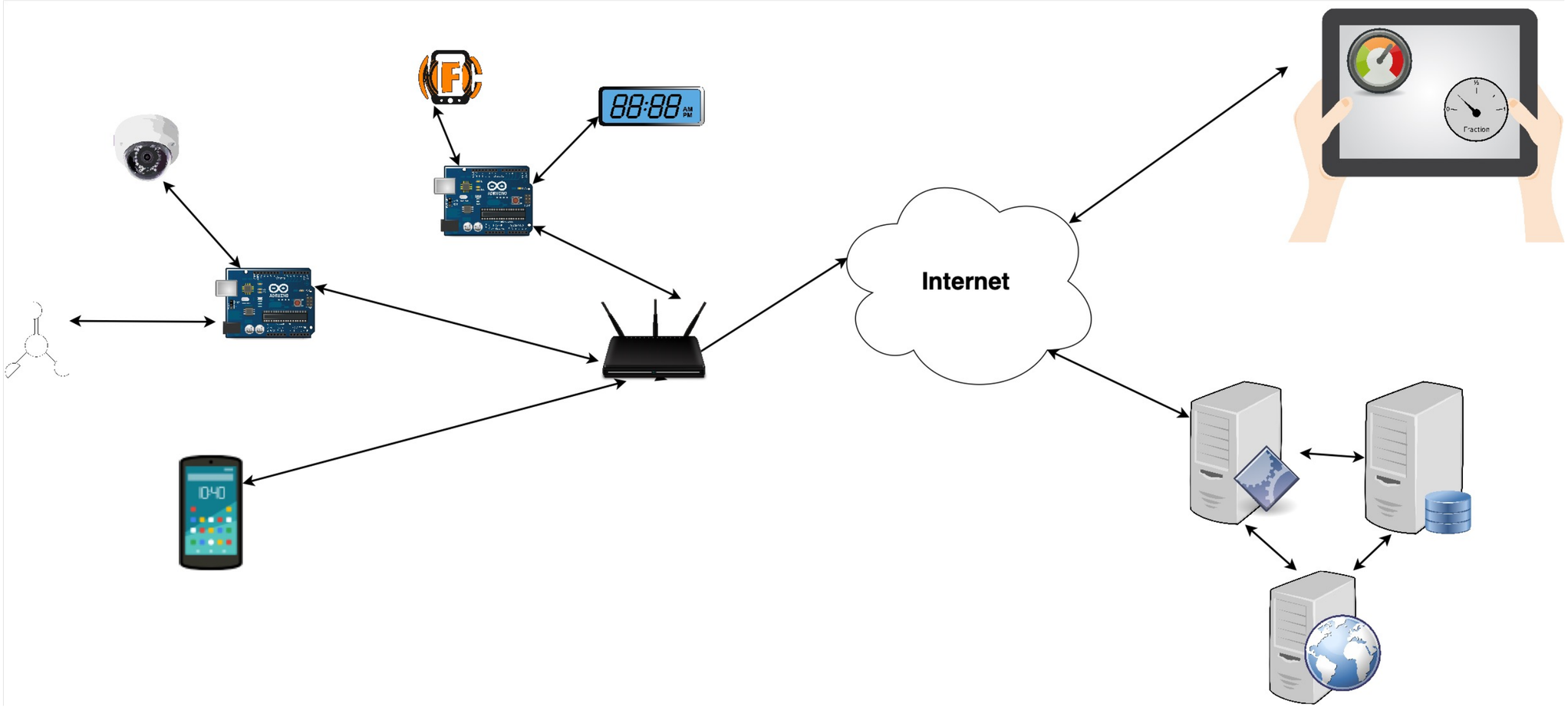
Intégration

- Des protocole IP et non IP
- Des capacités des différents éléments

Respect des contraintes

- Liés aux objets
- Liés à l’environnement

Schéma d’une application



Échange de données

Modèles de transmission

Définition des acteurs de la communication

Définition des initiatives de communication

Définition des flux de données

Impact sur
- Le développement
- L’architecture

Dirigé par
- Les capacités de objets
- L’écosystème de l’application

Communication directe

Échanges directs entre les acteurs :

