

## Research Topics in Software Quality

### Assignment 2: Detecting (Anti-)Patterns

Observer

---

Krista Puķe

Professor: Prof. Coen De Roover

Assistant: Quentin Stievenart

## Discussion Point 1: Detecting the prototypical implementation of a design pattern

In this section I briefly characterize the observer pattern, its members and relations. Furthermore I describe how I construct the *Ekeko* queries to detect the observer pattern.

### Observer pattern

The observer pattern define one-to-many dependency between the objects and if the object state changes then all its dependents automatically are notified and updated.

The observer pattern consists of four parts: *observer*, *subject*, *concrete subject* and *concrete observer*. The *observer* and the *subject* can only be interface. (See Fig.1)

Each of the subject can have many observers. The observer interface must include one method inside that updates (or refreshes) the observer when the subject's state changes. Also the subject interface must include methods to add, remove and notify the observer. The method invocation to add() and to remove() attaches and respectively removes the observer from the subject. The method notify alerts all the observers that a change has occurred in the subject state. (See Fig.1)

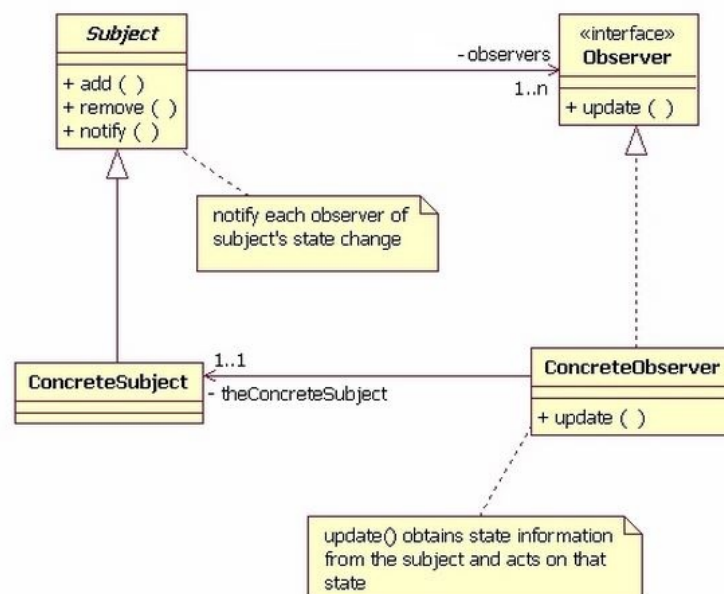


Figure.1. Observer pattern class diagram

### Ekeko: observer pattern detection in DesignPatterns project

To begin building the *Ekeko* query that detects the observer pattern, first of all I set up a number of small queries to verify my assumptions how to detect the pattern.

The first query finds all the possible subject classes and their observer interface. I started with the selection of all the methods that as the parameter use an interface (*the possible observer interface*) and has methods "add" invocation. Further I build the second query which as an interface take the possible observer interface from first query, and are looking for all the methods that uses parameter with the type interface *the possible subject interface*. After having the subject interface further I looking for the concrete observer class. To find the class I search for all the methods which uses as parameter previously found subject interface.

Using the previously created queries I started to build the *Ekeko* query to detect the design pattern *Observer*. First of all, the same as in the previously created small queries, I select all the methods (*?method*) that contains the condition to have the parameter (*?parameter*) which node type is an interface ( *11th line in the figure (type—interface ?nodeType)*). (See Fig.2)

```

1 | // Search for all method declarations
  | (ast :MethodDeclaration ?method)
3 | // Methods children - parameter
  | (child :parameters ?method ?parameter)
5 | // The parameter node type must be an interface
  | // Next line find parameter type
7 | (has :type ?parameter ?parameterType)
  | // Find node parameter type
9 | (ast|type-type ?parameterType ?nodeType)
  | // Nodes parameter type must be interface
11| (type|interface ?nodeType)

```

**Figure.2.** *Ekeko query: All methods with parameter type interface*

With the next figure I show how I used the *Ekeko ast-parent* to search for the method (*?method*) parent class (*the possible concrete subject*). (See Fig.3)

```

1 | (ast-parent ?method ?ConcreteSubjectClass)

