Projet Système Distribué : Implémentation d'un serveur générateur de certificats

BEUQUE Eric, CORNEVAUX Sébastien, MOUTENET Cyril

13 janvier 2009

Table des matières

1 Sujet

Afin d'appliquer plusieurs des technologies découvertes en module de Systèmes Distribués, les étudiants de Master 2 SDR avaient un projet à réaliser à deux ou à trois.

Le but du projet était l'implémentation d'un serveur Java délivrant des certificats X509 à deux clients, disposant de clés, pour permettre à ces derniers d'établir une connexion sécurisée et authentifiée.

L'acheminement des certificats jusqu'aux clients était fait de manière à ce que les technologies JMS et Corba soient entre autres utilisées.

Ce rapport revient sur nos choix de réalisation et de développement en partant de l'implémentation des clients jusqu'à celle du serveur.

2 Analyse

Si nous avions toutes les clés en main pour mener à bien le projet avec les TPs réalisés en amont de ce travail, l'utilisation de nombreuses technologies nécessitait un indispensable travail d'analyse.

Nous avons donc rapidement schématisé de la manière suivante notre vision du sujet :

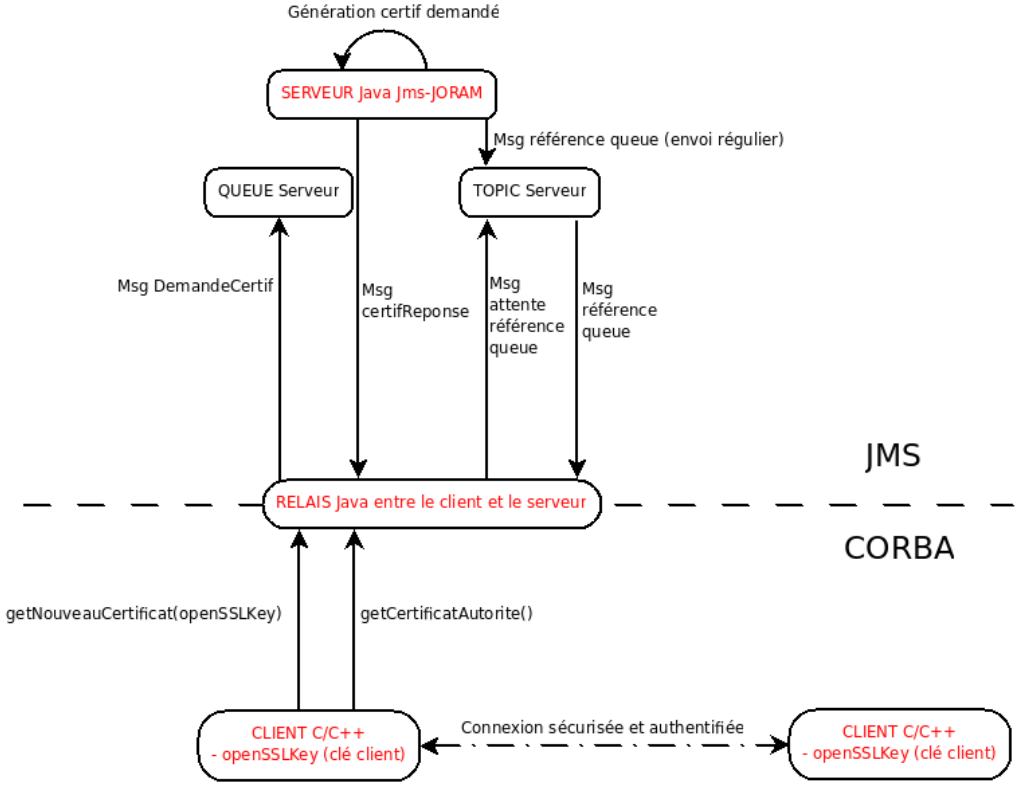

FIG. 1 – Analyse

3 Création clé client

Pour la génération des clés client et des requêtes à destination du serveur de certificats, nous nous sommes appuyé sur le TP OpenSSL.

L'opération se découpe donc en deux commandes :

1. openssl genrsa –out key 1024 : génération d'une clé RSA (1024 correspond à la taille),

2. openssl req –new –key key –out req : génération de la requête à destination d'une autorité de certification à partir de la clé client.

