Laboratoire réseau 2017

SmartHub

Routage applicatif

AA. DOSTIE – C. LORQUET – L. SWINNEN

Ce document est disponible sous licence Creative Commons indiquant qu'il peut être reproduit, distribué et communiqué pour autant que le nom des auteurs reste présent, qu'aucune utilisation commerciale ne soit faite à partir de celui-ci et que le document ne soit ni modifié, ni transformé, ni adapté.

https://creativecommons.org/licenses/by-nc-nd/3.0/deed.fr

La Haute Ecole Libre Mosane (HELMo) attache une grande importance au respect des droits d'auteur. C'est la raison pour laquelle nous invitons les auteurs dont une œuvre aurait été, malgré tous nos efforts, reproduite sans autorisation suffisante, à contacter immédiatement le service juridique de la Haute Ecole afin de pouvoir régulariser la situation au mieux.

Mars 2017

SmartHub – Routage applicatif

Introduction

Le sujet du laboratoire de réseau de cette année est de *développer une application capable de répondre, de manière efficace, à des requêtes en provenant de clients*. *Les informations sont fournies par des services Web à interroger*.

L'objectif est de vous faire travailler l'*implémentation d'un protocole (reprenant l'analyse syntaxique des messages, la compréhension et la réponse adaptée), la programmation concurrente (permettant de supporter plusieurs connexions simultanées), la communication unicast (avec TCP) et multicast (avec UDP)* et *la programmation de requêtes HTTP (pour interroger les services web proposés)*.

Le travail sera réalisé, par groupe **de 2 étudiants**. Exceptionnellement, et avec l'accord du responsable de laboratoire **un seul groupe de 3 étudiants** (avec des objectifs plus grands) sera possible.

Le laboratoire comporte **28 h** dont **24 h seront dédiées** à la réalisation de cet exercice. Si vous prenez du retard, il sera nécessaire d'avancer par vous-même.

Pour ce laboratoire, **vous devez utiliser le GitLab HELMo**. Ainsi, on vous demande de déterminer, rapidement, quel étudiant hébergera le dépôt pour ce travail.

Un planning est proposé à la suite de la présentation de l'énoncé. Veillez à le respecter scrupuleusement.