- Source de la donnée
- Destination de la donnée

Permet la communication point-à-point

- Unidirectionnelle
- Bidirectionnelle

Nécessite :

- L’identification des acteurs
- Le routage
- La mise en place de ressources systèmes

Communication indirecte

Communication via un troisième acteur : proxy de communication

- Assure la gestion de l'acheminement

Indépendance entre la source et la destination des données

- Pas de connaissance de la destination par la source
- Pas de connaissance de la source par la destination

Bénéfices

- Sécurité
- Montée en charge
- Dynamicité
- Tolérance aux fautes

Multiplicité des sources et des destinations

- Plusieurs sources de données possibles
- Plusieurs destinations possibles

Communication indirecte par file

Mise en place d'un serveur de file d'attente entre la source et la destination

Schéma général

- La source dépose une donnée dans une file
- La destination lit les données dans la file
- Extensible à :
  - Plusieurs sources
  - Plusieurs destinations

Possibilité de Gérer des files spécifiques par source / destination

Propriétés

Propriétés des files

- Nommées / anonymes
- Persistantes
- Authentifiées
- Cryptées

Propriétés des messages

- Lecture unique / multiple
- Durée de vie
- Cryptage du contenu
- Anonymes / signés

Communication indirecte en flux

Mise en place d'un gestionnaire de flux

Similaire à une solution de *chat*

Connexion de l'ensemble des acteurs au gestionnaire de flux

Schéma général

- Les acteurs écrivent des données dans le flux
- Les acteurs lisent des données dans le flux

Propriétés

Propriétés des canaux

- Publics / privés
- Avec / sans historique
- Ouverts en lecture / écriture
- Clairs / cryptés

Propriété des messages

- Horodatage
- Publics / privés

Consommation

Le modèle producteur consommateur

Modèle le plus simple

Schéma général

1. Production de la donnée sur la source (*producteur*)
2. Émission de la donnée vers la destination
3. Récupération de la donnée par la destination (*consommateur*)
4. Exploitation de la donnée par la destination

Pas de réponse

Application du modèle producteur / consommateur

Communication directe

- Emission d'un message en TCP ou UDP
- Modèle 1/1 (1 producteur / 1 consommateur)
- Couplage fort

Communication indirecte par file de messages

- Exploitation d'une file
- Consommation à l'initiative du consommateur : *polling*
- Modèles 1/1 , 1/n , n/1 , n/n
- Couplage faible

Communication indirecte en flux

- Exploitation d'un gestionnaire de flux
- Consommation à l'initiative du consommateur : écoute sur le flux
- Modèles 1/1 , 1/n , n/1 , n/n
- Couplage faible

Le modèle *publish / subscribe*

Association d'une propriété aux messages

Modèle basé sur un système d'abonnement

Schéma général

1. Définition des propriétés des messages
2. Inscription des abonnés aux propriétés choisies
3. Emission des messages *taggés* par leur propriété par les publiants
4. Réception automatique des messages par les abonnés

Application du modèle *publish / subscribe*

Modèle uniquement adapté à la communication indirecte

Communication indirecte par file de messages

- Association file / propriété (*topic)*
- Consommation automatique: *callback*
- Modèles 1/1 , 1/n , n/1 , n/n
- Couplage faible

Communication indirecte en flux

- Association canal / propriété *(chatroom)*
- Consommation automatique: *callback*
- Modèles 1/1 , 1/n , n/1 , n/n
- Couplage faible

Exécutions

Message simple

Application simple des modèles d’échange de données

Schéma

1. Production de la donnée sur la source (*producteur*)
2. Émission de la donnée vers la destination
3. Récupération de la donnée par la destination (*consommateur*)
4. Exploitation de la donnée par la destination

Pas de réponse requise

Client / Serveur

Modèle désignant deux acteurs

- Le client
- Le serveur

Schéma basé sur 2 messages

- La requête
- La réponse

Schéma général

- Le client émet la requête
- Le serveur reçoit la requête
  - traite la requête
- Le serveur émet la réponse
- Le client reçoit la réponse
  - consomme les données

