EMF Parsley: a toolbox for UI parts built on EMF

Lorenzo Bettini

Dipartimento di Informatica, Università di Torino, Italy
itemis Schweiz Consultant

joint work with **Francesco Guidieri** and **Vincenzo Caselli**
RCP-Vision

EclipseDay Florence 2013

Boilerplate code

```
adapterFactory = new ComposedAdapterFactory
    (ComposedAdapterFactory.Descriptor.Registry.INSTANCE);
adapterFactory.addAdapterFactory(new ResourceItemProviderAdapterFactory());
adapterFactory.addAdapterFactory(new MyModelItemProviderAdapterFactory());
adapterFactory.addAdapterFactory(new ReflectiveItemProviderAdapterFactory());

BasicCommandStack commandStack = new BasicCommandStack();
commandStack.addCommandStackListener
  (new CommandStackListener() {...});

editingDomain = new AdapterFactoryEditingDomain(adapterFactory, commandStack, ...);

Tree tree = new Tree(composite, SWT.MULTI);
TreeViewer viewer = new TreeViewer(tree);

viewer.setContentProvider(new AdapterFactoryContentProvider(adapterFactory));
viewer.setLabelProvider(new AdapterFactoryLabelProvider(adapterFactory));
viewer.setInput(editingDomain.getResourceSet());

new AdapterFactoryTreeEditor(viewer.getTree(), adapterFactory);
```

Customizing Generated Code

Follow the `@generated` pattern:

```
public class BookItemProvider extends ... {
  /** @generated NOT */
  @Override
  public Object getImage(Object object) {
    return overlayImage(object,
        getResourceLocator().getImage("mypath/custom_book.png"));
  }

  /** @generated NOT */
  @Override
  public String getText(Object object) {
    return "Book: " + " " + ((Book)object).getTitle();
  }
```

Emf Parsley

- A lightweight framework that allows easy and quick development of applications based on EMF.
- It can be configured to use all kinds of EMF persistence implementations (e.g., XMI, Teneo, CDO).
- It aims at providing a set of reusable components like trees, tables and detail forms that manage the model with the introspective EMF capabilities.
- Using these components one can easily build more complex widgets, editors and applications.
- The framework provides basic UI implementations which are customizable with injection mechanisms.
- A **DSL** to make customizations much easier.

Small Reusable and Customizable Blocks

- Split and delegate responsibilities into small classes.
- Easy to customizing a single aspect without customizing the components themselves.
- The custom version will be **injected** in the framework (Google Guice).
- All components relying on that aspect will be assured to use our specific version.

Specify Custom Bindings

A custom reflective way of specifying bindings (inspired by Xtext)

```
public class MyEmfComponentsModule extends EmfComponentsGuiceModule {

public Class<? extends ResourceLoader> bindResourceLoader() {
    return MyResourceLoader.class;
}

public Class<? extends EmfViewerMouseAdapter> bindEmfViewerMouseAdapter() {
    return MyEmfViewerMouseAdapter.class;
}

public Class<? extends FeatureLabelProvider> bindFeatureLabelProvider() {
    return MyFeatureLabelProvider.class;
}

}
```

What we provide

- Viewers and Composites:
  - TreeViewer, TableViewer, FormComposite
- Editors
- Views
  - Saveable view (based on Resource)
  - Based on selection