```

**Figure.3.** *Ekeko query: Methods class*

Next I verify - if the method children (*?methChildren*) is the invocation of the method "add". I chose to search for the method "add", because to use the observer pattern the concrete subject class needs to add the object to the list of observers. (See Fig.4)

Also I could search for the method "remove", but I was not sure if in all of the situations this method would be used in the code base, therefore I assume that the method "add" will be used to implement the observer pattern.

```

1 | // Search for the method children
2 | (child+ ?method ?methChildren)
3 | // Methods children must be the methods "add" invocation
4 | (methodinvocation|named ?methChildren "add")

```

**Figure.4.** Ekeko query: Methods children is the invocation of the method "add"

In the next figure I am continuing adding parameters to my query. The second line of the code in the figure 5 shows that I use the previously found methods parameter type (*?nodeType*) to detect its node (Observer interface name). (See Fig.5)

```

1 | //Find the observer Interface (node) name
2 | (typeddeclaration-type ?ObserverInterface ?nodeType)

```

**Figure.5.** Ekeko query: Interface node name

Till now the query selects the concrete subject class and the observer interface. Next I am searching for the observer interface body declaration to find its methods and I select all parameters used in these methods. Then I am using previously found parameters to verify if its node type is interface. Following the steps shown in figure I can find the subject interface nodes type out of the observer interface class. (See Fig.6)

```

1 | //Use the ?ObserverInterface interface to detect its body declaration
2 | (child :bodyDeclarations ?ObserverInterface ?observerClassMethods)
3 | //Select the parameters used in the observer interface
4 | (child :parameters ?observerClassMethods ?observerMethodParameters)
5 | // Select parameter type
6 | (has :type ?observerMethodParameters ?observerParameterType)
7 | // Select parameter nodes type
8 | (ast|type-type ?observerParameterType ?SubjectNodesType)
9 | // Condition : the node type is interface
10 | (type|interface ?SubjectNodesType)

```

**Figure.6.** Ekeko query: Select the body declaration and the parameters

In the figure below I show how to find the node (*?SubjectInterface*) using its type declaration. I am using predicate *typeddeclaration-type* that keep relation between node and the type it refers to, so I am giving to the predicate the nodes type I am getting the node. (See Fig.7)

```

1 | // Find class for subject interface
2 | (typeddeclaration-type ?SubjectInterface ?SubjectNodesType)

```

**Figure.7.** Ekeko query: Interface node name

In order to verify if the observer interface class is not the same as the subject interface class I set up the following condition - if the subject interface class is the same as the observer interface class then the condition fails.(see Fig.8)

```
1 | // The observer interface and subject interface can not be the same
2 | (fails (equals ?SubjectInterface ?ObserverInterface))
```

**Figure.8.** Ekeko query: Subject interface can not be the same as observer interface

Having the subject interface class next I can search for the concrete observer class. To search for the concrete observer class I use the predicate *ConcreteObserver—SubjectInterfaceName*. As an input to the predicate I give the subject interface class and I expect to get the concrete observer class (*?ConcreteObserver*). (See Fig.9)

```
1 | (ConcreteObserver|SubjectInterfaceName ?ConcreteObserver ?string)
```

**Figure.9.** Ekeko query: Use of the *ConcreteObserver—SubjectInterfaceName*

Next I will in details explain the predicate *ConcreteObserver—SubjectInterfaceName*. I build this predicate to search for the concrete observer class. In this predicate firstly I am searching for all the methods (*?method*), methods parameters (*?parameter*) and for the parameter type (*?parameterType*). Then I use the parameter type to select its node type (*?nodeType*) and next I verify if the node type is an type of the interface. (See Fig.10)

```
1 | // Select all the declared methods
2 | (ast :MethodDeclaration ?method)
3 | // Select methods parameters
4 | (child :parameters ?method ?parameter)
5 | // Select methods parameter type
6 | (has :type ?parameter ?parameterType)
7 | // Select node type
8 | (ast|type-type ?parameterType ?nodeType)
9 | // Node type must be an interface
10 | (type|interface ?nodeType)
```

