Autheur: Mathieu Guichaoua Nicolas EVANO	*TAO project Active Object*	26/01/2012

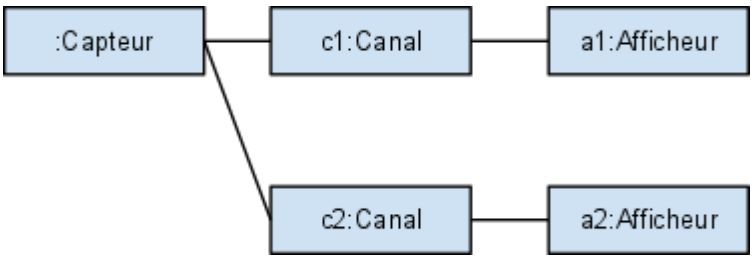
CR Projet TAO

Introduction:

Dans le cadre du module Technique Avancée Objet nous réaliserons un service de diffusion de données de capteur. La solution construite s'appuie sur des mécanismes de programmation par *threads* entièrement gérés par la bibliothèque standard de java et les patrons de conception Active Object et *Observer*.

Description du service de diffusion:

La mise en oeuvre permettra de diffuser un flot de valeurs vers des objets abonnés exécutés dans des threads différents de la source du service. L'objectif du TP étant la mise en oeuvre parallèle d'Observer (patron Active Object), les données diffusées seront une suite croissante d'entiers (c'est à dire un simple compteur). Le compteur sera incrémenté à intervalle fixe. La transmission de l'information vers les abonnés au service emploiera un canal avec un délai de transmission aléatoire.



Autheur: Mathieu Guichaoua Nicolas EVANO	*TAO project Active Object*	26/01/2012

L’architecture se compose:

- d’une source active (capteur), dont la valeur évolue de façon périodique.
- d’un ensemble de canaux de transmission avec des délais variables.
- d’un ensemble d’afficheurs réalisés en utilisant la bibliothèque graphique Swing.
- d’un ensemble de politiques de diffusion Observer, comprenant:
    - La diffusion atomique (tous les observateurs reçoivent la même valeur, qui est celle du sujet).
    - La gestion par époque.

Conception Analyse V2(active object):

