Single Sourceing for Eclipse 4.x and 3.x

Tom Schindl - BestSolution Systemhaus GmbH

EclipseCon 2011 - March 21st 2011

About Me

- CEO BestSolution Systemhaus GmbH
- Eclipse Committer
  - e4
  - Platform UI
  - EMF
- Projectlead: UFaceKit, Nebula
- Member of the Architectual Council

2 ways of single sourcing

- Leverage the Eclipse 4.x compat layer
  - Runs in 3.x
  - Runs in Eclipse 4.x Application Framework + compat layer (e.g. Eclipse 4.x SDK)

2 ways of single sourcing

- Leverage the Eclipse 4.x compat layer
  API clean 3.x bundles run in 4.x
- Adopt the Eclipse 4.x programming model
- Runs in 3.x
- Runs in Eclipse 4.x Application Framework + compat layer (e.g. Eclipse 4.x SDK)
- Runs in Eclipse 4.x Application Framework (=native e4 applications)

The e4 programming model

- JSR 330 (Dependency Injection)
  e.g. @Inject, @Named

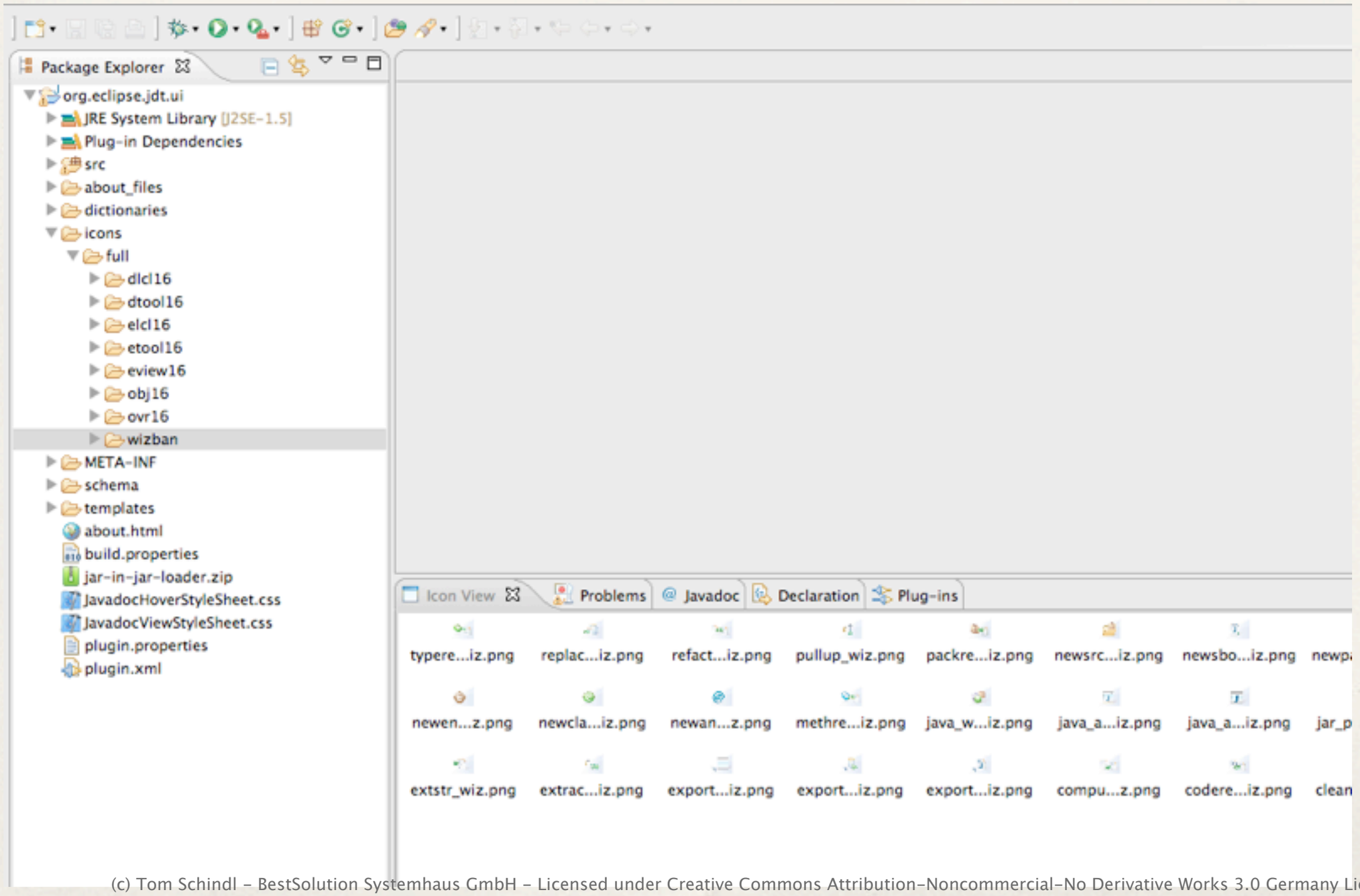
- JSR 250
  e.g. @PostConstruct, @PreDestroy

