Design Patterns Used in Eclipse

Ilya Shinkarenko
Bangalore, Eclipse Summit India 2009
18.07.2009

About me: Ilya Shinkarenko

» Software Engineer / Architect
  » Eclipse RCP, OSGi
  » Spring, JBoss AS
  » Other stuff: Android OS, Flash/Flex, Red5 etc.

» Working in IT since 1997
  » Java since 1999
  » Eclipse expertise since 2004

» Consulting, workshops, seminars
  » Eclipse RCP and basically everything around Eclipse
  » HCI, Usability, Screen Design

» E-Mail: ilya@shinkarenko.org
About the Eclipse Training Alliance

» Founded December 2005 by WeigleWilczek and Innoopract
  » WeigleWilczek now drives the initiative

» Offerings:
  » International one-stop delivery of high quality Eclipse training classes
  » Certificates
  » Coaching and consulting

The Partners

» We share expertise
Our Training Classes

» RCP
  » Developing RCP
  » Advanced RCP
» OSGi
  » Developing OSGi
  » Advanced OSGi
» Other
  » Administration Training
  » Architecture Training
  » Developer Training
  » Eclipse SOA Workshop
  » HYPERIC System Administration Training
  » Methodology Training

» Modelling
  » Eclipse Modeling (EMF & GMF)
  » EMF Professional
  » Graphical Modeling with GMF
  » Model-Driven Development with Eclipse Modeling
  » MDD in the context of Software Engineering
  » Advanced MDD with Eclipse Modeling and openArchitectureWare

Contact Europe

Please contact us for details:

Eclipse Training Alliance
c/o Weigle Wilczek GmbH
Heiko Seeberger
Martinstrasse 42-44
D-73728 Esslingen

Phone +49 711 45 99 98 0
Fax +49 711 45 99 98 29
www.eclipse-training.net
Contact India

Please contact us for details:

ANCiT Consulting
Mr. Imran S
84, Rajiv Gandhi Nagar,
Sowripalayam,
Coimbatore, TN, India
PIN 641028

T +91-422-6461778
certification@ancitconsulting.com

About You

» Are you at the moment / have you been actively involved in:
   » … development with Eclipse RCP?
   » … development of plug-ins for Eclipse IDE?
   » … development for OSGi / Equinox platform?
Classical Pattern Catalog

- Creational Patterns
  - Factory
  - Builder
  - Singleton
  - ...
- Structural Patterns
  - Adapter
  - Bridge
  - Composite
  - Proxy
  - Facade
  - ...
- Behavioral Patterns
  - Observer
  - Command
  - Memento
  - Strategy
  - Visitor
  - ...

Patterns in Eclipse

- Classical Pattern Catalog
  - Creational Patterns
    - Factory
    - Builder
    - Singleton
    - ...
  - Structural Patterns
    - Adapter
    - Bridge
    - Composite
    - Proxy
    - Facade
    - ...
  - Behavioral Patterns
    - Observer
    - Command
    - Memento
    - Strategy
    - Visitor
    - ...

- Eclipse Pattern Catalog
  - Platform
    - Singleton: getting Workbench / Plug-in / Service
  - OSGi
    - Bridge: OSGi Services
    - Whiteboard: pluggable listeners
  - Workspace Resources
    - Proxy and Bridge: Accessing File System
    - Composite: the workspace
    - Observer: tracking resource changes
    - Visitor: traversing the resource tree
  - Core Runtime
    - IAdaptable and Adapter Factories: Property View
  - SWT
    - Composite: composing widgets
    - Strategy: defining the layout
    - Observer: responding to events
  - JFace
    - Pluggable adapter: Connecting widget to model
    - Strategy: customize a viewer without subclassing
    - Command: Actions
  - UI Workbench
    - Memento: persisting UI state
    - Virtual Proxy: lazy loading with E.P.
Patterns in Eclipse

- Platform Runtime
  - IAdaptable and Adapter Factories: Property View
  - Singleton: getting a Workbench / Plug-in / Service instance
  - Bridge: OSGi Services
  - Whiteboard: pluggable listeners
- Workspace Resources
  - Proxy and Bridge: accessing File System
  - Composite: the workspace
  - Observer: tracking resource changes
  - Visitor: traversing the resource tree

- SWT
  - Composite: composing widgets
  - Strategy: defining the layout
  - Observer: responding to events
- JFace
  - Pluggable adapter: connecting widget to model
  - Strategy: customize a viewer without subclassing
  - Command: Actions
- UI Workbench
  - Memento: persisting UI state
  - Virtual Proxy: lazy loading with E.P.

