Université Bretagne Sud
UFR SSI

Réseaux et systèmes pour applications innovantes

OSGi – CoAP – MQTT

INF 4000

Master Informatique 2e année – Parcours SAIM

Yves MAHÉO

Réseaux et systèmes pour applications innovantes

Master 2 Informatique – Parcours SAIM
INF 4000

Septembre 2020

Yves Mahéo
Yves.Maheo@univ-ubs.fr

OSGi

OSGi

- Plate-forme OSGi
  - Canevas de déploiement et d'exécution de services Java
  - Plate-forme centralisée : une seule JVM
- Consortium OSGi
  - Fondé en 1999
  - Inclut de nombreux acteurs majeurs (IT, téléphonie, Eclipse, Apache,...)
- Marchés visés
  - Initialement : passerelles résidentielles (set top boxes,...)
  - Industrie automobile
  - Contrôle industriel
  - Téléphonie mobile
  - Élargissement aux applications généralistes

OSGi

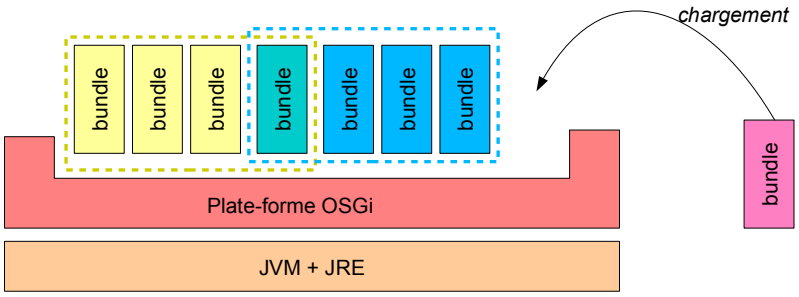
- Spécifications
  - Novembre 1999 : transfert du JSR008 (Open Service Gateway) à OSGi
  - 1.0, mai 2000 (189 pages)
  - 2.0, octobre 2001 (288 pages)
  - 3.0, mars 2003 (602 pages)
  - 4.0, octobre 2005 (1000 pages)
  - 5.0, juin 2012 (1400 pages)
- Principales implantations
  - Felix (Apache)
  - Eclipse-Equinox (IBM)
  - Knoplerfish
  - Concierge (limité à R3)

Motivations

- Objectifs généraux
  - Programmation orientée service en Java
  - Gestion du cycle de vie des applications
    - Mise à jour dynamique des applications
- Solutions techniques
  - Séparation interface/implantation
  - Conditionnement du code
  - Chargement/déchargement de classes
  - Résolution des dépendances de versions de code

Application OSGi

- Application = ensemble de bundles
  - Livrés dynamiquement
  - Éventuellement partagés par d'autres applications



Bundle OSGi

- Bundle
  - Unité fonctionnelle
    - offre des services
    - se lie à d'autres services
  - Unité de déploiement
    - sous forme d'archive JAR
- Dépendance entre bundles
  - Au niveau service
    - Un bundle contenant un client de service dépend du bundle contenant le (founisseur du) service requis
      - dépendances définies par programmation
  - Au niveau package
    - Résolution des dépendances de classes Java
      - dépendances déclarées par import/export de packages
      - résolues par la plate-forme OSGi

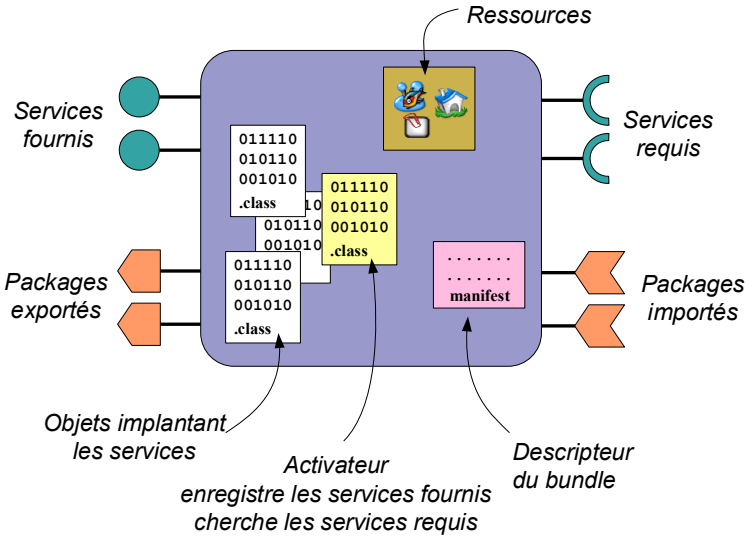
Service OSGi

- Interface de service
  - Interface de service = contrat = interface Java
    - Code présent dans un package Java
  - Descripteur meta-données (propriétes)
    - Implantée sous la forme de propriétés (table de hash associant String à Object)
    - permet le filtrage, l'évolution dynamique de contrat
  - Exemple :
    - interface org.osgi.PrinterService
    - propriétés : color=true, protocole=ftp,lpd, (dpi >=300 && dpi <=600)
  - Conditionnement
    - Il est recommandé de conditionner l'interface de service dans un bundle propre
      - une interface java est un binaire, il faut éviter de dupliquer le binaire dans les fournisseurs et les clients (incohérences, coût mémoire)
      - ce bundle ne fournit pas service, il contient juste l'interface

