OBSERVABLE: THE OBSERVER DESIGN PATTERN REVISITED

Ad van Gerven

Alten Nederland BV

Abstract

The Observer software design pattern (Gamma, Helm et a., 1995) suggests a tight coupling between the application problem domain and the pattern. Since this is often inconvenient, an adaptation of the pattern has been developed that minimizes this coupling and has been used for some years now with success.

Introduction

The Observer software design pattern (Gamma, Helm et a., 1995) suggests a coupling between the application problem domain and the Observer design pattern. Since this is often inconvenient, an adaptation of the pattern has been developed that minimizes this coupling and has been used for some years now with success.

The Observer software design pattern

The Observer software design pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically (Gamma, Helm et al., 1995, Object Behavioral).

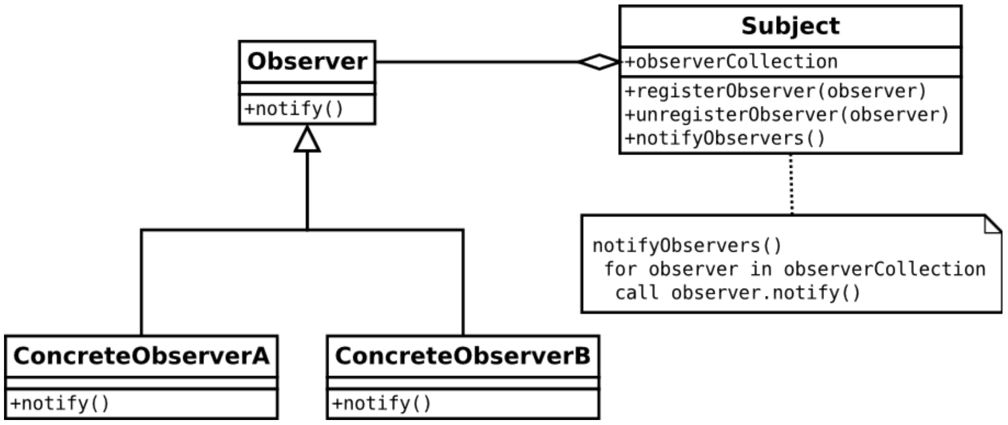

Figure 1 Class diagram for the Observer design pattern

The Observable software design pattern

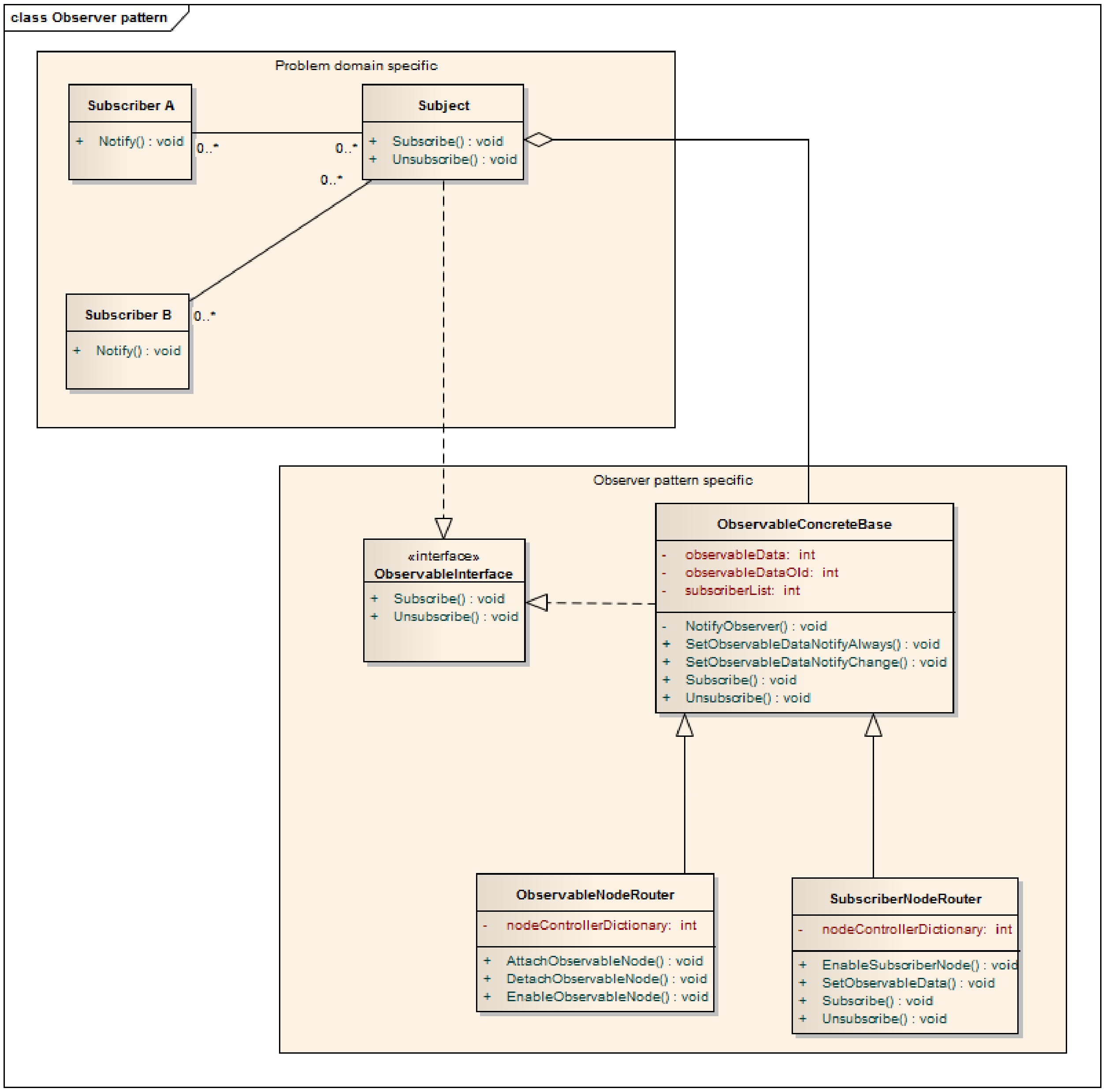

Figure 2 The observable design pattern

There are some obvious, (but hardly relevant) differences between the views of Figure 1 and Figure 2:

- Instead of ConcreteObserverA and ConcreteObserverB instances, the adapted pattern has ‘Subscriber A’ and ‘Subscriber B’ instances. These are different names for identical roles;
- The adapted Subject class in Figure 2 uses different names for similar methods:
  Subscribe is synonymous with registerObserver;
  Unsubscribe is synonymous with unregisterObserver

The significant (but less obvious) differences between the two views are:

- Concrete observers in the original pattern must be specializations of an Observer base class. In the adapted pattern any problem domain instance can subscribe to observable data as long as it provides a notification callback function for that data;