- Custom Annotation for Eclipse specific stuff
  e.g. @Focus, @Persist

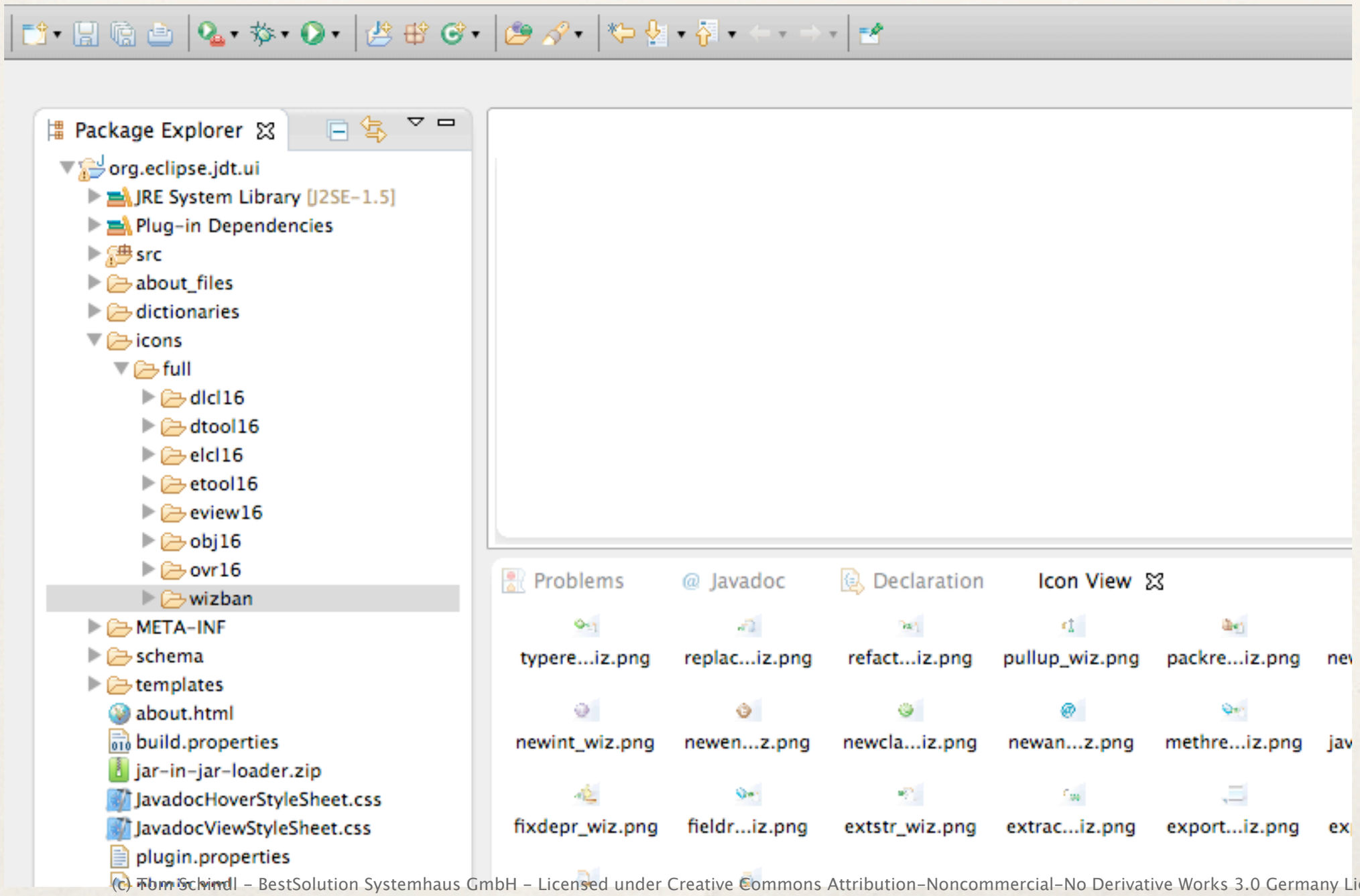
- Building blocks are POJOs with minimal to **NO** dependency on framework