Service OSGi

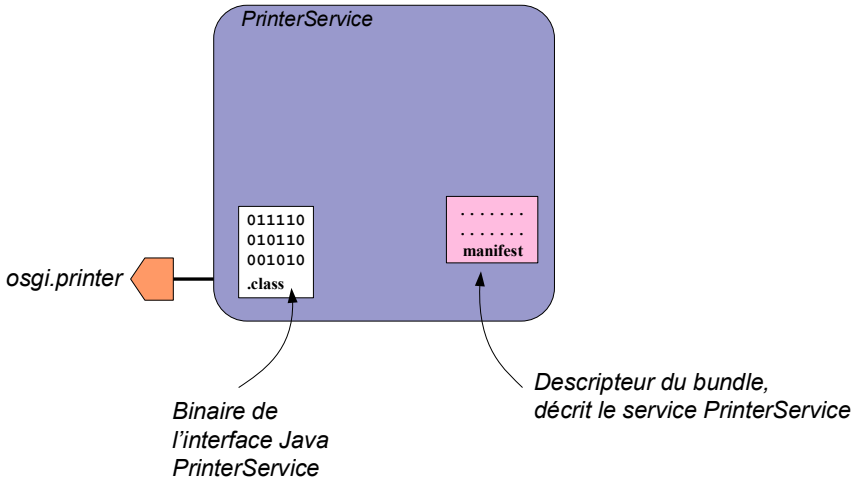
- Fournisseur de service
  - Souvent appelé « le service »
  - Implante le service
  - Instance d'une classe Java (définie dans un package Java)
  - La classe implémente une interface de service
- Client de service
  - Instance d'une classe Java (dans un package Java)
  - Se lie dynamiquement à (un fournisseur de) service
    - via la plate-forme
  - Manipule directement la référence Java à un objet implantant le service (fournisseur)

Struture d'un bundle



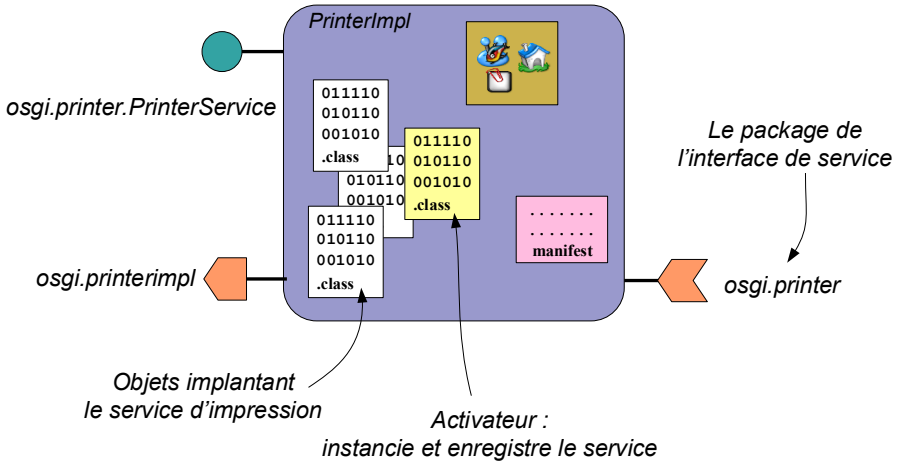
Exemple : interface d'un service d'impression

- Bundle conditionnant l'interface de service



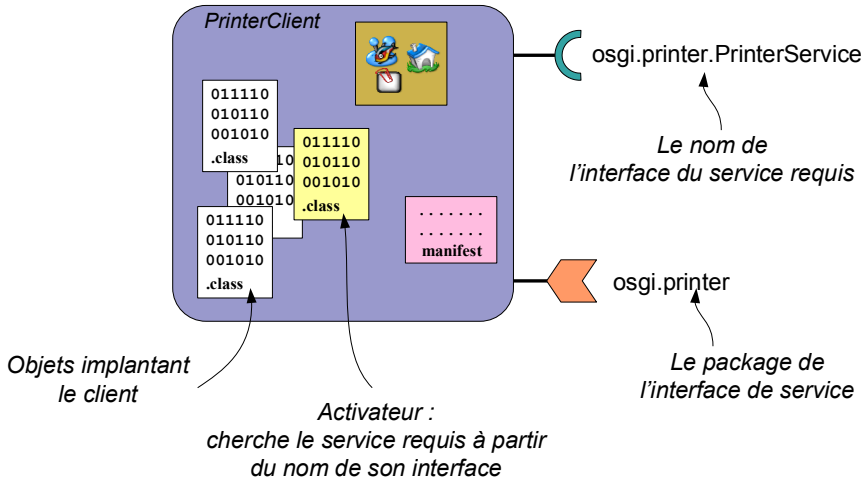
Exemple : fournisseur du service d'impression

- L'implantation du service

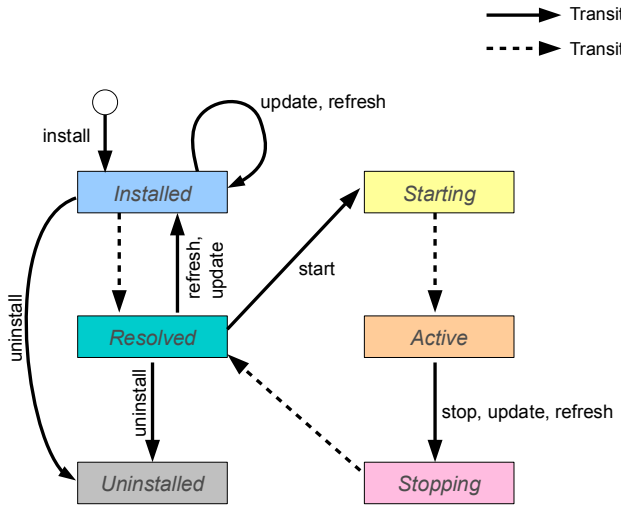


Exemple : client du service d'impression

- L'implantation du client



Cycle de vie



Développement OSGi

- Développement d'une application
  - Bundles interfaces
  - Bundles de services
    - bundle fournisseur
    - bundle client
    - bundle client et fournisseur
- Développement d'un bundle
  - Descripteur de déploiement
    - fichier manifest.mf
  - Fichiers binaires
    - classe activateur
    - classe(s) fournisseurs
    - classe(s) clientes

Activateur

- Définit une partie du cycle de vie du bundle
  - Bundle interface : normalement pas d'activateur
    - Le bundle ne contient que des interfaces et des exceptions
- Classe implémentant l'interface BundleActivator()
  - Offre deux méthodes
    - start()
    - stop()