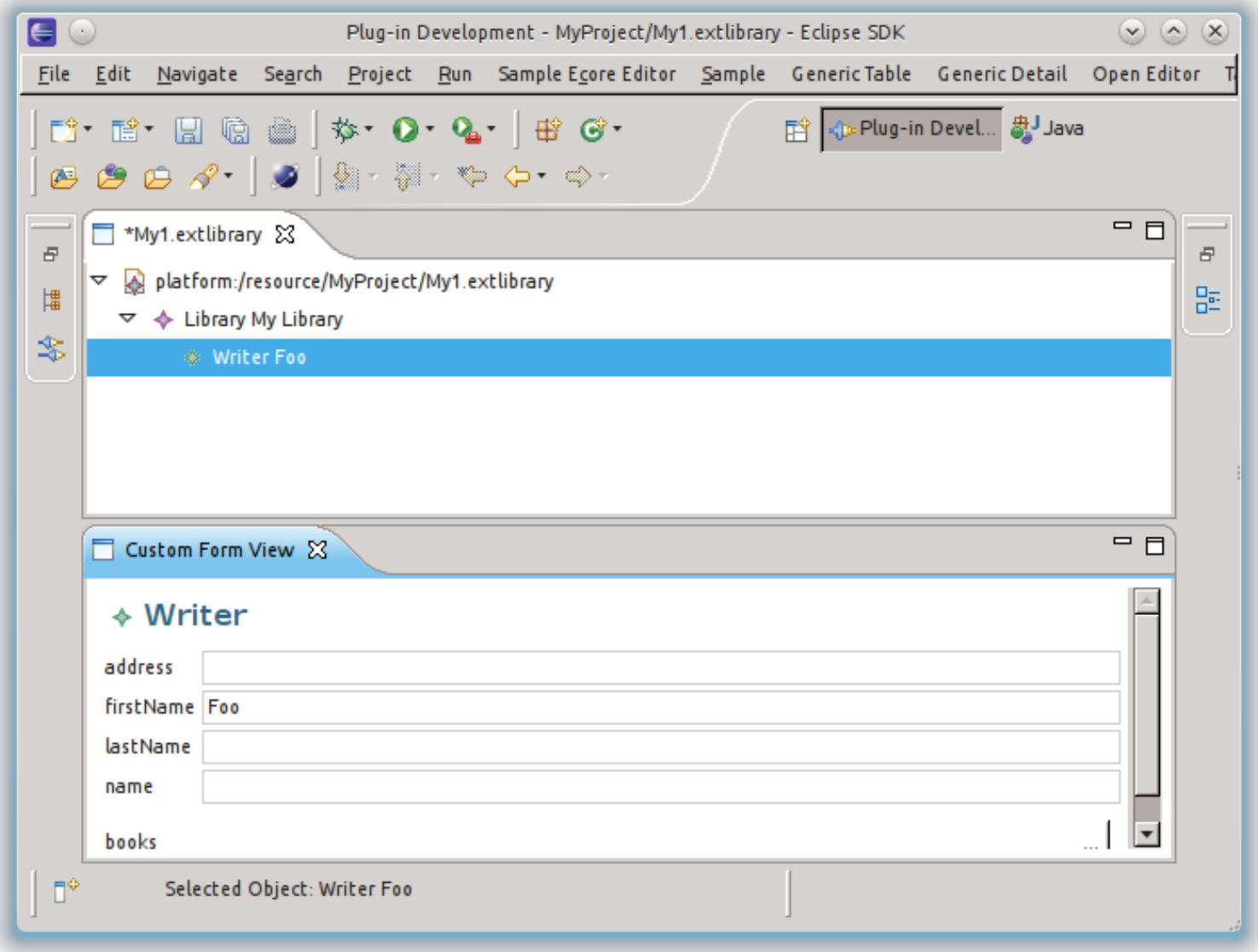
Create a TreeViewer

```
public class MyView extends ViewPart {

    @Inject ViewerInitializer initializer;

    @Override
    public void createPartControl(Composite parent) {
        ...
        viewer = new TreeViewer(parent, ...);
        // initialize with an EMF Resource
        initializer.initialize(viewer, resource);
        // or alternatively, if you have an EObject
        initializer.initialize(viewer, eObject);
    }
```

