# Rapport Projet Données réparties

EL ALOUT Ismail
KARMAOUI Oussama

Département Sciences du Numérique - Deuxième année- Parcours HPCBD
2022-2023

# Contents



# List of Figures

# 1 Introduction

L'objectif de ce projet est de développer une application de gestion d'objets dupliqués en JAVA. Cette application devrait autoriser un accès en lecture à plusieurs clients en même temps, mais de ne permettre l'écriture qu'à un seul client.

Dans ce service, les objets sont représentés par des descripteurs (instances de la classe SharedObject d'interface SharedObject itf) qui possèdent un champ obj qui pointe sur l'instance (ou une grappe d'objets) Java partagée. Toute référence à une instance partagée doit passer par une telle indirection. Dans une première etape, cette indirection est visible pour le programmeur qui doit adapter son mode de programmation. Dans une seconde etape, on implantera des stubs qui masquent cette indirection.

# 2 Etape 1

Dans cette première version, les SharedObject sont utilisés explicitement par les applications. Plusieurs applications peuvent accéder de façon concurrente au même objet, ce qui nécessite de mettre en œuvre un schéma de synchronisation globalement cohérent pour le service. On ne gère pas le stockage de référence à des objets partagés dans les objets partagés.

## 2.1 Gestion de la cohérence

Toutes les interactions entre le serveur et l'application se font au travers d'une couche : le Client. Ce client possède dans une liste (on a choisit de l'implémenter avec une HashMap qui associe à chaque objet un identifiant) tous les objets partagés connus de l'utilisateur.
La classe SharedObject possède une référence vers l'instance de l'objet effectivement partagé. L'état d'un objet est obtenu à travers la valeur du verrou: attribut Lock de type lockState; où lockState est une énumération définissant l'ensemble des états possibles pour un verrou (NL, RLT, WLT, RLC, WLC, RLT_WLC). La cohérence est assurée par le biais des méthodes de demande/réduction/invalidation de verrou. Ces méthodes étant appelées directement par l'application.

Les points importants concernant l'implémentation de la classe Client sont :

### 2.1.1 Initialisation du client (méthode init()) :

Concernant l'initialisation du client, elle s'effectue en deux etapes :

– Recherche de l'enregistrement du serveur dans le serveur de nom grace à la méthode Naming.lookup. On récupère ainsi toutes les informations nécessaires pour communiquer avec le serveur. Cependant chaque utilisateur doit connaitre l'adresse exacte du serveur (ce qui semble assez logique... En effet comme dans le système mis en place par IRC, l'utilisateur doit rentrer lui-même l'URL (voir clients IRC tels que GunScript, mIRC, ...)

–Initialisation d'une HashMap locale de sharedObject contenant la clé id et l'objet SharedObject associé à cet identifiant. Dans cette HashMap seront stockées en tant que clés, l'identifiant unique de chaque objet, mais egalement l'ensemble des SharedObject associés. Cette HashMap nous sera alors utile lorsque le seveur appellera des méthodes d'invalidation et de réduction de verrou.

#### 2.1.2 Création d'un SharedObject :

Pour créer un SharedObject par une application, elle doit passer par le client pour faire la demande de création. Le client transmet cette demande au serveur, qui crée un ServerObject en lui attribuant un identifiant unique. Ce dernier retourne ensuite cet identifiant au client qui peut alors créer le ServerObject.

#### 2.1.3 Enregistrement de l'identifiant d'un objet dans la HashMap nameMap (méthode register) :

De même, après la création d'un objet partagé, il faut l'enregistrer dans le serveur de nom. Le protocole d'enregistrement se fait alors en suivant le même parcours que précédemment.

#### 2.1.4 Recherche d'un objet (méthode lookup) :

On demande la recherche d'objet portant le nom "nom" en recherchant l'identifiant associé au nom , après on demande le lock_read sur le ServerObject portant ce identifiant qui, à son tour, retourne un objet au client pour créer un SharedObject et l'enregistrer dans le ServerObject local. On retourne le SharedObject au client.

### 2.2 Accès concurrent au service et synchronisation

Le mot clé synchronized est utilisé pour implémenter la gestion des accès concurrents à une ressource critique. Il permet de maintenir un accès exclusif à cette ressource en utilisant les méthodes wait() et notify() pour gérer les threads en attente et les réveiller lorsque la ressource est disponible.

La méthode wait() permet à un thread d'attendre que la ressource critique soit libérée pour y accéder, tandis que la méthode notify() (appelé toujours lors de la libération du verrou avec la méthode unlock()) permet de réveiller un thread mis en attente pour lui permettre d'accéder à la ressource. Nous avons utilisé ces méthodes dans la class Client pour gérer les lectures et les écritures sur les objets partagés.

Nous avons également utilisé un attribut mutex (de type ReentrantLock) pour mettre en oeuvre le concept de l'exclusion mutuelle vu en cours.

### 2.3 Les test réalisés

Pour tester notre implémentation, nous avons utilisé la classe fournie Irc pour réaliser des tests manuellement, et nous avons également défini une classe Tester.java qui permet de tester la synchronisation de notre modèle.

## 3 Etape 2

Dans cette partie, l'objectif est de soulager le programmeur de l'utilisation des SharedObject. On doit donc implanter un générateur de stubs.

### 3.1 Générateur de Stubs

Nous avons réalisé un programme GeneratorStub qui permet de générer le fichier _stub.java d'une classe ou d'une interface. Ce programme se base sur le package java.lang.reflect, on detérmine les méthodes à implémenter et pour chaque méthode, on récupère l'objet encapsulé dans le SharedObject et on appelle la méthode correspondante sur l'objet. Le code est engendré dans le fichier _stub.java grace à printWriter.

### 3.2 Modifications apportées au code de l'étape 1

Nous avons modifié les méthodes create et lookup de la classe Client. En effet, on ne manipule plus directement des SharedObject, mais plutot des objets Sentence. Pour utiliser le Stub, le Client doit créer une instance de celui-ci en connaissant le nom de la classe de l'objet recherché par l'application.

## 4 Conclusion

Ce projet nous a permis de mettre en pratique les concepts de synchronisation abordés dans les cours de systèmes concurrents. Il nous a aussi donné l'occasion de manipuler plus les notions de Java RMI vu en Intergiciels. L'étape 2 nécessitait l'utilisation de méthodes moins courantes comme l'introspection et la manipulation de classes. Cette partie du projet nous a posé une difficulté majeure. Nous n'avons pas pu traité l'étape 3 à cause de l'accumulation de projets et en raison des difficultés rencontrées sur ce projet.