Activateur

- Méthode start()
  - Invoquée par la plate-forme lors du passage à l'état starting
  - Rôle standard
    - instancier et enregistrer les services fournis (côté fournisseur)
    - interroger la plate-forme pour trouver les services requis et se lier à eux (côté client)
    - démarrer des threads
      - l'exécution de la méthode start elle-même doit être courte
- Méthode stop()
  - Invoquée par la plate-forme lors du passage à l'état stopping
  - Rôle standard
    - désenregistrer les services fournis (côté fournisseur)
    - relâcher les références vers le services utilisés (côté client)
    - arrêter des threads

Manifeste

- Liste de headers prédéfinis attendus par la plate-forme
  - Certains headers sont optionnels
- Exemples de headers
  - `Bundle-Name` : nom du bundle (pour lecteur humain)
  - `Import-Package` : listes des packages Java à importer (au chargement)
  - `Export-Package` : liste des packages Java exportés
  - `Bundle-Activator` : classe de l'activateur
  - `Bundle-ClassPath` : classpath (pour les sous-fichiers jar)
  - `Bundle-Update-Location` : URL pour obtenir une mise à jour
  - `Dynamic-Import-Package` : liste de package à importer à l'exécution
  - `Bundle-Version` : numéro de version du bundle (0.0.0.0 par défaut)

Exemple d'application

- Fourniture et utilisation d'un service d'impression
- Trois bundles
  - bundle interface : PrinterService
  - bundle fournisseur : PrinterImpl
    - Implante l'interface
  - bundle client : PrinterClient
- Packages
  - hp.services.printerservice
  - hp.prod.printerimpl
  - perso.printerclient

Bundle interface

- Descripteur de déploiement
  - Fichier META-INF/manifest.mf

```
Bundle-Name: PrinterService
Bundle-Description: Service d'impression
Bundle-Version: 1.0.0
Bundle-Vendor: Hepsonne
Bundle-ContactAdress: tech@hepsonne.com
Bundle-Copyright: Library GNU Public Licence
Export-Package:hp.services.printerservice
```


Bundle interface

- Interface Java

```
package hp.services.printerservice;

public interface PrinterService {
   public void print(String nomfic);
   public void cancel();
}
```

Bundle fournisseur

- Descripteur de déploiement
  - Fichier META-INF/manifest.mf

```
Bundle-Name: PrinterImpl
Bundle-Description: Implementation HP du service d'impression
Bundle-Version: 1.0.0
Bundle-Vendor: Hepsonne
Bundle-ContactAdress: tech@hepsonne.com
Bundle-Copyright: Library GNU Public Licence
Bundle-Activator: hp.prod.printerimpl.Activator
Import-Package: hp.services.printerservice, org.osgi.framework
```

package ou se trouve l'interface du service implanté

package de la plate-forme OSGi

Bundle fournisseur

- Classe du fournisseur
  - Classe Java codant l'implantation du service

```
package hp.prod.printerimpl;

public class PrinterImpl implements PrinterService {
   public void print(String nomfic){
      //imprime le fichier
      ...
   }

   public void cancel() {
      // Interrompt l'impression en cours
      ...
   }
}
```


Bundle fournisseur

- Classe de l'activateur
  - La méthode start
    - enregistre le service
    - donne les propriétes au service

```
package hp.prod.printerimpl;

public class Activator implements BundleActivator {
   private ServiceRegistration reg

   public void start(BundleContext ctx) throws BundleException {
     PrinterImpl pimpl = new PrinterImpl();
     Dictionary props = new Hashtable();
     props.put("papier", "A4");
     reg = ctx.registerService("PrinterService",
                               pimpl, props);
   }

   ...
```

Bundle fournisseur

- Classe de l'activateur (suite)
  - La méthode stop désenregisre le service

```
   ...

   public void stop(BundleContext ctx) throws BundleException {
      if (reg != null)
         reg.unregister();
   }
}
```


Bundle client

- Descripteur de déploiement
  - Fichier META-INF/manifest.mf

```
Bundle-Name: PrinterClient
Bundle-Description: Client du service d'impression
Bundle-Version: 1.0.0
Bundle-Vendor: Home
Bundle-ContactAdress: contact@home.org
Bundle-Copyright: Library GNU Public Licence
Bundle-Activator: perso.printerclient.Activator
Import-Package: hp.services.printerservice, org.osgi.framework
```

package ou se trouve l'interface du service requis

package de la plate-forme OSGi

Bundle client

- Classe de l'activateur
  - La méthode start
    - cherche à se lier à un service d'impression (le premier trouvé)
      - elle interroge la plate-forme selon le type du service et selon un filtre LDAP pour les propriétés
    - invoque le service puis relâche la référence au service

```
package perso.printerclient;

public class Activator implements BundleActivator {
    public void start(BundleContext ctx) throws BundleException {
      ServiceReference[] refs =
           ctx.getServiceRererences("PrinterService",
                                    "(papier=A*)");
      PrinterService pr = ctx.getServices(refs[0]);
      pr.print("monfichier.ps");
      ctx.ungetService(refs[0]);
   }
}
```

Gestion d'événements

- Il est possible d'attacher des gestionnaires d'événements (observateurs ou listeners) relatifs
  - aux bundles
  - aux services
  - à la plate-forme
- L'observateur peut être
  - Synchrone
    - l'observateur est exécuté avant que l'événement soit effectif
  - Asynchrone
    - l'observateur est appelé alors que l'événement se poursuit
    - tous les observateurs sont exécutés par un seul thread de la plate-forme

Mise en place d'un observateur

- Définir une classe d'observateur
  - La classe implémente l'interface de listener adéquate
- Définir dans cette classe la méthode adéquate de traitement de l'événement
- Dans l'activateur
  - Créer une instance de l'observateur
  - L'ajouter à la liste des listeners
- Tâche fastidieuse
  - Problématique lorsque l'on doit gérer un grand nombre de service
  - Tâche déléguée à un service approprié (e.g. ServiceTracker)

