

Dr. Michael Eichberg

Software Engineering

Department of Computer Science

Technische Universität Darmstadt

Software Engineering

The Observer Design Pattern

For details see Gamma et al. in "Design Patterns"



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Observer Design Pattern

Intent

Define a one-to-many dependency between objects so that when an object changes its state, all its dependents are notified and updated automatically.



TECHNISCHE
UNIVERSITÄT
DARMSTADT

The Observer Design Pattern

Alternative Implementation using AspectJ

The screenshot shows a web browser window with the title "Design pattern implementation in Java and aspectj". The browser's address bar shows the URL "http://www.cs.ubc.ca/~...". The page features the ACM Portal logo and navigation links such as "Subscribe (Full Service)", "Register (Limited Service, Free)", and "Login". A search bar is present with radio buttons for "The ACM Digital Library" and "The Guide". The main content area displays the article title "Design pattern implementation in Java and aspectj" and provides metadata including the source (ACM SIGPLAN conference), authors (Jan Hannemann and Gregor Kiczales), and sponsors (ACM and SIGPLAN). At the bottom, there are links for "Additional Information" (abstract, references, cited by, index terms, collaborative colleagues, peer to peer) and "Tools and Actions" (Find similar Articles, Review this Article).

Design pattern implementation in Java and aspectj

Patterns AspectJ - Goo... pattern implementation... http://www.cs.ubc.ca/~... iCalamus.net :: iCalamu... D120.de/forum • Them...

acm **PORTAL** HeBIS

Subscribe (Full Service) Register (Limited Service, Free) Login

Search: The ACM Digital Library The Guide

SEARCH

THE GUIDE TO COMPUTING LITERATURE

Feedback Report a problem Satisfaction survey

Design pattern implementation in Java and aspectj

Full text Pdf (367 KB)

Source Conference on Object Oriented Programming Systems Languages and Applications [archive](#)
Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications [table of contents](#)
Seattle, Washington, USA
SESSION: Aspects [table of contents](#)
Pages: 161 - 173
Year of Publication: 2002
ISSN:0362-1340
[Also published in ...](#)

Authors [Jan Hannemann](#) University of British Columbia, Vancouver B.C. V6T 1Z4
[Gregor Kiczales](#) University of British Columbia, Vancouver B.C. V6T 1Z4

Sponsors [ACM](#): Association for Computing Machinery
[SIGPLAN](#): ACM Special Interest Group on Programming Languages

Publisher [ACM](#) New York, NY, USA

Additional Information: [abstract](#) [references](#) [cited by](#) [index terms](#) [collaborative colleagues](#) [peer to peer](#)

Tools and Actions: [Find similar Articles](#) [Review this Article](#)

The Observer Design Pattern

Alternative Implementation using AspectJ

- We want to...
 - **avoid the decision between Push or Pull mode observers**
 - **better support observers interested only in specific events**



The Observer Design Pattern

Alternative Implementation using AspectJ

Parts Common to Potential Instantiations of the Pattern

1. The existence of **Subject** and **Observer** roles
(i.e. the fact that some classes act as Observers and some as Subjects)
2. Maintenance of a mapping from **Subjects** to **Observers**
3. The general update logic: **Subject** changes trigger Observer updates

Will be implemented in a reusable ObserverProtocol aspect.

Parts Specific to Each Instantiation of the Pattern

4. Which classes can be **Subjects** and which can be **Observers**
5. A set of changes of interest on the **Subjects** that trigger updates on the **Observers**
6. The specific means of updating each kind of **Observer** when the update logic requires it

The Observer Design Pattern

Alternative Implementation using AspectJ

```
public abstract aspect ObserverProtocol {  
  
    // Realization of the Roles of the Observer Design Pattern  
    protected interface Subject { }  
    protected interface Observer { }  
  
    ...  
}
```

The part
common to
instantiations
of the pattern.

The Observer Design Pattern

Alternative Implementation using AspectJ

```
public abstract aspect ObserverProtocol {  
    ...  
    // Mapping and Managing Subjects and Observers  
    private WeakHashMap<Subject, List<Observer>> perSubjectObservers;  
    protected List<Observer> getObservers(Subject s) {  
        if (perSubjectObservers == null)  
            perSubjectObservers = new WeakHashMap<Subject, List<Observer>>();  
        List<Observer> observers = perSubjectObservers.get(s);  
        if (observers == null) {  
            observers = new LinkedList<Observer>();  
            perSubjectObservers.put(s, observers);  
        }  
        return observers;  
    }  
    public void addObserver(Subject s, Observer o){  
        getObservers(s).add(o);  
    }  
    public void removeObserver(Subject s, Observer o){  
        getObservers(s).remove(o);  
    }  
    ...  
}
```

The part
common to
instantiations
of the pattern.