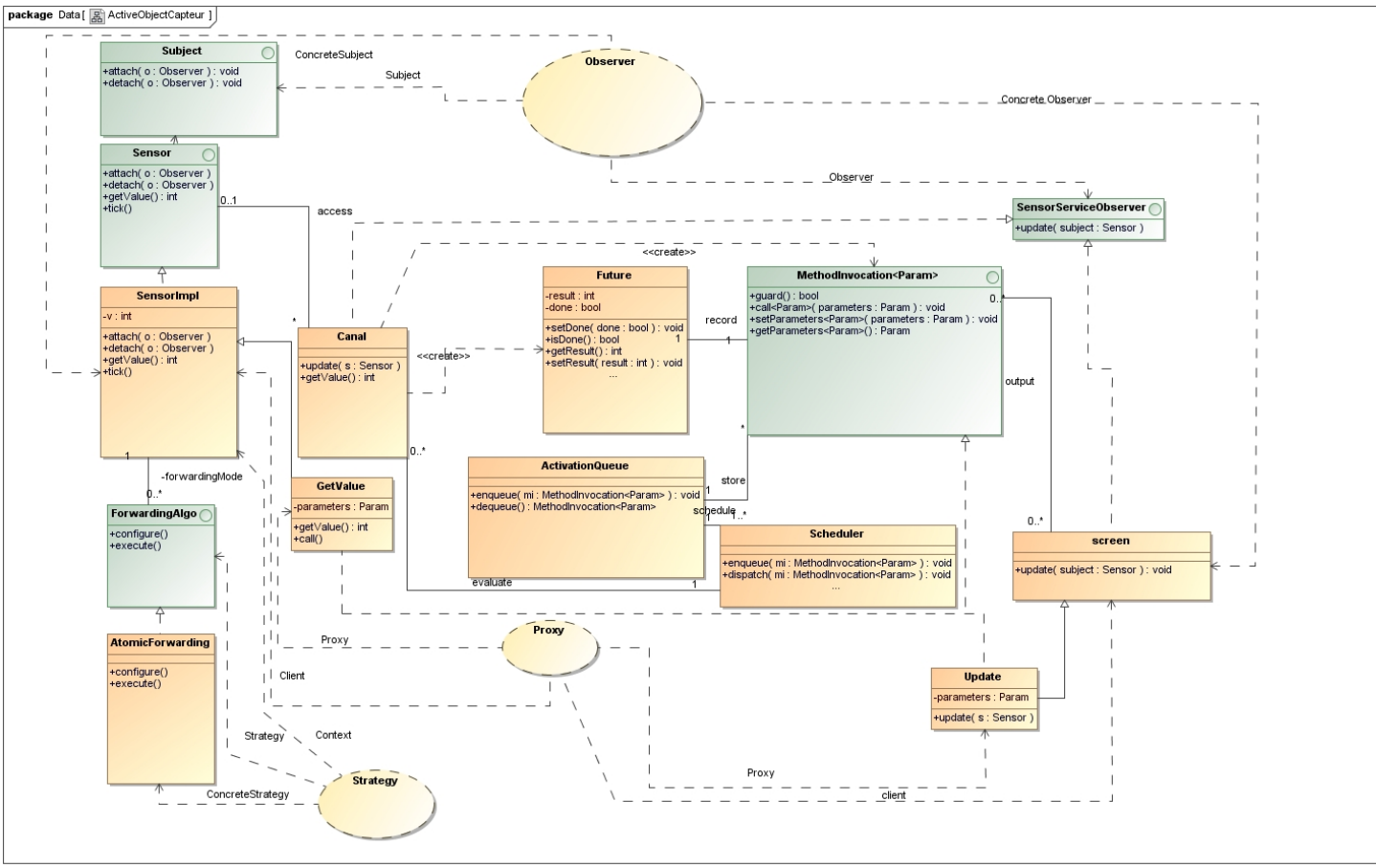
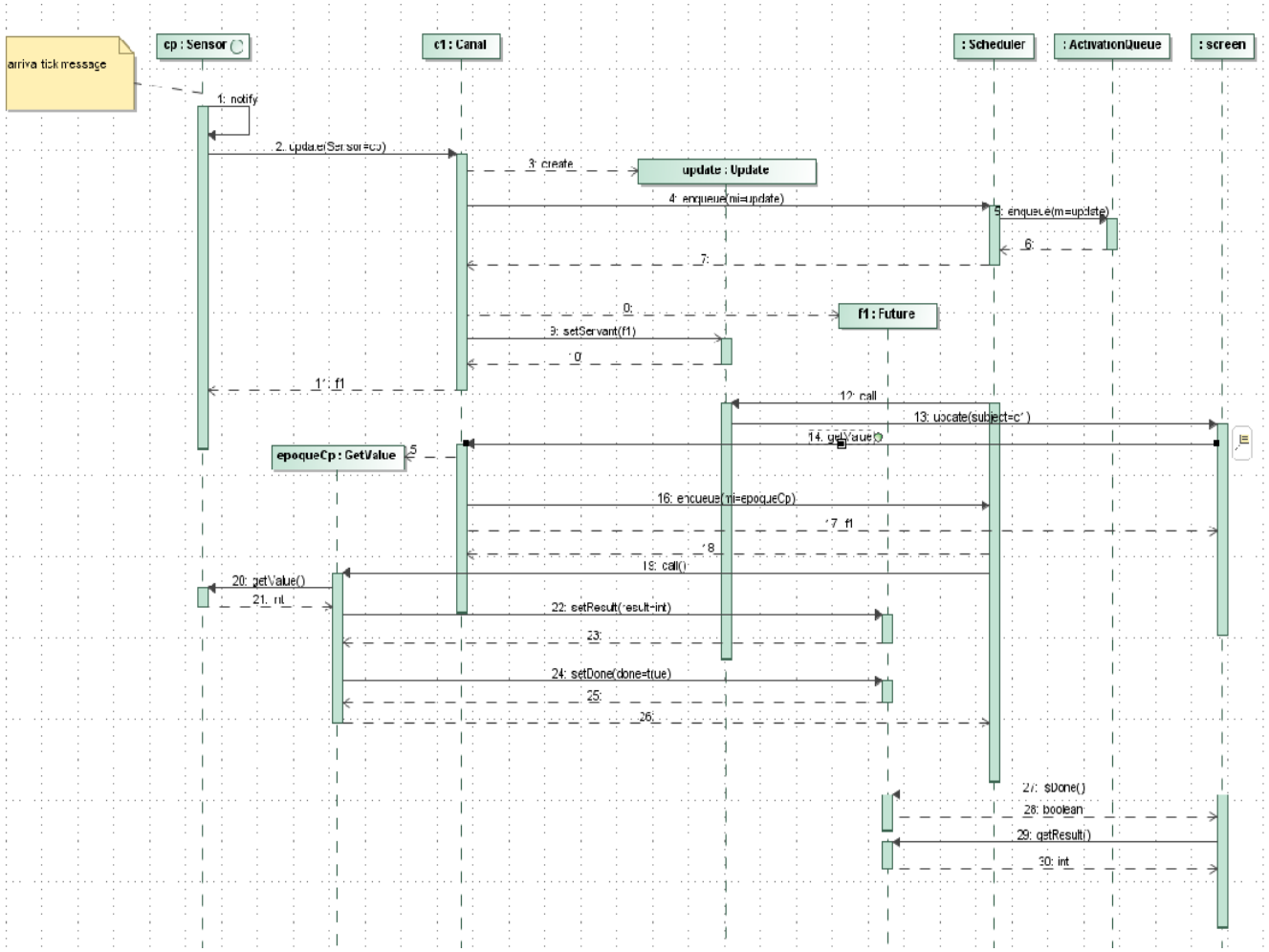
