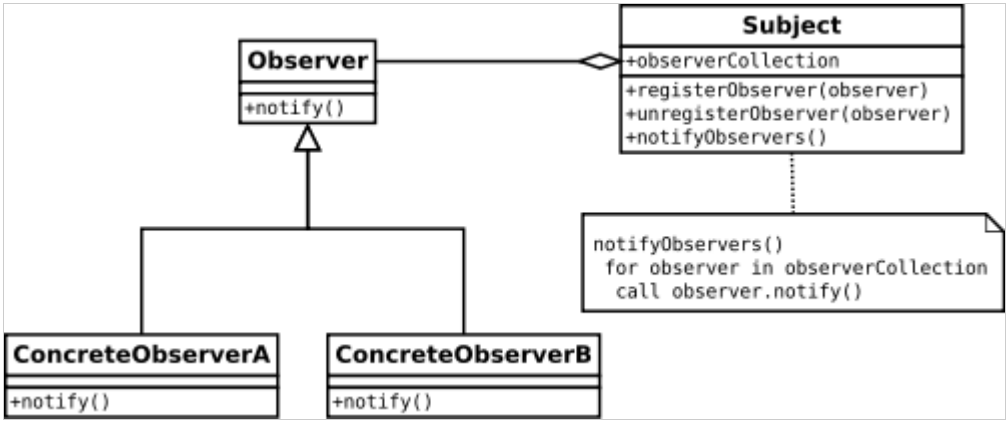
Observer pattern

From Wikipedia, the free encyclopedia

The **observer pattern** (a subset of the publish/subscribe pattern) is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods. It is mainly used to implement distributed event handling systems.

Contents

Structure



UML diagram of Observer pattern

Definition

The essence of the Observer Pattern is to "Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. ".[1]

Example

Below is an example that takes keyboard input and treats each input line as an event. The example is built upon the library classes `java.util.Observer` (http://java.sun.com/javase/7/docs/api/java/util/Observer.html) and `java.util.Observable` (http://java.sun.com/javase/7/docs/api/java/util/Observable.html) . When a string is supplied from System.in, the method `notifyObservers` is then called, in order to notify all observers of the event's occurrence, in the form of an invocation of their 'update' methods - in our example, `ResponseHandler.update(...)`.

The file `MyApp.java` contains a `main()` method that might be used in order to run the code.

```
/* File Name : EventSource.java */
package obs;

import java.util.Observable;          //Observable is here
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class EventSource extends Observable implements Runnable {
    public void run() {
        try {
            final InputStreamReader isr = new InputStreamReader( Syst
            final BufferedReader br = new BufferedReader( isr );
            while( true ) {
                String response = br.readLine();
                setChanged();
                notifyObservers( response );
            }
        }
        catch (IOException e) {
            e.printStackTrace();
```

```
        }
    }
}
```

```
/* File Name: ResponseHandler.java */

package obs;

import java.util.Observable;
import java.util.Observer;  /* this is Event Handler */

public class ResponseHandler implements Observer {
    private String resp;
    public void update (Observable obj, Object arg) {
        if (arg instanceof String) {
            resp = (String) arg;
            System.out.println("\nReceived Response: "+ resp );
        }
    }
}
```

```
/* Filename : MyApp.java */
/* This is the main program */

package obs;

public class MyApp {
    public static void main(String args[]) {
        System.out.println("Enter Text >");

        // create an event source - reads from stdin
        final EventSource evSrc = new EventSource();

        // create an observer
        final ResponseHandler respHandler = new ResponseHandler();

        // subscribe the observer to the event source
        evSrc.addObserver( respHandler );

        // starts the event thread
        Thread thread = new Thread(evSrc);
        thread.start();
    }
}
```

Implementations

The observer pattern is implemented in numerous programming libraries and systems, including almost all

GUI toolkits.

Some of the most notable implementations of this pattern:

ActionScript

- flash.events (http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/events/package-detail.html) , a package in ActionScript 3.0 (following from the mx.events package in ActionScript 2.0).

C

- GObject, in GLib - an implementation of objects and signals/callbacks in C. (This library has many bindings to other programming languages.)

C++

- libsigc++ (http://libsigc.sourceforge.net) - the C++ signalling template library.
- sigslot (http://sigslot.sourceforge.net/) - C++ Signal/Slot Library
- Cpp::Events (http://code.google.com/p/cpp-events) - Template-based C++ implementation that introduces separation of connection management interface of the event object from the invocation interface.
- XLObject (http://xlobject.sourceforge.net/) - Template-based C++ signal/slot model patterned after Qt.
- Signals (http://github.com/pbhogan/Signals) - A lightweight and non-intrusive C++ signal/slot model implementation.
- libevent (http://www.monkey.org/~provos/libevent/) - Multi-threaded Crossplatform Signal/Slot C++ Library
- Boost.Signals (http://www.boost.org/doc/html/signals.html) , an implementation of signal/slot model
- MFC's CDocument-CView-framework
- The Qt C++ framework's signal/slot model
- The MRPT robotics C++ framework's observer/observable (http://reference.mrpt.org/svn/classmrpt_1_1utils_1_1_c_observable.html) model.

Objective C

- NSKeyValueObserving (http://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Protocols/NSKeyValueObserving_Protocol/Reference/Reference.html) - The Objective C NSKeyValueObserving protocol.