C'est donc cette requête générée qui sera envoyée en direction du serveur de certificats.

Dans notre projet, on appel les deux commandes en exécutant le script de lancement "genkey.sh" avec l'option "-c".

4 Client C/C++

Notre client va d'abord "charger" la requête générée avec sa clé dans le bon fichier.

Après l'initialisation de Corba, le client peut aller "récupérer" (protocole RPC) son nouveau certificat chez le relais en lui passant la requête.

Le client stocke alors le certificat dans le fichier "key.pem".

Le client va de la même manière aller (mais sans passer de clé cette fois-ci) ensuite "récupérer" le certificat de l'autorité de certification.

Ce second certificat est stocké dans "cacert.pem".

Une fois que nos deux clients ont réalisé ces opérations, ils peuvent communiquer (échange de messages de bienvenue) de manière sécurisée et authentifiée.

En pratique, les deux clients vérifient d'abord que le nouveau certificat reçu est compatible avec leurs clés privées. Ensuite, ils établissent la connection SSL par laquelle ils vérifient d'abord la validité entre le certificat de l'autre client et celui d'autorité.

5 Communication Java/C++ entre client et relais : Corba

Corba est un standard qui permet la communication sur le modèle RPC en environnement hétérogène et l'accès normalisé à des services.

Pour faire communiquer notre client C/C++ et notre relais Java, il fallait qu'ils partagent la même interface Idl ("RelaisApp.idl").

6 Les relais Java

Le relais fait l'intermédiaire entre le client et le serveur.
La fonction du relais est de communiquer avec le serveur de certificat au moyen d'un *topic* et d'une *queue* JMS. La *queue* et le *topic* sont crées par le

serveur de certificat. Le relais possède deux méthodes pouvant être appelées par un client corba.

6.1 Fonctionnement

Au démarrage du relais, celui-ci se connecte au *topic*. La référence du *topic* est stockée dans le JNDI ("fr.sd.topic"). On y récupére aussi le Connection-Factory du *topic*. Nous avons ajouté la possibilité de réessayer plusieurs fois de récupérer le *topic* au moment du démarrage du relais.

La récupération des messages du *topic* est réalisée de façon asynchrone avec validation automatique des messages lus. Au moment de la réception d'un message *MessageServeurPret* dans le *topic*, on récupére la référence de la *queue* contenue dans le message transmit.

Pour la *queue*, nous avons choisi de récupérer les messages de façon asynchrone avec validation des messages lus par le relais. En effet, la *queue* contient un message pour le serveur de certiticat et un message de réponse pour le relais à un moment donné. Seul les messages concernant le relais sont consommés avec *msg.acknowledge()*

6.2 Modélisation

Nous avons modélisé les classes suivantes pour le relais :

- RelaisServeur : Classe gérant les fonctionnalités métier du relais.
- TopicRelaisServeur : Classe gérant la communication avec le *topic*, donc seulement la réception de messages *ServeurPret*.
- QueueRelaisServeur : Classe gérant la communication avec la *queue* ; Elle permet l'envoi et l'écoute de message sur la *queue* ; Dès qu'un messsage *CertificatReponse* arrive dans la *queue* celle-ci appelle la méthode *retournerCertificat* correspondant au certificat demandé. Les messages de demande situés dans la *queue* sont ignorés. Lorque le serveur est prêt, un message de demande est envoyé dans la *queue*.

Lorsqu'un client demande un certificat, celui-ci est mis en attente par un système wait notify jusquà reception du certificat. une fois la demande effectuée, l'indication serveur prêt débloque l'envoi d'un message de demande dans la *queue*. Lorsque le message certificat est capté dans la *queue* cela déclenche de déblocage du client. Le client suivant peut ainsi être traité à la suite.

7 Communication entre relais et serveur : JMS

La spécification JMS suppose que les objets de connexion et de communication soient déjà créés avant l'exécution des programmes Donc on créer les objets et on les déclare à Jndi avant d'utiliser le Serveur JMS-Joram avec nos programmes.

8 Serveur Java

Le serveur JMS-Joram intègre un service Jndi pour la déclaration des objets qu'il gère.