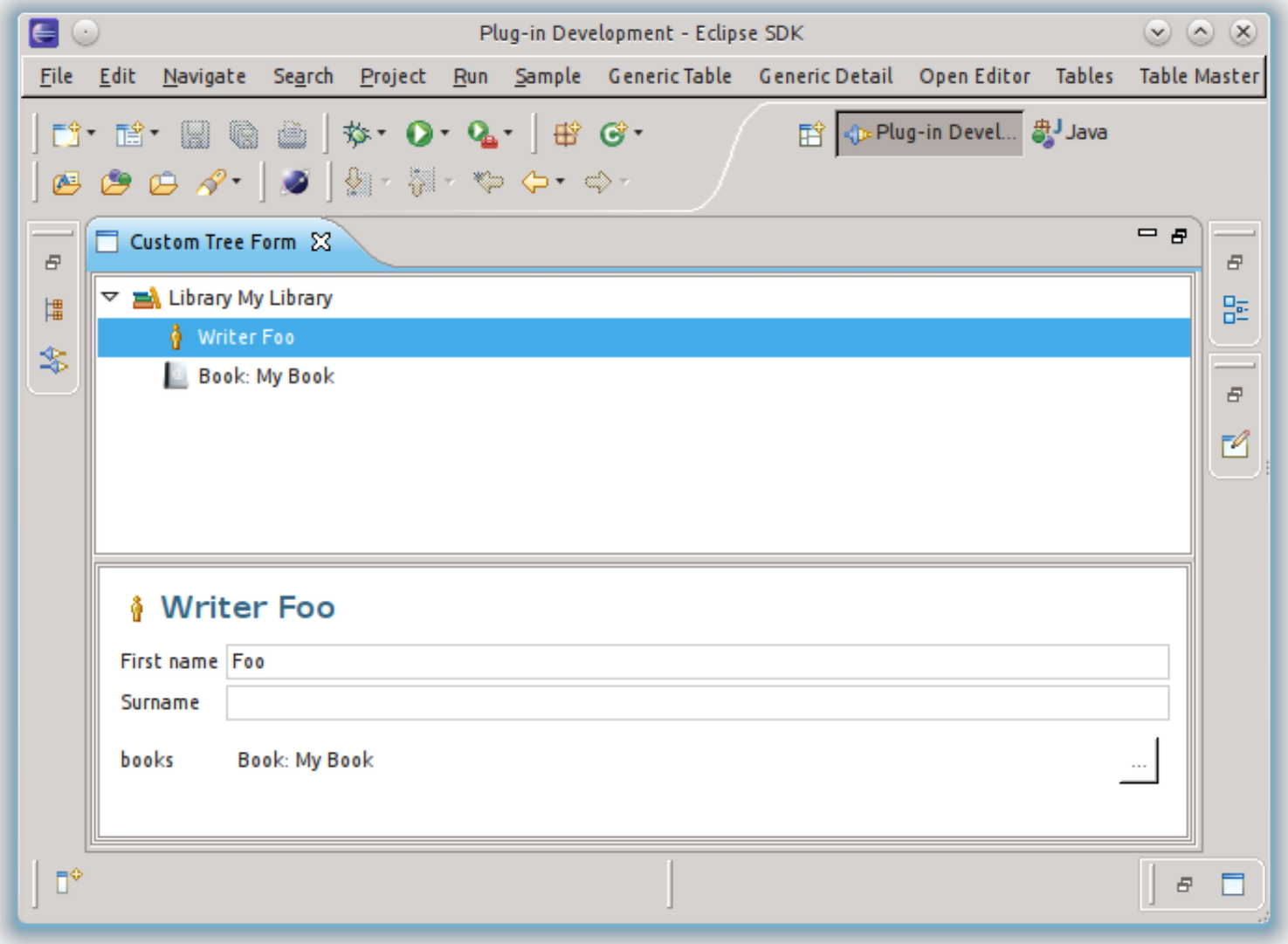
Form Composites

```
public class MyView extends ViewPart
    implements ISelectionChangedListener {

    @Inject FormFactory factory;

    public void selectionChanged(SelectionChangedEvent event) {
        EObject selectedObject = getFirstSelectedEObject
                (event.getSelection());

        // relevant lines
        FormDetailComposite detailForm = factory.
            createFormDetailComposite(parent, SWT.BORDER);
        detailForm.init(selectedObject);
    }
```

