IN201 : 9 – Diagrammes d'états-transitions

Author : Christophe Garion <garion@isae.fr>
Public : SUPAERO 2A
Date :

Résumé

Ce sujet de TP est un sujet récapitulatif de conception. Il vous permettra également de construire un diagramme d'états-transitions.

1 Objectifs

Les objectifs du TP sont les suivants :
– construire un diagramme de classes à partir d'un « cahier des charges » ;
– construire un diagramme d'états-transitions pour modéliser le comportement d'un objet complexe.

2 Présentation du problème

L'objectif de cet exercice est de modéliser un système de navigation par satellites (comme le GPS ou Galileo) pour réaliser un logiciel de simulation. On ne s'attachera pas ici à décrire le cœur du simulateur, mais simplement à construire un modèle du système réel.
Les systèmes de navigation comme GPS ou Galileo utilisent une constellation de satellites. Une constellation est constituée d'un ensemble de satellites. On peut considérer qu'une constellation ne possède pas de satellites si ceux-ci ne sont pas encore lancés. Une constellation est caractérisée par sa taille, et on peut ajouter un satellite dans une constellation si celle-ci n'est pas encore complète.
Chaque satellite évolue sur une orbite. Il existe plusieurs types d'orbites différentes : par exemple, une orbite géostationnaire ou une orbite de Kepler sont deux orbites particulières. Une orbite de Kepler de type J2 est elle même une orbite de Kepler particulière.
Un satellite peut embarquer des capteurs comme par exemple un radar, un altimètre ou un capteur infra-rouge.
Un satellite est caractérisé par une position courante, une vitesse courante et une date courante. On peut demander à un satellite de renvoyer toutes les informations concernant sa position.
Sur la Terre, les satellites d'une constellation peuvent être vus par des stations sol. Celles-ci ont une position représentée par des coordonnées cartésiennes. Une station sol peut être par exemple un récepteur, qui peut lui même être un récepteur Galileo ou GPS.

3 Questions

1. proposer un modèle de classes UML de conception préliminaire (analyse) identifiant les classes, les rôles et les multiplicités (ou cardinalités) ;

2. proposer une description détaillée de la classe `Satellite` (attributs et opérations). Vous vous limiterez à un petit nombre d'opérations nécessaires à la réalisation de cette classe. Vous considérerez que vous avez à votre disposition une classe `Date` ;

3. nous nous intéressons maintenant à la création des capteurs embarqués sur un satellite. Nous ne considérerons ici que les capteurs `Radar` et `Altimetre`. On suppose que chaque capteur a besoin de trois éléments pour pouvoir être construit et embarqué sur le satellite :
   - un support physique pour mettre en place le capteur sur le satellite
   - les instruments constituant le capteur lui-même
   - une interface logicielle et matérielle permettant au capteur de dialoguer avec le système informatique du satellite

   On suppose que l'on dispose donc de la hiérarche présentée sur la figure 3. On suppose également que la classe `Capteur` a la structure présentée sur la figure 4. Quelques précisions :
   - la méthode `initComponents` est une méthode abstraite dont le rôle est d'initialiser le support, l'instrument et l'interface satellite du capteur.
   - la méthode `createCapteur` est une *factory method* permettant de créer un capteur à partir d'une chaîne de caractères. Cette méthode sera utilisée pour créer des capteurs.

On cherche à assurer la cohérence des composants de chaque capteur : par exemple, un capteur de type `Radar` ne pourra avoir utiliser que `SupportRadar`, `InstrumentRadar` et `InterfaceSatRadar`.
Le code de la classe `Capteur` est donné sur le listing 1.

Listing 1– La classe `Capteur`

```
public abstract class Capteur {

    protected Support support;
    protected Instrument instrument;
    protected InterfaceSat interfaceSat;

    public static Capteur createCapteur(String type) {
        Capteur c = null;

        if (type.equals("radar")) {
            c = new Radar();
        } else if (type.equals("altimetre")) {
            c = new Altimetre();
        }

        if (c != null) {
            c.initComponents();
        }

        return c;
    }

    public abstract void initComponents();

}
```

(a) écrire un diagramme de séquence représentant la création d'un capteur de type `Radar`.

