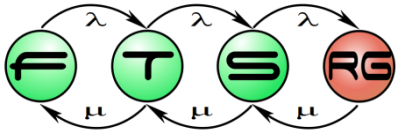
The New Generation of the Eclipse Platform



Eclipse RCP

- For developing client applications
  - Based on the Eclipse workbench model
  - But without IDE functionality
    - Or possibly including a limited function set
- Since Eclipse 3.0 (2004)
  - Eclipse IDE is a specific Eclipse application

Eclipse 4.x application platform

Eclipse Application Platform 4.x

- Rework of Eclipse RCP platform
- Goals
  - Easier programming
  - Better reuse
  - Compatibility
    - Using a compatibility layer

Eclipse Application Platform 4.x

- Most important new functions
  - Context handling
    - Platform service access
    - Dependency Injection
  - EMF-based workbench model
    - Editor and view are not separate entities
  - CSS based theming
    - RCP application should look different to IDE

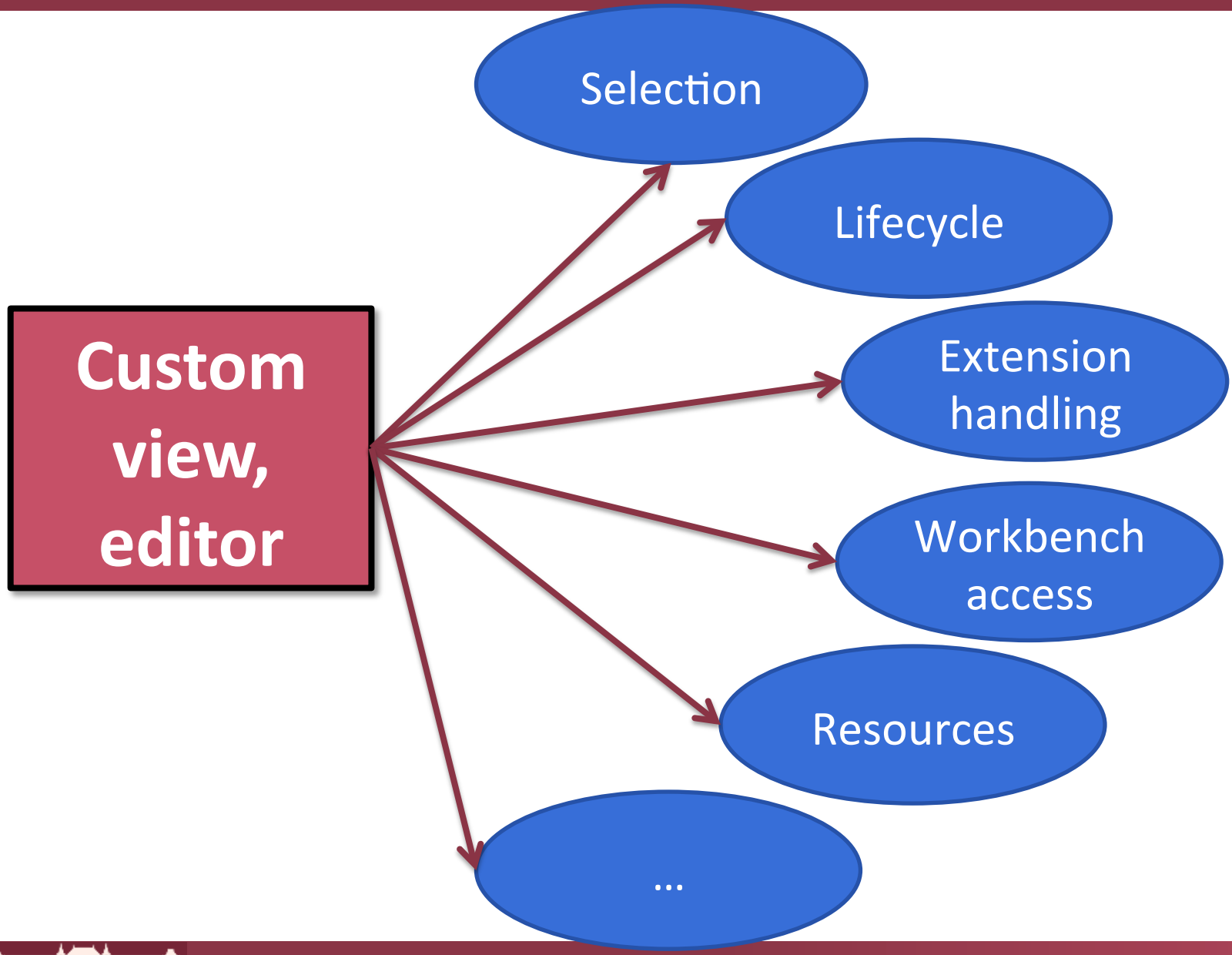
Glossary

- e4
  - Incubation project at eclipse.org
  - Collection of new generation techniques
  - Eclipse 4.x Application Platform started here
- Eclipse 4.x
  - New release of the platform
  - New services
  - Can be used a stable base software