Using Components



Using Components



Declarative Customization

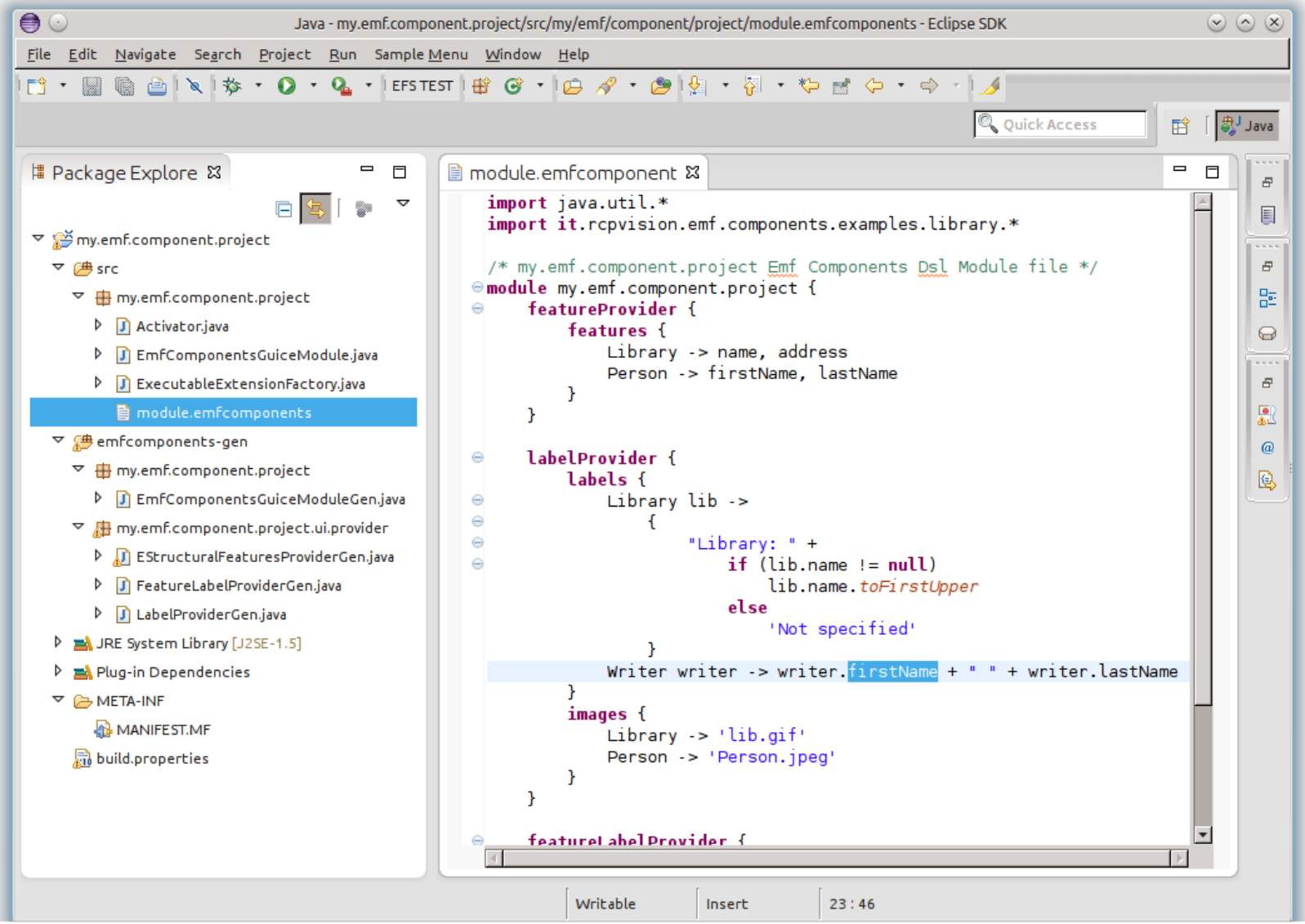
```
public class CustomLabelProvider extends ViewerLabelProvider {
    public String text(Book book) {
        return "Book: " + book.getTitle();
    }

    public String image(Book book) {
        return "book.png";
    }

    public String text(Borrower b) {
        return "Borrower: " + b.getFirstName();
    }
}
```

Methods selected at run-time with polymorphic dispatch according to the run-time type of the argument.