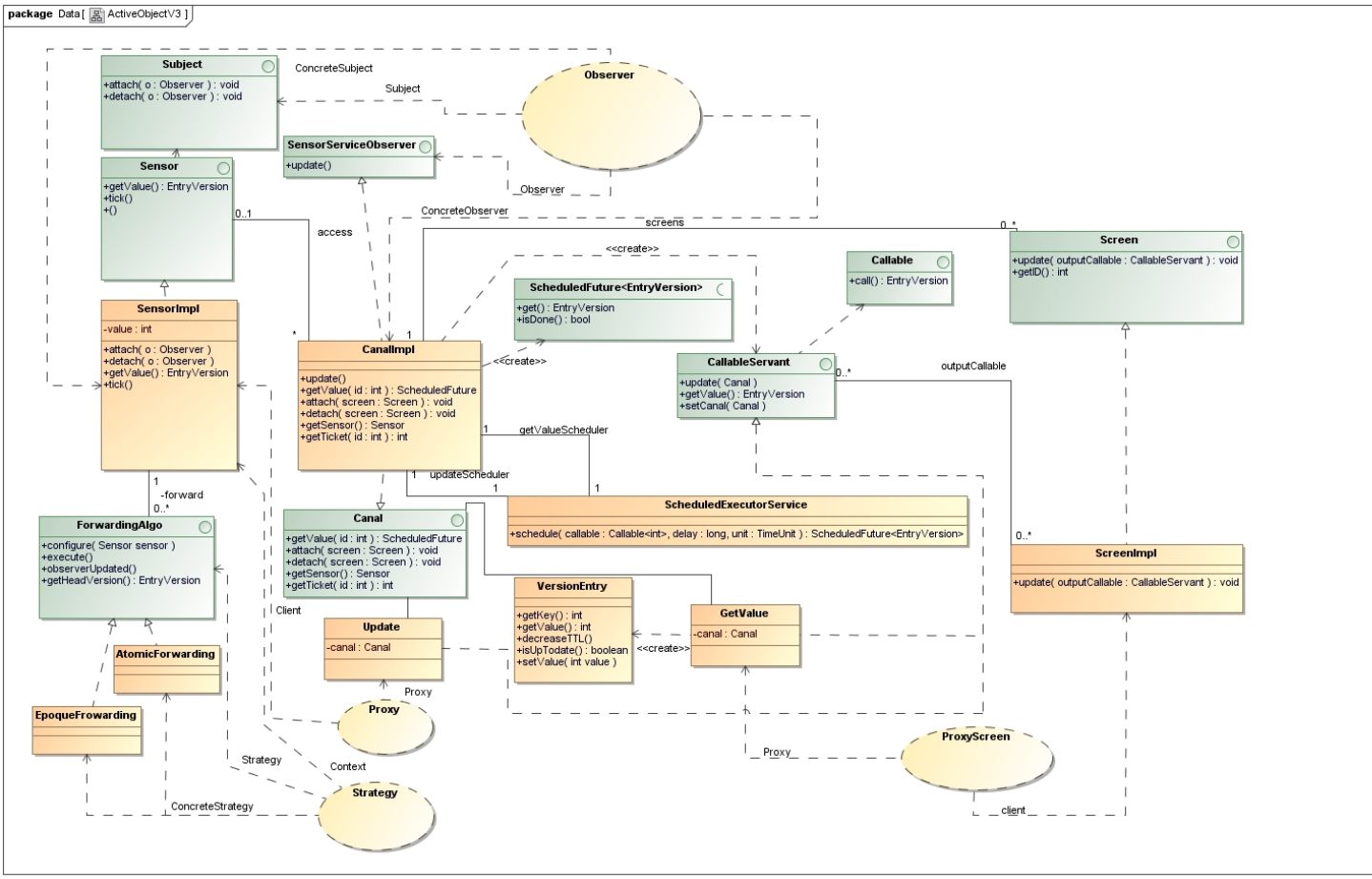