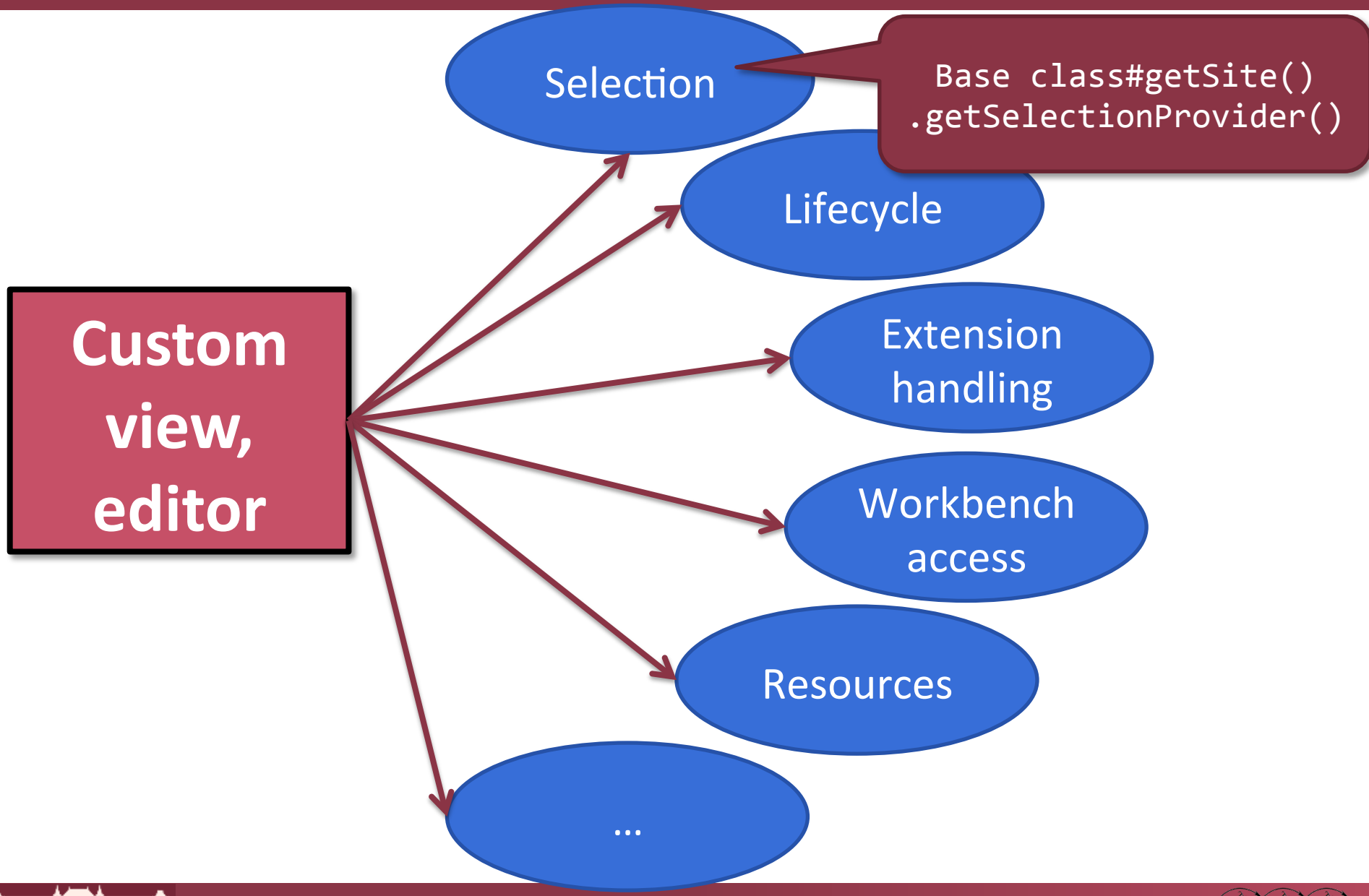
Accessing services in the 3.x platform

Based on the “What’s the context?” talk
from EclipseCon 2009

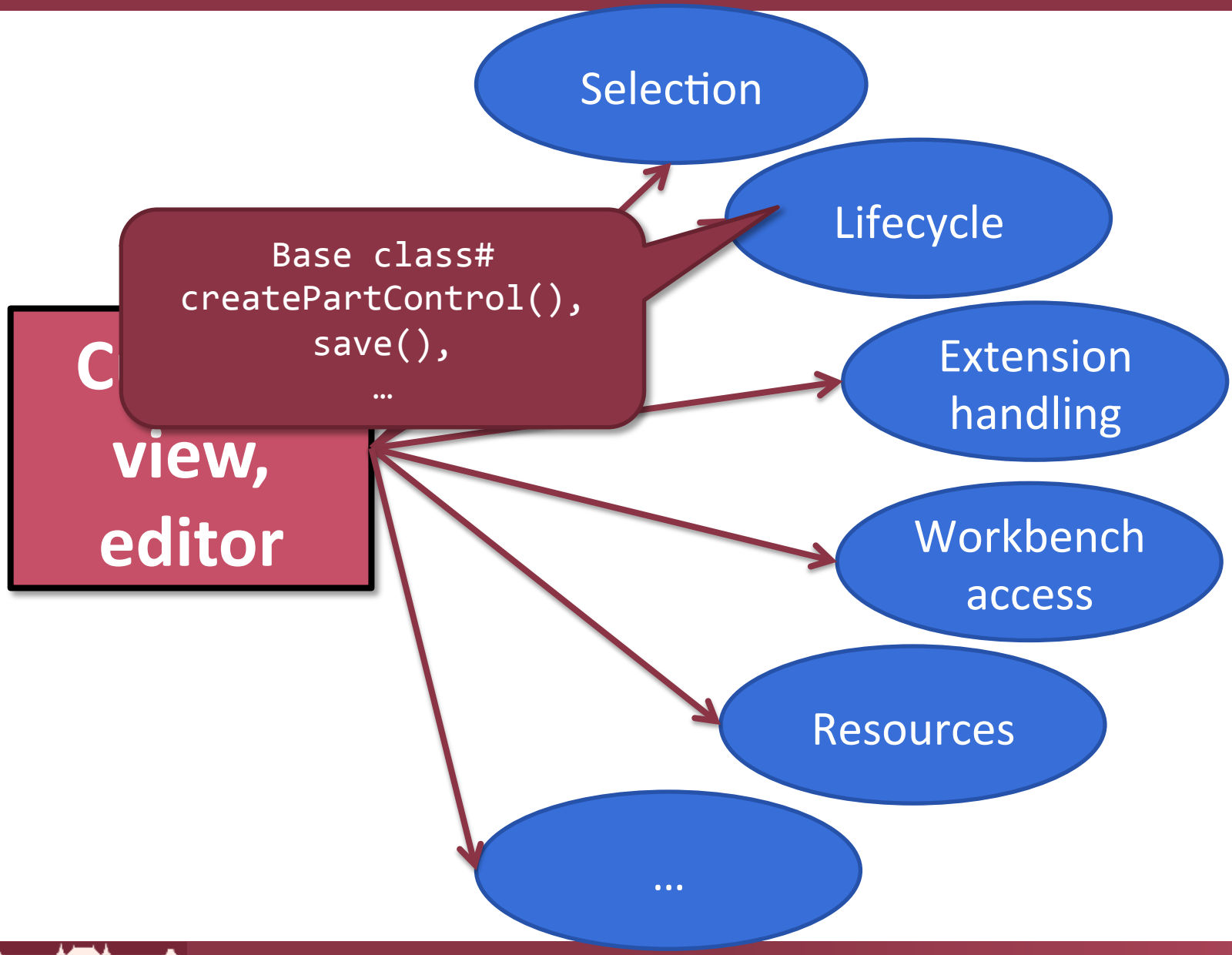
Service access



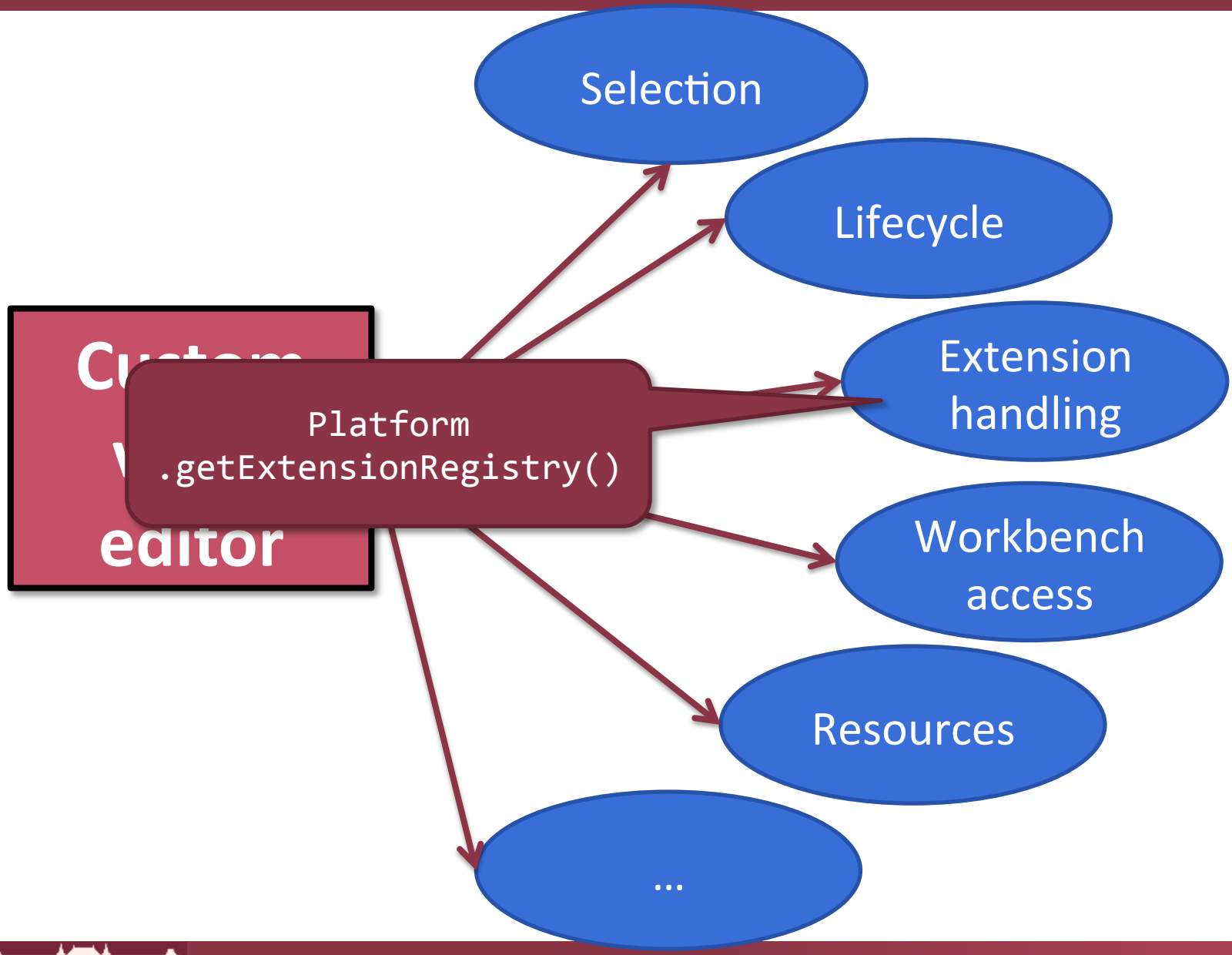
Service access



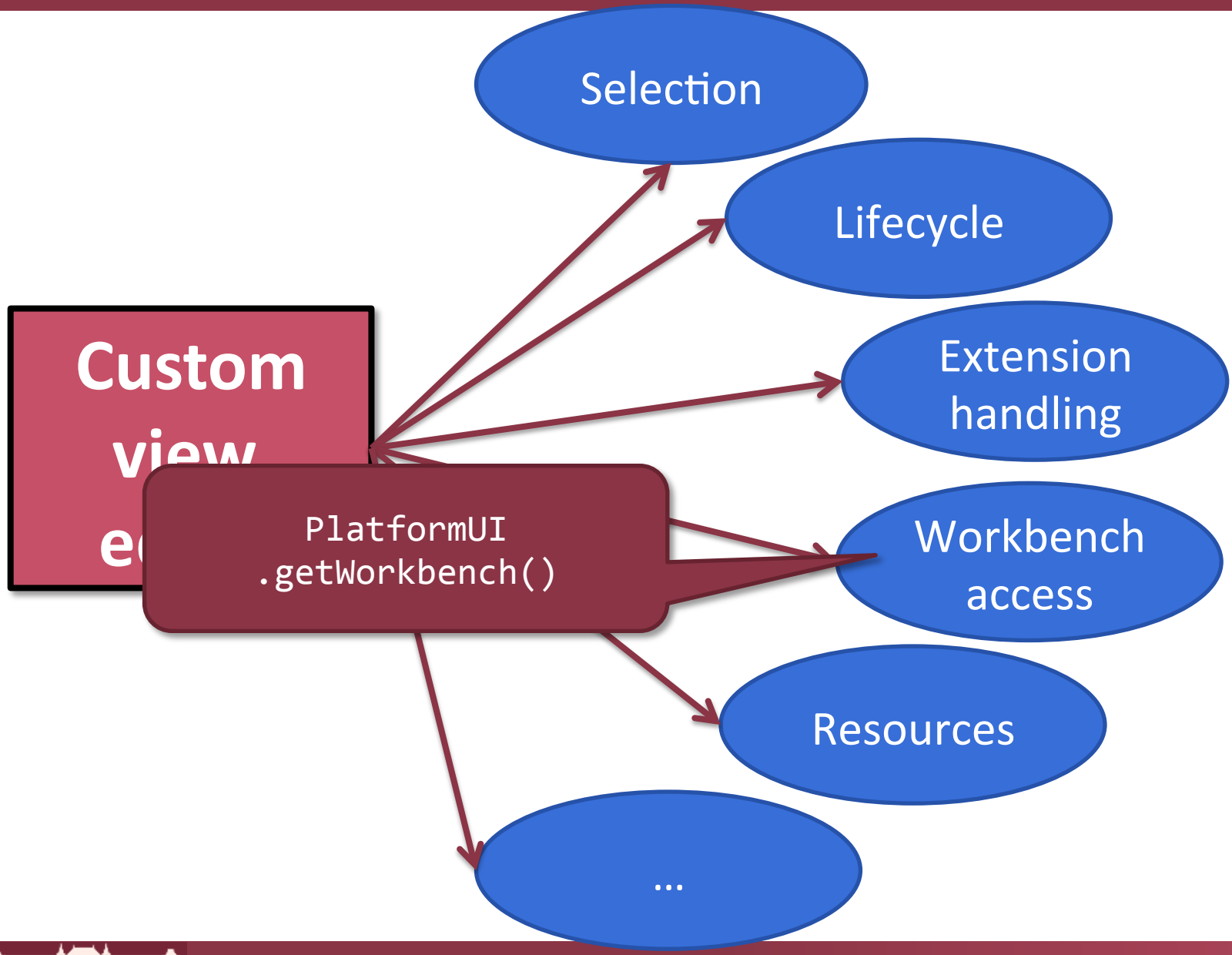
Service access



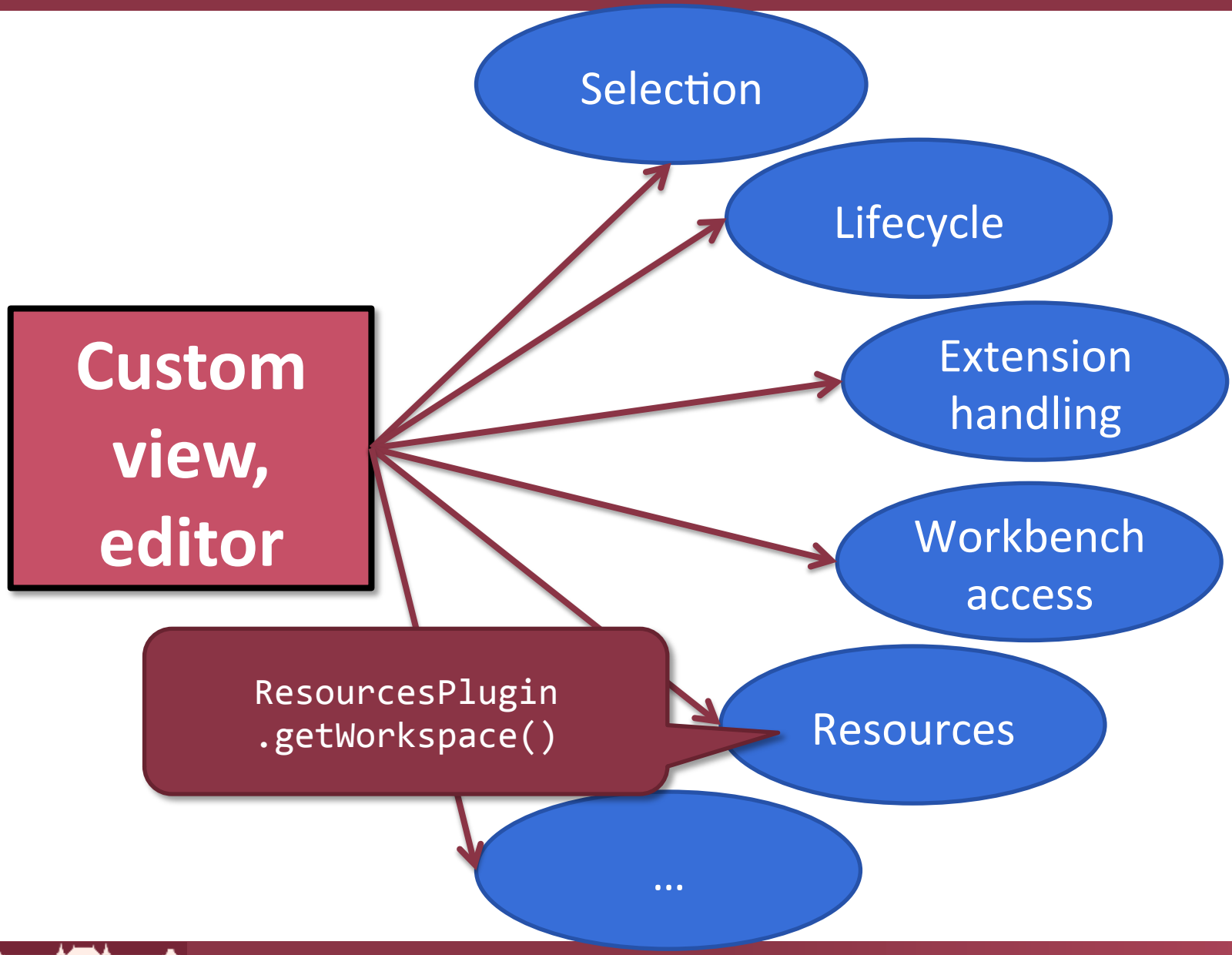
Service access



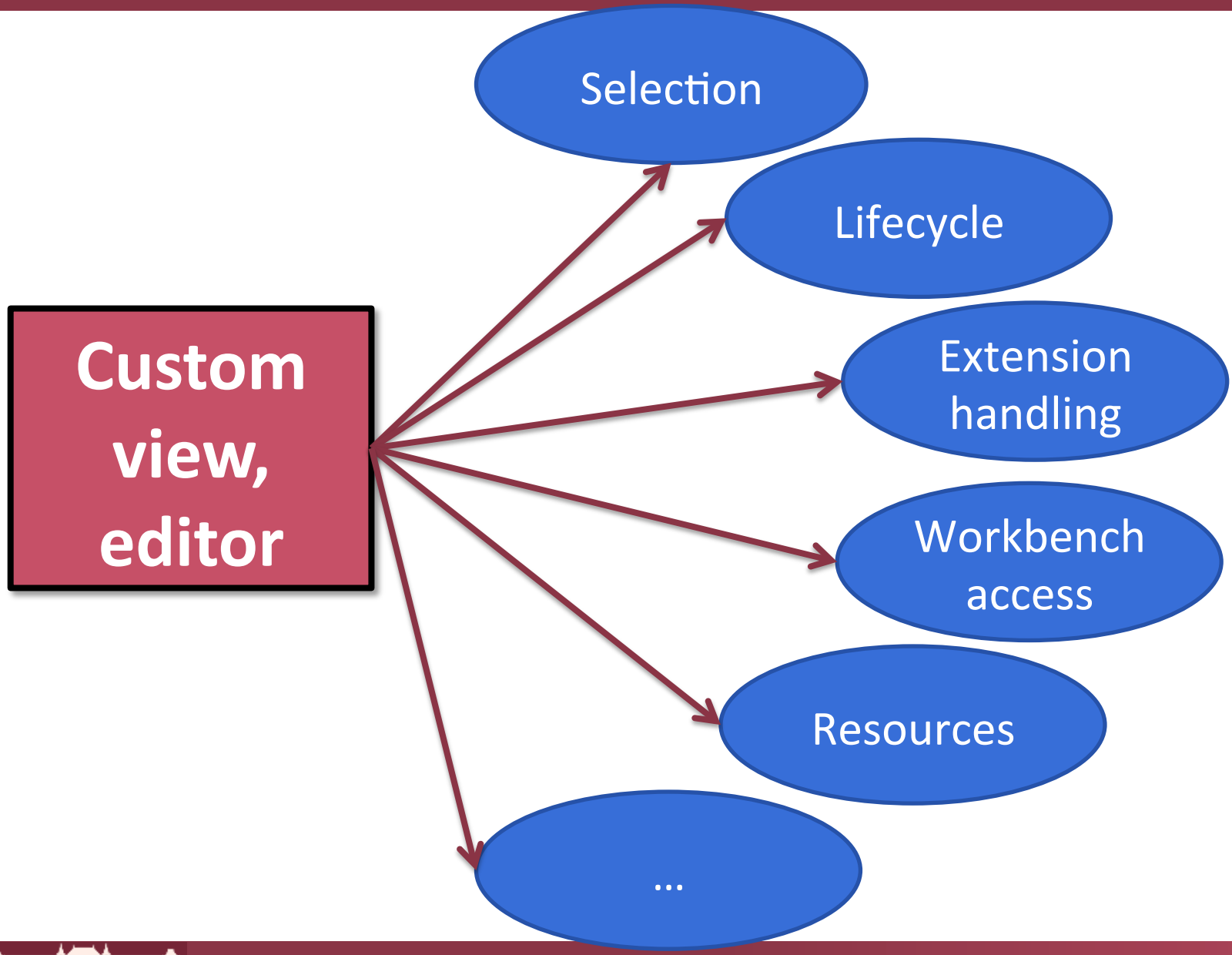
Service access



Service access



Service access



Singleton service providers

- Accessing common services
  - Not always provided by the Platform itself
- Nice and confortable solution
  - Simple client code
  - Information hiding: returning service interface
- …At first glance
  - What is the problem?

Singleton service providers

- Accessing common services