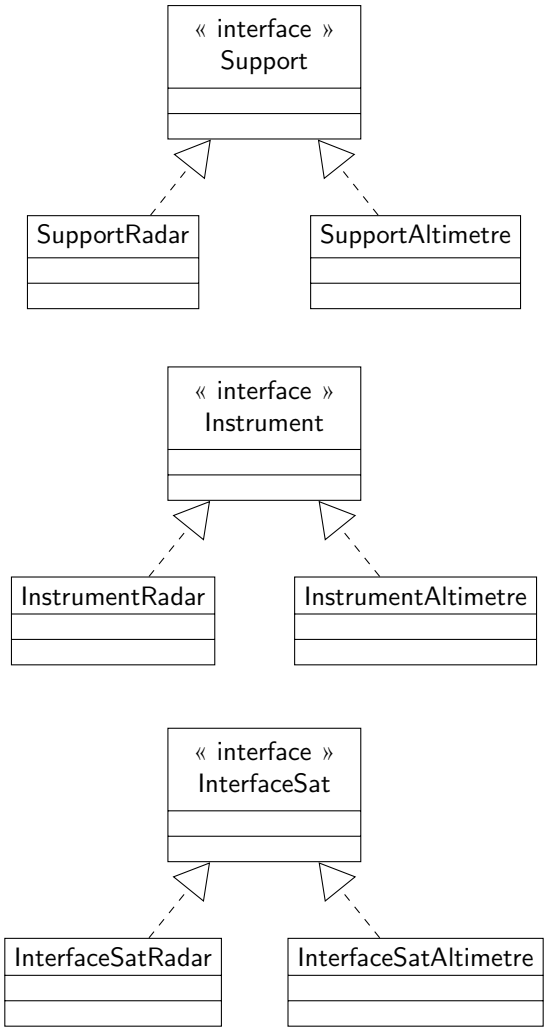

FIGURE 3 – Hiérarchie des composants des capteurs

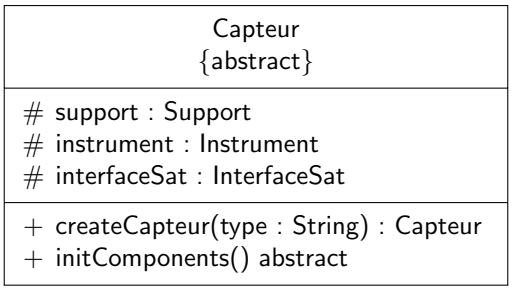

FIGURE 4 – La classe abstraite `Capteur`

(b) peut-on garantir qu'un capteur sera bien construit avec les composants avec cette architecture ?

(c) pour pouvoir construire une famille d'objets, on peut utiliser le patron de conception *abstract factory* présenté sur la figure 6. Adaptez le patron de conception au problème de la création de capteur. La méthode `initComponent` reste-t-elle abstraite ?