Client / serveur

Exemple de protocole client / serveur

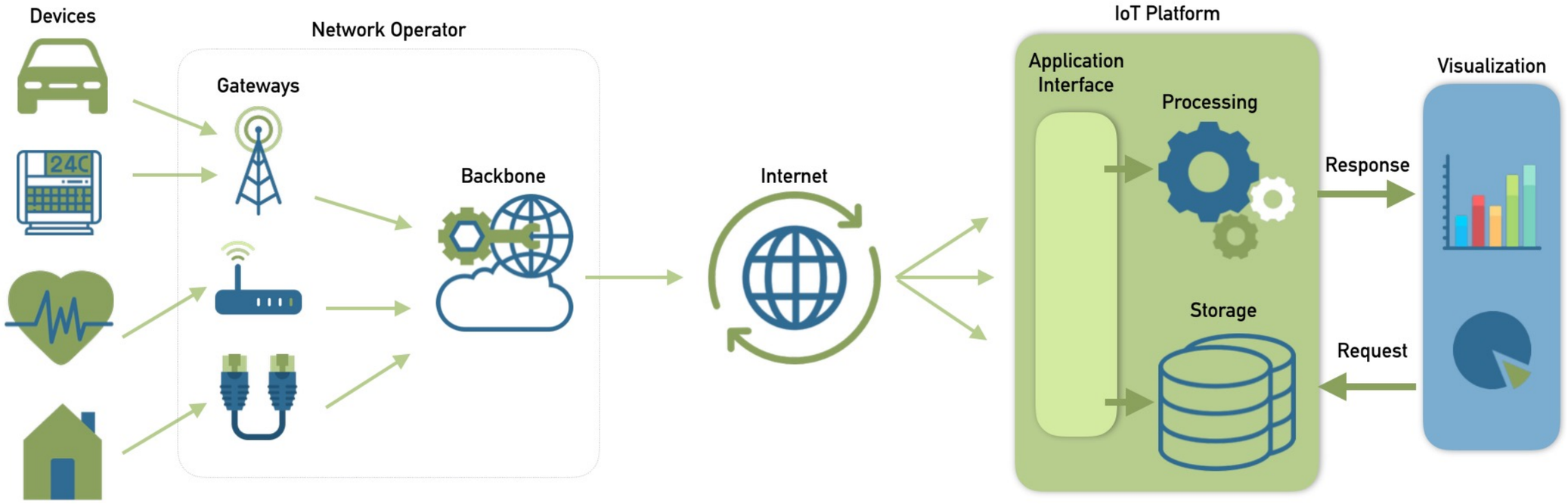
- HTTP, DNS, POP, SMTP …

Nécessite une attente du client

Toujours une réponse, même en cas d’erreur

Mise en œuvre dans l’Internet des objets

Une application IoT



Contraintes des objets

Contraintes liées au calcul

- Objets connectés de faible capacité
  - Peu de CPU
  - Peu de mémoire
- Programme de petite taille
- Pas de traitement local de l'information

Contraintes liées à l'énergie

- Communication limitée
  - Faible distance de communication
  - Protocoles les plus légers possibles

Protocoles et applications

Nécessité de concevoir des applications

- Fortement dynamiques
- Minimisant la configuration des objets
- Fortement disponibles
- Intégrables : communication M2M (*machine to machine*)