  - Not always provided by the Platform itself
- Nice and confortable solution
  - Simple client code
  - Information hiding: returning service interface
- …At first glance
  - What is the problem?

Singleton service providers

- Accessing common services
  - Not always provided by the Platform itself
- Nice and confortable solution
  - Simple client code
  - Information hiding: returning service interface
- …At first glance
  - What is the problem?

Singleton service providers

- Alternative providers?
  - Service stub for testing
  - Remote services
- Reuse services at different locations
  - Using editor (or view) inside a dialog
    - E.g. textual editor in a Preference page

Singleton service providers

- Alternative providers?
  - Service stub for testing
  - Remote services
- Reuse services at different locations
  - Using editor (or view) inside a dialog
    - E.g. textual editor in a Preference page

Environment hard-coded in a single plug-in

Hardcoded environment - example

- Eclipse IDE
  - Exactly one IWorkspace
  - Client: ResourcesPlugin.getWorkspace()
- Bespin IDE
  - Experimental project from 2009 (Mozilla)
  - IDE in the web
  - Goal: different IWorkspace instances for each user
  - Client
    - Removing singleton references is a lot of work

Defining custom services

- Singleton provider
  - Such as Eclipse 3.x platform
- OSGi service
  - Easier to integrate
    - Manager classes for service access
    - OSGi code written by hand
  - Declarative services
    - Direct integration problematic
    - Instantiation by DS or Eclipse?