- In the original pattern the Subject instance completely implements the responsibilities associated with providing observable access to the data. In the adapted pattern any problem domain instance can provide observable data as long as it implements the ObservableInterface and has an aggregate instance of the ObservableConcreteBase class, which completely implements the responsibilities for the observer pattern. In most cases implementation of the ObservableInterface by the Subject class simply is a dispatching of the functionality to the ObservableConctreteBase instance;
- In the original pattern view, Figure 1, the Subject class must provide a public notifyObservers method to initiate the observer pattern activity. In the adapted pattern this is not needed at all. The Subject instance can privately invoke the SetObservableData method of its aggregate ObservableConcreteBase class instance;

Note the ObservableNodeRouter and SubscriberNodeRouter classes in Figure 2 are not parts of the original Observer pattern and not required in most cases. These classes are specializations that allow N-to-M observing relationships that can individually be enabled / disabled during run-time, as opposed to the one-to-many dependency in the original pattern. Further details of these specializations are out of scope for this document.

Without doubt there will be cases in which the original pattern or other alternatives are better suited than the alternative described here. In our case however the adapted pattern of Figure 2 has supported and endured many different designs in which a decoupling of data generators and data viewers is needed. In addition we found that extending existing implementations with observer functionality through this pattern was easy because of the decoupling between problem domain and the observer pattern.

References

Gamma, Helm, Johnson & Vlissides (1995). *Design Patterns*. Addison-Wesley, ISBN 0201633612