Patrons de Conception
-
Observer

Simon Urli
urli@i3s.unice.fr

Master 1 MIAGE
2014-2015

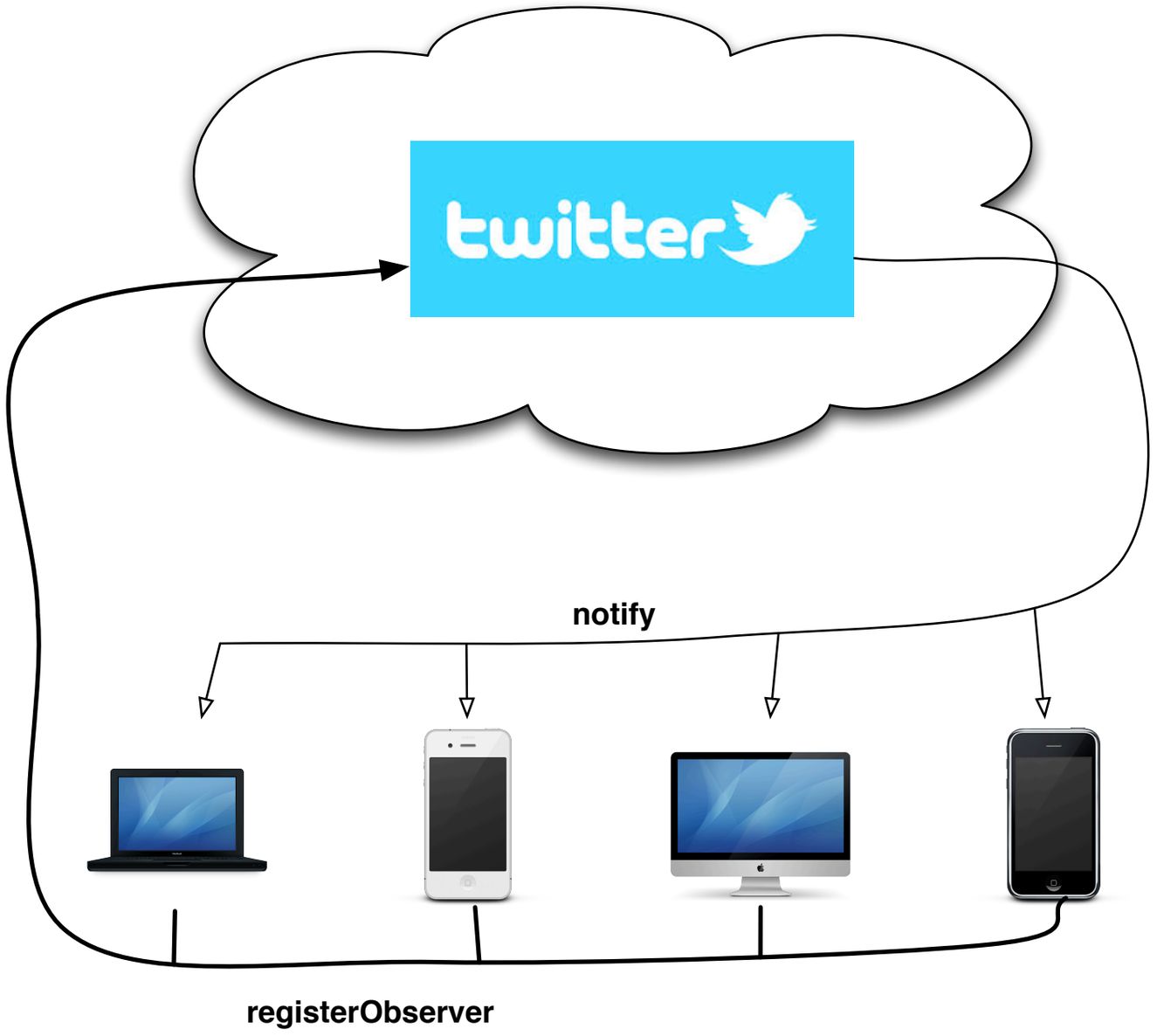
Objectif

“ Définir une dépendance un-à-plusieurs (1-N) entre des objets de telle façon que si un objet change d’état tous les objets dépendants en soient notifiés et puissent se mettre à jour. ”