- Advantage Components can be reused in different contexts

Example - IconView in 3.x



Example - IconView in 4.x



Analyze 3.x source code

```
public class IconViewPart extends ViewPart {

  public void createPartControl(Composite parent) {
    parent.setLayout(new FillLayout());
    this.viewer = new GalleryTreeViewer(parent, SWT.MULTI | SWT.V_SCROLL);

    this.workspace = ResourcesPlugin.getWorkspace();
    this.workspace.addResourceChangeListener(resourceListener);

    this.selectionListener = new ISelectionListener() {
      public void selectionChanged(IWorkbenchPart part, ISelection selection) {
        setFolder(selection);
      }
    };
    getSite().getWorkbenchWindow().getSelectionService().addSelectionListener(selectionListener);
  }

  public void dispose() {
    getSite().getWorkbenchWindow().getSelectionService().addSelectionListener(selectionListener);
    this.workspace.removeResourceChangeListener(resourceListener);
    super.dispose();
  }

  public void setFocus() {
    viewer.getControl().setFocus();
  }
}
```

Analyze 3.x source code

```
public class IconViewPart extends ViewPart {

  public void createPartControl(Composite parent) {
    parent.setLayout(new FillLayout());
    this.viewer = new GalleryTreeViewer(parent, SWT.MULTI | SWT.V_SCROLL);

    this.workspace = ResourcesPlugin.getWorkspace();
    this.workspace.addResourceChangeListener(resourceListener);

    this.selectionListener = new ISelectionListener() {
      public void selectionChanged(IWorkbenchPart part, ISelection selection) {
        setFolder(selection);
      }
    };
    getSite().getWorkbenchWindow().getSelectionService().addSelectionListener(selectionListener);
  }

  public void dispose() {
    getSite().getWorkbenchWindow().getSelectionService().addSelectionListener(selectionListener);
    this.workspace.removeResourceChangeListener(resourceListener);
    super.dispose();
  }

  public void setFocus() {
    viewer.getControl().setFocus();
  }
}
```

Split your 3.x source code

```
public class IconView {
  private IWorkspace workspace;

  public IconView(IWorkspace workspace) {
    this.workspace = workspace;
    this.workspace.addResourceChangeListener(resourceListener);
  }

  public void createPartControl(Composite parent) {
    parent.setLayout(new FillLayout());
    this.viewer = new GalleryTreeViewer(parent, SWT.MULTI | SWT.V_SCROLL);
  }

  public void dispose() {
    this.workspace.removeResourceChangeListener(resourceListener);
  }

  public void setFocus() {
    viewer.getControl().setFocus();
  }
}
```

