Chapitre 7

Design Patterns et communication réseau

DP Observateur distant et MVC distant

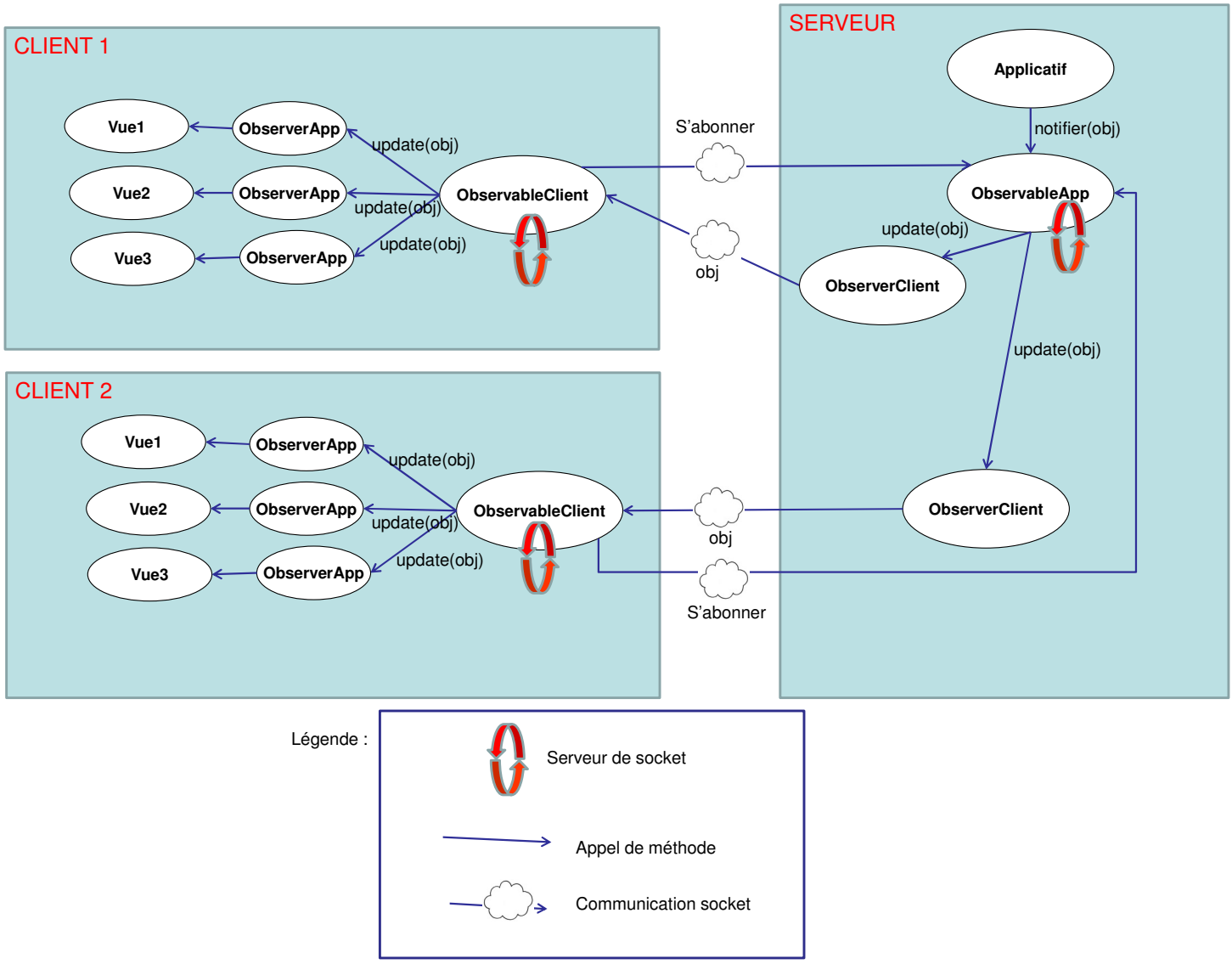
1. Le DP Observateur distant

L'objectif ici est de réaliser le DP Observateur alors que l'Observable est sur un serveur et les Observers sont sur un poste client.

La solution est la suivante : les observateurs (les vues) ne pouvant pas s'abonner à l'observable se trouvant sur le serveur, s'abonnent à un observable créé en local qui communique en socket avec l'observable qui se trouve sur le serveur.