1. L'architecture

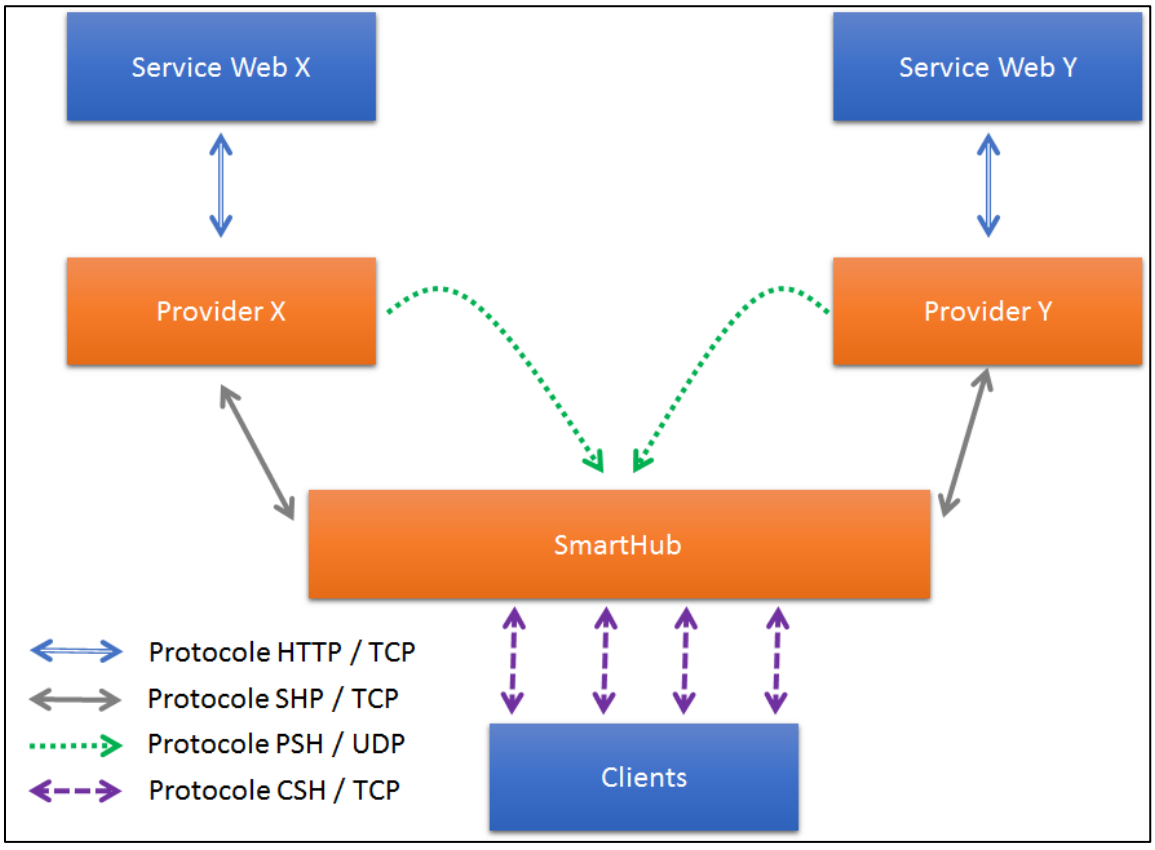

Figure 1 : Architecture du système

Comme nous pouvons le voir sur *la figure 1*, l'architecture du système est assez variée, on y trouve divers protocoles de transports (UDP et TCP) et applicatifs (standard comme HTTP ou personnalisé comme SHP, PSH ou CSH).

Dans cette architecture, seuls les éléments en orange sont à implémenter (à savoir **le SmartHub** et **le(s) provider(s)**). Les autres éléments (le *client* et le *service web*) sont fournis.

1.1 Fonctionnement

Le **client** souhaite obtenir une information ou ressource, par exemple une *mesure d'une température*. Pour ce faire, il va **interroger le smart hub**. Celui-ci doit, en fonction de la nature de la requête (par exemple *demande de relevé de température*), **interroger le fournisseur** attaché au service web fournissant cette information ou ressource. Le **fournisseur** interroge alors le service web en question et retourne le résultat **au smart hub**. Celui-ci doit, ensuite, **retourner le résultat au client qui a effectué la demande**.

Le **Smart Hub** doit, en fonction des **fournisseurs qui se feront connaître**, proposer **les services demandés aux clients**. Ainsi, il doit être possible de démarrer et/ou d'arrêter un fournisseur de ressources sans que le **Smart Hub** ne soit perturbé.

Les **fournisseurs doivent se faire connaître** par le **Smart Hub** afin que celui-ci puisse leur transmettre les requêtes adéquates. Chaque fournisseur est spécialisé en fonction du service qu'il rend (ie. de la ressource qu'il fournit). Ainsi, un fournisseur de relevé de température peut être très différent d'un fournisseur de relevé d'humidité, de données météo, d'horodateur certifié, …

Il y a **deux services web** que vous **devez** supporter : **une mesure de température** et **une mesure du taux d'humidité**. Dans les améliorations possibles, vous avez la possibilité d'écrire d'autres fournisseurs liés à des services web disponibles sur Internet.

Il y a donc plusieurs applications différentes à écrire. Le choix langage (C# ou Java) est à votre discrétion. Vous pouvez varier les langages dans l'implémentation de vos applications.

1.2 Scénario

Dans ce paragraphe, nous allons découvrir le scénario le plus courant de cette application :

1) Le **smart hub** est démarré. Par défaut, il écoute pour entendre, en multicast, les annonces faites par les fournisseurs. Il attend également toute connexion en provenance d'un client. Le **smart hub** doit maintenir une liste de services ou de ressources qu'il est capable de proposer au client, en fonction des fournisseurs auxquels il est connecté.
2) Les applications **fournisseurs** sont démarrées. Chacune, en utilisant le multicast, **s'annonce** en répétant le message `AWAKE` toutes les 30 secondes (voir le protocole).
3) Le **smart hub** ayant reçu les annonces de chaque fournisseur va, pour chacun, démarrer une connexion directe (TCP, unicast, protocole SHP décrit ci-dessous) pour transmettre toute requête provenant d'un client. S'il a pu établir cette connexion avec le fournisseur, il met à jour la liste des ressources qu'il peut proposer aux clients.
4) Le **client** se connecte au **smart hub** directement (TCP, unicast, protocole CSH décrit ci-dessous). Il peut ainsi **obtenir la liste des services / ressources disponibles** et **faire une requête sur une ressource donnée**.
    a) Le **smart hub** réceptionne toute requête et, via la connexion directe qu'il a établie avec le fournisseur, transmet cette demande.
    b) Le **fournisseur** reçoit la demande du **smart hub** et effectue une requête HTTP vers le service web concerné pour obtenir la ressource demandée.

c) La réponse du service web est transmise au **fournisseur** qui est alors chargé de la faire suivre **au smart hub** qui lui, la transmettra au client ayant effectué la demande.