Split your 3.x source code

```
public class IconViewPart extends ViewPart {
  public IconView view;

  public void createPartControl(Composite parent) {
    this.view = new IconView(ResourcesPlugin.getWorkspace());

    this.selectionListener = new ISelectionListener() {
      public void selectionChanged(IWorkbenchPart part, ISelection selection) {
        view.setFolder(selection);
      }
    };
    getSite().getWorkbenchWindow().getSelectionService().addSelectionListener(selectionListener);
  }

  public void dispose() {
    getSite().getWorkbenchWindow().getSelectionService().addSelectionListener(selectionListener);
    this.view.dispose();
    super.dispose();
  }

  public void setFocus() {
    view.setFocus();
  }
}
```

Split up your bundles

- Split up 3 bundles
  - UI-Bundle: Depends only on SWT, JFace, if needed Resources
  - Framework-Integration-Bundle: Integration into 3.x framework
  - Broker-Bundle: Custom Services brokers to seal off from framework services

Summary

- Decoupled UI Code from Eclipse Framework
  - Easier to test
- UI-Code can be used in multiple contexts (e.g. in a Dialog)

Summary

- Decoupled UI Code from Eclipse Framework
  - Easier to test
  - UI-Code can be used in multiple contexts (e.g. in a Dialog)
- ... but we still need to write a lot of glue code because we are not yet using dependency injection

Too much glue, what can we do?

```
public class IconViewPart extends ViewPart {
  public IconView view;

  public void createPartControl(Composite parent) {
    this.view = new IconView(ResourcesPlugin.getWorkspace());

    this.selectionListener = new ISelectionListener() {
      public void selectionChanged(IWorkbenchPart part, ISelection selection) {
        view.setFolder(selection);
      }
    };
    getSite().getWorkbenchWindow().getSelectionService().addSelectionListener(selectionListener);
  }

  public void dispose() {
    getSite().getWorkbenchWindow().getSelectionService().addSelectionListener(selectionListener);
    this.view.dispose();
    super.dispose();
  }

  public void setFocus() {
    view.setFocus();
  }
}
```

Introducing the e4-bridge

- Provides a DI-Container inside the 3.x workbench
- Provides often needed Broker-Services (e.g. publishing selection, C&P integration)
- Provides services for NLS and Resource Management (Font,Color,Image)

Let‘s reduce the glue!

```
public class IconViewPart extends ViewPart {
  public IconView view;

  public void createPartControl(Composite parent) {
    this.view = new IconView(ResourcesPlugin.getWorkspace());

    this.selectionListener = new ISelectionListener() {
      public void selectionChanged(IWorkbenchPart part, ISelection selection) {
        view.setFolder(selection);
      }
    };
    getSite().getWorkbenchWindow().getSelectionService().addSelectionListener(selectionListener);
  }

  public void dispose() {
    getSite().getWorkbenchWindow().getSelectionService().addSelectionListener(selectionListener);
    this.view.dispose();
    super.dispose();
  }

  public void setFocus() {
    view.setFocus();
  }
}
```

Let‘s reduce the glue!

```
public class IconViewPart extends DIViewPart<IconView> {

    public IconViewPart() {
        super(IconView.class);
    }
}
```

Let‘s reduce the glue!

```
public class IconViewPart extends DIViewPart<IconView> {

    public IconViewPart() {
        super(IconView.class);
    }
}
```

How the hell does that work?

We use some annotations

```
public class IconView {
  private IWorkspace workspace;

  public IconView(IWorkspace workspace) {
    this.workspace = workspace;
    this.workspace.addResourceChangeListener(resourceListener);
  }

  public void createPartControl(Composite parent) {
    parent.setLayout(new FillLayout());
    this.viewer = new GalleryTreeViewer(parent, SWT.MULTI | SWT.V_SCROLL);
  }

  public void dispose() {
    this.workspace.removeResourceChangeListener(resourceListener);
  }

  public void setFocus() {
    viewer.getControl().setFocus();
  }

  public void setFolder(Object selection) {
  }
}
```