**Figure.10.** Ekeko query: Methods and its parameters

The the figure below shows the condition that the methods parent needs to be of the type class. First I use the method to select its node (*?methodClass*) and then I find the nodes type and set the condition that the type must be a class. (See Fig.11)

```
1 | //Find methods parent class
2 | (ast-parent ?method ?methodClass)
```

```

3 | //Next find the class type
4 | (typedeclaration-type ?methodClass ?classType)
5 | //verify that the type must be the type of the class
6 | (type|class ?classType)

```

**Figure.11.** Ekeko query: Method parameter type - class

At the end of this predicate I find the concrete observer class by using predicate that based on the nodes type finds the node. And this node is the concrete observer. (See Fig.12)

```

1 | // Selects parameters ?nameOfSubjectInterface name
2 | (typedeclaration-type ?parameterClass ?nodeType)

```

**Figure.12.** Ekeko query: Parameters name must be the same as subject interface

## Ekeko query results

To this end, I defined the two predicates: *ConcreteObserver—SubjectInterfaceName* and *DesignPattern*. As previously discussed the *ConcreteObserver—SubjectInterfaceName* predicate is used in the *DesignPattern* predicate.

To detect the design pattern observer in the *Ekeko* I build the query as it is shown in the figure below.

```









1 | (ekeko* [?ConcreteSubject ?ObserverInterface
2 |         ?SubjectInterface ?ConcreteObserver]
3 |         (DesignPattern ?ConcreteSubject ?ObserverInterface
4 |         ?SubjectInterface ?ConcreteObserver))

```

**Figure.13.** The *DesignPattern* predicate call

Query results demonstrate that the project *DesignPatterns* use the design pattern Observer. The result show that the *Ekeko* query detects the subject interface *ChangeSubject* and the observer interface *ChangeObserver*. The query also detects two concrete subject classes *Point* and *Screen*, that attaches the observer interface. As well as query detects the concrete observer class - *Screen*. The figure 14 shows that in the project *DessignPatterns* the concrete observer and the concrete subject can also be the same class.

In this concrete coda base the results do not contains any false positive or false negative.

?SubjectInterface	?ObserverInterface	?ConcreteSubject	?ConcreteObserver
 ChangeSubject	 ChangeObserver	 Point	 Screen
 ChangeSubject	 ChangeObserver	 Screen	 Screen

**Figure.14.** The results of the *Ekeko* query

## Discussion Point 2: Detecting alternative implementations of the pattern

To start the second discussion point at the beginning I ran my previously created query on the project *JHotDraw51*. Firstly the query execution took a lot of time and results contained false positive and false negative.

Based on the results I understand that I need to rebuild my previously defined predicates. To do that I keep the idea how to find the observer pattern in the code base but I started to build predicates in a different manner.

Before to start building the new predicates I looked at the file *PMARt.xml* and discover the pattern classes in the code base. The observer pattern implementation in the project *JHotDraw51* code base is different than it was in the previous project. The main difference is that the first project concrete subject classes implement the subject interface and respectively the concrete object class implement the object interface while the *JHotDraw51* project concrete subject classes implement an abstract class in between to implement the subject interface and the same principle is applied to the concrete object classes.

Based on this knowledges, I am looking for the classes that implements the observer pattern interfaces to detect the concrete classes.

### Predicate implementation

This time I also started the main predicate with selecting all method declarations that as the parameter uses the interface and as the children have the methods *add* invocation to add the object to the list of observers. Also to restrict the results I created a condition that the methods return type is of the type *primitive type*. (See Fig.15)