Event Handling – I.

- Event
  - If a data is interesting, usually its changes as well!
- Many event types
  - Service appearance/disappearance
  - Service specific listener
    - Resource changes
    - Selection Service
  - User interface
    - Data binding
    - Editor/view lifecycle events

Example: Workbench window change

- Several thousand events
  - Main cause: Complex event chains (event storm)

Name	Invocation Count	
ExpressionAuthority.getCurrentState()	10,593	100%
ces.EvaluationService.getCurrentState()	2,196	21%
ces.ExpressionAuthority.evaluate(IEvaluationResultCache)	8,397	79%
ntexts.ContextAuthority.containsActive(Collection)	108	1%
ntexts.ContextAuthority.sourceChanged(int)	135	1%
rvices.EvaluationAuthority.refsWithSameExpression(EvaluationReference[])	8,154	77%
l.services.EvaluationAuthority.sourceChanged(String[])		
rnal.services.ExpressionAuthority.sourceChanged(int, String[])		
internal.services.ExpressionAuthority.sourceChanged(int, Map)		
.ui.AbstractSourceProvider.fireSourceChanged(int, Map)		
lipse.ui.internal.services.WorkbenchSourceProvider.access$10(WorkbenchSourceProvider, int, Map)	3,150	30%
lipse.ui.internal.services.WorkbenchSourceProvider.checkActivePart(boolean)	5,004	47%
.eclipse.ui.internal.services.WorkbenchSourceProvider.checkActivePart()		
org.eclipse.ui.internal.services.WorkbenchSourceProvider$2.windowDeactivated(IWorkbenchWindow)	1,668	16%
org.eclipse.ui.internal.services.WorkbenchSourceProvider$2.windowActivated(IWorkbenchWindow)	1,668	16%

Source: *What's the context?* slide

Event Handling – II.

- Problems
  - Multiple event handling
  - Complex, implicit effects between listeners
    - Unnecessary event handling in inconsistent states
    - Event storm
  - For simple cases
    - Many, repeated code
    - Easy to miss something