Exemple d'observateur de service

- Définition de l'observateur

```
package nsa.surveillance;

public class Observateur implements ServiceListener {
   public void serviceChanged(ServiceEvent evt) {
     ServiceReference ref = evt.getServiceReference();
     switch (evt.getType()) {
        case ServiceEvent.REGISTERED:
          affiche(ref+" enregistré par"+ref.getBundle().getLocation());
          break;
        case ServiceEvent.UNREGISTERING:
          affiche(ref+" est sur le point d'être désenregistré");
          // il faudrait ici relacher la reférence
          break;
        case ServiceEvent.MODIFIED:
          affiche(ref+" a été modifié");
          break;
     }
   }
```

Exemple d'observateur de service

- Création / suppression de l'observateur
  - Dans l'activateur

```
package nsa.surveillance;

public class Activator implements BundleActivator {
   private ServiceRegistration reg;
   private Observateur obs = null;

   public void start(BundleContext ctx) throws BundleException {
     Observateur obs  = new Observateur(context);
     ctx.addServiceListener(obs,
                  "objectClass="+PrinterService.class.getName());
   }

   public void stop(BundleContext ctx) throws BundleException {
     ctx.removeServiceListener(obs);
   }
}
```


Complexité de l'observation

- Limite du modèle de gestion manuelle de la dynamicité
  - La mise en place d'observateurs est fastidieuse et source d'erreurs
    - blocage possible de la plate-forme
  - Difficilement applicable pour un grand nombre de services
- Alternatives
  - ServiceTracker
    - Délégation du suivi d'un type de service à un service ServiceTracker
    - Simplification du la gestion des références au service
    - Ne gère pas le cycle de vie du client
  - Modèle à composants orienté service
    - Approche déclarative de la gestion de dépendances entres services
    - Automatise le cycle de vie des clients de services en fonction de la disponibilité des services requis
    - Service Component Runtime (OSGi R4), FROGi

Services standard

- Standard OSGi = core + services standard
- Spécification de services (optionnels)
  - Décrits dans le document «OSGi Compendium »
  - Services généraux et services spécifiques à un domaine (e.g. automobile)
  - Sous ensemble orienté serveur d'applications : « OSGi Enterprise Release »
- Exemples
  - Service Tracker (suivi de service)
  - Log Service (journalisation)
  - Event Admin (publish/subscribe)
  - Http Service (publication d'application web)
  - Wire Admin Service (gestion de producteurs/consommateurs type capteur)
  - ...

Conclusion sur OSGi

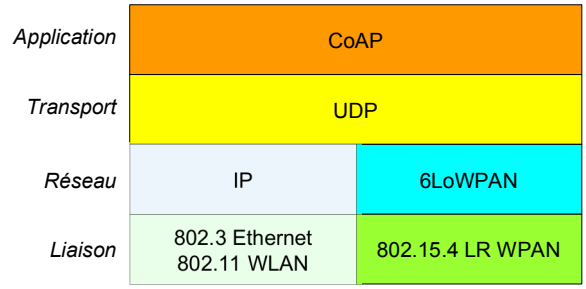
- OSGi : modèle de plate-forme à service locale en Java
- Avantages
  - Très performant
    - Un client possède une référence directe sur l'objet fournisseur
  - Support du monde industriel
  - Champ d'application assez vaste
- Inconvénients
  - Uniquement local
    - Mais utilisation possible de services distants (cf. Remote Services du Compendium)
  - Problème de passage à l'échelle de la gestion de la dynamicité
    - Problème encore ouvert

CoAP

Constrained Application Protocol (CoaP)

- Cible : Internet des objets (IoT), Web des objets (WoT)
- Protocole de la couche application
  - Implante le modèle REST d'HTTP (en le simplifiant)
  - Combinable facilement avec HTTP pour intégration dans le web
  - RFC 7252
- Vision M2M
  - Nouveaux formats de données
  - Remplacement de la sémantique orientée présentation
- Adapté pour être mis en œuvre sur des dispositifs légers (typiquement des capteurs)
  - peu de puissance de calcul
  - peu de mémoire

Pile protocolaire

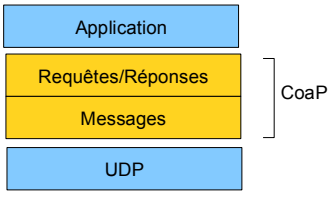


Caractéristiques générales

- Protocole web contraint pour le M2M
- Exploitation d'UDP
  - fiabilisation optionnelle
  - supporte unicast et multicast
- Découverte
- Échange de messages asynchrone
- Surcoût faible (entêtes courts, parsing aisé)
- Supporte les URI et Content-type
- Possibilité de proxy simple et de cache
- Mapping HTTP stateless
- Sécurité (Datagram Transport Layer Security)

Modèle abstrait

- Rôle client / serveur (similaire à HTTP)
  - Le client envoie une requête au serveur pour une action à effectuer sur une ressource (désignée par une URI)
    - Méthodes : GET, PUT, POST, DELETE
  - Le serveur renvoie une réponse avec éventuellement la représentation d'une ressource
- Échanges asynchrones sur UDP
  - Fiabilisation potentielle par back-off exponentiel
  - Port par défaut : 5683



Format de messages

- Messages entre deux *endpoints*

  - Longueur limitée à un paquet IP (pas de fragmentation)
    - par défaut : payload 1 ko
    - dans certains contextes, payload beaucoup plus petit

Format de messages

- Type de message
  - Confirmable (CON), Non Confirmable (NON), Acknowledgment (ACK), Reset (RST)
- Code : définit une requête ou une réponse
  - Entier donnant une classe et des détails, noté *c.dd*
  - Classes : vide, requête, réponse succès, réponse erreur client, réponse erreur serveur
- Message ID (2 octets)
  - Permet de détecter les duplicats et associer un ACK à un message
- Token (longueur variable)
  - Assimilable à un id de requête
  - Permet de corréler les requêtes et les réponses

