Bit Listener

Pieter van den Hombergh

Fontys Hogeschool voor Techniek en Bedrijfsmanagement
Software Engineering

September 24, 2009

Fontys Hogescholen

Independence of the application

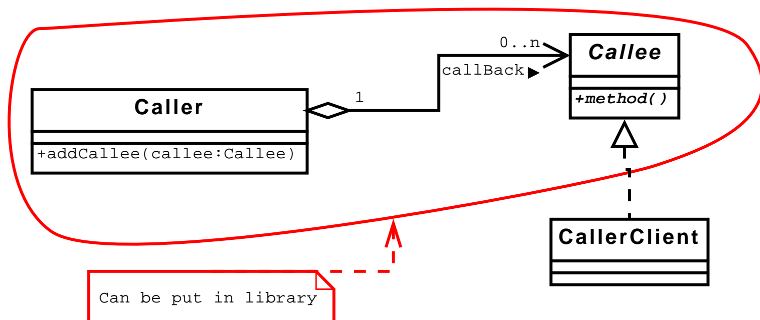
Output is rather straightforward.

Input requires that you efficiently read the relevant io words and distribute the detected changes amongst the interested classes. But how do you inform (call a method) on a application that you don't know at the time of the development of your IO library.

The solution lies in call-back functions after the famous Hollywood saying:
"Do not call us; w'll call you..."
That works wonders to keep *would be* actors off your line; In our case: if you know or can prescribe how you would call a client, then you are out of the mess.

Fontys Hogescholen

Hollywood in a class diagram

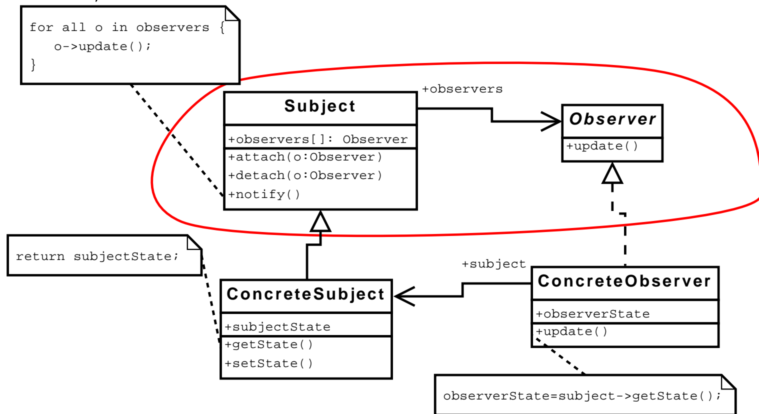


The stuff in the red group can be put in a library. Thereby the library is independent of the client and still usable to inform the client application of interesting things like events.

Fontys Hogescholen

Fancy name for Hollywood principle : Observer

The official "Observer" pattern has a two parties: the `Observer` and the `Observable` or `Subject`. Also known as *Publish/Subscribe*.



Division of responsibility

Very often, as in our elevator case, a number of binary inputs should be watched. This works well if we interogate or poll the aggregate word for changes. For each changed bit in the word we inform the relevant Bit instance. This can be arranged after the Observer pattern.

The word gets the responsibility to monitor "it's" changes.

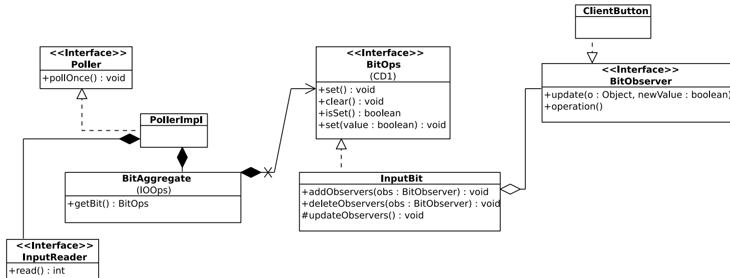
The bit itself should be defined with it's `setTo(newValue)` in such a way that it in turn informs its observers.

In our case, we inform the observer with the new value of the relevant bit upon changes only, so that it can react to true to false and false to true transitions.