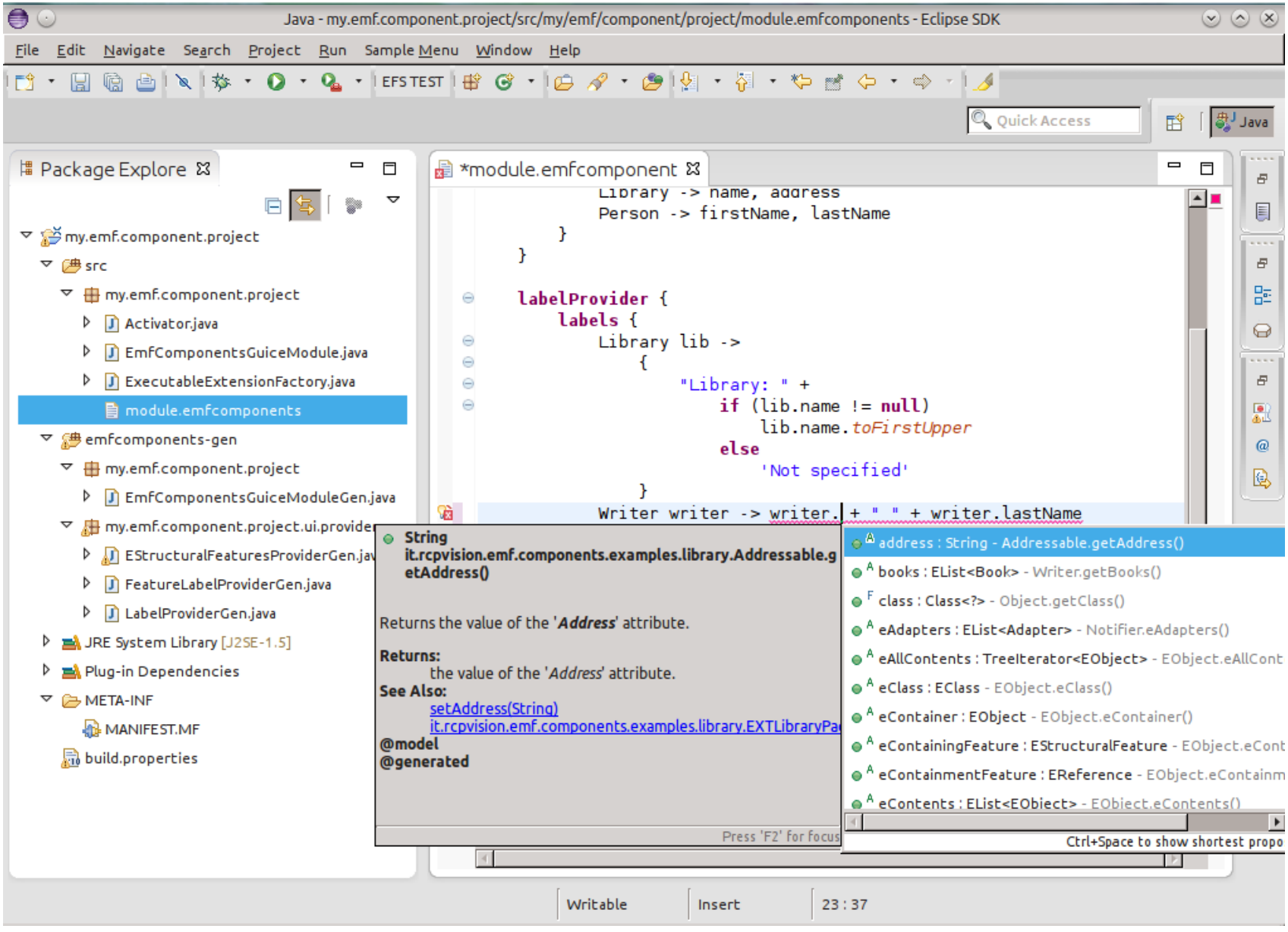
Custom Bindings

```
public class MyCustomModule extends EmfComponentsGenericModule {
    public Class<? extends CompositeLabelProvider>
            bindCompositeLabelProvider() {
        return CustomLabelProvider.class;
    }
    ...
}
```