Nécessité de disposer de protocoles

- Les plus légers
- Les plus efficaces

Protocole de files d’attentes

MQTT

Messaging Queuing Telemetry Transport

Normalisé en 1999

Exploité à l'origine dans l'industrie pétrolière

Protocole de type *publish / subscribe*

- Création des topics à la publication

Exploitation d'un serveur MQTT (*broker)*

Solution intégrée de gestion de QoS

- QoS élevée = latence et bande passante élevée

Protocole

Transport TCP

Protocole léger

- header de 2 octets
- Peu de messages

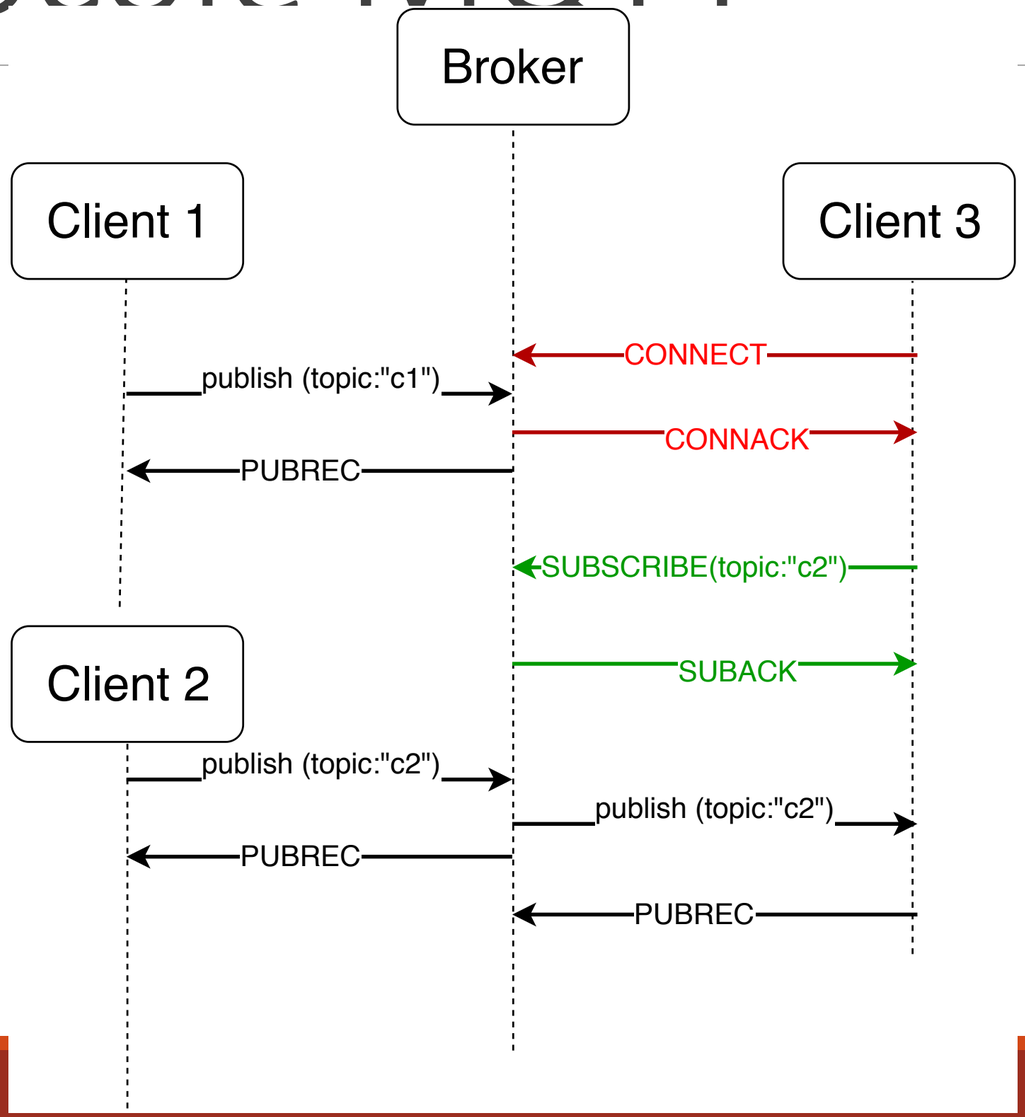
Contrôle des échanges et de la connexion

- Mise en place de messages pour les subscriber en cas de déconnexion

Sécurisable

- Transport SSL/TLS
- Intégration d’un mot de passe

Le protocole MQTT



Qualité de service

QoS0 (Unacknowledged Service)

- Séquence PUBLISH vers le broker sans réémission
- Relais unique vers les abonnés
- Pas d'ACK, pas de sauvegarde du message

QoS1 (Acknowledged Service)

- Mise en place d'une séquence PUBLISH / PUBACK
  - Publication / acquittement de publication
- Risque de duplication de messages

Qualité de service

QoS2 (Assured Service)

- Message délivré avec 2 paires de paquets
  - PUBLISH / PUBREC : publication / réception de publication
  - PUBREL / PUBCOMP : finalisation de publication / fin de publication
- Message délivré juste une seule fois