Runtime

Platform

- Workbench
- JFace
- SWT

Core Resources

Core Runtime
"Anticipate that an object’s interface needs to be extended in the future. An Extension Object lets you add interfaces to a class and lets clients query whether an object has a particular extension."

Extension Interface: `IAdaptable`

```java
IAdaptable
getAdapter(Class)
```

**Need for Adapters**

» Add a service interface to a type without exposing it in that type

» Add behavior to preexisting types such as `IFile`, `Person`, etc.

» Our goal: we want to adapt `Person` to `IPropertySource`
Getting an Adapter:

```
public static Object getAdapter(Object sourceObject, Class adapter) {
    // 1. Check instance
    if (adapter.isInstance(sourceObject)) {
        return sourceObject;
    }

    // 2. Check if can adapt to
    if (sourceObject instanceof IAdaptable) {
        IAdaptable adaptable = (IAdaptable) sourceObject;
        Object result = adaptable.getAdapter(adapter);
        if (result != null) {
            // Querying check
            ensureAdaptableDoesNotChange(result);
            return result;
        }
    }

    // 3. Load adapter from the Platform Adapter Manager
    Platform.getAdapterManager().loadAdaptable(sourceObject, adapter.getName());
}
```

Use of AdapterManager in Platform

```
public abstract class PlatformObject implements IAdaptable {

    public Object getAdapter(Class adapter) {
        return AdapterManager.getDefault().getAdapter(this, adapter);
    }
}
```
How does the Properties View work?

The object in the Current Selection must be an IPropertySource.

What does it mean: „the object must be an IPropertySource“?

- be a direct implementor of IPropertySource:
  ```java
  public class Person implements IPropertySource { /**/ }
  ```

- or can adapt to IPropertySource

What does it mean „the object can adapt to IPropertySource“?

- to implement IAdaptable and be able to return an IPropertySource on request
  ```java
  public class Person implements IAdaptable { /**/ }
  ```

- or to have a registered IAdapterFactory which would be able to adapt Person to IPropertySource
What is an Adapter?

» In object-oriented software systems, an adapter simply adapts (converts) an object of type A to another object of relevant type B

» Eclipse provides the interface IAdaptable to address the adaption of an object:

```java
public interface IAdaptable {
    public Object getAdapter(Class adapter);
}
```

» Since model objects should not depend on Eclipse, AdapterFactories can adapt all objects. Even if the objects do not implement IAdaptable...

» How does this work?

IAdapterFactory

![Diagram showing the relationship between IAdaptable, PlatformObject, and IAdapterManager with AdapterFactories.](image)
Such a factory provides adapters for given adaptable types:

```java
public class AdapterFactory implements IAdapterFactory {
    public Class[] getAdaptation(Class<?> [] c, Class<?> adaptee) {
        return new Class[] { IPropertySource.class, IQuery.class, IQueryFeature.class, IAdapterFactory.class };
    }

    public Object getAdapterManager(Class adaptee) {
        if (AdapterFactory.class.isAssignableFrom(adaptee) && IPropertySource.class.isAssignableFrom(adaptee)) {
            final Person p = (Person) adaptee.cast(Person.class);
            return new IPropertySource() { ... return null; 
        }
    }
}
```

Implementations of IAdapterFactory can be registered with the platform:

- programatically:
  ```java
  Platform.getAdapterManager().
  .registerAdapters(adapterFactory, Person.class);
  ```

- or declaratively:
  ```xml
  <extension
      point="org.eclipse.core.runtime.adapters">
    <factory
        adaptee="org.eclipse.core.runtime.IAdapterFactory">
    </factory>
  </extension
```