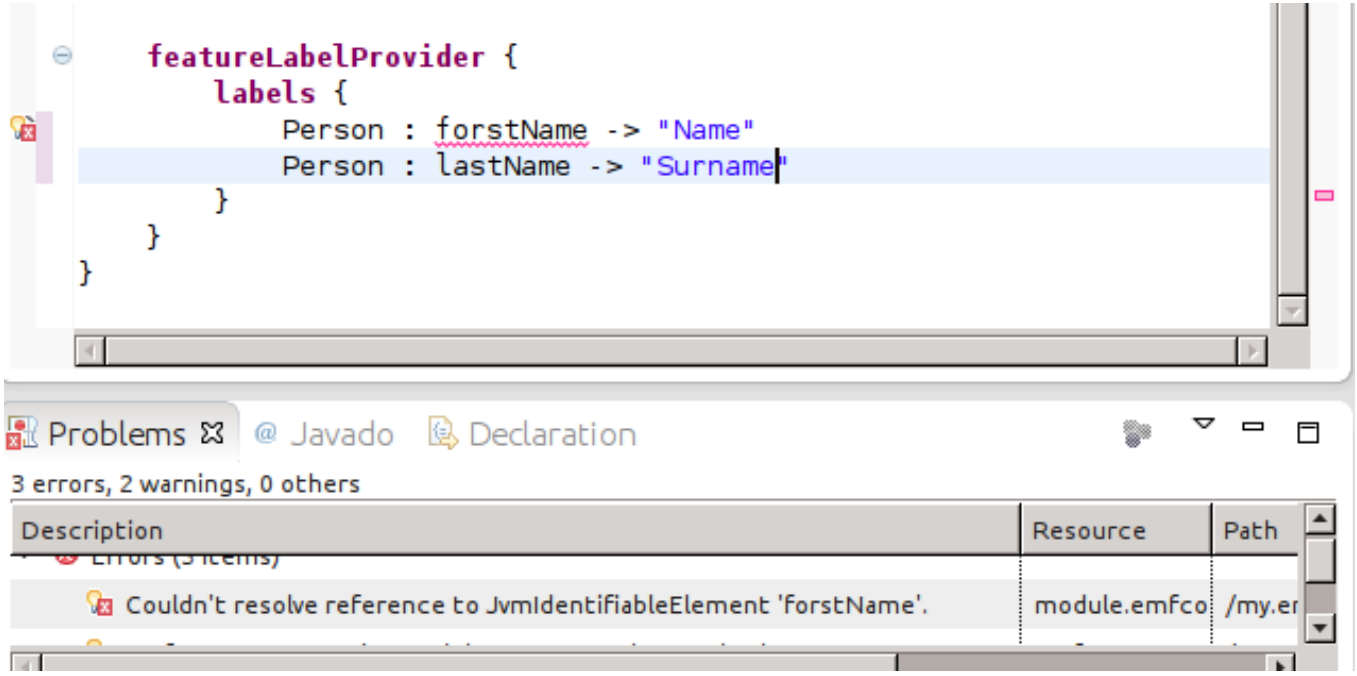
A DSL implemented in Xtext/Xbase



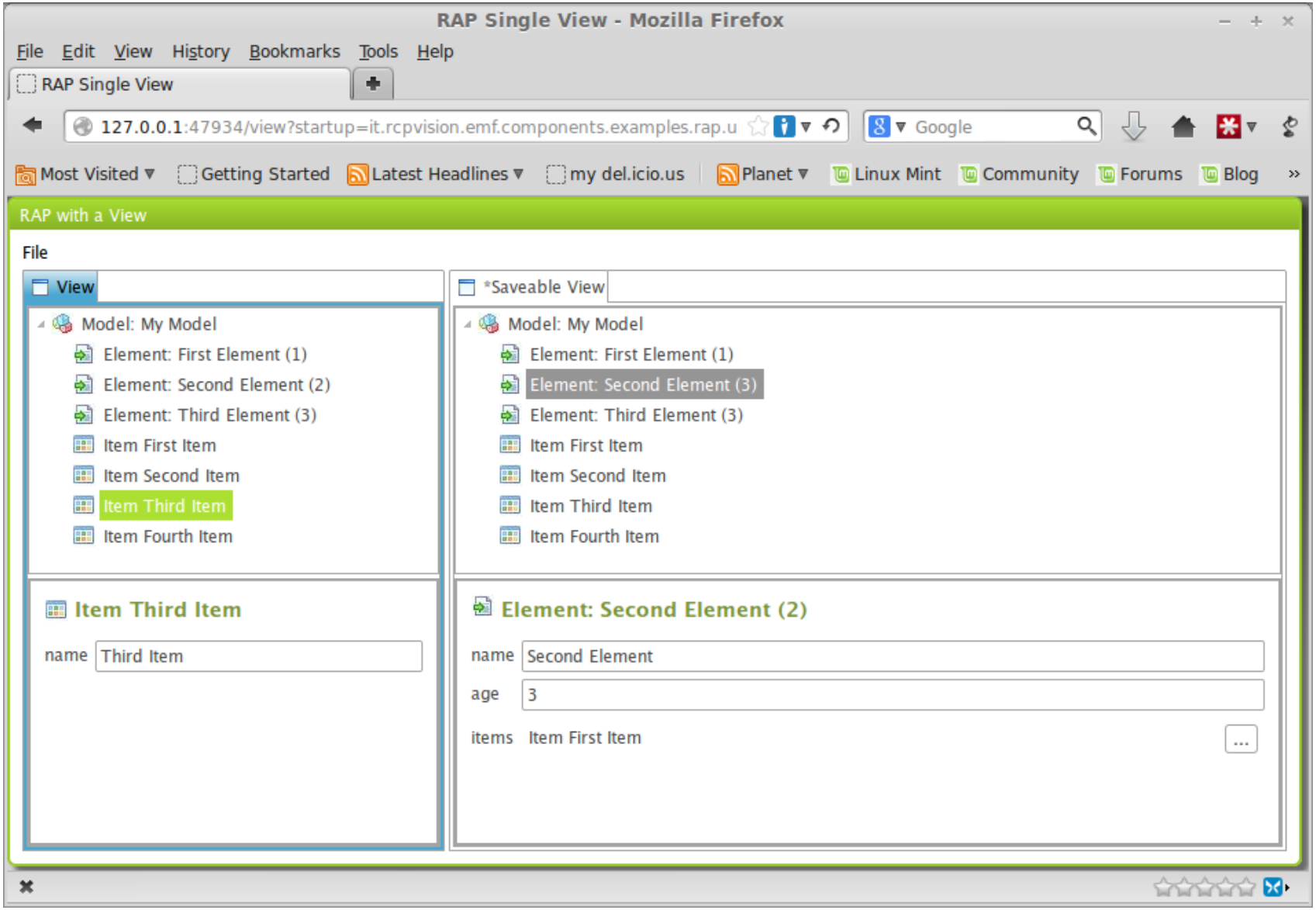
Tooling fully integrated with Java



More Static Checks



Works with RAP



Links

Eclipse project proposal:
http://eclipse.org/proposals/modeling.emfparsley/
https://bugs.eclipse.org/bugs/show_bug.cgi?id=400609

Current implementation:
http://code.google.com/a/eclipselabs.org/p/emf-components/

Discussion thread:
http://www.eclipse.org/forums/index.php/t/457873/

Links

Eclipse project proposal:
http://eclipse.org/proposals/modeling.emfparsley/
https://bugs.eclipse.org/bugs/show_bug.cgi?id=400609

Current implementation:
http://code.google.com/a/eclipselabs.org/p/emf-components/

Discussion thread:
http://www.eclipse.org/forums/index.php/t/457873/

Thanks!