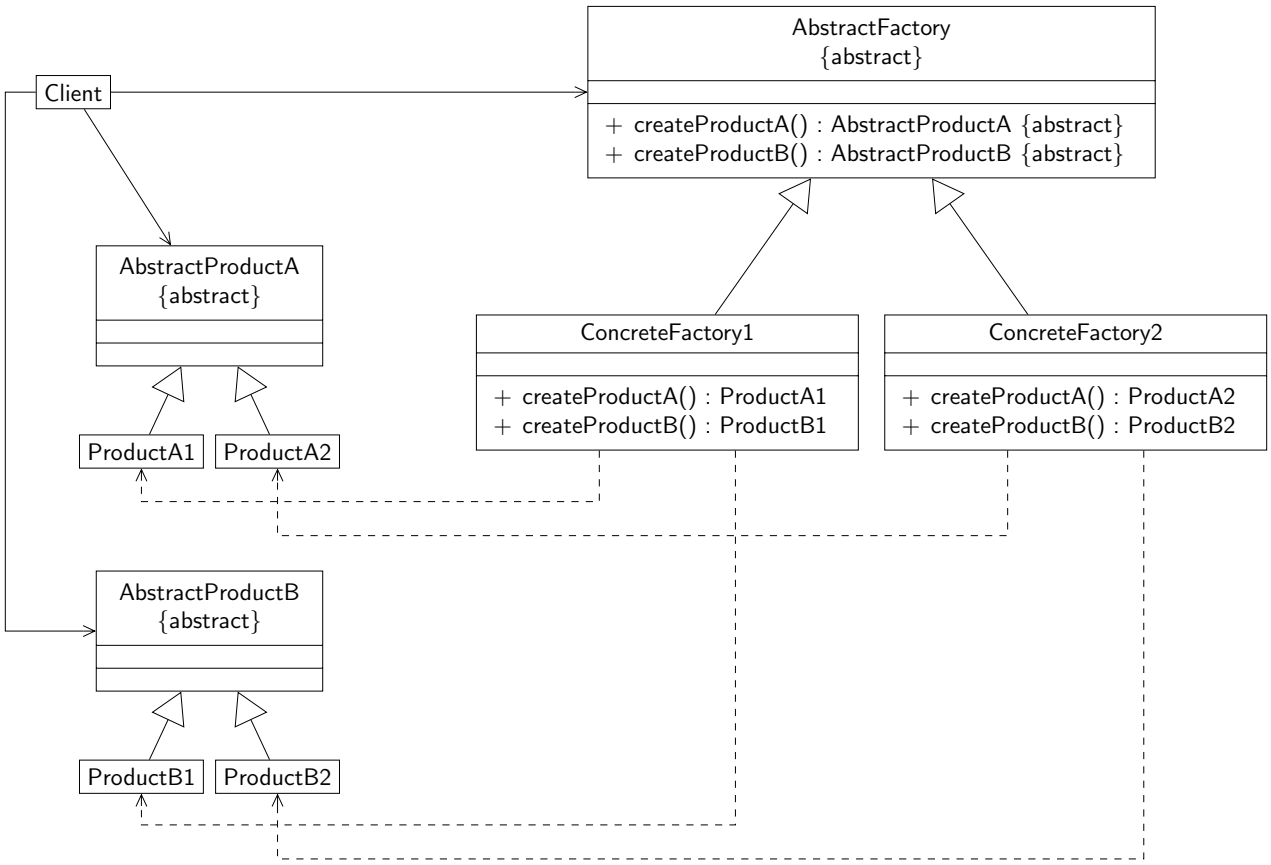

FIGURE 6 – La patron de conception *abstract factory*

4. nous proposons dans cette question de modéliser grâce à un diagramme d'états-transitions le comportement d'un récepteur de type Galileo[2].

   Lorsque l'on met en marche un récepteur Galileo, celui-ci cherche tout d'abord à savoir s'il dispose d'une première approximation de la date actuelle. S'il ne l'a pas, il entre dans un état et :
   - soit il dispose d'une interface réseau et récupère une approximation de la date via un serveur ntp (*Network Time Protocol*) par exemple ;
   - soit il attend que l'utilisateur lui fournisse une date en utilisant un clavier par exemple.

   Lorsque le récepteur dispose d'une approximation de la date actuelle, il entre dans un état dans lequel il cherche à orienter son antenne pour trouver un premier satellite en utilisant les éphémérides dont il dispose. Cette phase peut prendre un certain temps. Lorsque le récepteur a trouvé un satellite, il entre dans un nouvel état dans lequel il ajuste la date actuelle.

   À ce moment, dès que le récepteur va « voir » 4 satellites de la constellation, il va entrer dans un état `NavigationSolution` qui va lui permettre de calculer sa position et la date actuelle. Sinon, il entre dans un mode `NoNavigationSolution`.

2. Ce modèle de comportement est très simplifié par rapport à la réalité !

Ce mode est obligatoirement atteint dès que moins de 4 satellites sont visibles. Dans le mode `NoNavigationSolution`, si un certain laps de temps s'écoule (modélisé par une fonction f que l'on suppose connue) le récepteur va de nouveau rechercher un premier satellite. Si 4 satellites sont de nouveau visibles avant ce laps de temps, le récepteur est de nouveau capable de calculer sa position et la date actuelle.

On suppose également que le récepteur Galileo est couplé à une centrale inertielle. Celle-ci permet de connaître une position pendant un certain temps. Si la position donnée par la centrale est valide, la condition `INS` (pour *Inertial Navigation System*) est vraie. Lorsque le récepteur est dans l'état `NoNavigationSolution`, la position donnée par la centrale peut être utilisée par l'utilisateur pour connaître sa position.

De plus, si la durée f s'est écoulée depuis l'entrée du récepteur dans l'état `NoNavigationSolution` et que la centrale inertielle est valide, le récepteur entre dans une une phase de recalibrage utilisant la centrale inertielle plutôt que de retourner directement dans l'état de recherche du premier satellite. Au bout d'une unité de temps, si 4 satellites sont vus, le récepteur devient alors opérationnel, sinon le récepteur recherche à nouveau un premier satellite. Cela permet de gagner éventuellement du temps par rapport à la phase d'initialisation.

Donner le diagramme d'états-transitions correspondant au fonctionnement du récepteur.