Diagramme de séquence V2 (scénario update d'un capteur):



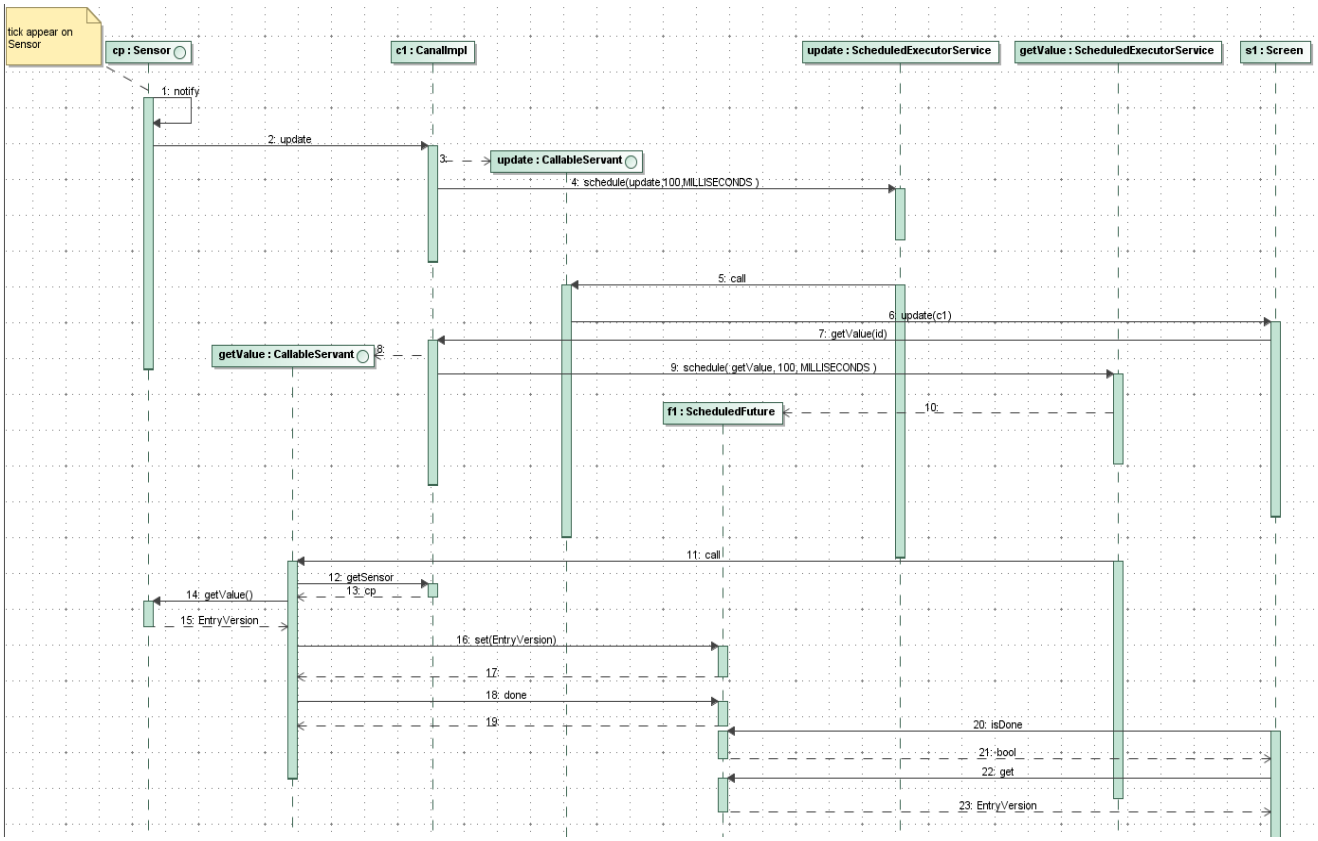
Ce scénario décrit l'update d'un écran dans la version 2 utilisant l'implantation standard d'Active Object. Il fait intervenir les instances suivantes :
-Sensor : implémentation de l'interface Sensor, contenant la valeur physique à mettre à jour sur l'afficheur.
-GetValue : implémentation de MethodInvocation, permettant de récupérer la grandeur physique sur le capteur et proxy du capteur.
-Canal : proxy du capteur permettant la communication entre le capteur et l'afficheur
-Update : implémentation de MethodInvocation, Proxy de l'afficheur.
-Future : ticket de Update, mis à jour avec la grandeur physique récupérée à afficher sur l'écran
-Scheduler : exécute périodiquement des demandes de mise à jour pour les afficheurs.
-ActivationQueue : enregistre les commandes à exécuter.
-Screen : implémentation de l'afficheur dans la solution.