Modèle du niveau message

- Message confirmable : CON
  - Retransmis jusqu'à réception d'un accusé de réception (ACK)
    - L'ACK porte le même Message ID que le message CON
  - Attente d'un timeout avant retransmission, avec backup exponentiel
  - Lorque l'endpoint recevant le message CON ne peut pas traiter le message (même pas pour générer une réponse d'erreur), il renvoie un message de reset (RST)
- Message non confirmable : NON
  - Utile si la fiabilité n'est pas requise (pas d'ACK renvoyé)
  - Retransmission possible (gestion de la duplication grâce au message ID))
  - L'endpoint renvoie RST s'il ne peut pas traiter le message
- Communication de groupe possible
  - Fondée sur l'utilisation du multicast IP

Contrôle de transmissions

- Ensemble de paramètres de l'implémentation
  - Valeurs par défaut

name	default value
ACK_TIMEOUT	2 seconds
ACK_RANDOM_FACTOR	1.5
MAX_RETRANSMIT	4
NSTART	1
DEFAULT_LEISURE	5 seconds
PROBING_RATE	1 byte/second

  - Paramètres dérivés

name	default value
MAX_TRANSMIT_SPAN	45 s
MAX_TRANSMIT_WAIT	93 s
MAX_LATENCY	100 s
PROCESSING_DELAY	2 s
MAX_RTT	202 s
EXCHANGE_LIFETIME	247 s
NON_LIFETIME	145 s

Modèle du niveau requête/réponse

- Similaire au modèle HTTP
  - Mais requêtes et réponses échangées de manière asynchrone (pas de session)
- Requêtes
  - Méthodes GET, PUT POST, DELETE
- Réponses
  - Réponse embarquée dans l' ACK (*piggybacked response*)
    - cas le plus courant
  - Réponse séparée
    - si la réponse est longue à calculer, pour éviter les retransmissions de requêtes

Codes

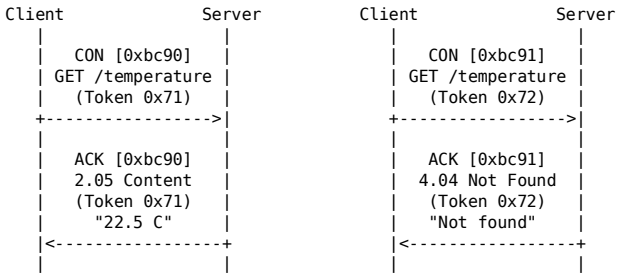
- Requêtes
  - Méthodes `0.01 GET, 0.02 POST, 0.03 PUT, 0.04 DELETE`
- Réponses
  - Succès
    - `2.01 Created, 2.02 Deleted, 2.03 Valid, 2.04 Changed, 2.05 Content`
  - Erreur client
    - `4.00 Bad Request, 4.01 Unauthorized, 4.02 Bad Option, 4.03 Forbidden, 4.04 Not Found, 4.05 Method Not Allowed, 4.06 Not Acceptable, 4.12 Precondition Failed, 4.13 Request Entity Too Large, 4.15 Unsupported Content-Format`
  - Erreur serveur
    - `5.00 Internal Server Error, 5.01 Not Implemented, 5.02 Bad Gateway, 5.03 Service Unavailable, 5.04 Gateway Timeout, 5.05 Proxying Not Supported`

Options

- Infos-méta-données sur les requêtes/réponses
  - Certaines options sont répétables
- Exemples d'options :
  - `Uri-Host, Uri-Port, Uri-Path, Uri-Query`
    - ressource cible d'une requête (très souvent nécessaire)
  - `Proxy-Uri, Proxy-Scheme`
    - destinées au forward-proxy
  - `Content-Format`
    - Indique le format du payload (utilisation largement recommandée)
  - `Accept`
    - Indique les formats de payload acceptables par le client
  - `If-Match`
    - Pour rendre une requête conditionnelle à l'existence d'une ressource
  - `ETag`
    - Identifiant différenciant des valeurs successives d'une même ressource

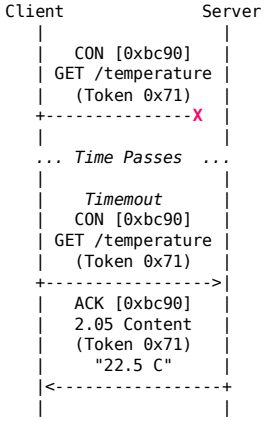
Exemples

- Requêtes GET avec payloads adossés aux réponses



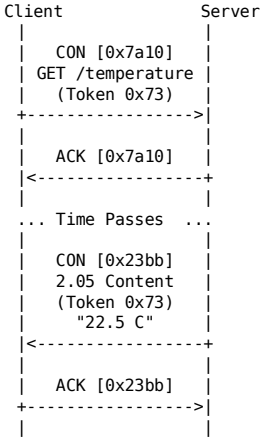
Exemples

- Perte de message



Exemples

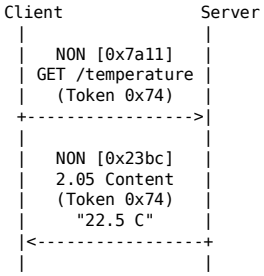
- Requête GET avec une réponse séparée
  - Le serveur n'a pas immédiatement la température à sa disposition



Exemples

- Requête GET (et réponse) dans messages non-confimables

  (une réponse CON aurait pu aussi être utilisée)



Caching

- Un endpoint (client) CoAP peut stocker en cache des réponses
  - But : réutiliser une réponse déjà présente pour satisfaire une requête
- Réutilisation d'une réponse suffisamment fraîche
  - La réponse en cache porte une option `Max-Age` donnant l'âge de péremption (60 s par défaut)
  - Inutile de générer une requête réseau
- Validation de ressource
  - Quand un endpoint possède en cache une ou plusieurs valeurs d'une réponse pas suffisamment fraîche, il peut générer une requête GET avec l'option ETag
  - Le serveur répond avec une réponse simple `2.03 (Valid)` avec l'option ETag pour désigner la valeur valide
  - Inutile de renvoyer une réponse complète

