Projet Java RMI

Advanced Middleware

Sommaire

Implémentation Java RMI des Web Hooks

Notre architecture repose de manière très simple sur 2 interfaces Java présentes à la fois sur le client et sur le serveur :

- `ClientHook.java - Cette interface possède les méthodes suivantes :`
  - `boolean receive(String message) throws RemoteException;`
- `PubSubServer.java - Cette interface possède les méthodes suivantes :`
  - `boolean subscribe(ClientHook client) throws RemoteException;`
  - `boolean unsubscribe(ClientHook client) throws RemoteException;`

L'interface *ClientHook* est implémentée côté client alors que l'interface *PubSubServer* est implémentée côté serveur. Une fois le serveur lancé, le client va instancier un objet héritant de *UnicastRemoteObject* et implémentant *ClientHook*. Il peut, dès lors, souscrire au serveur en faisant appel à la méthode *subscribe* en passant en paramètre l'objet typé *ClientHook.*

Côté serveur, à chaque souscription, une sauvegarde de l'objet distant reçu est effectuée; Lorsqu'un message est produit, le serveur appelle la méthode *receive* sur l'ensemble des clients sauvegardés, de façon à afficher le message du côté client.

Implémentation du protocole HTTP Streaming avec Java RMI

Le protocole HTTP Streaming repose sur un échange en continu de blocs d'octets répondant à une unique requête GET. L'exemple donné dans la thèse de Benjamin Billet comporte le schéma de la figure n°1 ci-dessous.

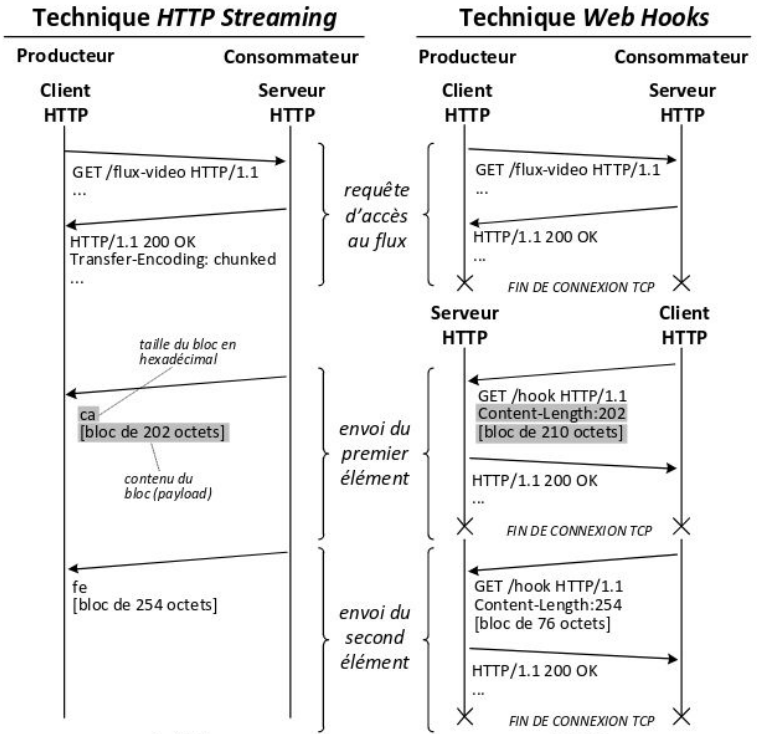

Figure n°1 : Échanges réseaux pour HTTP streaming et Web hooks

On remarque ici, dans le cas du streaming HTTP, qu'une unique requête GET sur l'URI /flux vidéo déclenche une première réponse dont le code est 200 et comportant la propriété *Transfer-encoding: chuncked.* Cette dernière permet, depuis la version 1.1 du protocole HTTP, d'indiquer un transfert de blocs du serveur au client sans connaître à l'avance la taille totale de la donnée transférée.

Java RMI ne permet pas d'effectuer un envoi de ce type. En effet, à chaque fin d'appel de méthode distante, la connexion est fermée. Voyons donc quelles méthodes pourraient être utilisées pour simuler le HTTP Streaming :

Méthode 1 :

Il est possible d'imaginer un client exposant une méthode prenant en paramètre un tableau de bytes et un numéro d'ordre. Suite à la requête GET initiale du client, le serveur pourrait alors diviser la donnée en N blocs d'octets et faire des appels successifs à la méthode de réception du bloc côté client.

Méthode 2 :

Cette seconde méthode est celle qui s'éloigne le plus de Java RMI. L'idée est ici d'utiliser un appel de méthode distante en Java RMI pour ouvrir une connexion TCP servant à l'envoi des blocs d'octets. La connexion peut alors rester active.

Conclusion

De ces deux alternatives, aucune ne correspond à une réelle implémentation du protocole HTTP Streaming car la technologie Java RMI ne s'y prête pas. En effet, à cause de la perte de la connexion à chaque message envoyé, il est compliqué de simuler une connexion qui reste vivante après plusieurs messages à sens unique envoyés.