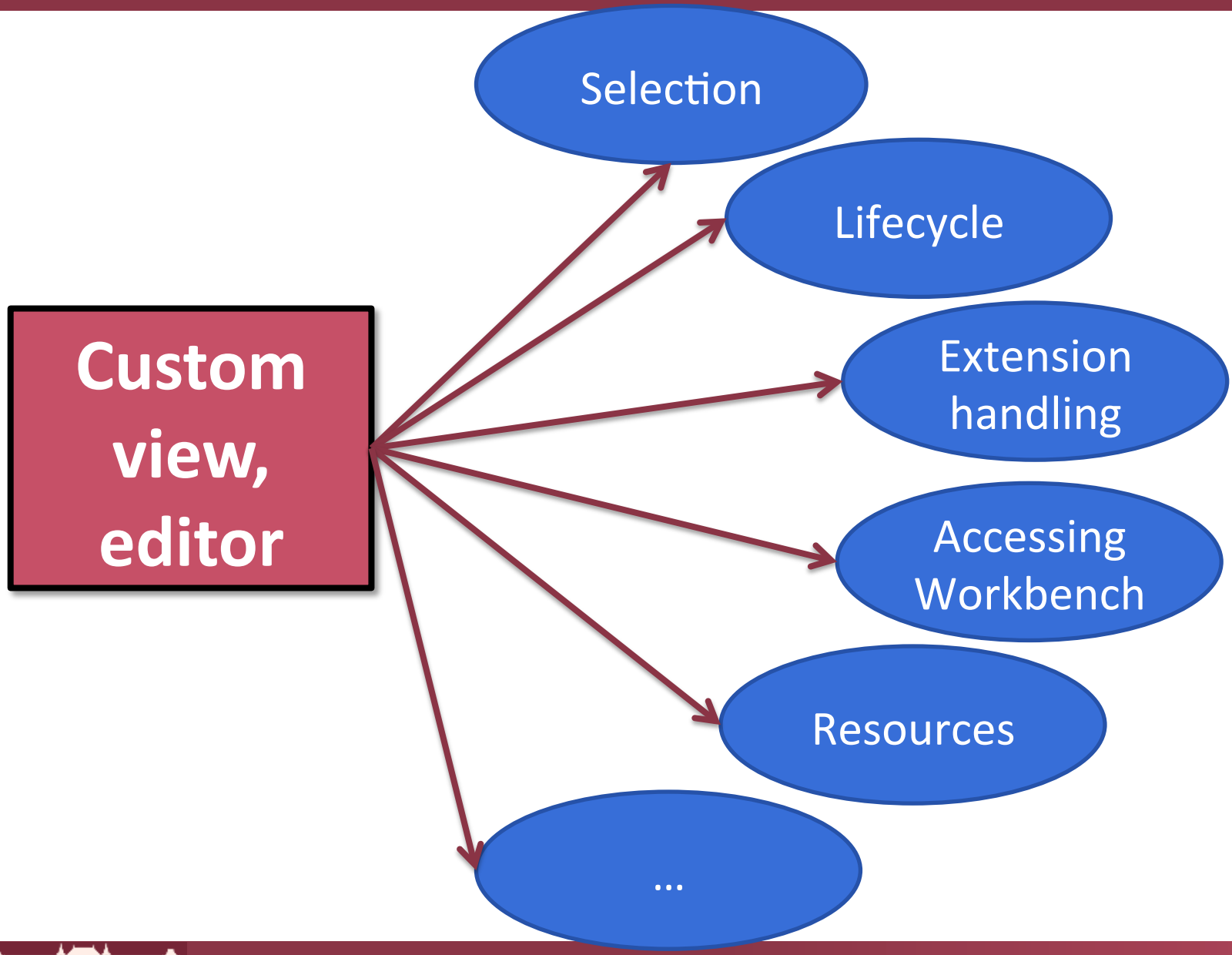
Eclipse 4.x - Context

Based on the “What’s the context?” talk
from EclipseCon 2009

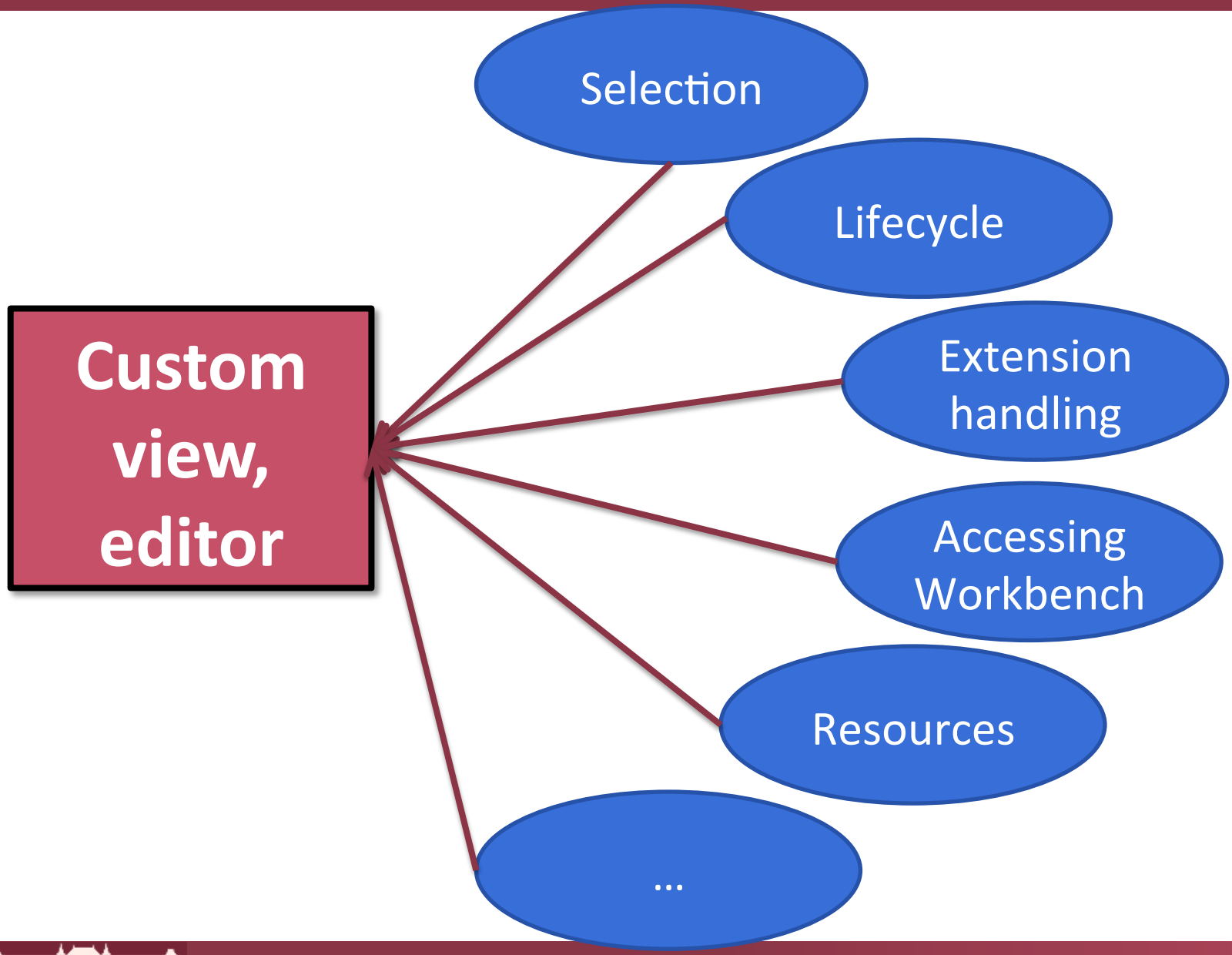
Base problem

- Problem
  - Extension knows service provider
- Solution: indirection
  - Service provider is a parameter of a class
  - On extension instantiation
    - Collect services
    - Set up parameters
  - Platform can do all of this
    - If it knows the list of available services

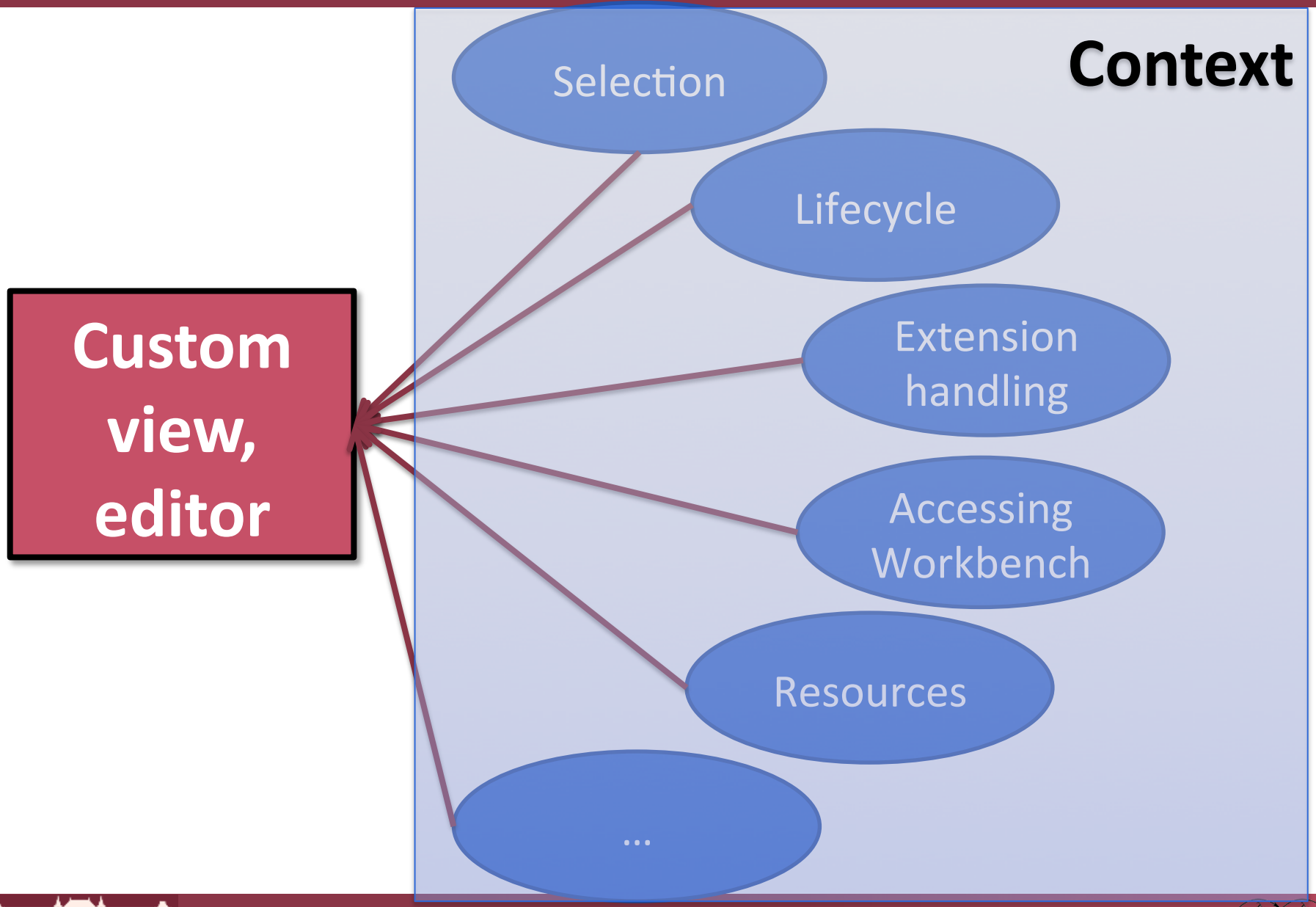
Service access – 3.x



Service access – 4.x



Service access – 4.x



Context

- Platform service access
  - Built-in services
    - Resources, selection, etc.
  - Properties
    - Preferences
    - Hierarchic key-value pairs
  - OSGi services
  - Common event handling!