» see E.P. org.eclipse.core.runtime.adapters
“Ensure that a class only has one instance, and provide a global point of access to it.”

```java
Singleton
static getInstance()
static uniqueInstance
return uniqueInstance;
```

Usage examples in Eclipse

- `PlatformUI.getWorkbench()`
- `Platform.getAdapterManager()`
- `ResourcesPlugin.getWorkspace()`
- `PMCoreActivator.getInstance().getPersonRepository()`
### Singleton Drawbacks

» Use of `static`:
  » Classloading issues
  » May behave in an unpredictable fashion in dynamic OSGi environments

» Coupling between the Singleton and the Client(s)
  » Bridge pattern will help ;-)
PM Core Retrospective

» Core bundle is designed fairly good (it is already a bridge in a fact), but it is not really flexible in terms of Components:

- Core bundle is designed fairly good (it is already a bridge in a fact), but it is not really flexible in terms of Components:

- Have to call Core methods directly 😊
- IPersonRepository implementation is coupled with Core 😊

Solution: introduce a bridge

» Make possible for others to contribute core implementations:
Bundles may register services in an OSGi Service Registry

```java
public void start(BundleContext context) throws Exception {
    super.startContext();
    plugins = false;
    final IPersonRepository personRepository = new DummyPersonRepository();
    context.registerService(IPersonRepository.class, name, personRepository, null);
}

public void stop(BundleContext context) throws Exception {
    plugins = null;
    super.stopContext();
}
```

Other bundles may get the registered services:

```java
final String serviceName = IPersonRepository.class.getName();
final ServiceReference serviceReference = context.getServiceReference(serviceName);
if (serviceReference == null) {
    throw new Exception("Person Repository service cannot be found");
}
personRepository = (IPersonRepository) context.getService(serviceReference);
```

Or listen to their lifecycle:

```java
final String serviceName = IPersonRepository.class.getName();
final ServiceTracker serviceTracker = new ServiceTracker(null, serviceName,
    new BundleActivator.ServiceTrackerObserver()
) {
    @Override
    public void removedService(ServiceReference reference, Object service) {
        log("Removed service " + service);
    }
    @Override
    public void modifiedService(ServiceReference reference, Object service) {
        log("Modified service " + service);
    }
    @Override
    public void addingService(ServiceReference reference) {
        log("Adding service with reference " + reference + " to personRepository");
        return personRepository;
    }
}
```
Listeners Considered Harmful: The “Whiteboard” Pattern


Listener support in IPersonRepository

```java
public interface IPersonRepository {
    * * Creates the given person.
     * String create(Person person) throws PMException;
    * * Loads a person by the given id.
     * Person load(String id) throws PMException;
    * * Loads all persons from the repository sorted by ID.
     * List<Person> loadAll() throws PMException;
    * * Updates the given (link Person)
     * void update(Person person) throws PMException;
    * * Deletes the given (link Person)
     * void delete(Person person) throws PMException;
    * * Adds the given (link IPersonListener) to the repository's listener list.
     * void addListener(IPersonListener listener);
    * * Removes the given (link IPersonListener) to the repository's listener list.
     * void removeListener(IPersonListener listener);
}
```
1. Mark listener support methods in IPersonRepository as deprecated or remove at all

2. Let all clients register IPersonListeners via OSGi Service Registry

3. Set up a ServiceTracker in your core implementation to update the local cache of Listeners

4. Test carefully (View open-close, Bundle start-stop etc.)
Proxy and Bridge: Accessing File System

» Proxy: "Provide a surrogate or place holder for another object to control access to it"

» Proxy structure at a runtime:

```
+------------------+-+------------------+-+
| aClient           | aClient           | aClient           |
| subject           | realSubject       |                  |
```

IFile as a Proxy

» How to address a resource in a Workspace?

» give a handle for a resource, not the full resource

» the handle acts like a key for a resource
The handles are defined as interfaces IFile, IFolder, IProject, and IWorkspaceRoot.

Small objects. Once created, none of their fields will ever change. Use them in maps as keys.

Handles define the behavior of a resource, but they do not keep any resource state information.

