Replace and extend the Eclipse Framework
Lars Vogel

Lars Vogel Eclipse platform and e4 committer

Founder of vogella GmbH, Trainer and Consultant for Eclipse and Android.

Author of the Eclipse 4 RCP book
Framework design rule #1

As simple as possible
DnD is provided by a model add-on
DnDAddon is registered via a model processor at runtime
DnDAddon get triggered via events
DnDAddon modifies the application model
Framework design rule #2

Extensible and flexible
Presentation is flexible
What about the structure?
Eclipse Modeling Framework (EMF) is the underlying basis of the application model.
EMF extension + Custom renderer
Demo

Solution App in Github repo
com.vogella.e4.appmodel.app

We have this also as exercise but it will be a loooooong exercise if you do not know EMF
Example for model extensions:

WAZAABI talk on Thursday 11:00 – 11:35
All the flexibility at your fingertips

+Context Modifications

+Model Add-ons
Lets mix some things
Propagating the selection
Dependency injection

OSGi

Application

Window

Perspective

Dynamically queried

Selection persisted in IEEclipseContext of the window

Part

Part
Setting the selection in one Part:

```java
@Inject
ESelectionService selectionService;

viewer.addSelectionChangedListener(new ISelectionChangedListener() {
    @Override
    public void selectionChanged(SelectionChangedEvent event) {
        IStructuredSelection selection = (IStructuredSelection) viewer.getSelection();
        service.setSelection(selection.getFirstElement());
    }
});
```

Consume active selection in another Part:

```java
@Inject
public void setTodo(@Optional @Named(IServiceConstants.ACTIVE_SELECTION) Todo todo) {
    if (todo != null) {
        // Remember the todo as field
        this.todo = todo;
        // update the user interface
        updateUserInterface(todo);
    }
}
```
Setting the selection in one Part:

```java
@Inject
ESelectionService selectionService;

viewer.addSelectionChangedListener(new ISelectionChangedListener() {
    @Override
    public void selectionChanged(SelectionChangedEvent event) {
        IStructuredSelection selection = (IStructuredSelection) viewer.getSelection();
        service.setSelection(selection.getFirstElement());
    }
});
```

Consume active selection in another Part:

```java
@Inject
public void setTodo(@Optional @Named(IServiceConstants.ACTIVE_SELECTION) Todo todo) {
    if (todo != null) {
        // Remember the todo as field
        this.todo = todo;
        // update the user interface
        updateUserInterface(todo);
    }
}
Selection is based on the MWindow context
I deleted the ESelectionService
Current solution

Dynamically queried

Selection persisted in IEclipseContext of the window
Desired solution

Dynamically queried

Put the selection here!
Implementation
We write a service which can modify a given context
Joining the Eclipse framework with context functions
Searching the context

OSGi

Application

Window

Perspective

Part

Part

Dynamically queried
Context Functions

Dynamically queried

- OSGi
  - Application
    - Window
      - Perspective
        - Part
          - Part
Dependency injection with Context Functions

CF registers for a type and has access to the requesting context

```java
@Inject @Named(key) Type s;
```
Model Add-ons
Use a model add-on to register the context function for the new service
Demo application
Exercise
Questions?
Evaluate This Session

1. Sign-in: www.eclipsecon.org
2. Select session from schedule
3. Evaluate: +1 0 -1