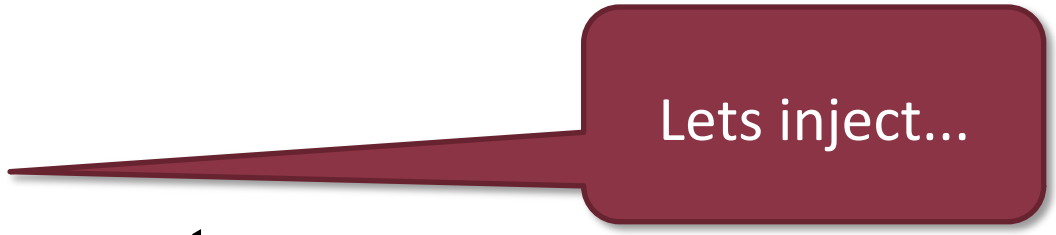
Service access – I.

- Context is accessed via
  - Dependency injection (@inject annotation)

```
@Inject
public void execute
(@Named(IServiceConstants.SELECTION) @Optional
IResource file) {
  // do something
}
```

Service access – I.

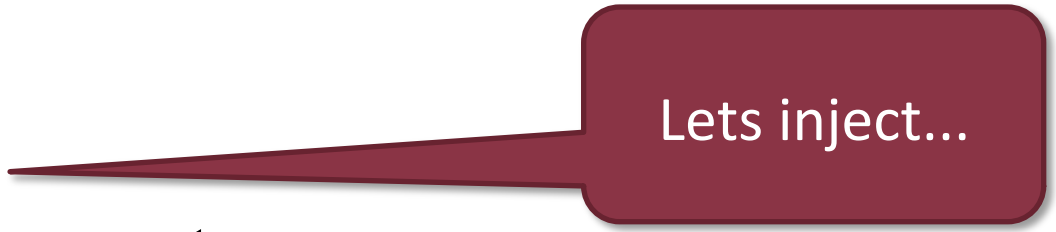
- Context is accessed via
  - Dependency injection (@inject annotation)



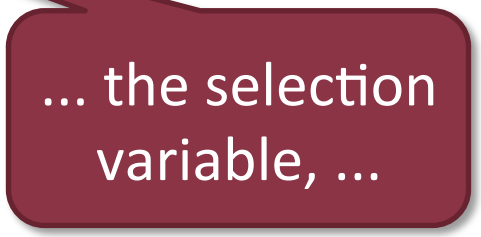

```
@Inject
public void execute
(@Named(IServiceConstants.SELECTION) @Optional
IResource file) {
  // do something
}
```

Service access – I.

- Context is accessed via
  - Dependency injection (@inject annotation)




```
@Inject
public void execute
(@Named(IServiceConstants.SELECTION) @Optional
IResource file) {
  // do something
}
```



Service access – I.

- Context is accessed via
  - Dependency injection (@inject annotation)

```
@Inject
public void execute
(@Named(IServiceConstants.SELECTION) @Optional
IResource file) {
  // do something
}
```

Lets inject...

... the selection variable, ...

... that might be empty ...

Service access – I.

- Context is accessed via
  - Dependency injection (@inject annotation)

```
@Inject
public void execute
(@Named(IServiceConstants.SELECTION) @Optional
IResource file) {
  // do something
}
```

Lets inject...

... the selection variable, ...

... that might be empty ...

and has a type of IResource.

Service access – II.

- What happens in the background?
  - Find variables from context
  - If matches parameters, it is transferred
  - If not, and
    - Optional => null value, callee handles this;
    - Not optional => inject error
  - Used annotations based on JSR-330 specification
    - JSR 330: Dependency Injection for Java
    - http://jcp.org/en/jsr/detail?id=330

Event Handling

- Base class
  - Queried value changes
  - Solved by dependency injection
    - In case of changes framework recalls method
    - Much fewer event handler required
- Internal optimalization
  - Notification only in consistent context states

Model-based Workbench

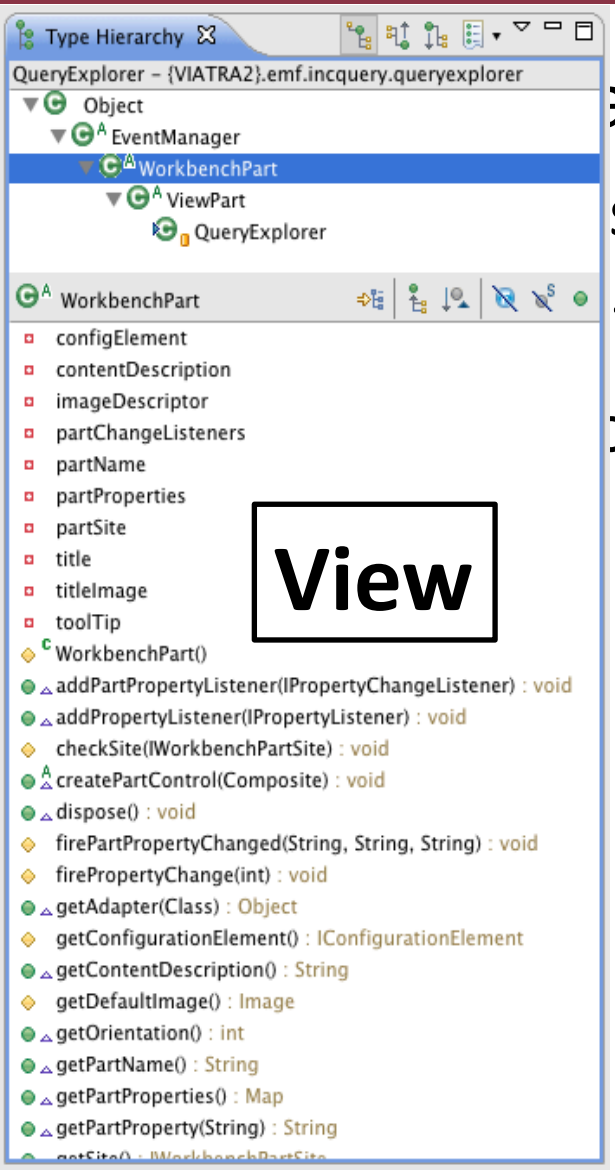
Workbench stucture

- Strict hierarchy
  - Window
  - Site
  - Perspective
  - Editors and views
- In 3.x, defined in code
  - New instances set up via extensions

Inheritance hierarchy

- Common behaviour
  - Defined in base classes
  - Not required to code all the time
  - Same behaviour all the time