The Observer Design Pattern

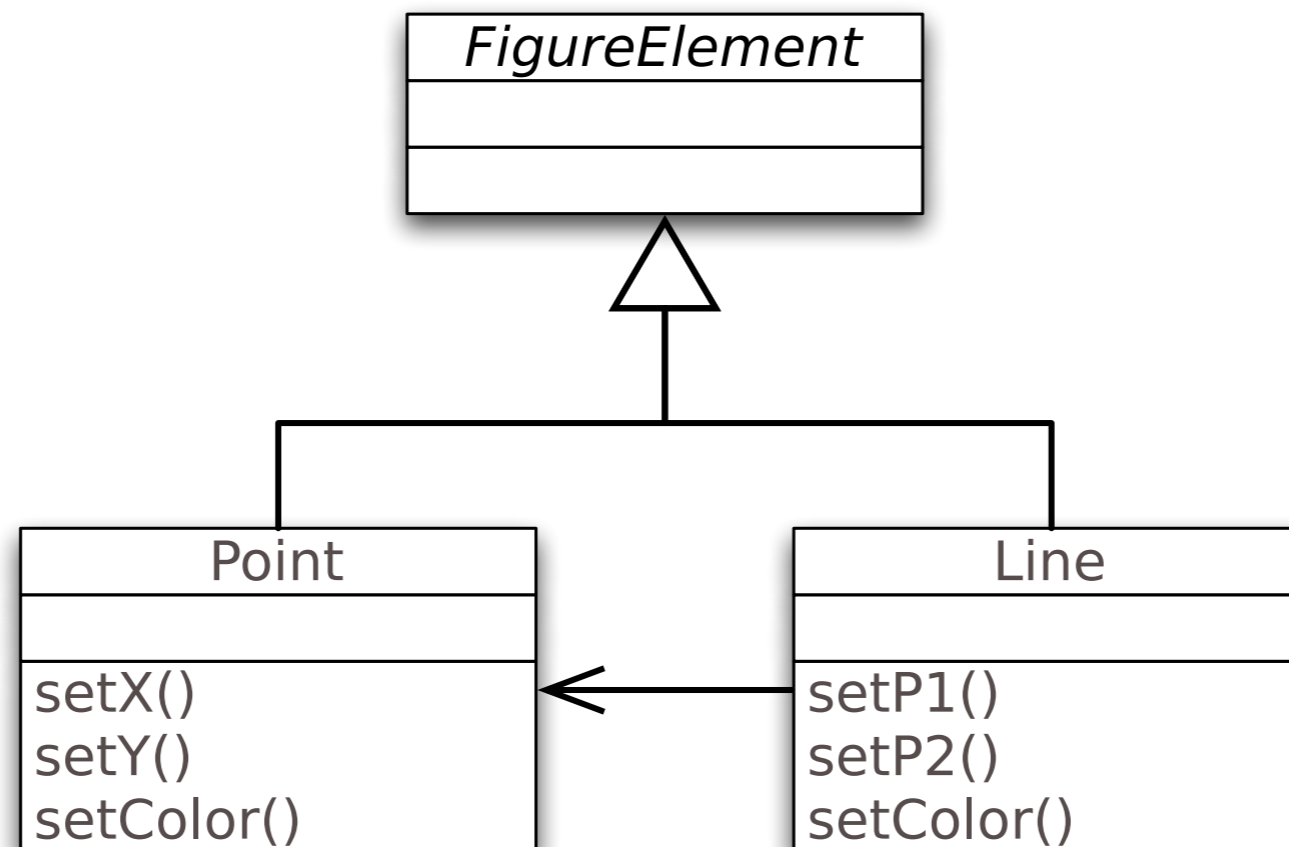
Alternative Implementation using AspectJ

```
public abstract aspect ObserverProtocol {  
    ...  
    // Notification related functionality  
    abstract protected pointcut subjectChange(Subject s);  
  
    abstract protected void updateObserver(Subject s, Observer o);  
  
    after(Subject s): subjectChange(s) {  
        Iterator<Observer> iter = getObservers(s).iterator();  
        while ( iter.hasNext() ) {  
            updateObserver(s, iter.next());  
        }  
    }  
}
```

The part
common to
instantiations
of the pattern.

The Observer Design Pattern

Alternative Implementation using AspectJ - Example



The Observer Design Pattern

Alternative Implementation using AspectJ - Example

Task: Observe Changes of the Color

```
public aspect ColorObserver extends ObserverProtocol {  
  
    declare parents: Point implements Subject;  
    declare parents: Line implements Subject;  
    declare parents: Screen implements Observer;  
  
    protected pointcut subjectChange(Subject s):  
        (call(void Point.setColor(Color)) ||  
         call(void Line.setColor(Color)) ) && target(s);  
  
    protected void updateObserver(Subject s, Observer o) {  
        ((Screen)o).display("Color change.");  
    }  
}
```

To create a mapping between an Observer and a Subject:

```
ColorObserver.aspectOf().addObserver(P, S);
```

The Observer Design Pattern

Alternative Implementation using AspectJ - Assessment

- **Locality**

All code that implements the Observer pattern is in the abstract and concrete observer aspects, none of it is in the participant classes; there is no coupling between the participants.

Potential changes to each Observer pattern instance are confined to one place.

- **Reusability**

The core pattern code is abstracted and reusable. The implementation of ObserverProtocol is generalizing the overall pattern behavior. The abstract aspect can be reused and shared across multiple Observer pattern instances.

The Observer Design Pattern

Alternative Implementation using AspectJ - Assessment

- **Composition transparency**

Because a pattern participant's implementation is not coupled to the pattern, if a Subject or Observer takes part in multiple observing relationships their code does not become more complicated and the pattern instances are not confused. Each instance of the pattern can be reasoned about independently.

- **(Un)pluggability**

It is possible to switch between using a pattern and not using it in the system.

Observer Design Pattern

How it is implemented depends on the available programming language mechanisms; the consequences may also change!