Les principes sont les suivants :
- L'Observable du serveur gère autant d'observateur qu'il existe de poste client.
- L'Observable d'un poste client se déclare (s'abonne) à travers le réseau à l'Observable du serveur.
- Le rôle de l'observateur du serveur est d'envoyer à travers le réseau l'objet à notifier à l'Observable du poste client
- L'Observable du poste client notifie naturellement l'objet à ses observateurs (les vues).

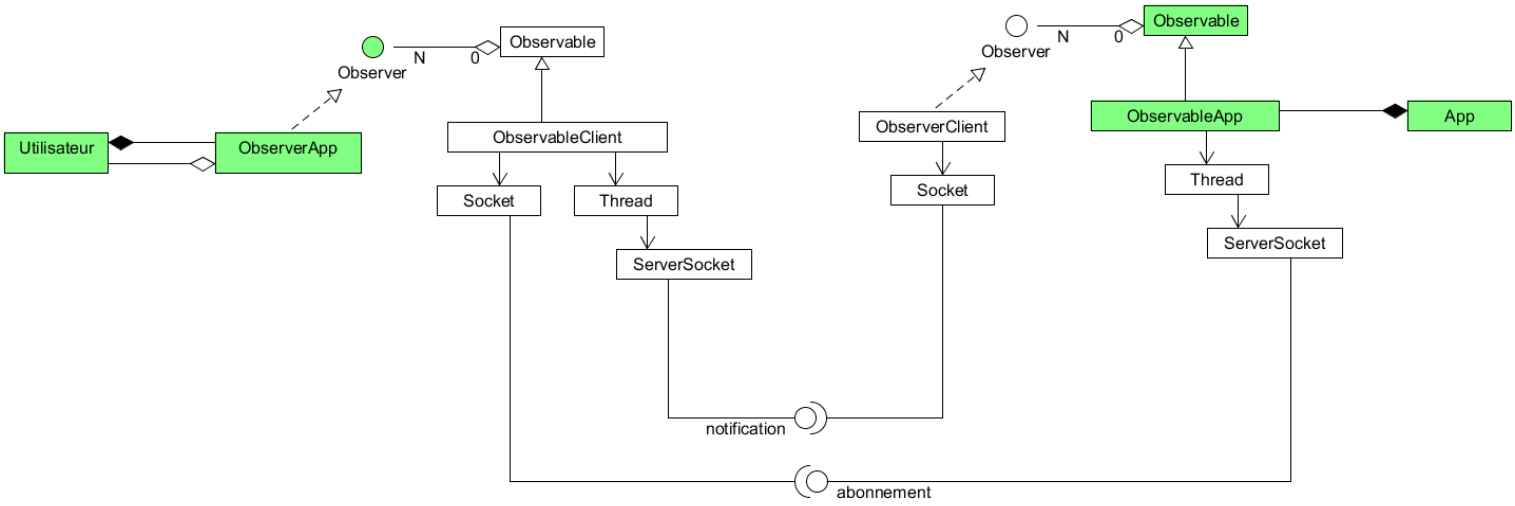
On peut décrire ce principe par le schéma suivant :



Voir sur le site de NFP 121 l'exemple : **Exemple04 Observateur Distant**

Cet exemple est dérivé de l'exemple vu dans le chapitre précédent sur le DP Observateur :