We use some annotations

```
public class IconView {
  private IWorkspace workspace;

  @Inject
  public IconView(IWorkspace workspace) {
    this.workspace = workspace;
    this.workspace.addResourceChangeListener(resourceListener);
  }

  @PostConstruct
  public void createPartControl(Composite parent) {
    parent.setLayout(new FillLayout());
    this.viewer = new GalleryTreeViewer(parent, SWT.MULTI | SWT.V_SCROLL);
  }

  @PreDestroy
  public void dispose() {
    this.workspace.removeResourceChangeListener(resourceListener);
  }

  @Focus
  public void setFocus() {
    viewer.getControl().setFocus();
  }

  @Inject
  public void setFolder(@Named(IServiceConstants.ACTIVE_SELECTION) @Optional Object selection) {
  }
}
```

e4 bridge DI container

- Provides fairly the same features default e4 DI-Container does
  - Injection of OSGi-Services
  - Support for @Focus, @Persist
  - Support for @EventTopic, @UIEventTopic and IEventBroker
  - Support for @Preference

e4 bridge DI container

- Provides fairly the same features default e4 DI-Container does
  - Injection of OSGi-Services
  - Support for @Focus, @Persist
  - Support for @EventTopic, @UIEventTopic and IEventBroker
  - Support for @Preference
- ... and it goes beyond that and adds new concepts for pure e4 and 3.x integration

e4 bridge DI extension

- Resource Management - Image-Management in particular

```
public class ModelEditor extends EditorPart {

  public void createPartControl(Composite composite) {
    Label l = new Label(composite, SWT.NONE);
    l.setImage(
      Activator.imageDescriptorFromPlugin(Activator.PLUGIN_ID, "/resource/myimage.png").createImage()
    );
  }
}
```

e4 bridge DI extension

- Resource Management - Image-Management in particular

```
public class ModelEditor {

  @Inject
  private final IResourcePool resourcePool;

  @PostConstruct
  void init(Composite composite) {
    Label l = new Label(composite, SWT.NONE);
    l.setImage(resourcePool.getImageUnchecked(ResourceProvider.IMG_Obj16_cross));
  }

}
```

e4 bridge DI extension

- NLS / Externalized String support

```
public class Messages {
  public static String MyLabel;

  NLS.initializeMessages(Messages.class.getName(), Messages.class);
}
```

```
public class ModelEditor {

  public void createPartControl(Composite composite) {
    Label l = new Label(composite, SWT.NONE);
    l.setText(Messages.MyLabel);
  }

}
```

e4 bridge DI extension

- NLS / Externalized String support

```
public class Messages {
    public String MyLabel;
}
```

```
public class ModelEditor extends EditorPart {

  @Inject
  @Translation
  private Messages messages;

  @PostConstruct
  void init(Composite composite) {
    Label l = new Label(composite, SWT.NONE);
    l.setText(messages.MyLabel);
  }

}
```

e4 bridge - broker services

- IClipboardService: Provide access to C&P-Framework of the workbench (currently only available in 3.x and 4.x when running with compat layer)

```
public class ModelEditor {

  @Inject
  @Optional
  private IClipboardHandler clipboardService;

  @Focus
  public void setFocus() {
    if (clipboardHandler == null) {
      clipboardHandler = new ClipboardHandler();
    }
    if (clipboardService != null) {
      clipboardService.setHandler(clipboardHandler);
    }
    viewer.getControl().setFocus();
  }
}
```

e4 bridge - broker services

- IClipboardService: Provide access to C&P-Framework of the workbench (currently only available in 3.x and 4.x when running with compat layer)