A handle can refer to non-existing resources.

Handles are created from a parent handle:
```
IProject project;
IFolder folder = project.getFolder("someFolder");
```

Handles are used to create the underlying resource:
```
folder.create(...);
```
IResource Is a Proxy and a Bridge

Composite: IWorkspace

- Composite: “Compose object into tree structures to represent part/whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.”

- A typical Composite object structure:
IWorkspace as a Composite

IWorkspace is a Composite of IContainers and IFiles

Access the Singleton workspace instance from the static accessor
ResourcesPlugin.getWorkspace()
Observer: tracking resource changes

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

Subject
- Attach(Observer)
- Detach(Observer)
- Notify()

ConcreteSubject
- GetState()
- SetState()

Observer
- Update()

ConcreteObserver
- Update()
- observerState = subject->GetState()

SubjectState
- IResourceChangeListener

Observer: Subject
- Observer
- IWorkspace

addResourceChangeListener()
removeResourceChangeListener()

public void resourceChanged(IResourceChangeEvent event) {
    if (IResourceChangeEvent.STRUCTURAL_CHANGE == event.getType()) {
        //
    }
}

IResourceChangeListener
- addResourceChangeListener()
- removeResourceChangeListener()
IResourceDelta Records a Tree of Changes

- A resource delta describes a single change and multiple changes using the same structure.
- It is easy to process a resource delta recursively top-down when updating an observer.
- You can reuse the traversal logic of a resource delta with an IResourceDeltaVisitor:

```java
public interface IResourceDeltaVisitor {
    public boolean visit(IResourceDelta delta) throws CoreException;
}
```

 Processing the Resource Delta

```java
IWorkspace ws = ResourcesPlugin.getWorkspace();
ws.addResourceChangeListener(new IResourceChangeListener() {
    @Override
    public void resourceChanged(IResourceChangeEvent event) {
        IResourceDelta delta = event.getDelta();
        IResourceDeltaVisitor deltaVisitor = new IResourceDeltaVisitor() {
            @Override
            public boolean visit(IResourceDelta delta) throws CoreException {
                return true;
            }
        };
        try {
            delta.accept(deltaVisitor);
        } catch (CoreException e) {
        }
    }
});
```
Visitor: traversing the resource tree

Visitor: "Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates."

Visitor Interaction Diagram

ObjectStructure
  \[\text{Accept(Visitor)}\]

\[\text{Element} \rightarrow \text{Visitor} \]

ConcreteElementA
  \[\text{Accept(Visitor \(v\))} \rightarrow \text{OperationA()}\]

ConcreteElementB
  \[\text{Accept(Visitor \(v\))} \rightarrow \text{OperationB()}\]

Visitor
  \[\text{VisitConcreteElementA(ConcreteElementA)} \rightarrow \text{VisitConcreteElementB(ConcreteElementB)}\]

ConcreteVisitor1
  \[\text{VisitConcreteElementA(ConcreteElementA)} \rightarrow \text{VisitConcreteElementB(ConcreteElementB)}\]

ConcreteVisitor2
  \[\text{VisitConcreteElementA(ConcreteElementA)} \rightarrow \text{VisitConcreteElementB(ConcreteElementB)}\]

Client
  \[\text{VisitConcreteElementA(ConcreteElementA)} \rightarrow \text{VisitConcreteElementB(ConcreteElementB)}\]
Visiting a Resource API

```java
final IResource resource = ResourcesPlugin.getWorkspace().getRoot();

final IFacetVisitor visitor = new IFacetVisitor() {
    public boolean visit(IResource resource) throws CoreException {
        if (resource.getType(29) == IResource.FILE) {
            IFile file = (IFile) resource;
            System.out.println(file.getName());
        }
        return true;
    }
};

try {
    resource.accept(visitor, IResource.ÊOSETperature, false);
} catch (CoreException e) { }
```

---

**SWT**

- **Platform**
  - Workbench
  - JFace
  - SWT
  - Core Resources
  - Core Runtime
» Composite: “Compose object into tree structures to represent part/whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.”