2. Les protocoles

Comme nous l'avons mentionné, il y a plusieurs protocoles applicatifs utilisés : on trouve le protocole standard HTTP, et les protocoles spécifiques nommés SHP, PSH et CSH. Chacun d'entre-eux fera l'objet d'une description détaillée puisqu'il vous faudra les implémenter précisément dans vos applications.

Tous les protocoles sont décrits sous la forme d'une *grammaire ABNF*[1]. Ainsi, la syntaxe est strictement spécifiée et vos applications doivent s'y conformer précisément.

2.1 Définitions standards

```
lettre = %d65-90 / %d97-122   ; [A-Z][a-z]
chiffre = %d48-57             ; [0-9]
lettre_chiffre = lettre / chiffre   ; une lettre ou un chiffre
crlf = %d13 %d10              ; \r suivi de \n
port = 1*5digit               ; port : 1 à 4 chiffres
caractere = %d32-255          ; tous les caractères imprimables et l'espace
sp = %d32                     ; espace
type = "INTEGER" / "STRING"   ; type est soit la chaine INTEGER, soit STRING
id = 1*50lettre_chiffre       ; 1 à 50 lettres et/ou chiffres
id_ressource = id
id_client = 1*4chiffres       ; 1 à 4 chiffres
valeur = 1*500caractere       ; 1 à 500 caractères
ressource = id_ressource "," type   ; un id suivi d'une virgule et d'un type
```

2.2 Le protocole CSH

Le protocole CSH est utilisé **par un client** pour transmettre ses requêtes **au smart hub**. De ce fait, il respecte le modèle client/serveur traditionnel. Le client devra connaitre l'adresse IP du serveur. Le numéro de port dédié à la discussion entre le client et le serveur sera `15000+x` (où `x` représente votre numéro de groupe).

Le client (déjà programmé) et le smart hub doivent respecter le protocole suivant :

```
ressource_disponible = "LISTR" crlf
liste_ressource = "RLIST" (sp ressource)+ crlf
requete_valeur = "QUERY" sp id_ressource crlf
reponse_requete = ("VALUE" sp id_ressource SP valeur crlf) / ("ERROR" crlf)
quit = "SAYONARA" crlf
message = requete_valeur / reponse_requete
```

Typiquement[2], le client commencera par demander la liste des ressources disponibles via le message `LISTR`. Le smart hub lui répondra par un message `RLIST` mentionnant toutes les ressources qu'il connait.

Le client demande la valeur d'une ressource par le message `QUERY`, qui mentionne l'identifiant de la ressource souhaitée. Le serveur peut répondre à ce message par un message `VALUE` donnant la valeur demandée ou `ERROR` si aucune ressource n'est connue avec cet identifiant.

[1] Augmented Backus-Naur Form (RFC 5234 – https://tools.ietf.org/html/rfc5234)
[2] Mais sans obligation de commencer par ce message

Enfin, lorsque le client a terminé son échange, il transmet un message `SAYONARA` pour annoncer sa déconnexion.

2.3 Le protocole PSH

Le protocole PSH est utilisé **par un fournisseur** pour annoncer au **smart hub** la ressource qu'il fournit. Cette annonce permettra au smart hub de proposer le service aux clients.

Ce protocole doit être *multicast* et utiliser le protocole de transport UDP. Cela signifie que les messages d'annonce seront envoyés, en multicast, régulièrement. Le Smart Hub doit être configuré pour recevoir ces messages.

L'adresse IP multicast réservée pour les échanges dans le groupe multicast sera : `239.255.10.x` (`x` représente votre numéro de groupe, cela permet de traiter uniquement vos messages et pas ceux des autres groupes). Le port à utiliser est `15 000`.

Ce protocole propose le message suivant :

```
hello = "AWAKE" sp port sp ressource crlf
```

Le *port* indique le numéro de port à utiliser pour établir la connexion directe (TCP, unicast, protocole SHP) vers le fournisseur. L'adresse IP du fournisseur est l'adresse IP source du message multicast.