C#

- The IObserver<T> Interface (http://msdn.microsoft.com/en-us/library/dd783449.aspx) - The .NET Framework supported way of implementing the observer pattern.
- Exploring the Observer Design Pattern (http://msdn.microsoft.com/en-us/library/ee817669.aspx) - the C# and Visual Basic .NET implementation, using delegates and the Event pattern

ColdFusion

- http://www.cfdesignpatterns.com/behavioral-patterns/observer-design-pattern-in-coldfusion/

Delphi

- Delphi Observer Pattern (http://blogs.teamb.com/joannacarter/2004/06/30/690) , a Delphi implementation

Java

- The class `java.util.Observer`[2] provides a simple observer implementation.
- Events are typically implemented in Java through the callback pattern: one piece of code provides a common interface with as many methods as many events are required[3], another piece of code provides an implementation of that interface, and another one receives that implementation, and raises events as necessary.[4]

JavaScript

- JavaScript Observer Pattern (https://github.com/rgr-myrg/DevShop-JS) . Pocket-sized minimalist framework of common design patterns in JavaScript.
- EventDispatcher singleton (http://www.refactory.org /s/eventdispatcher_singleton_a_core_api_based_signals_and_slots_implementation/view/latest) , a JavaScript core API based Signals and slots implementation - an observer concept different from Publish/subscribe - pretty lightweighted but still type-safety enforcing.

Lisp

- Cells (http://common-lisp.net/project/cells/) , a dataflow extension to Common Lisp that uses meta-programming to hide some of the details of Observer pattern implementation.

Perl

- Class::Observable (http://search.cpan.org/~cwinters/Class-Observable-1.04/lib/Class /Observable.pm) Basic observer pattern implementation
- Class::Publisher (http://search.cpan.org/~simonflk/Class-Publisher-0.2/lib/Class/Publisher.pm) a slightly more advanced implementation

PHP

- Event_Dispatcher (http://pear.php.net/package/Event_Dispatcher) , a PHP implementation
- SPL (http://www.php.net/~helly/php/ext/spl/main.html) , the Standard PHP Library
- Symfony Event Dispatcher (http://components.symfony-project.org/event-dispatcher/) , a standalone library by the Symfony team

Python

- Louie (http://github.com/11craft/louie) , an implementation by Patrick K. O'Brien.
- PyDispatcher (http://pydispatcher.sourceforge.net/) , the implementation on which the Django (http://www.djangoproject.com/) web framework's signals are based.
- Py-notify (http://home.gna.org/py-notify/) , a Python (plus a little C) implementation
- Observer Pattern using Weak References (http://radio-weblogs.com/0124960/2004/06/15.html) implementation by Michael Kent
- PyPubSub (http://sourceforge.net/projects/pubsub/) an in-application Pub/Sub library for Observer

behavior
- NotificationFramework (http://pypi.python.org/pypi/NotificationFramework/) classes directly implementing Observer patterns
- Blinker (http://pypi.python.org/pypi/blinker/1.1) , an implementation which can be used with decorators.

Ruby

- Observer (http://ruby-doc.org/stdlib/libdoc/observer/rdoc/index.html) , from the Ruby Standard Library.

Other/Misc

- CSP (http://ptolemy.eecs.berkeley.edu/presentations/06/FutureOfEmbeddedSoftware_Lee_Graz.ppt) - *Observer Pattern* using *CSP-like Rendezvous* (each actor is a process, communication is via rendezvous).
- YUI Event utility (http://developer.yahoo.com/yui/event/) implements custom events through the observer pattern
- Publish/Subscribe with LabVIEW (http://www.labviewportal.eu/viewtopic.php?f=19&t=9) , Implementation example of Observer or Publish/Subscribe using G.

Critics

The Observer pattern is criticized[5] for being too verbose, introducing too many bugs and violating software engineering principles, such as not promoting side-effects, encapsulation, composability, separation of concepts, scalability, uniformity, abstraction, resource management, semantic distance. The recommended approach is to gradually deprecate observers in favor of reactive programming abstractions.

See also

- Design Patterns (book), the book which gave rise to the study of design patterns in computer science
- Design pattern (computer science), a standard solution to common problems in software design
- Implicit invocation
- Model-view-controller (MVC)
- Client–server model

References

1. ^ Gang Of Four
2. ^ java.util.Observer (http://download.oracle.com/javase/6/docs/api/java/util/Observer.html)
3. ^ Which will typically implement interface java.util.EventListener (http://download.oracle.com/javase/6/docs/api/java/util/EventListener.html)
4. ^ http://download.oracle.com/javase/tutorial/uiswing/events/generalrules.html
5. ^ Deprecating the Observer Pattern by Ingo Maier, Tiark Rompf, Martin Odersky (2010) PDF (http://lamp.epfl.ch/~imaier/pub/DeprecatingObserversTR2010.pdf)

- http://www.research.ibm.com/designpatterns/example.htm
- http://msdn.microsoft.com/en-us/library/ms954621.aspx
- "Speaking on the Observer pattern" (http://www.javaworld.com/javaworld/javaqa/2001-05/04-qa-0525-observer.html) - JavaWorld

External links

- Observer Pattern implementation in JDK 6 (http://java.sun.com/javase/6/docs/api/java/util/Observable.html)
- Observer Pattern in Java (http://www.javaworld.com/javaworld/javaqa/2001-05/04-qa-0525-observer.html)
- Definition, C# example & UML diagram (http://www.dofactory.com/Patterns/PatternObserver.aspx)
- Jt (http://www.fsw.com/Jt/Jt.htm) J2EE Pattern Oriented Framework
- Subject Observer example in C++ (http://rtmatheson.com/2010/03/working-on-the-subject-observer-pattern/)
- Observer Pattern recipe in Python (http://code.activestate.com/recipes/131499-observer-pattern/)
- SourceMaking Tutorial (http://sourcemaking.com/design_patterns/observer)
- Observer Pattern in Objective-C (http://www.a-coding.com/2010/10/observer-pattern-in-objective-c.html)
- Observer Pattern in Java (Portuguese) (http://www.patternizando.com.br/2011/03/design-pattern-observer-com-aplicacao-swing-jse/)