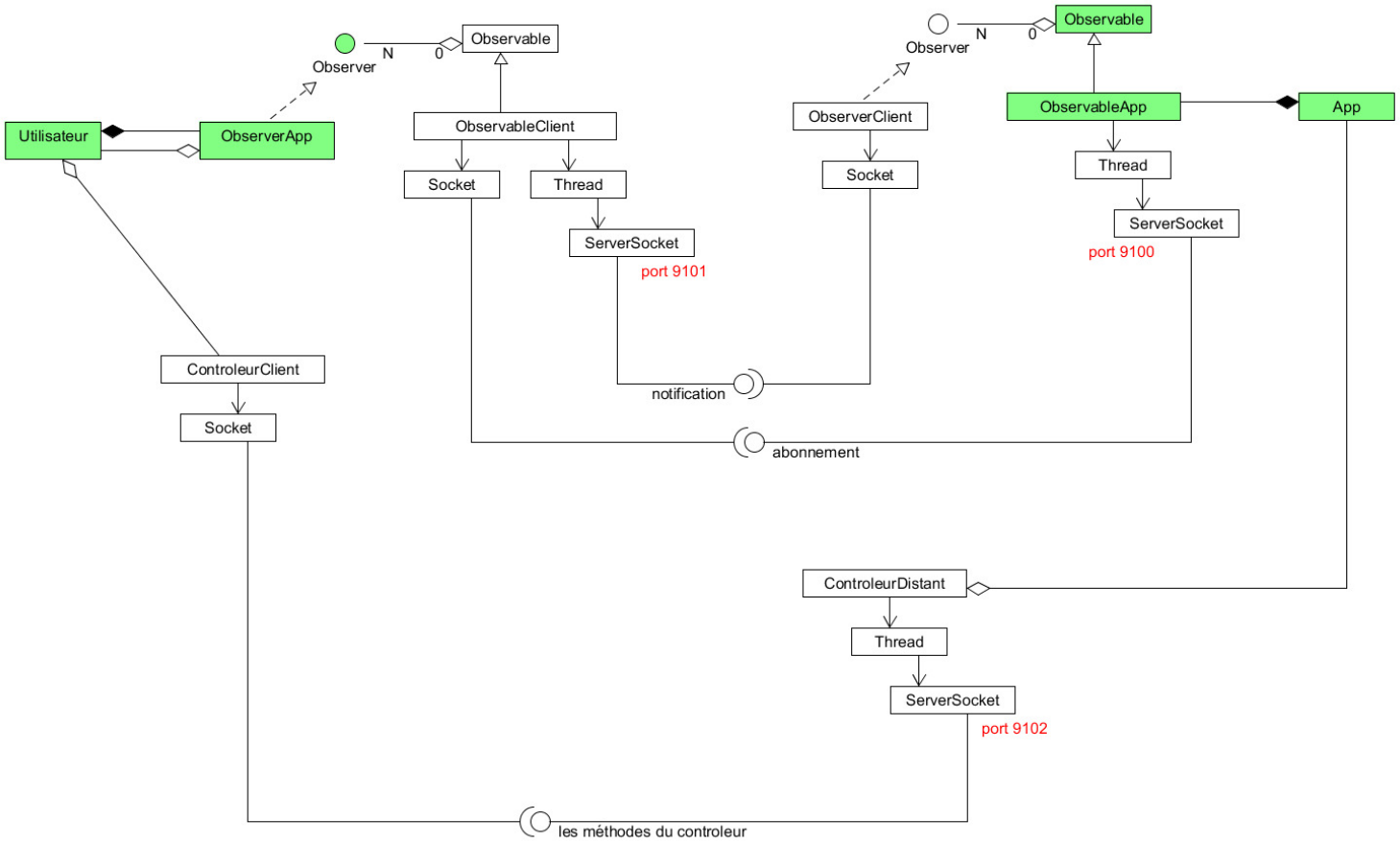
Qui devient dans l'exemple **Exemple04 Observateur Distant :**



2. Le MVC Distant

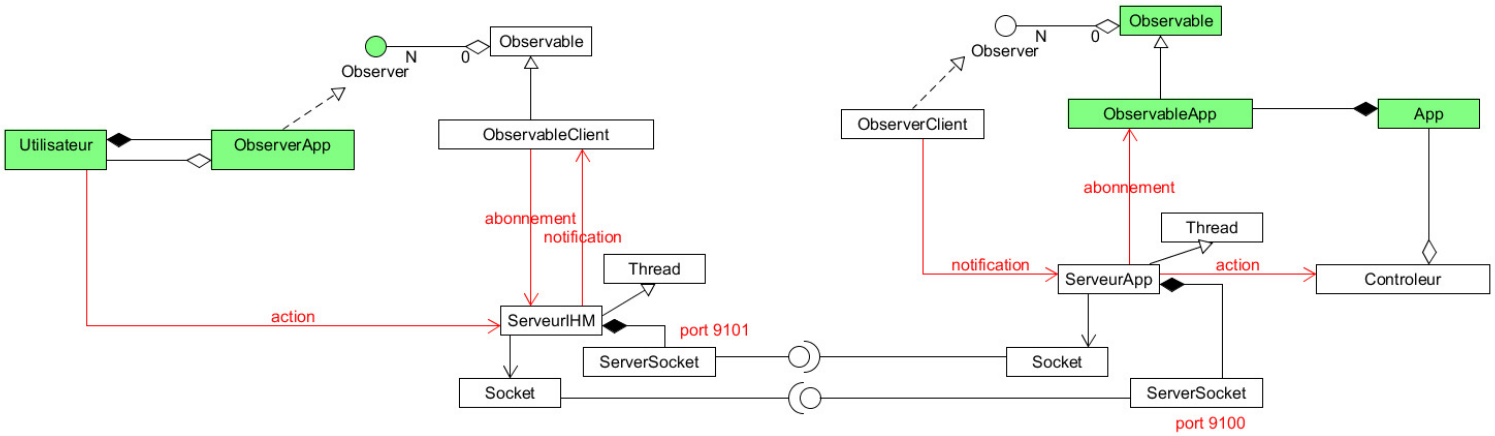
Pour faire un modèle MVC Distant, il suffit de créer dans le CONTROLEUR un Thread de Serveur de Socket sur lequel les vues du client écrivent des objets qui traduisent les différentes méthodes appelées.