Classification : patron de comportement

Synonyme : publish/subscribe

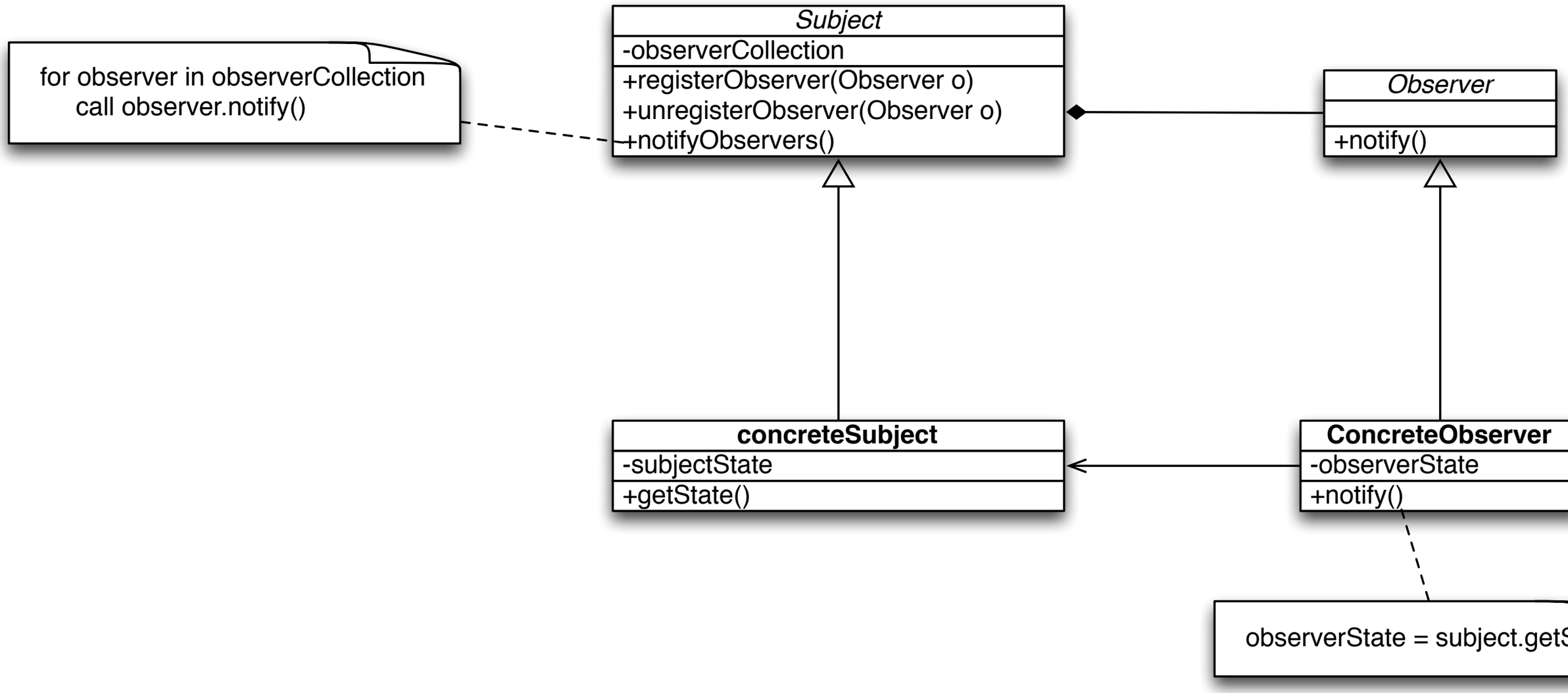
Exemple



Applications

- Quand une abstraction possède deux aspects inter-dépendants. Encapsuler ces aspects dans des objets séparés permet de les faire varier et de les réutiliser indépendamment.
- Quand un changement sur un objet nécessite de modifier les autres et qu'on ne peut pas savoir à l'avance combien d'objets doivent être modifiés.
- Quand un objet doit être capable de notifier d'autres objets sans faire de suppositions sur qui sont ces objets : les objets ne doivent donc pas être fortement couplés.

Structure


for observer in observerCollection
call observer.notify()
Subject
-observerCollection
+registerObserver(Observer o)
+unregisterObserver(Observer o)
+notifyObservers()
Observer
+notify()
concreteSubject
-subjectState
+getState()
ConcreteObserver
-observerState
+notify()
observerState = subject.getState()

Implémentation

- Le sujet a une référence sur les observateurs : Collections (de préférence hashtable si beaucoup d'observateurs)
- Un observateur peut observer plusieurs sujets : étendre la méthode notify() pour savoir qui le notifie.
- Attention aux problèmes de synchronisation lors de l'ajout ou de la suppression d'un observateur.

Exemple 1/2

```
public class ConcreteTwitterEngine implements ITwitterEngine {
  private Map<String, TwitterClient> mapObserver;
  private List<String> tweets;
  public ConcreteTwitterEngine() {
       this.mapObserver = new HashMap<String,TwitterClient>();
       this.tweets = new ArrayList<String>();
  }
  public void addTweet(String tweet) {
       this.tweets.add(tweet);
       this.notify(tweet);
  }
  public void notify(String tweet) {
       for (TwitterClient tc : this.mapObserver.values()) {
           tc.notify(tweet);
       }
  }
  public void addTwitterClient(TwitterClient tc) {
       this.mapObserver.put(tc.getIdentifier(), tc);
  }
}
```

Exemple 2/2

```
public abstract class TwitterClient {
     private ITwitterEngine moteur;

     public TwitterClient(ITwitterEngine moteur) {
        this.moteur = moteur;
        this.moteur.addTwitterClient(this);
     }
     public void sendTweet(Tweet t) {
        this.moteur.addTweet(t);
     }
     public void notify(Tweet t) {
        this.displayNotification(t);
     }
     public abstract String getIdentifier();
     public abstract void displayNotification(Tweet t);
}
```

Où est-il utilisé ?

- Patron d'architecture MVC : communication entre les différents éléments.
- Bibliothèque Swing de Java : EventListener

Exercice

- Modéliser une installation électrique : une centrale délivre du courant, des équipements peuvent se brancher et recevoir de l'énergie. Lorsque la centrale s'arrête les équipements ne reçoivent plus d'énergie. Si un équipement provoque une surtension la centrale doit s'arrêter.