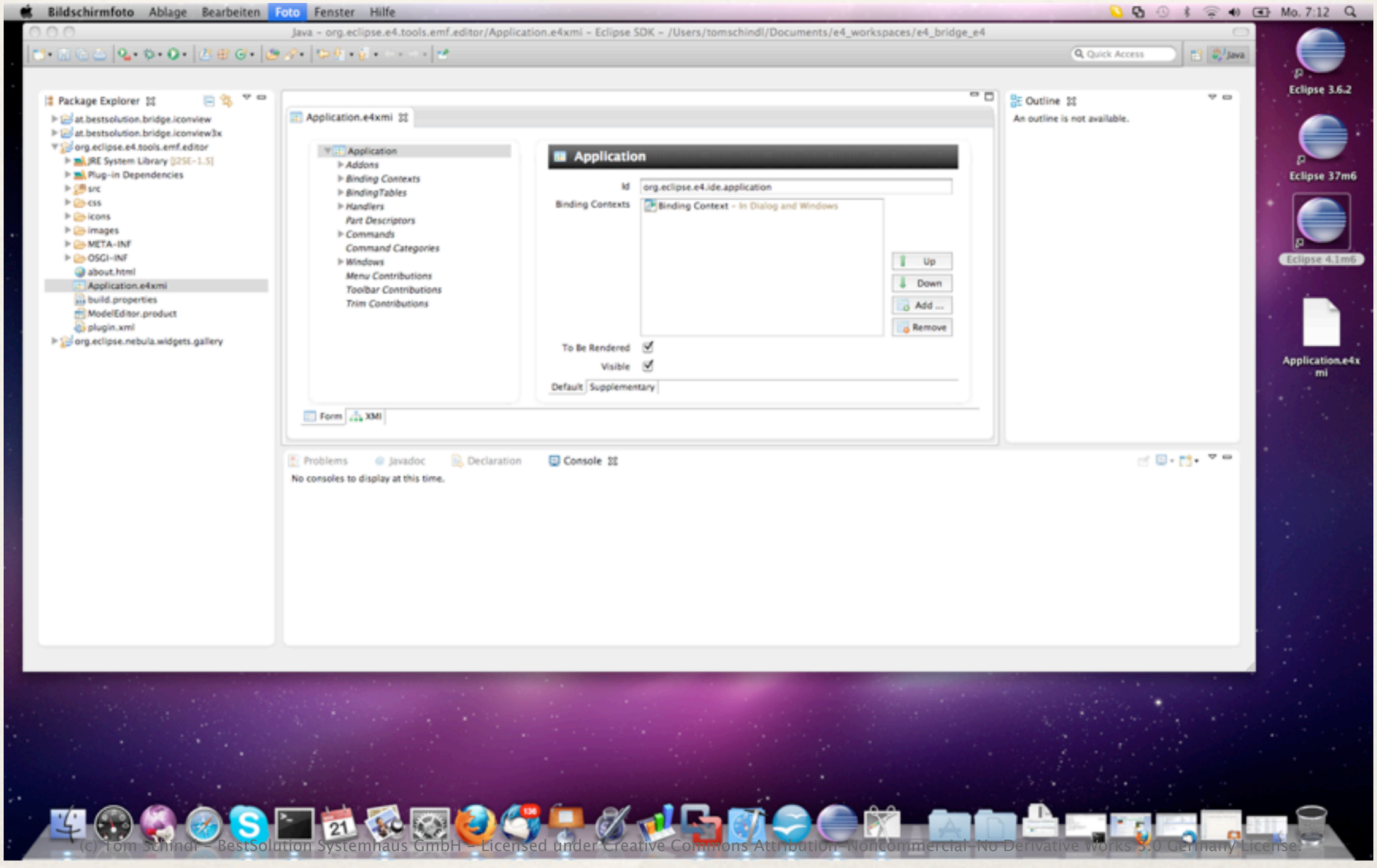
```
public class ModelEditor {

  @Inject
  @Optional
  private IClipboardHandler clipboardService;

  @Focus
  public void setFocus() {
    if (clipboardHandler == null) {
      clipboardHandler = new ClipboardHandler();
    }
    if (clipboardService != null) {
      clipboardService.setHandler(clipboardHandler);
    }
    viewer.getControl().setFocus();
  }
}
```

```
public interface Handler {
  public void paste();
  public void copy();
  public void cut();
}
```