Avantages du MQTT

Particulièrement léger

- Faible surcoût du protocole

Adapté à une faible bande passante

Peu énergivore

- En calcul
- En émission

MQTT & API client

Les serveurs

- Active MQ
- JoramMQ
- Mosquitto
- RabbitMQ

Les clients

- API clients spécifiques par langages
- Projet Eclipse Paho

Gestion des messages

Publication

1. Création d'un identifiant de publiant
2. Définition des options du client
3. Publication du message
   - Création du message
   - Définition de la QoS
   - Publication

Lecture des messages

1. Définition du *callback* de traitement
2. Inscription à la file
3. Lecture automatique et exécution du *callback*

Programmation d’une publication

ETAPE 1

```
String publisherId = UUID.randomUUID().toString();
IMqttClient publisher = new MqttClient("tcp://iot.eclipse.org:1883",publisherId);
```

ETAPE 2

```
MqttConnectOptions options = new MqttConnectOptions();
options.setAutomaticReconnect(true);
options.setCleanSession(true);
options.setConnectionTimeout(10);
publisher.connect(options);
```

ETAPE 3

```
byte[] payload = "data-data-data".getBytes();
MqttMessage m = new MqttMessage(payload);
m.setQos(0);
m.setRetained(true);
client.publish("example",m);
```

Programmation d’une lecture

ETAPE 1

```
…
@Override
public void messageArrived(String topic, MqttMessage message) throws Exception {
    System.out.println("message is : "+ message);
}
…
```

ETAPE 2

```
…
MqttClient client = new MqttClient("tcp://localhost:1883","clientid");
client.setCallback(this); MqttConnectOptions mqOptions=new MqttConnectOptions();
mqOptions.setCleanSession(true);
client.connect(mqOptions); //connecting to broker
client.subscribe("test/topic");
…
```