» A typical Composite object structure:

![Composite diagram]

SWT basic and compound widgets

![SWT widget diagram]
SWT basic and compound widgets

» Basic widgets
  » do not contain other widgets
  » are the leaves in a widget tree
  » Button, Label, Text, ...

» Compound widgets
  » contain other widgets
  » are the inner nodes of a widget tree
  » have Composite as the base class

```java
private void initControl(Composite parent) {
    Composite composite = new Composite(parent, SWT.NONE);
    // ...
    Label label = new Label(composite, SWT.NONE);
    label.setText("hello");
    // ...
}
```

Strategy: Defining UI Layout

» Strategy: "Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it."

```
Context
  ContextInterface()

Strategy
  AlgorithmInterface()

ConcreteStrategyA
  AlgorithmInterface()

ConcreteStrategyB
  AlgorithmInterface()

ConcreteStrategyC
  AlgorithmInterface()
```
**SWT Layout Managers**

- Control
  - setLayoutData(Object)
  - getLayoutData()

- Composite
  - getLayout()

- Strategy
  - Context

- Layout
  - computerSize()
  - getLayout()

- FillLayout
- FormLayout
- GridLayout
- RowData
- RowLayout

**SWT Layout Example**

```java
public class ExampleComposite extends Composite {

    EmailText txtFirstName;

    public ExampleComposite(Composite parent, final int style) {
        super(parent, style);

        // 1. Create widgets
        final Label label = new Label(this, SWT.NONE);
        txtFirstName = new Text(this, SWT.BORDER);

        // 2. Set the layout manager
        final GridData layoutData = new GridData(GridData.FILL_HORIZONTAL);
        txtFirstName.setLayoutData(layoutData);

        // 3. Set the layout data (if needed)
        final GridData layoutData = new GridData(GridData.FILL_HORIZONTAL);
        label.setLayoutData(layoutData);

        // 4. Set some data (if needed)
        label.setText("First Name");
    }
}
```
Observer: responding to Events

Observer

SelectionListener

widgetSelected(SelectionEvent)

SelectionAdapter

widgetSelected(SelectionEvent)
widgetDefaultSelected(SelectionEvent)

Observer-Subject

Button

addSelectionListener()

Adding Listeners to SWT widgets

```java
Test txtName = new Text(smp, SWT.FLUSH);

// Typed listener:
txtName.addModifyListener(new ModifyListener() {
    @Override
    public void modifyText(ModifyEvent e) {
        // handle modify
    }
});

// Generic (untyped) listener:
txtName.addListener(SWT.Modify, new Listener() {
    @Override
    public void handleEvent(Event event) {
        // handle modify
    }
});
```
Pluggable Adapters: Label- and Content Providers

» Adapt the domain knowledge so that the Viewer can understand it and render properly:
  » Content: IContentProvider
  » Rendering: ILabelProvider
A Content Viewer ...

- ... delegates handling of input changes to a content provider:
  - The viewer queries the content provider for (an) element(s) to be shown
- ... delegates mapping the elements to be displayed to labels and images to a label provider

![Content Viewer Diagram]

Strategy: Customizing without Subclassing

- ViewerSorter and ViewerFilter are strategies

![Strategy Diagram]
» Command: "Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support for undoable operations."

**Command: Actions**

- Client
- Inkover
- Receiver
- Command
  - Execute()
- ConcreteCommand
  - Execute()
  - receiver -> Action();
  - state

**JFace IAction**

- IAction
  - run()
  - getImageDescriptor()
  - getToolTipText()
  - isEnabled()
  - addPropertyChangeListener()
- IPropertyChangeListener
  - propertyChange()
- Observer
  - Subject

- Command
- Observer: Subject
IAction defines a run() method to be called to execute the request.

Stores all the “decorative” information:
- to present the action in a menu / toolbar / button
- action’s label, icon, tooltip, and enablement state

Can be used by multiple menu items / toolbar items / buttons at the same time. Create an Action once and share it.

Fires a property change event when an action’s state changes.
- This allows JFace to keep the state of the widget in sync with the action.