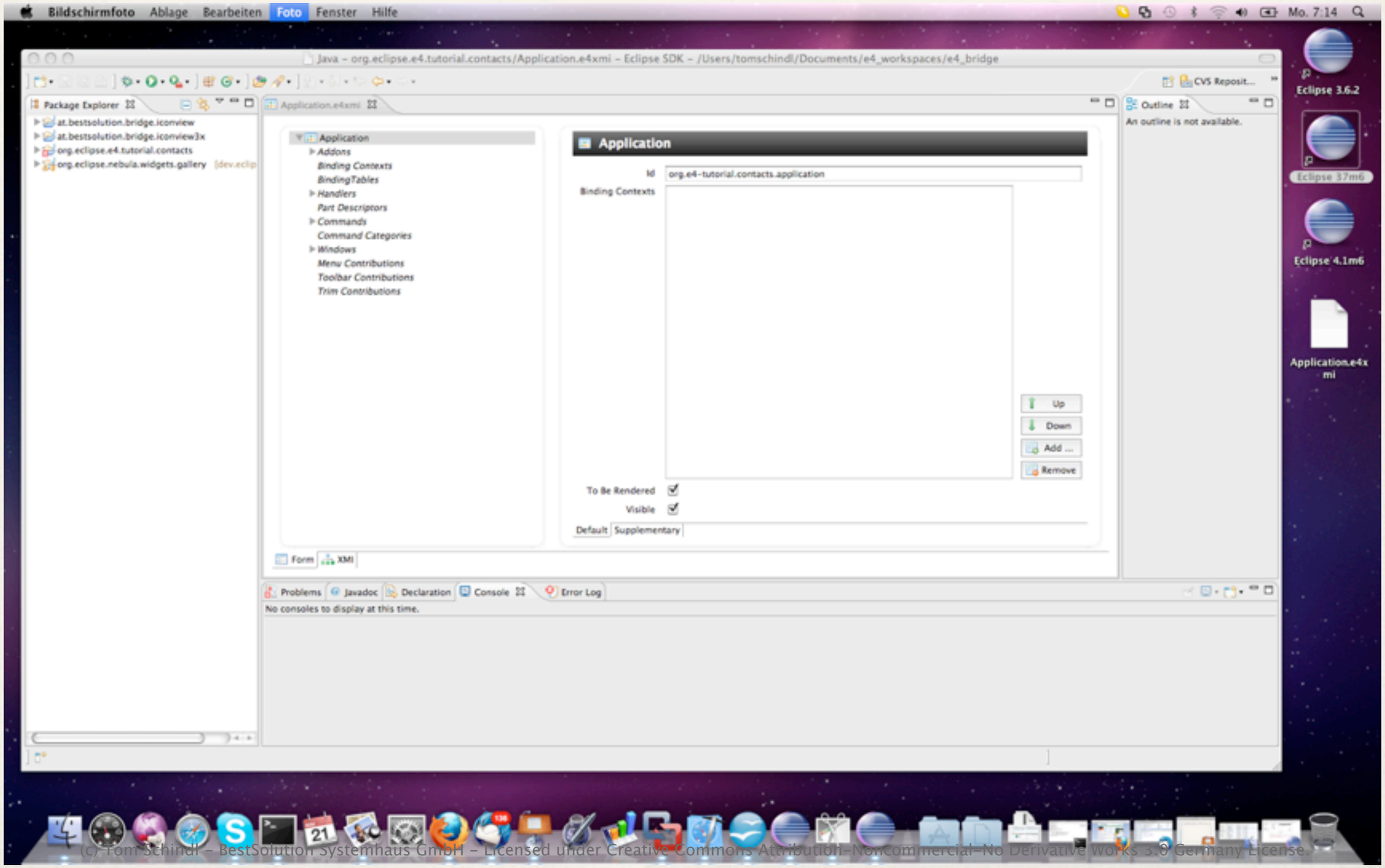
e4 bridge - broker services

- ISelectionProviderService: Can be used to push selection back into the framework
- IDirtyProviderService: Can be used to set the dirty state of a View/EditorPart