Si on applique ce principe sur l'exemple précédent, on obtient :

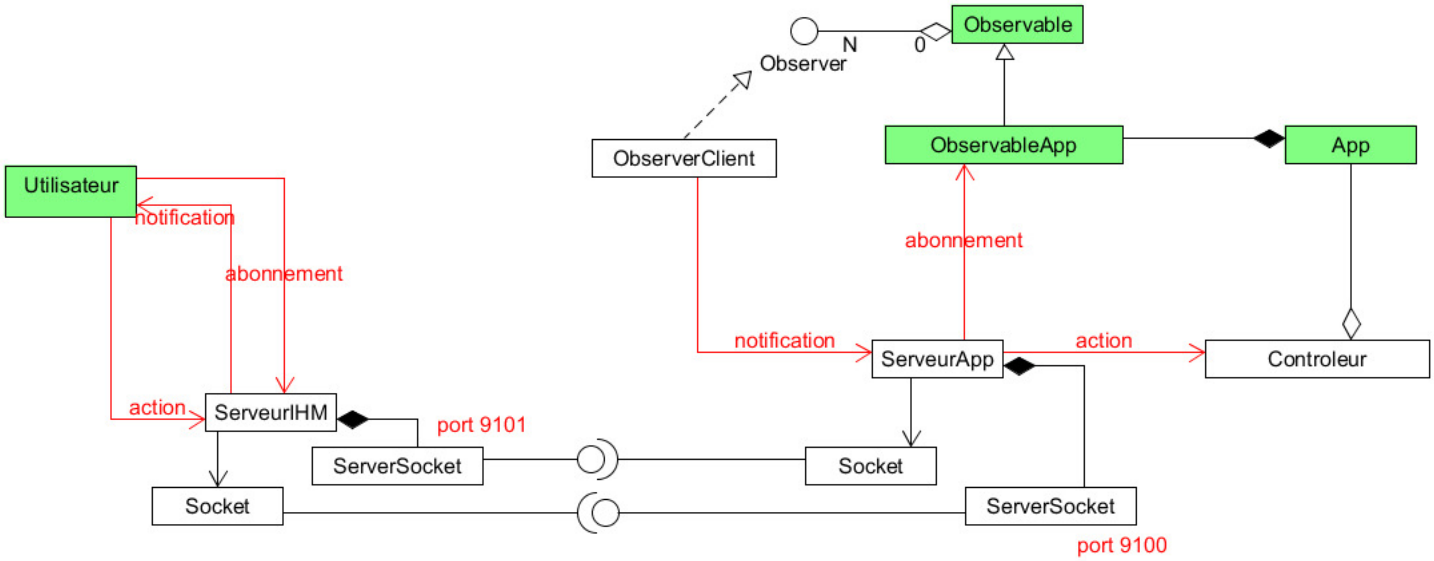


Mais, cela multiplie les serveurs de sockets. On préfère donc une autre solution : créer un seul point d'entrée unique avec un seul serveur de socket. C'est ce point d'entrée unique qui appellera les bonnes méthodes en fonction de la requête envoyée.

De plus, autant aussi faire le point de sortie par la même classe qui implémente ce point d'entrée. On obtient ceci :



Si en plus, comme c'est le cas dans notre projet, il n'existe qu'un seul Observer (l'IHM elle-même), et que en plus, comme en Java les IHM sont des threads, il est inutile de créer des threads, on obtient ceci :


Observable
N
0
Observer
ObserverClient
ObservableApp
App
Utilisateur
notification
abonnement
abonnement
notification
ServeurApp
action
Controleur
action
ServeurIHM
port 9101
ServerSocket
Socket
Socket
ServerSocket
port 9100