Next, the same as previously, I search for the methods parameter and verify that the parameter node type is of the type *interface*. Then based on nodes type I search for the class that is the Observer interface. To find methods class I am using predicate *ast-parent* and I named this classes as abstract class of the concrete subject (*?AbstractClassOfConcSubj*), further concrete subject. (See Fig.15)

```
1  // All method declarations
2  (ast :MethodDeclaration ?method)
3  // Has return type is of type primitive type
4  (has :returnType2 ?method ?returnType2)
5  (has :primitiveTypeCode ?returnType2 ?typevoid)
6  // method has methods add invocation
7  (child+ ?method ?methChildren)
```

```

8 | (methodinvocation|named ?methChildren "add")
9 | //methods parameter nodes type is of type interface
10 | (child :parameters ?method ?parameter)
11 | (has :type ?parameter ?parameterType)
12 | (ast|type-type ?parameterType ?nodesType)
13 | (type|interface ?nodesType)
14 | //Find observer interface class from its nodes type
15 | (typeddeclaration-type ?ObserverClass ?nodesType)
16 | //Find methods parent that is the abstract class of concrete subject
17 | (ast-parent ?method ?AbstractClasofConcSubj)

```

**Figure.15.** To find Observer class and abstract concrete subject class

To find the subject interface I use predicate *SubjectClass|ObserverClass* and the observer interface. This predicate selects all type declarations that methods as parameter uses an interface and this interface node is the same as the observer interface. In this predicate I use *Ekeko* rule *one* to select the subject interface just once. (See Fig.16)

The implementation of the predicate *SubjectClass|ObserverClass* is found in appendix subsection *JHotDraw51 project predicates*.

To verify that the previously found concrete subject classes are the ones I am searching for I use predicate *Class|SuperInterfaceName* to find the given classes super interface types name as a string. Next I use other predicate *Clas|Name* to find the super interface class based on its name. The super interface from the concrete subject class needs to be the same as the subject interface, therefore I compare them using predicate *equal*. (See Fig.16)

The code of the predicates *Class|SuperInterfaceName* and *Clas|Name* is found in the appendix subsection *JHotDraw51 project predicates*.

```

1 | //Find Subject interface
2 | (SubjectClass|ObserverClass ?SubjectClass ?ObserverClass)
3 | //Selects the super interface type of the concrete subject classes
4 | (Class|SuperInterfaceName ?AbstractClasofConcSubj ?SubInterfacname)
5 | //Gets the super interface class from its name
6 | (Clas|Name ?SubInterfacname ?superClasNameSub)
7 | //Compare with subject interface
8 | (equals ?superClasNameSub ?SubjectClass)

```

**Figure.16.** Concrete subject classes

After the super interface type of the concrete subject class verification I use *type-type|sub+* to find all the subtypes of the given observer interface type. Next I select all the subtype classes, for further verification, using the predicate *typeddeclaration-type*. In the same way as I verified the concrete subject classes I verify the concrete observer classes. (See Fig.17)



```

1  //Select all subtypes of given observer interface type
2  (type-type|sub+ ?nodesType ?obAllsubType)
3  //Select all subtype classes
4  (typeddeclaration-type ?AbstractClasofConcObj ?obAllsubType)
5  //Finds the super interface types name from subtype classes
6  (Class|SuperInterfaceName ?AbstractClasofConcObj ?obInterfacname)
7  //Finds the super interface from its name
8  (Clas|Name ?obInterfacname ?superClasNameob)
9  //Compare to the observer interface
10 (equals ?superClasNameob ?ObserverClass)

```

**Figure.17.** Concrete observer classes

## Results

To detect the design pattern observer in the code base I built the *Ekeko* query shown in the figure below.

















```

1  (ekeko* [?AbstractClasofConcSubj ?ObserverClass
2      ?SubjectClass ?AbstractClasofConcOb]
3      (ObserverPatern ?AbstractClasofConcSubj ?ObserverClass
4      ?SubjectClass ?AbstractClasofConcOb))

```

**Figure.18.** Ekeko query

The following figure shows the results of the query in the project *JHotDraw51*. As it was in the *PMARt.xml* in the result as the subject interface is the class *Figure* and as the observer interface is the class *FigureChangeListener*. The figure below shows that only one class that implements the subject interface is the abstract class *AbstractFigure*. In the figure the column *?AbstractClasof-ConcOb* represents all classes that implements observer interface. Three of the concrete observer classes extends also the class *AbstractFigure*, except one *FigureChangeEventMulticaster*. This class is used as the list to store all observers.