Proxying

- Proxy : CoAP endpoint qui peut être invoqué par des clients afin qu'il émette des requêtes à leur place
  - Utile quand la requête ne peut pas être émise par le client (e.g, pour des contraintes de routage IP)
  - Utile pour servir la requête depuis un cache
- Rôles
  - Proxy désigné explicitement par le client : forward-proxy
  - Proxy positionné en lieu et place d'un serveur : reverse-proxy
  - Proxy transformant une requête CoAP en une requête vers un autre protocole (typiquement HTTP) et/ou vice-versa : cross-proxy

Comportement d'un proxy

- Requête à un forward proxy (par opposition à une requête à un serveur)
  - Utilisation de l'option Proxy-Uri à la place de [Uri-Host, Uri-Port, Uri-Path, Uri-Query]
    - Alternative : option Proxy-Scheme + [Uri-Host, Uri-Port, Uri-Path, Uri-Query]
  - Le proxy transforme les options pour transmettre une requête normale
  - Le proxy peut aussi exploiter son cache
- Requête reçue par un reverse-proxy
  - Transparent pour le client (par de Proxy-Uri ou Proxy-Scheme)
  - Le proxy détermine la destination de la requête en fonction
    - de l'URI portée par la requête
    - de sa propre configuration (libre)

Découverte de ressources

- Pour faciliter la découverte de ressources, un serveur CoAP doit supporter la spécification CoRE Link Format
  - Voir RFC 6690 (et RFC 5988)
  - Le serveur fournit une description de ses ressources en réponse à une requête GET (port 5683) portant l'URI `/.well-known/core`
  - Le format de la réponse est standardisé, destiné à être interprétable par une machine
    - attributs `rt` (resource type), `if` (Interface [~sémantique]), `sz` (maximum size estimate), `ct` (content type)
  - Possibilité de faire des requêtes filtrantes
- Exemples

```
REQ: GET /.well-known/core

RES: 2.05 Content
     </sensors>;ct=40
```

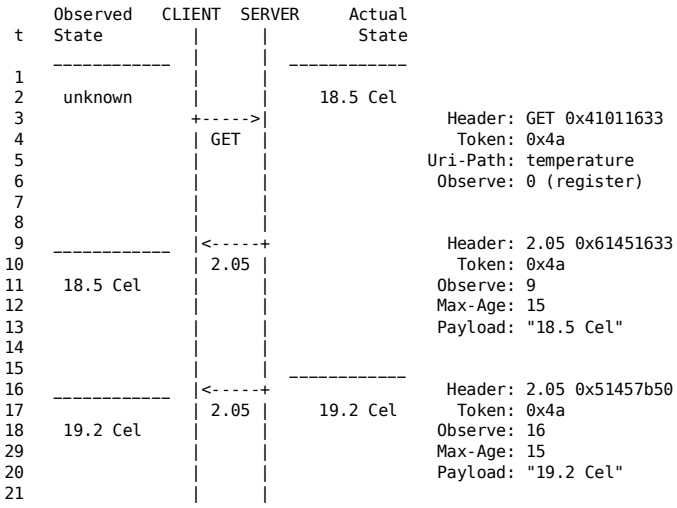
application/link-format

```
REQ: GET /sensors

RES: 2.05 Content
     </sensors/temp>;rt="temperature-c";if="sensor",
     </sensors/light>;rt="light-lux";if="sensor"
```

Extension pour l'observation

- Extension de CoAP pour qu'un client puisse observer les états successifs d'une ressource
  - RFC 7641 *Observing Resources in the Constrained Application Protocol*
- Patron « observer »
  - Enregistrement auprès du serveur de son intérêt pour une ressource via une requête GET étendue
    - Option `Observe`, portant la valeur `0 (register)` ou `1 (unregister)`
  - Réponse classique + ajout par le serveur d'un observateur sur la ressource
  - Notifications spontanées par le serveur des changements d'état de la ressource
    - Réponses `2.05 Content` avec option `Observe`
      - L'option porte un numéro croissant (équivaut à une date [logique ou non])
      - Messages CON ou NON

Exemple d'observation



Block-Wise Transfer

- Extension de CoAP pour transmettre des longs contenus sur plusieurs cycles requête/réponse
  - Transfert d'un « body » dans plusieurs « payloads »
  - Contourne la contrainte de payloads limités à un seul paquet IP
  - RFC 7959 : *Block-Wise Transfers in the Constrained Application Protocol*
- Combinaison d'échanges (requêtes/réponses)
  - Négociation possible sur la taille des blocs
- Options `Block1` et `Block2` dans les messages
  - Indiquent que payload du message est un bloc d'un contenu plus grand (body)
  - `Block1` : relatif à une requête, `Block2` : relatif à une réponse
  - Paramètres usuels d'une option
    - taille du bloc : puissance de 2 entre 2^4 et 2^{10}
    - booléen indiquant si d'autres blocs suivent
    - numéro du bloc dans la séquence de blocs

Implantations de CoAP

- Plusieurs implantations libres et commerciales
  - Dans différents langages (C, Java, Javascript, Python)
  - Pour des dispositifs plus ou moins contraints
  - Spécialisées (ou pas) côté client ou serveur
- Exemples d'implantations libres
  - libcoap, smcp, microcoap, Californium
  - node-coap, CoAPthon

Californium (Cf)

- Implantation en Java de CoAP
  - Pour dispositifs pas très contraints
    - mémoire et CPU pour une JVM
- Offre complète du RFC 7252 + extensions
  - Observe, Block-wise transfer, Discovery
- Caractéristiques
  - Mise en œuvre modulaire
  - Exploite des *connectors* (support de la communication)
    - UDP, DTLS, et autres potentiellement
  - Représentation objet des ressources (hiérarchie d'objets)
- Outil associé
  - Copper (Cu) : extension Firefox implantant un agent utilisateur (client) CoAP