Fontys
Hogescholen

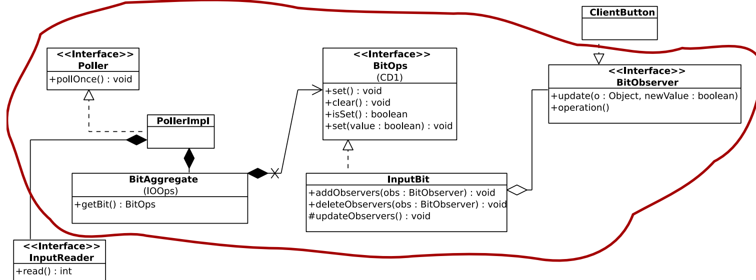
Class diagram

The IOBit could be implemented in this way. In the elevator project your milage may vary...



Class diagram

The IOBit could be implemented in this way. In the elevator project your milage may vary...



The red encircled part is library, ClientButton is application specific.

Class definition detail of InputBit and Byte

The `pollOnce()` method is the essential driver of this event chain. It should be called frequently, for instance once every 10 milliseconds.

```
class PollerImpl {

    private BitAggregate ba;
    private InputReader input;
    private int oldValue;
    [...]
    public void pollOnce(){

        int bitIndex=0;
        int mask = 1;
        int newValue = input.read();
        int diff = newValue ^ oldValue; // 0 if same
        while ( bitIndex < ba.size() && diff !=0 ) {
            if ((diff & mask) !=0) {
                ba.getBit(bitIndex)->set((newValue & mask)!=0);
            }
            diff &= ~mask; // kick diff bit
            mask <<=1;
            bitIndex++;
        }
        oldValue = newValue; // remember what we saw last.
    }
}
```

The implementation of BitSubject etc. is left as an exercise.

Unit testing works with embedded almost just as well.

Choosing hardware does not prevent the use of unit testing. On the contrary: If you have hardware you can test just as well. If you have a proper design, you could replace the hardware layer and the associated layer classes with simulations and test the rest if the application (logic, behavior). Of course, in the end you shoudl test with real hardware.
So I expect you to do proper testing in the hardware projects for the essential components (io, scheduler, state etc).

Designing for Portability helps testability

It is very common in the embedded computer world, that not all software developers have a copy of the IO hardware available. In the industry this has to do with mundane facts such as:

- The hardware is only available in prototype form and therefor unique of very expensive.
- The hardware is bulky.
- The hardware is designed to work in a hazardous or otherwise unfriendly environment.

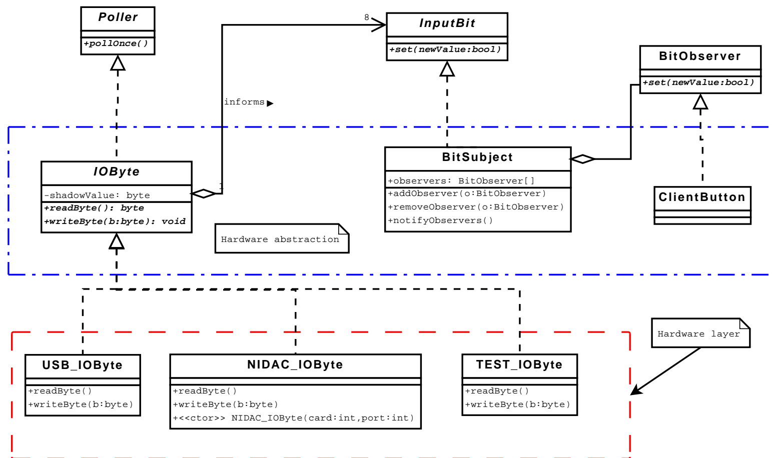
It therefor makes sense to be able to test most of the software without using the real hardware.
If you can make (most of your) software working on a workstation, the developer can do his work and testing on that workstation. This is actually an exercise in protability.

Fontys
Hogescholen

Portable extension of BitGroup

You can apply a simple extension to the current classes to make them portable. You could even call it a layered design. The hardware is abstracted away in to the hardware abstraction layer.

Using a abstract hardware layer



Fontys Hogescholen