?AbstractClasofConcSubj	?SubjectClass	?ObserverClass	?AbstractClasofConcOb
 AbstractFigure	 Figure	 FigureChangeListener	 CompositeFigure
 AbstractFigure	 Figure	 FigureChangeListener	 DecoratorFigure
 AbstractFigure	 Figure	 FigureChangeListener	 FigureChangeEventMulticaster
 AbstractFigure	 Figure	 FigureChangeListener	 TextFigure

**Figure.19.** The results of the Ekeko query on the project *JHotDraw51*

To verify my new predicates I also ran *Ekeko* query in the previous project code base and the results are shown in the figure below. As the results show my new predicates and query is general enough to detect design pattern in two different code bases.

?AbstractClasofConcSubj	?SubjectClass	?ObserverClass	?AbstractClasofConcOb
➤ Point	➤ ChangeSubject	➤ ChangeObserver	➤ Screen
➤ Screen	➤ ChangeSubject	➤ ChangeObserver	➤ Screen

**Figure.20.** *The results of the Ekeko query on the project DesignPatterns*

## References

Geary, D. (2003). An inside view of Observer. *Java Design Patterns*. Retrieved January 01, 2014, from <http://faculty.washington.edu/stepp/courses/2005winter/tcss360/readings/13-observer.html>

## Appendix

### DesignPattern project predicates

#### ConcreteObserver|SubjectInterfaceName

```
1 (defn ConcreteObserver|SubjectInterfaceName [?methodClass ?parameterClass]
2   (fresh [?method ?parameter ?parameterType ?nodeType ?classType]
3     //Find all methods with parameter type interface
4     (ast :MethodDeclaration ?method)
5     (child :parameters ?method ?parameter)
6     (has :type ?parameter ?parameterType)
7     (ast|type-type ?parameterType ?nodeType)
8     (type|interface ?nodeType)
9     //Methods class needs to be a class
10    (ast-parent ?method ?methodClass)
11    (typeddeclaration-type ?methodClass ?classType)
12    (type|class ?classType)
13    (typeddeclaration-type ?parameterClass ?nodeType)
14  ))
```

#### DesignPattern

```
1 (defn DesignPattern [?ConcreteSubjectClass ?ObserverInterface
2   ?SubjectInterface ?ConcreteObserver]
3   (fresh [?method ?parameter ?parameterType ?nodeType ?methChildren
4     ?observerClassMethods ?observerMethodParameters
5     ?observerParameterType ?SubjectNodesType]
6     //Find all methods that has parameter and parameter type is interface
7     (ast :MethodDeclaration ?method)
8     (child :parameters ?method ?parameter)
9     (has :type ?parameter ?parameterType)
10    (ast|type-type ?parameterType ?nodeType)
11    (type|interface ?nodeType)
12
13    //Check if in method is method add invocation
14    (child+ ?method ?methChildren)
15    (methodinvocation|named ?methChildren "add")
16
17    //find methods class
18    (ast-parent ?method ?ConcreteSubjectClass)
19
20    //Find Observer Interface Class
21    (typeddeclaration-type ?ObserverInterface ?nodeType)
22
23    //Find Observer interface methods and parameters
24    (child :bodyDeclarations ?ObserverInterface ?observerClassMethods)
```

```

25         (child :parameters ?observerClassMethods ?observerMethodParameters)
26
27         //check if observer interface parameter type is interface ->
28         //to find subject interface
29         (has :type ?observerMethodParameters ?observerParameterType)
30         (ast|type-type ?observerParameterType ?SubjectNodesType)
31         (type|interface ?SubjectNodesType)
32
33         //Find subject interface class
34         (typeddeclaration-type ?SubjectInterface ?SubjectNodesType)
35
36         //Observer interface and Subject interface can't be the same
37         (fails (equals ?SubjectInterface ?ObserverInterface))
38
39         //Based on Subject class name. Search for Concrete Observer
40         // class that as a property has subject interface
41         (ConcreteObserver|SubjectInterfaceName ?ConcreteObserver ?SubjectInterface)
42     ))