Pour le serveur de message et l'accès aux entités Queue et Topics pour la communication par message, nous avons utilisé l'implantation JORAM de JMS comme en TP

La fonction du relais est de communiquer avec le serveur de certificat au moyen d'un *topic* et d'une *queue* JMS. La *queue* et le *topic* sont crées par le serveur de certificat. Le relais possède deux méthodes pouvant être appelées via un client corba.

8.1 Fonctionnement

Au démarrage du relais, crée le *topic* et le référence dans le JNDI ("fr.sd.topic"). Même chose pour ConnectionFactory du *topic*. De même, le queue est créée mais n'est pas enregistrée dans le JNDI. Seule son instance est stockée dans un message *ServeurPret* envoyé dans le *topic* lorsque celui-ci est prêt à traiter une demande de certificat.

Le message serveur prêt est envoyé dans le *topic* toutes les cinq secondes.

Pour la *queue*, nous avons choisi de récupérer les messages de façon asynchrone avec validation des messages lus par le serveur de certificats. En effet, la *queue* contient un message pour le serveur de certiticat et un message de réponse pour le relais à un moment donné. Seuls les messages concernant le serveur de certificats sont consommés avec *msg.acknowledge()*

8.2 Modélisation

Nous avons modélisé les classes suivantes pour le relais :

- CertificatServeur : Classe gérant les fonctionnalités métier du serveur de certificats.
- TopicCertificatServeur : Classe gérant la communication avec le *topic*, donc seulement l'envoi de messages *ServeurPret*.

– QueueCertificatServeur : Classe gérant la communication avec la *queue* ; Elle permet l'envoi et l'écoute de message sur la *queue* ; Dès qu'un message *DemandeCertificat* arrive dans la *queue* celle-ci appelle la méthode métier *genererCertificatClient* ou *genererCertificatAutorite* en fonction du message reçu. Dès que le certificat est fini d'être généré, celui-ci est retourné au relais.

9 Gestion des certificats

Comme pour la gestion des clés clients, nous nous sommes appuyés sur le TP OpenSSL pour la gestion des certificats.

Dans notre projet, on appel les commandes en exécutant "genkey.sh" avec l'option "-s".

10 Guide d'utilisation

10.1 Arborescence du dossier projet

Configuration de départ :

– Fichiers racine : on retrouve à la racine du dossier de notre projet les fichiers lanceurs de celui-ci.

– Dossier **config** : contient les fichiers de configuration (omniORB, Joram, Jndi...).

– Dossier **doc** : contient le sujet du projet ainsi que ce rapport.

– Dossier **src** : contient les sources Java ou C++ des client, relais, serveur ainsi que l'interface Idl commune au client et au relais pour la communication via Corba.

Dossiers créés au fil de l'exécution du projet :

– Dossier **bin** : va accueillir les fichiers exécutables du projet.

– Dossier **run** : va accueillir une copie des fichiers de configuration.

– Dossier **dirclient1** : va accueillir la clé et la requête du client 1.

- Dossier **dirclient2** : va accueillir la clé et la requête du client 2.

- Dossier **CA** : va accueillir le certificat serveur.

10.2 Exécution

Après s'être positionné à la racine du dossier de notre projet :

1. Génération du certificat serveur :
   ./genkey.sh -s CA

2. Génération des clés des clients :
   ./genkey.sh -c dirclient1
   ./genkey.sh -c dirclient2

3. Compilation du projet :
   ant

4. Lancement du serveur CORBA :
   tnameserv -ORBInitialPort 1664

5. Lancement su serveur JMS :
   ./single_server.sh

6. Lancement du serveur de certificats et du relais :
   ant run_all

7. Lancement du client 1 :
   ./bin/client/client -s dirclient1/ 8888

8. Lancement du client 2 :
   ./bin/client/client -c dirclient2/ 8888 127.0.0.1

11 Bilan

Ce projet nous a d'abord permis de parfaire nos connaissances sur les technologies utilisées, à savoir la manipulation des clés et certificats avec

openssl pour la sécurité et l'authentification de la connexion entre les deux clients, ainsi que Corba et Jms pour les communications entre nos entités client, relais et serveur.

Ce travail a donc constitué un bon complément aux TPs du module de Systèmes Distribués.

De plus, une piqûre de rappel de C++ n'était pas inutile tant on oublie vite lorsqu'on utilise trop rarement un langage de programmation.