Action: Default IAction implementation

Encapsulate the code you want to run in “run” method
Set the decorative attributes
Use the action in menu / toolbar / …

```java
final IAction action = new Action("Edit JFace") {
    @Override
    public void run() {
        System.out.println();
    }
}

// do not override getters, use setters instead!
action.setEnabled(true);
action.setDescription("Edit JFace");
action.setToolTipText("Edit JFace");

// attach the action
final MenuManager menuManager = new MenuManager("File");
final ToolBarManager toolbarManager = new ToolBarManager(SWT.FLAT);
menuManager.add(action);
toolbarManager.add(action);
```
Memento: Persisting UI State

» **Memento:** "Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later."

```java
class Originator

public void SetMemento(Memento m) {
    // Set the state
}

public Memento CreateMemento() {
    // Create a new Memento
    return new Memento(state);
}

public void UseMemento(Memento m) {
    // Use the Memento
}

```

```java
class Memento

public Memento() {
    // Initial state
}

public void setState(int state) {
    this.state = state;
}

public int getState() {
    return state;
}

```

```java
class Caretaker

public Memento getMemento() {
    // Get the Memento
}

public void setMemento(Memento m) {
    // Set the Memento
}

```
Memento: Persisting UI State

```java
public class MementoViewport extends ViewPort {
    private static final String ANY_NAME = "name";
    private static final String ANY_VALUE = "";
    private Text txtName;
    private String savedName = ANY_NAME;

    @Override
    public void init(ViewPort view) throws PanelliException {
        super.init(view, memento);
        if (memento != null) {
            final String name = memento.getString(ANY_NAME);
            this.txtName = new Text(name); // new Text(name )
        }
    }

    @Override
    public void setTextView(String name) {
        this.txtName.setName(name);
    }

    @Override
    public void setFullName() {
        this.txtName.setText();
    }

    public void passMemento(Memento memento) {
        memento.putString(ANY_NAME, txtName.getText());
    }
}
```

Virtual Proxy: lazy loading with E.P.

» Proxy: "Provide a surrogate or place holder for another object to control access to it."

» Proxy structure at runtime:

![Diagram of proxy structure](image)

» Virtual: the proxy has only a descriptor, the instance is created on demand
Virtual Proxies: the Lazy Loading Rule

» PluginAction Lazily Loads the Real Action:

```
Proxy:Subject
  IAction
  PluginAction
    run(Action)
    selectionChanged(ISelection)
    getDelegate()

Proxy:RealSubject
  run(Action)
  run(Action)
  selectionChanged(IAction, ISelection)
  NewRepositoryAction
    run(Action)
    selectionChanged(IAction, ISelection)
```
IActionDelegate implementation

```java
/**
 * Our sample action implements workbench action delegate.
 * The action proxy will be created by the workbench and shows in the UI.
 * Then the user tries to use the action, this delegate will be created and
 * execution will be delegated to it.
 */

public class ActionDelegate implements IActionDelegate {

    public void init(IWorkbenchWindow window) {
    }

    public void selectionChanged(IAction action, ISelection selection) {
    }

    /**
     * The action has been activated. The argument of the method represents the
     * "real" action sitting in the workbench UI.
     * @param action The action.
     */

    public void run(IAction action) {
        System.out.println("Hello, my real name is " + action.getText());
    }

    public void dispose() {
    }

    }
```

Eclipse RCP Certification Program

Authorized Eclipse Personnel Only
Eclipse RCP Certification Program

» Prove your competence – get certified!
  » „Certified Eclipse RCP Developer”
  » „Certified OSGi & Equinox Developer”

» Which skills are required?
  » The exams are based on the training classes of the ETA.
  » Of course you are allowed to start the exam without having joined any class.

» How can I get certified?

  » Visit http://www.eclipse-training.net/india/certification

  » Special offer for Eclipse Summit India!

Copyright notice

» This presentation contains some excerpts from the following books:

  » Design Patterns
    Elements of Reusable Object-Oriented Software
    by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

  » Contributing to Eclipse: Principles, Patterns, and Plug-Ins
    by Erich Gamma, Kent Beck

» This presentation is supposed to be used ONLY in non-for-profit activities