Rapport de Projet Données Réparties PODP 1

Mathieu Campan, Besson Germain, Simon Hautesserres
Groupe L1

Département Sciences du Numérique
2022-2023

Table des matières

Introduction

Ce rapport décrit nos choix de conception et les algorithmes que nous avons implémenté au cours de l'étape PODP 1. Nous nous sommes appuyé sur notre version de l'étape 1 du S7, afin de limiter les risques d'erreur et avoir une base fonctionnelle.

1 Ajout apportés :

Dans cette étape, nous avons ajouté des méthodes qui permettent à un client de s'abonner à un objet et permette à une notification de se propager aux clients abonnés, après l'écriture sur un objet d'un client.

1.1 Abonnement

Pour **l'abonnement**, nous avons ajouté les méthodes *lookupAndSuscribe* et *registerAndSuscribe* dans Server.java et Client.java et *subscribe* dans ServerObject.java.

Dans le code du **client**, *lookupAndSuscribe* et *registerAndSuscribe* réalisent les mêmes fonctionnalités que lookup et register, à la seule différence que ces méthodes font appel respectivement à *lookupAndSuscribe* et *registerAndSuscribe* du Server.
Dans le code du **Server** :

- *lookupAndSuscribe* fait appel à la méthode *subscribe* du ServerObject correspondant au nom donné en paramètre, à condition que ce nom soit déjà enregistré dans le Server.

- *registerAndSubscribe* réalise la même chose en appelant *subscribe* du ServerObject correspondant.

Dans le code du **ServerObject** nous avons ajouté, en tant qu'attribut, une liste des clients abonnés à l'objet. La méthode subscribe est chargé d'ajouter le client passé en paramètre à cette liste.

1.2 Notification

Pour les notifications, nous avons ajouté la méthode *notification* dans Client.java. Elle est appelée dans la méthode *unlock* du SharedObject lorsque le verrou est en WLT uniquement. La notification est propagée jusqu'au ServerObject correspondant.

Dans le ServerObject, *notification* parcourt le tableau des clients abonnés. Pour chaque client, la méthode *getNotification* est appelée.

getNotification est définie dans le code du SharedObject. Elle incrémente un nouvel attribut du SharedObject nommé notif. Cet attribut est initialisé à zéro et est incrémenté à chaque

appel de *notification* pour tous les clients abonnés au SharedObject. Enfin, il est remis à zéro à chaque fois qu'un client fait un lockread (mais seulement pour lui, les autres clients devront faire un lockread à leur tour).

1.3 Irc_notif

Nous avons repris les mêmes bases que pour l'IRC du semestre 7 en ajoutant comme particularité qu'il met à jour le compteur (attribut notif du SharedObject). Celui-ci est affiché au centre de la fenêtre sous la forme d'une zone de texte et nous voyons en temps réel sa valeur incrémenter ou être mise à zéro.

2 Choix de conception :

2.1 Observateur

Afin de mettre à jour le compteur, nous avions mis un appel statique dans SharedObject vers Irc_notif mais cela ne nous satisfaisait pas car SharedObject doit rester indépendant de toute application qui le lance.
Nous avons donc défini une interface Observateur_itf.java qui possède une seule fonction *getNotification(int compteur*).
Nous avons également créé une classe l'implémentant pour l'exemple du Irc_notif : Observateur_Irc_notif.java. L'Irc_notif s'occupe de créer cet observateur qui sera passé en paramètre dans *lookupAndSuscribe* et surtout *registerAndSuscribe*.
À noter que lookupAndSubscribe nécessite également un observateur en paramètre car nous nous sommes assuré que le registerAndSubscribe ne stocke pas l'observateur afin d'éviter qu'un client soit abonné malgré lui. Ainsi, l'observateur passera par un appel au Client puis sera enregistré dans le SharedObject pour avoir accès en temps réel (sans possibilité de le modifier) à la valeur de l'attribut notif, qui représente le compteur à afficher.

2.2 Opération unlock()

Pour le moment notre unlock() est bloquant à cause de l'appel à *Client.notification* qui se propage vers le serveur puis aux abonnés. Dans le cadre de nos premiers tests simples cela ne pose pas de problème, mais nous pensons le rendre par la suite non bloquant afin d'éviter des problèmes d'interblocage liés à la concurrence, ou simplement des problèmes de baisse de performance par attente de la propagation de la notification alors que le client ayant fait le *unlock()* n'a pas d'intérêt à attendre cette propagation.

2.3 Gestion de blocages

À cette étape du projet, la notification ne fournit pas d'objets de rappels, elle n'est qu'une simple information de modification. Il n'y a donc pas de modifications de verrous ou de l'objet partagé en dehors des appels clients read et write. Ce point sera principalement abordé lors de l'étape suivante, où nous devrons géré les copies antérieures de l'objet partagé pour former une sorte d'historique.