Conception Analyse V3 (utilisant l'API standard de java):



Le diagramme des classes fait intervenir deux stratégies de diffusion : une en mode atomique et une autre en mode époque.
La stratégie atomique garantit que l'ensemble des afficheurs observant un capteur seront mis à jour avec une unique valeur avant qu'une autre notification ne soit émise.
La stratégie par époque diffuse une nouvelle valeur à chaque *tick* ayant une durée de vie équivalente au nombre d'afficheurs observant le capteur. Si un afficheur notifié demande une valeur alors que la queue de version est vide, le capteur renvoie un instantané de sa valeur non versionnée (le membre *key* de l'instance de membre initialisé à -1).

Diagramme de séquence V3 (scénario update d'un capteur):



Ce scénario décrit l'update d'un afficheur, dans la version 3 qui utilise l'implantation d'Active Object de la librairie standard de Java. Il fait intervenir les instances suivantes :

-*Sensor* : implémentation de l'interface *Sensor*, contenant la valeur physique à mettre à jour sur l'afficheur.

-*CallableServant* : interface étendant *Callable*, implémentée de façon anonyme pour les instances de *getValue* et *update*.

-*getValue* : récupère la grandeur physique sur le capteur et Proxy du capteur.

-*update* : proxy de l'afficheur.

-*ScheduledFuture* : ticket de *getValue*, mis à jour avec la grandeur physique récupérée à afficher sur l'écran.

-2 *ScheduledExecutorService* : qui exécutent périodiquement des demandes de mise à jour pour les afficheurs, un spécifique pour les *CallableServant getValue* et un autre pour *update*.