Ce message devra être envoyé par le fournisseur, en multicast, toutes les 30 secondes au groupe mentionné. Aucune réponse n'est attendue à ce message. Le **smart hub** ne transmet aucune information en multicast, il « se contente » de les recevoir et de les traiter.

2.4 Le protocole SHP

Le protocole SHP est utilisé entre le **Smart Hub** et le **fournisseur**. Il utilise la couche de transport TCP (transfert fiable des informations). Il permet au **Smart Hub** d'interroger un **fournisseur donné** pour obtenir une ressource.

Les messages de ce protocole sont :

```
forward = "FORWARD" sp id_client sp message crlf
quit = "SAYONARA" crlf
```

Le **smart hub** communique, grâce à la connexion directe établie avec le fournisseur, et propage les requêtes provenant du client. Le message `FORWARD` permet d'encapsuler la requête du client (`message` étant soit `QUERY` ou `VALUE/ERROR`). Afin de pouvoir transmettre la réponse qu'il recevra au client (qui sera conforme au message `VALUE/ERROR`, encapsulée dans un `FORWARD`), il ajoutera **un identifiant numérique** désignant ce client. Ainsi le **smart hub** pourra transmettre la réponse au client ayant effectué la requête.

2.5 Le protocole HTTP

Le protocole HTTP 1.1 fait déjà l'objet d'une spécification complète et précise (décrite dans le RFC 2616 – https://www.ietf.org/rfc/rfc2616.txt). Ce protocole est employé **entre un service web et son fournisseur**. Nous attirons votre attention sur le fait que, dans tous les langages actuels, des composants pour manipuler HTTP sont fournis. Le fournisseur **est spécialisé** en fonction du service web qu'il doit interroger. Ainsi, l'URL à utiliser et la séquence des méthodes HTTP à appeler sont propres au service web.

Les URLs des services web de base sont les suivantes :

- Température : http://192.168.128.7:8000/temperature/, méthode GET, résultat JSON, valeur en °C
- Humidité : http://192.168.128.7:8000/humidity/, méthode GET, résultat JSON, valeur en %

3. Implémentation

L'implémentation à réaliser devra :

- Etre écrite en C# / Java en respectant, tant que possible, les principes de la programmation objets. Vous pouvez choisir d'implémenter une application en C# et l'autre en Java.
- Vérifier tous les messages du protocole par des *expressions régulières*[3]. Les langages C# et Java supporte ces expressions nativement.
- Compter, pour chaque application, le nombre de threads créés (au total).

4. Planning

Voici le planning à respecter pour ce laboratoire :

Phase	Description	Heures
1	Développement du Hub et de la connexion avec le client. Utilisation d'un FakeProvider	6
2	Connexion avec le Webservice depuis un provider. Implémentation d'un provider spécialisé en fonction du service web. Discussion unicast et requêtes HTTP (découverte des composants JAVA/C# adéquats)	2 - 4
3	Développement de la communication multicast : annonce des providers - ressources	2
4	Connexion entre SmartHub et les providers et intégration de l'ensemble.	6
5	Optimisation et améliorations	6
Eval	Présentation du projet	4

Les phases 1 à 4 **doivent être terminées** pour espérer la réussite dans ce laboratoire. Si les phases 1 à 4 **sont terminées et conformes EXACTEMENT à ce qui est demandé**[4], la note obtenue sera de MAXIMUM 14/20. Pour espérer obtenir une note supérieure, il faut terminer la 5ème phase. S'il y a un groupe composé de 3 personnes, il est obligatoire d'implémenter les 5 phases pour espérer réussir le projet.

Une présence **d'au moins à la moitié des laboratoires de tous les membres du groupe** est requise pour être autorisé à défendre le projet.

La version qui sera défendue sera celle **déposée le mercredi 17 mai 2017 à 23h55** sur le GitLab HELMo au plus tard.

La dernière séance est réservée, dans chaque groupe, à la présentation orale de votre projet et son évaluation sur base de votre défense.

Une évaluation des pairs vous sera également demandée, individuellement. Celle-ci pourra être utilisée pour ajuster la note de chaque membre du groupe.

[3] Pour tester vos expressions régulières, le site http://regexr.com peut être utile

[4] Autrement dit, si nous n'avons aucune remarque concernant l'implémentation et le fonctionnement de votre application