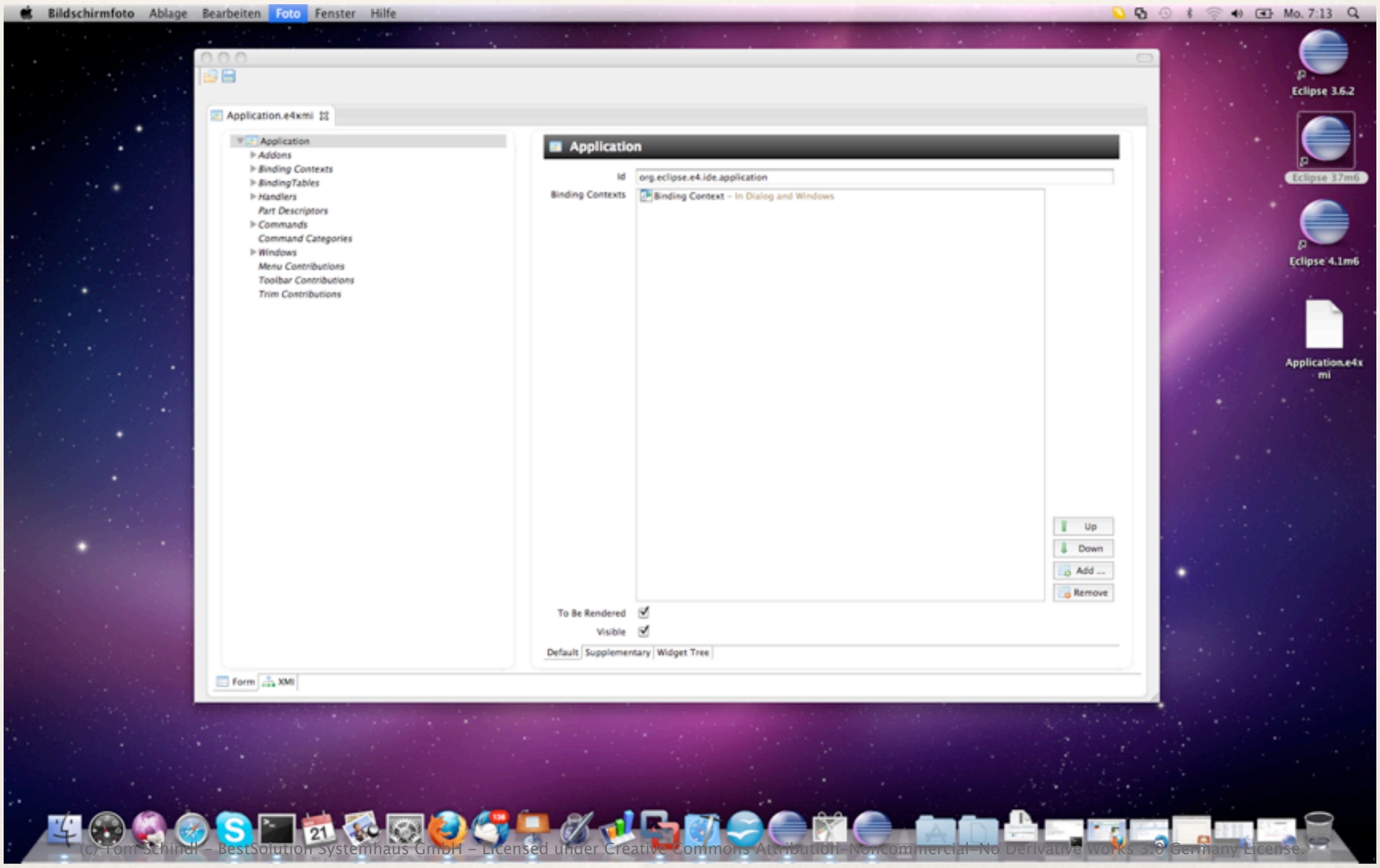
e4 model tooling is THE reference



e4 model tooling is THE reference



e4 model tooling is THE reference



Additional Information

- Blog: http://tomsondev.bestsolution.at
- e4 Newsgroup: eclipse.e4
- e4 Mailingliste: e4-dev@eclipse.org
- e4 wiki: http://wiki.eclipse.org/e4