Système de vidéo-surveillance

Documentation développeur

Description (rapide) de l'architecture de l'application

Afin de mieux comprendre comment l'application est extensible à un nouveau protocole, il est nécessaire de rappeler l'architecture de l'application. Celle-ci a pour objectif de gérer des catalogues (ou serveurs) qui possèdent des flux. Un flux utilise un protocole réseau pour transmettre des images. On retrouve donc assez naturellement une classe Stream à laquelle est associée un protocole (qui étend IProtocol), l'intérêt est ainsi d'avoir un découpage entre l'aspect réseau/communication et la notion de flux (caractéristiques plus générales). Au sein du protocole on trouvera notamment la gestion des données (connexion de données) et le contrôle sur le protocole (connexion de contrôle). Il a été privilégié une architecture sous forme de briques modulaires. Ainsi, on retrouve dans notre implémentation actuelle des classes destinées à la gestion du transfert de données en TCP, en UDP, en Multicast, de même pour le contrôle. On pourrait alors imaginer de concevoir un protocole basé sur une connexion de contrôle en TCP et permettant le transfert en UDP. Les briques déjà programmées seraient réutilisables telles quelles.

Ce choix d'architecture implique une faible factorisation du code mais laisse une plus grande liberté d'évolution à notre application puisque la logique même du protocole est faiblement couplée à la représentation du protocole. Cependant, il est vrai que la notion de protocole en elle même (commandes possibles, états) est assez rigide et que dans notre cas nous nous limitons à la gestion d'images.

On notera que l'application intègre une gestion d'erreurs sommaire qui permet principalement de centraliser les erreurs en provenance des différents threads et de les rediriger vers des " listeners ".

D'une manière générale, la remontée des événements des différents threads de l'application passe par un mécanise de " listeners " (exemple : lorsqu'une image a été reçue). Il ne faut pas oublier que ces listeners sont invoqués dans le thread de la tâche en question et que cela peut poser problème pour des mises à jours de l'interface graphique par exemple (voir SwingUtilities et la gestion du multithreading avec Swing).

Extension pour nouveau protocole

Afin de concevoir un nouveau protocole, nous allons détailler les différentes briques qu'il est nécessaire d'implémenter :

- Implémentation de IProtocol : le noyau du protocole, elle contiendra la logique qui va avec, notamment la réponse aux différentes commandes (démarrer, pause, arrêt, …). On associera donc les changements d'états à ces commandes. Cette classe exposera l'adresse et le port du serveur. Comme elle contient la logique du protocole, elle sera amenée à décrire les commandes à envoyer au serveur (si il y en a).

    - Cette classe gérera de manière autonome l'envoi de commandes, on conseille donc de le réaliser à travers un thread chargé de l'écriture de la commande sur une socket ouverte avec le serveur à la manière de TcpControlThread par exemple.
    - Elle s'occupera aussi de gérer la réception de données ce qui sera certainement fait à travers un autre thread (à moins que l'on fasse des lectures bloquantes, il n'y a pas de limitation à ce niveau là). C'est à ce niveau que l'on pourra notifier les " listeners " d'arrivée d'images.
- Ajouter le nouveau protocole à la fabrique de protocole ProtocolFactory (avec une énumération pour l'identifier ainsi que sa représentation textuelle, utile pour les fichiers de configurations par exemple).

Quelques recommandations pour implémenter au mieux un nouveau protocole :

- Gérer correctement le cycle de vie des différentes thread créés, à savoir veiller à ce que la destruction se fasse correctement sans pour autant fermer les connexions de manière barbare en laissant le serveur gérer des déconnexions brutales !
- Les différentes appels aux commandes du protocole contiennent le flux en paramètre ce qui permet de lui remonter des informations notamment les statistiques.

Pour résumer l'extension d'un protocole est assez simple puisqu'il s'agit principalement d'implémenter l'interface IProtocol, le développeur est ensuite assez libre et surtout responsable des erreurs qu'il fera. Les classes XXXControlThread, XXXDataThread et XXXProtocol sont de bons exemples pour voir ce qu'il est possible de faire.

Améliorations possibles

Du fait du contexte du projet, certains points n'ont pas été abordés de manière très complète, notre application souffre de quelques défauts donc voici un aperçu :

- il serait intéressant d'avoir une gestion des erreurs beaucoup plus complète, avec notamment une meilleure traçabilité sur la source de l'erreur.
- introduire de la reprise sur erreurs, tenter de rétablir les connexions en cas de déconnexions serait un plus certain.
- le code n'a pas été complétement optimisé (création de chaînes de caractères par exemple) sur des sections que nous n'avons pas estimé critiques, cependant il serait certainement plus propre d'éviter certaines (coûteuses) créations d'objets.

Diagrammes de classes UML