AMQP

Advanced Message Queuing Protocol

Protocole de messagerie open Source

Alternative aux solutions de MOM ou JMS

Exploitation d'un serveur AMQP (*broker)*

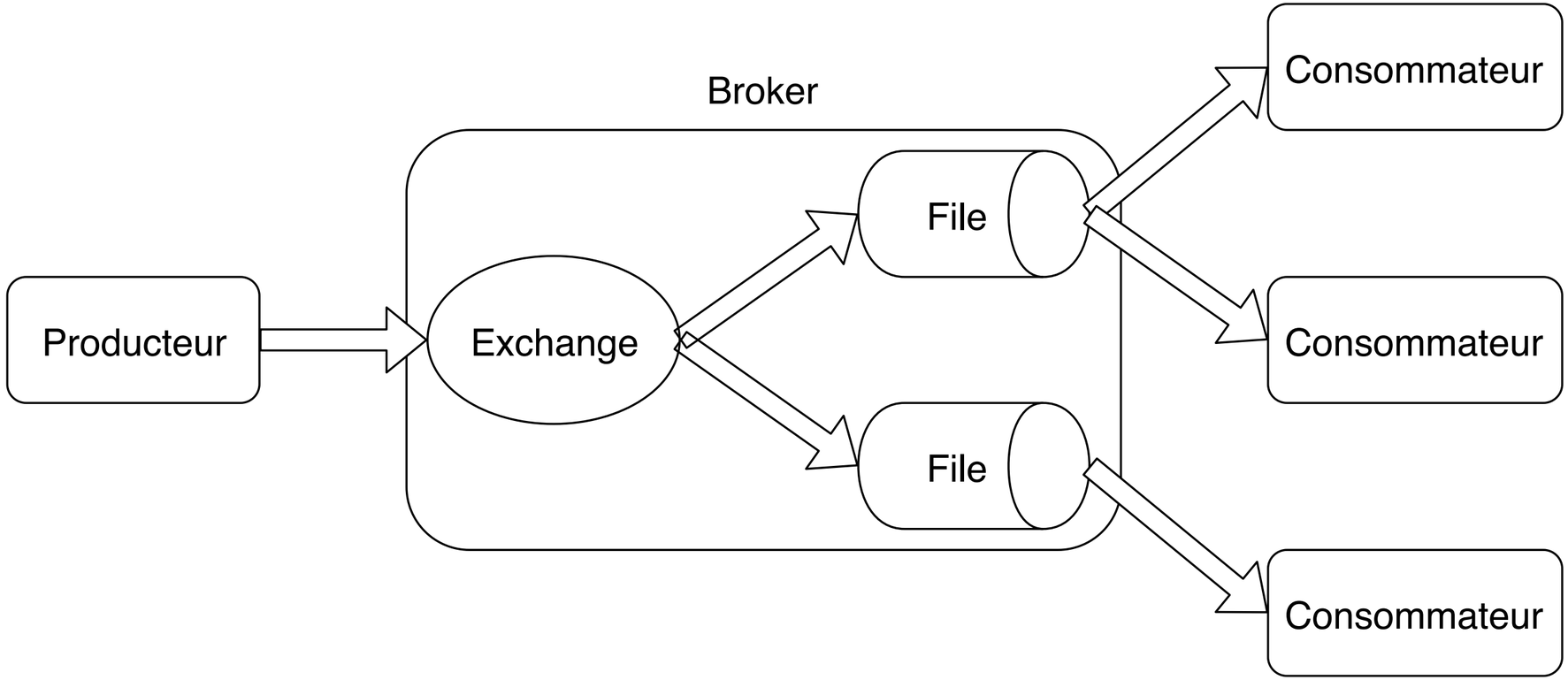
Intègre une solution de routage interne

- Routage d'un message reçu vers des files
- Beaucoup de stratégies de routage (contenu, *topic*, clés …)
- Possibilité de consommation multiple

Basé sur un modèle producteur / consommateur

Extension au modèle *publish / subscribe*

Fonctionnement du protocole AMQP



AMQP & API client

Les serveurs

- Active MQ
- QPid
- RabbitMQ

Les clients

- API client spécifiques par langages

Gestion des messages

Emission d’un message

1. Création d’une connexion
2. Création d’un canal
3. Association à une File
4. Emission

Lecture d’un message

1. Création d’une connexion
2. Création d’un canal
3. Association à une File
4. Consommation

Programmation d’une production

ETAPE 1

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost(QUEUE_ADDR);
Connection connection = factory.newConnection();
```

ETAPE 2

```
Channel channel = connection.createChannel();
```

ETAPE 3

```
channel.queueDeclare(QUEUE_NAME, false, false, false, null);
```

ETAPE 4

```
String message = "Hello World!";
channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
channel.close();
connection.close();
```

Programmation d'une consommation

ETAPE 1

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost(QUEUE_ADDR);
Connection connection = factory.newConnection();
```

ETAPE 2

```
Channel channel = connection.createChannel();
```

ETAPE 3

```
channel.queueDeclare(QUEUE_NAME, false, false, false, null);
```

ETAPE 4

```
QueueingConsumer consumer = new QueueingConsumer(channel);
channel.basicConsume(QUEUE_NAME, true, consumer);
QueueingConsumer.Delivery delivery = consumer.nextDelivery();
String message = new String(delivery.getBody());
System.out.println(" Réception de" + message + "'");
```

Programmation en flux

XMPP

Extensible Messaging and Presense Protocol

Protocole de messagerie instantanée (Jabber, Google talk)

Utilisé dans le cadre de la VOIP, transfert de fichiers …

Basé sur une architecture client / serveur

Exploite des échanges XML

Tolérant aux fautes par interconnexion de serveurs

Deux modes d'échanges

- Directs
- Publics

Principes

Distribution des messages par connexion serveur à serveur

Définition de passerelles

- Assure l'interopérabilité

Bus générique de données

- Messagerie instantanée : juste une application

Protocole simple

Eléments XMPP

JID : identification unique de chaque utilisateur

- [nœud ¨@¨]domaine_xmpp[¨/¨ ressource]
  - Ressource : gestion des connexions multiples
- olivier.flauzac@masterRT/salleTP