```

## JHotDraw51 project predicates

### SubjectClass|ObserverClass

```

1  // to find Subject class from ObserverClass
2  (defn SubjectClass|ObserverClass [?SubjectClass ?ObserverClass]
3      (one
4          (fresh [?type ?classBody ?parameter ?parameterType ?nodesType ]
5              (ast :TypeDeclaration ?SubjectClass)
6              (typeddeclaration-type ?SubjectClass ?type)
7              (type|interface ?type)
8
9              (child :bodyDeclarations ?SubjectClass ?classBody)
10             (child :parameters ?classBody ?parameter)
11             (has :type ?parameter ?parameterType)
12             (ast|type-type ?parameterType ?nodesType)
13             (type|interface ?nodesType)
14             (typeddeclaration-type ?ObserverClass ?nodesType)
15             //Subject interface can not be the same as the Observer interface
16             (fails (equals ?ObserverClass ?SubjectClass))
17         )))

```

### Clas|Name

```

1  //to get class out of name
2  (defn Clas|Name [?name ?class]
3      (fresh [?classname ?nameString ?stname]

```

```

4      (ast :TypeDeclaration ?class)
5      (has :name ?class ?classname)
6      (name|simple-string ?classname ?nameString)
7      (name|simple-string ?name ?stname)
8      (equals ?nameString ?stname)
9  ))

```

### Class|SuperInterfaceName

```

1  // to get from abstract class the super interface name it is using
2  (defn Class|SuperInterfaceName [?concOBCLas ?obInterfacname]
3    (fresh [?superInterfaceType ?stypesList ?obInterfacn ?concOBCLasType]
4      (ast :TypeDeclaration ?concOBCLas)
5      (has :superInterfaceTypes ?concOBCLas ?superInterfaceType)
6      //Take name out of the list
7      (value-raw ?superInterfaceType ?stypesList)
8      (contains ?stypesList ?obInterfacn)
9      (has :name ?obInterfacn ?obInterfacname)
10
11      (typedecclaration-type ?concOBCLas ?concOBCLasType)
12      (type|class ?concOBCLasType)
13    ))

```

### ObserverPatern

```

1  (defn ObserverPatern [?AbstractClasofConcSubj ?ObserverClass
2    ?SubjectClass ?AbstractClasofConcObj]
3    (fresh [?returnType2 ?typevoid ?method ?parameter ?parameterType
4      ?modifier ?methChildren ?nodesType
5      ?obAllsubType ?obInterfacname ?superClasNameob
6      ?superClasNameSub ?SubInterfacname]
7      // Find all methods that has return type void
8      (ast :MethodDeclaration ?method)
9      (has :returnType2 ?method ?returnType2)
10     (has :primitiveTypeCode ?returnType2 ?typevoid)
11     // Methods children is methods add invocation
12     (child+ ?method ?methChildren)
13     (methodinvocation|named ?methChildren "add")
14
15     // Method use parameter that is interface
16     (child :parameters ?method ?parameter)
17     (has :type ?parameter ?parameterType)
18     (ast|type-type ?parameterType ?nodesType)
19     (type|interface ?nodesType)
20     // Find parameter class
21     (typedecclaration-type ?ObserverClass ?nodesType)
22

```

```

23
24 // Find Subject Interface
25 (SubjectClass|ObserverClass ?SubjectClass ?ObserverClass)
26 // Concrete Subject class must to have superInterfaceType
27 // name the same as the subject interface name
28 (ast-parent ?method ?AbstractClasofConcSubj)
29 (Class|SuperInterfaceName ?AbstractClasofConcSubj ?SubInterfacname)
30 (Clas|Name ?SubInterfacname ?superClasNameSub)
31 (equals ?superClasNameSub ?SubjectClass)
32
33 // Find all classes that implement Observer class
34 (type-type|sub+ ?nodesType ?obAllsubType)
35 (typeddeclaration-type ?AbstractClasofConcObj ?obAllsubType)
36 // Observer Sub classes superInterfaceType name must be the same
37 // as the observer interface name
38 (Class|SuperInterfaceName ?AbstractClasofConcObj ?obInterfacname)
39 (Clas|Name ?obInterfacname ?superClasNameob)
40 (equals ?superClasNameob ?ObserverClass)
41 ))

```