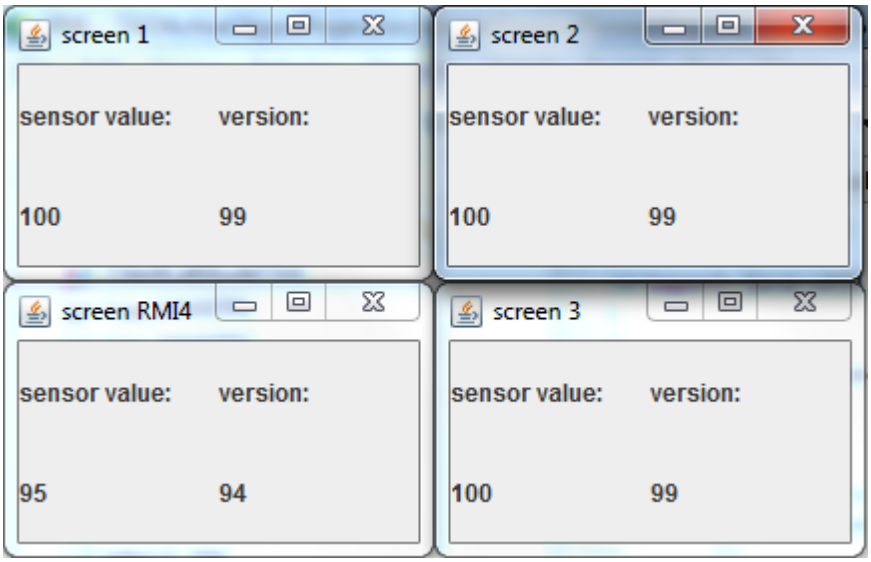
-Screen : implémentation de l'afficheur dans la solution.

Mise en oeuvre:

La version V3 est développée en Java en utilisant “Spring” pour la configuration de l’application. “Spring” est particulièrement commode pour modifier la configuration, passer d’une stratégie atomique à une stratégie par époque. De plus un des canaux observant le capteur joue le rôle de canal distant accessible via RMI (Remote Méthode Invocation), aussi “Spring” permet de masquer presque complètement l'implémentation de RMI.

Constatation:

La version de diffusion atomique attribut à chaque Observer un jeton dans une file. Tant que l’ensemble des jetons n’a pas été consommé aucune autre diffusion ne sera effectué.
Dans les fait sur 100 diffusions l’ensemble des abonnés au capteur ont obtenu l’ensemble des valeurs diffusées.
La version de diffusion par époque attribue un jeton pour l’ensemble des abonnés au capteur ; cependant elle peut commencer une nouvelle diffusion alors que l’ensemble des jetons n’a pas été consommrmé. Aussi un abonné peut très bien récupérer deux fois la même version et ainsi en priver un autre. Donc pour les afficheurs fonctionnant en local, la valeur la plus récente est partagée pour garantir une homogénéité entre les afficheurs. L’afficheur distant (R.M.I.) ne dispose pas d'accès avec les afficheurs locaux ; sa version de valeur est donc souvent à la fin de 100 diffusions plus ancienne que celle des afficheurs locaux.



Les configuration Spring disponible en fonctionnement local sont les suivantes:

TAOActiveObject, 100 notification avec une stratégie époque.
TAOActiveObject50, 50 notification avec une stratégie époque.
TAOActiveObject25, 25 notification avec une stratégie époque.
TAOActiveObjectAtomic50, 50 notification avec une stratégie atomique.
TAOActiveObjectAtomic25, 25 notification avec une stratégie atomique.

Chacune de c'est configuration (disponible dans *src*) va générer un capteur sur lequel un canal va s'abonner. Trois afficheurs y seront rattachés. Un canal supplémentaires va s'abonner également au capteur et sera accessible pour des afficheurs distant.

Un jar exécutable est disponible à la racine du projet "*tao.jar*". Il utilise la configuration **TAOActiveObject**.
Il est possible de lancé un afficheur distant en appelant le main contenu dans la classe *RMIChannelBridge* du package *client* .

Conclusion :

L'implémentation d'Active Object avec la librairie standard de Java permet une meilleure compréhension du comportement à l'exécution du patron. Il offre une solution pour gérer les problèmes de concurrence avec des architectures parallèles.