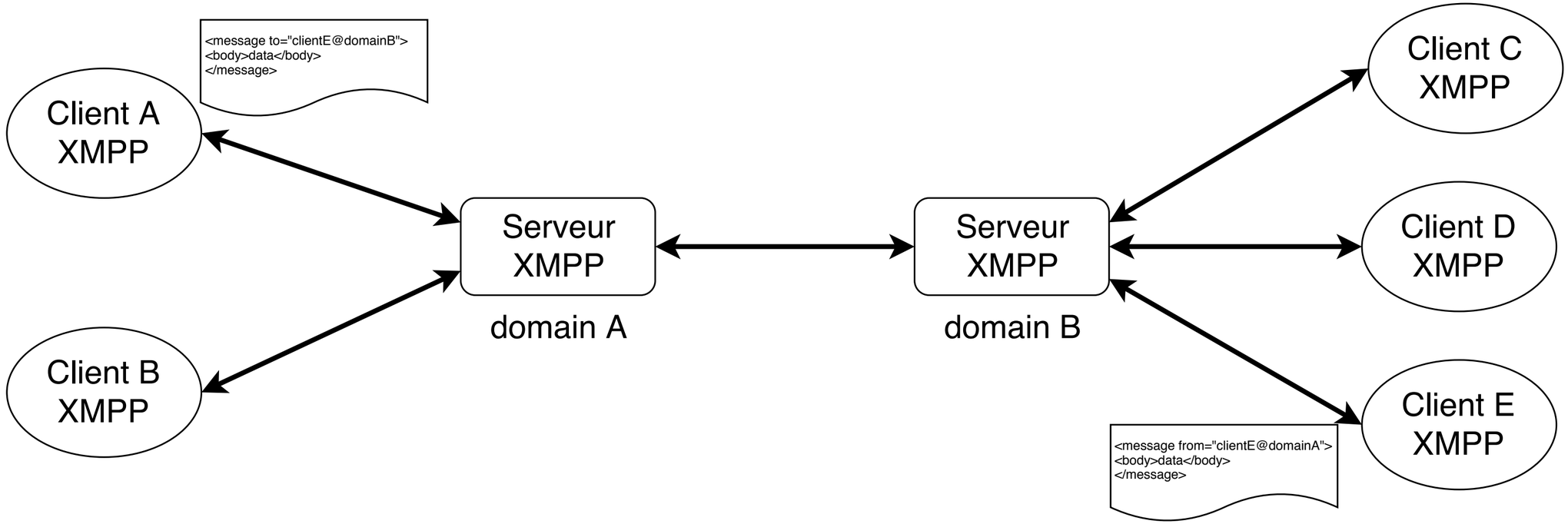
Session

- Session TCP mise en place par le client
- Permet le mode PUSH étendu (à la reconnexion)

Strophe

- Structure des messages (UTF-8)
- Message, presence, info/query

Fonctionnement du XMPP



XMPP & API Client

Serveurs XMPP

- Ejabberd
- OpenFire
- Prosody

API XMPP

- Smack en java
- Vysper
- Strophe en javascript

Clients XMPP

- Gajim
- Jitsi
- Psi

Solutions client / serveur

COAP

Constrained Application Protocol

Protocole client / serveur IETF

Inclus un système de découverte

Définition par le client de la qualité de service souhaitée

- Demande ou non d’ACK

Asynchrone

Protocole

Dérivé de HTTP / REST

Réduction aux verbes : GET, PUT, POST, DELETE

Exploitation de UDP et non de TCP

En-tête fixe de 4 octets

Exploitation d'un *status code*

Exploitation de deux couches

- Couche de messagerie (gestion de la non-fiabilité de UDP)
  - CON, ACK, NON, RST
- Couche d'interactions
  - GET, PUT, POST, DELETE

Message COAP

4 octets

- Version du protocole (1)
- Type de message : confirmable, non confirmable, Acknowledgement, Reset
- Token Length
- Code
- Message ID
- Token

Eléments de COAP

Fiabilité :

- Définition dans la requête de messages confirmable ou non-confirmable
- Assure la retransmission en cas de perte

Jeton : assure la liaison requête / réponse

Label : détection de doublons

Découverte de services

Decouverte

- Unicast : connaissance de l'adresse du serveur
- Multicast
  - Problème de fiabilité: comment s'assurer d'avoir atteint tous les serveurs

Emission d'une requête GET à l'URI `/.well-known/core`

COAP et API

Mise en place de serveur dans les objets

- Serveurs légers

Frameworks

- Californium en java
- Erbium en C
- Copper en Javascript