Code Serveur

- Serveur
  - Héberge un arbre de ressources
  - Les ressources sont exposées à travers un ou plusieurs endpoints
  - Un endpoint est lié à une interface réseau
- Création de ressources (en étendant CoapResource)

```
public class MyResource extends CoapResource {

   @override public void handleGET(CoapExchange exchange) {
       exchange.respond("hello world");  // 2.05 content avec payload texte
   }

   @override@override public void handlePOST(CoapExchange exchange) {
     exchange.accept();  // la réponse sera séparée (renvoie ici un ACK)

     if (exchange.getRequestOptions()...)     {
        // teste et ajoute/modifie une option
     }
     exchange.respond(CREATED); // renvoie la réponse (avec seulement le code)
   }
}
```

Code Serveur

- Ajout des ressources au serveur
- Démarrage du serveur

```
public static void main(String[] args) {

     CoapServer server = new CoapServer() ;

     server.add(new MyResource("hello"));

     server.start() ;
}
```


Code Client

- Client synchrone

```
String uri = new String("coap://serveur.example.com:5683/hello");
CoapClient client = new CoapClient(uri);
CoapResponse response = client.get();

System.out.println(response.getCode());
System.out.println(response.getOptions());
System.out.println(response.getResponseText());
```

- Client asynchrone

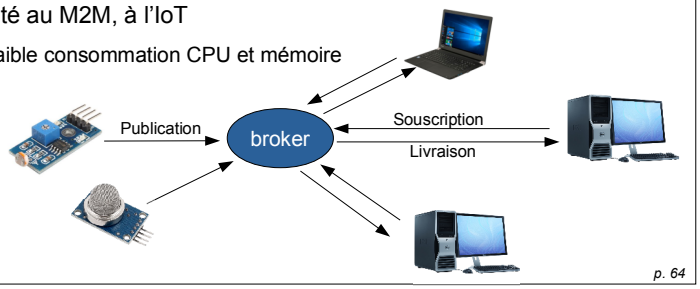
```
CoapClient client = new CoapClient(uri);

client.get(new CoapHandler() {

      @Override public void onLoad(CoapResponse) {
          System.out.println(response.getResponseText());
      }

      @Override public void onError() {
          System.err.println("Erreur") ;
      }

});
```


MQTT

MQTT

- MQ Telemetry Transport
  - À l'origine (1999), un système propriétaire (A. Standford-Clark, A. Nipper)
    - Connexion de systèmes de télémétrie de pipeline par satellite
  - Libre depuis 2010, standardisé (OASIS) depuis 2013

- Système de messagerie (MOM) publish/subscribe
  - Clients + Broker (serveur intermédiaire entre clients)
  - Léger, adapté au M2M, à l'IoT
    - Simple, faible consommation CPU et mémoire



Protocole MQTT

- Protocole client/serveur
  - Client ↔ Broker
  - Transport : TCP/IP (port 1883)
    - Alternatives : web sockets, TLS (port 8883)
- Mode opératoire
  - Échanges de paquets entre client et broker (*control packets*)
    - Paquet = entête fixe
      + entête variable [facultatif] : paramètres (e.g., nom de topic)
      + charge utile binaire [facultatif] : message applicatif (e.g., valeur de capteur)
  - Déroulement :
    - Le client établit une connexion avec le broker (paquet CONNECT)
      - Juste après l'établissement de la connexion TCP
    - Échange de paquets de publication/ souscription
    - Déconnexion
      - Normalement initiée par le client (paquet DISCONNECT)
      - Éventuellement par le serveur (sur faute, panne réseau...)

Paquets de contrôle

- Entête fixe

bit	7-4	3	2-1	0
1er octet	Type	DUP flag	QoS	RETAIN flag
2e octet...	taille restante			

  - Types

```
CONNECT     [C→S]  Client request to connect to server
CONNACK     [S→C]  Connect acknowledgement
PUBLISH     [C↔S]  Publish message
PUBACK      [C↔S]  Publish acknowledgement
PUBREC      [C↔S]  Publish received   }
PUBREL      [C↔S]  Publish released   }  Niveaux 1, 2 et 3 de livraison
PUBCOMP     [C↔S]  Publish complete   }
SUBSCRIBE   [C→S]  Client subscribe request
SUBACK      [S→C]  Subscribe acknowledgement
UNSUSBCRIBE [C→S]  Unsusbcribe request
UNSUBACK    [S→C]  Unsubscribe aknowledgement
PINGREQ     [C→S]  PING request
PINGRESP    [S→C]  PING response
DISCONNECT  [C→S]  Client is disconnecting
```

Paquets de contrôle

- Entête Variable
  - Dépend du type de paquet
  - Contient une série de paramètres, par exemple :
    - flags (indiquant pour certains des valeurs à trouver dans le payload).
    - valeurs numériques (e.g., valeur du Keep Alive)
    - identifiant de paquet
- Payload
  - Contient un ou plusieurs champs (selon le type du paquet), par exemple :
    - Contenu du message (paquet PUBLISH)
    - Valeurs associées aux paramètres de l'entête (paquet CONNECT)

Sessions

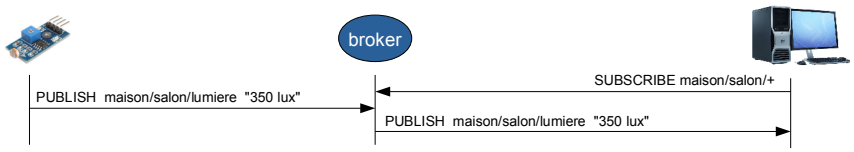
- Session = Interaction avec état entre un client et le serveur
  - Cas simple : la session dure le temps de la connexion réseau
    - Entre paquets CONNECT et DISCONNECT
  - *Clean session* : la session couvre plusieurs connexions consécutives
    - Flag « clean session » à 1 dans le paquet CONNECT
- Nécessité de stocker des infos de session
  - Côté client et côté serveur
  - Fermeture de session à prévoir si stockage dépassé
- Identification de client :
  - Paramètre « Client ID » dans paquet CONNECT
    - Chaîne de caractères alphanumériques (max 23 car. en UTF-8)
  - Nécessaire pour les *clean sessions*

Sessions

- Authentification
  - Une session peut être authentifiée
  - Login/mot de passe (en clair) dans paquet CONNECT
    - Flags « User Name Flag » et « Password Flag » + valeurs dans le payload
- Keep Alive
  - Le client doit émettre des paquets suffisamment souvent
    - Le serveur coupe la connexion sinon
    - Paramètre temporel « Keep Alive » donné par le client dans le CONNECT
    - Émission de PING si nécessaire

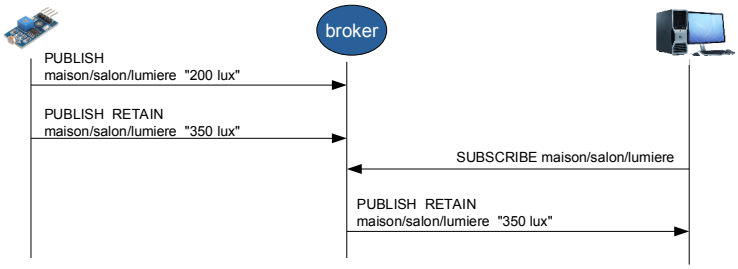
Publication / Souscription

- Topics hiérarchiques
  - Spécifiés dans les entêtes variables
  - Séparateur de niveau : /
    - Exemple : maison/salon/temperature
    - Commencer par / est possible (1er niveau vide) mais est déconseillé
  - Wildcards dans les filtres de topic
    - # remplace plusieurs niveaux de topic
    - + remplace un (seul) niveau
    - $ permet de spécifier des topics réservés
      - un client ne peut pas créer un topic commençant par $
      - convention : souscription à $SYS/ pour obtenir des infos « système » du broker



RETAIN Flag (persistance)

- Le publieur peut demander à ce que le broker conserve les messages pour les futurs souscripteurs
  - Flag RETAIN dans les paquets de contrôle (PUBLISH)
  - Le broker ne conserve qu'un seul message par topic
- Le client recevra le message en réponse à une nouvelle souscription qui matche le topic du message avec RETAIN
  - Permet a un client de recevoir immédiatement la dernière valeur valide



RETAIN Flag

- Le publieur peut publier des messages sans RETAIN *après* un message avec RETAIN
  - Les nouveaux souscripteurs recevront le message avec RETAIN
- On supprime le message RETAIN sur le broker en publiant un message RETAIN vide
- La conservation sur le broker est en réalité temporaire
  - C'est l'implantation du broker qui décide combien de temps elle conserve le message.
    - Il faut s'assurer que l'implantation du broker utilisée est capable de stocker suffisamment de messages pour l'application

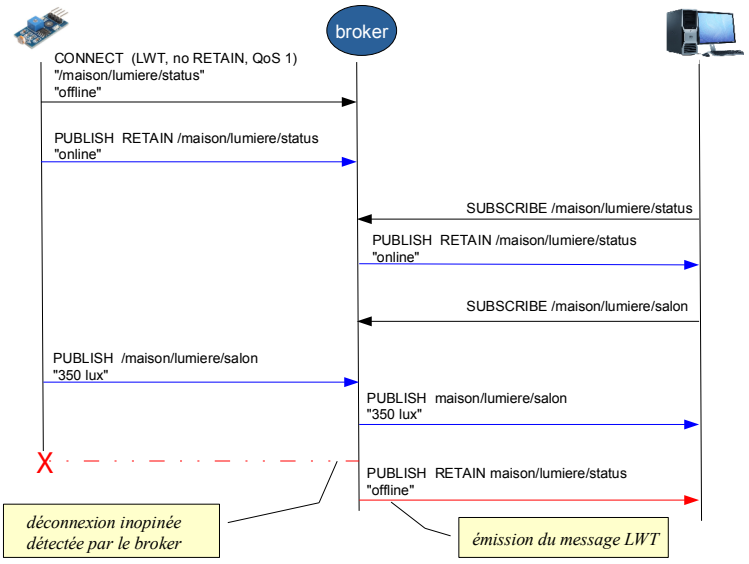
QoS

- Trois niveaux de QoS : 0, 1, 2
  - QoS assurée pour les messages émis par un client vers le broker ou par le broker vers un client (pas de bout en bout)
  - Niveau spécifié dans PUBLISH, niveau max accepté dans SUBSCRIBE
- QoS 0 : *At most once delivery (fire and forget)*
  - Utile quand on peut se permettre de perdre des messages
  - Rapide, pas de stockage sur le serveur
- QoS 1 : *At least once delivery*
  - Le récepteur accuse réception (PUBACK)
  - Livraison garantie, mais l'application doit pouvoir gérer les duplicats
- Qos 2 : *Exactly once delivery*
  - Double accusé de réception (PUBREC, PUBREL, PUBCOMP)
  - Livraison garantie, sans duplicat

Last Will and Testament

- Utilisé par un publieur pour notifier les souscripteurs d'une déconnexion inopinée
  - Lors de sa connexion, le publieur indique un message LWT
    - Paquet CONNECT contenant
      - flags « Will Flag », « Will Retain », « Will QoS »
      - Will topic et Will message dans payload
  - Quand le broker détecte une rupture de connexion inopinée avec le publieur, il publie le message LWT
    - Rupture de connexion inopinée = pb réseau, batterie épuisée...
    - Rupture de connexion inopinée ≠ déconnexion (paquet DISCONNECT)
- Le contenu et le topic du message LWT sont généralement prédéfinis
  - e.g., topic=maison/lumiere/status, contenu= « offline » avec message RETAIN initial contenant « online »

Last Will And Testament



Implantations de MQTT

- Client MQTT
  - Très nombreuses implantations
    - Langages et plateformes diverses
  - Notamment pour des systèmes contraints
    - e.g. bibliothèque C pour Arduino
    - Parfois limités (e.g., pas de support de QoS 2)
  - Problèmes d'intéropérabilité potentiels
- Broker MQTT
  - Pas prévus pour dispositif contraint
  - Intégré à un serveur MOM généraliste
    - e.g., Apache Artemis, IBM Websphere MQ
      - MQTT en plus de JMS ou d'un protocole propriétaire...
  - ou spécialisé
    - e.g., Eclipse Mosquitto