Inheritance hierarchy



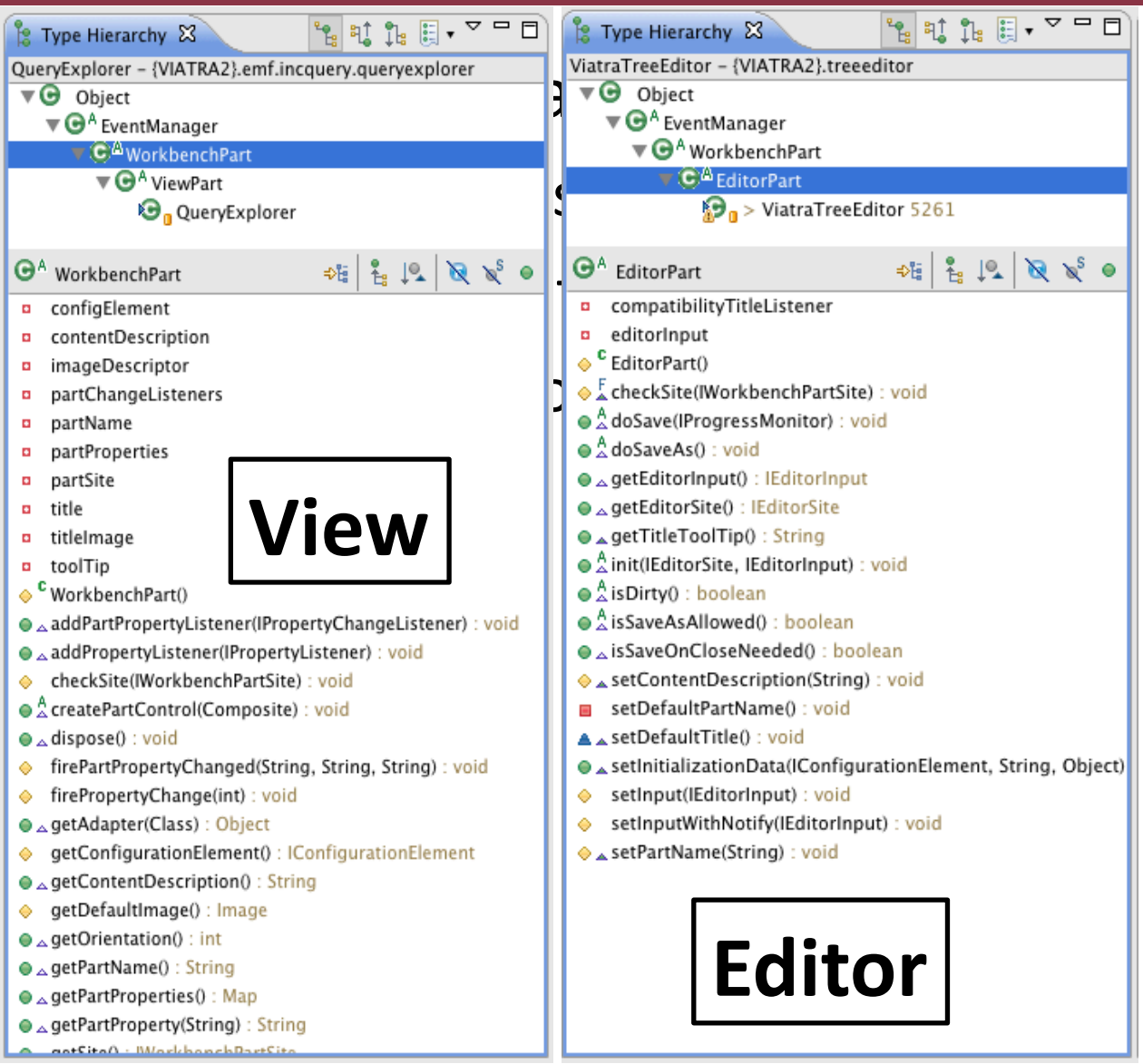
aviour

se classes

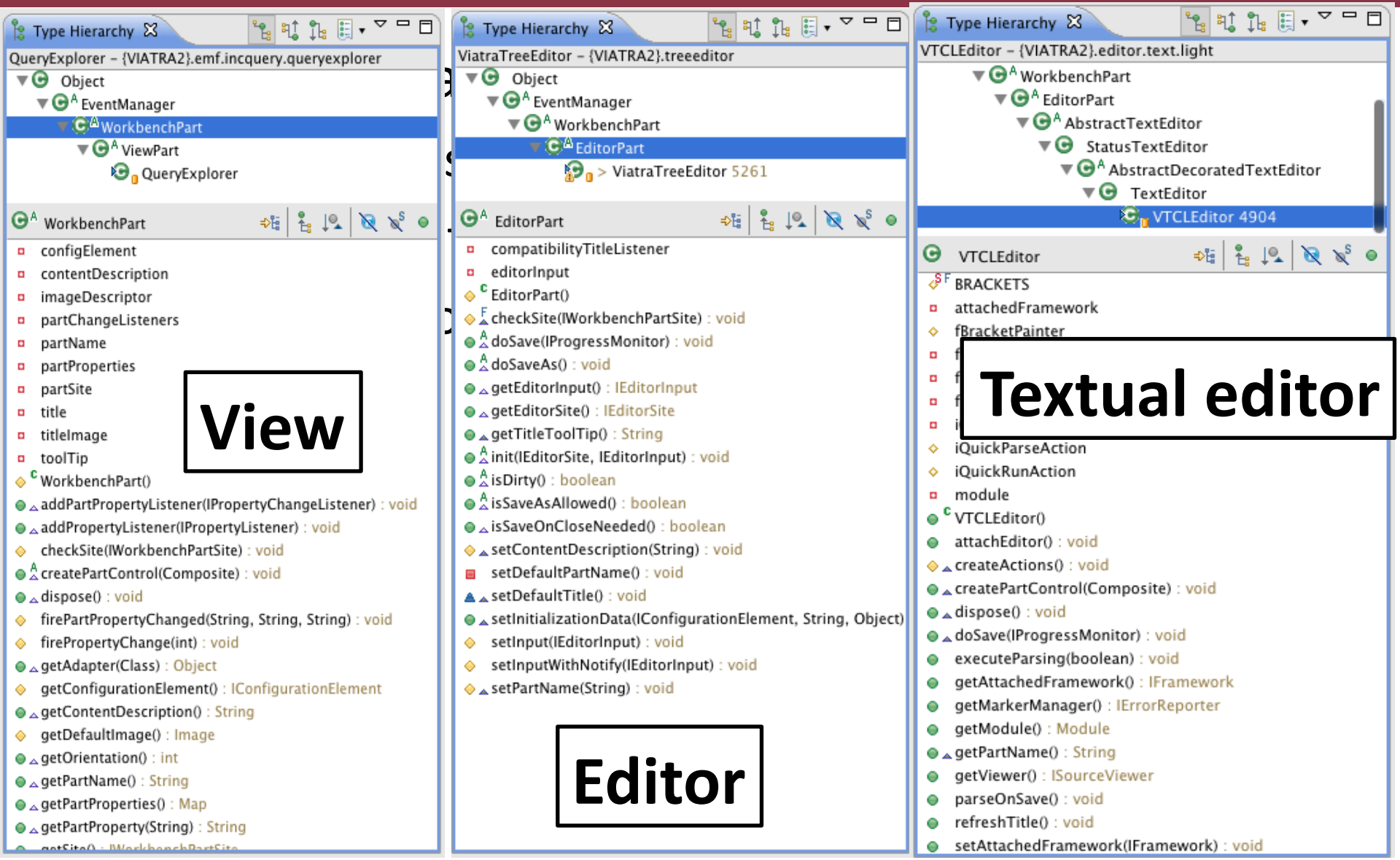
to code all the time

our all the time

Inheritance hierarchy



Inheritance hierarchy



New approach

- Application model
  - Describing the workbench structure
    - Does not contain the entire GUI (on the widget level)
    - Widget set independent
  - Available during runtime
    - AND modifiable

Custom model

- Application model
  - Basically one for an RCP application
  - Structure of the entire application
- Model fragment
  - Extensions to running application model
  - A single fragment replaces a large set of extensions
    - More understandable than scattered extensions
- Registration of models and fragments
  - Using a single extension point

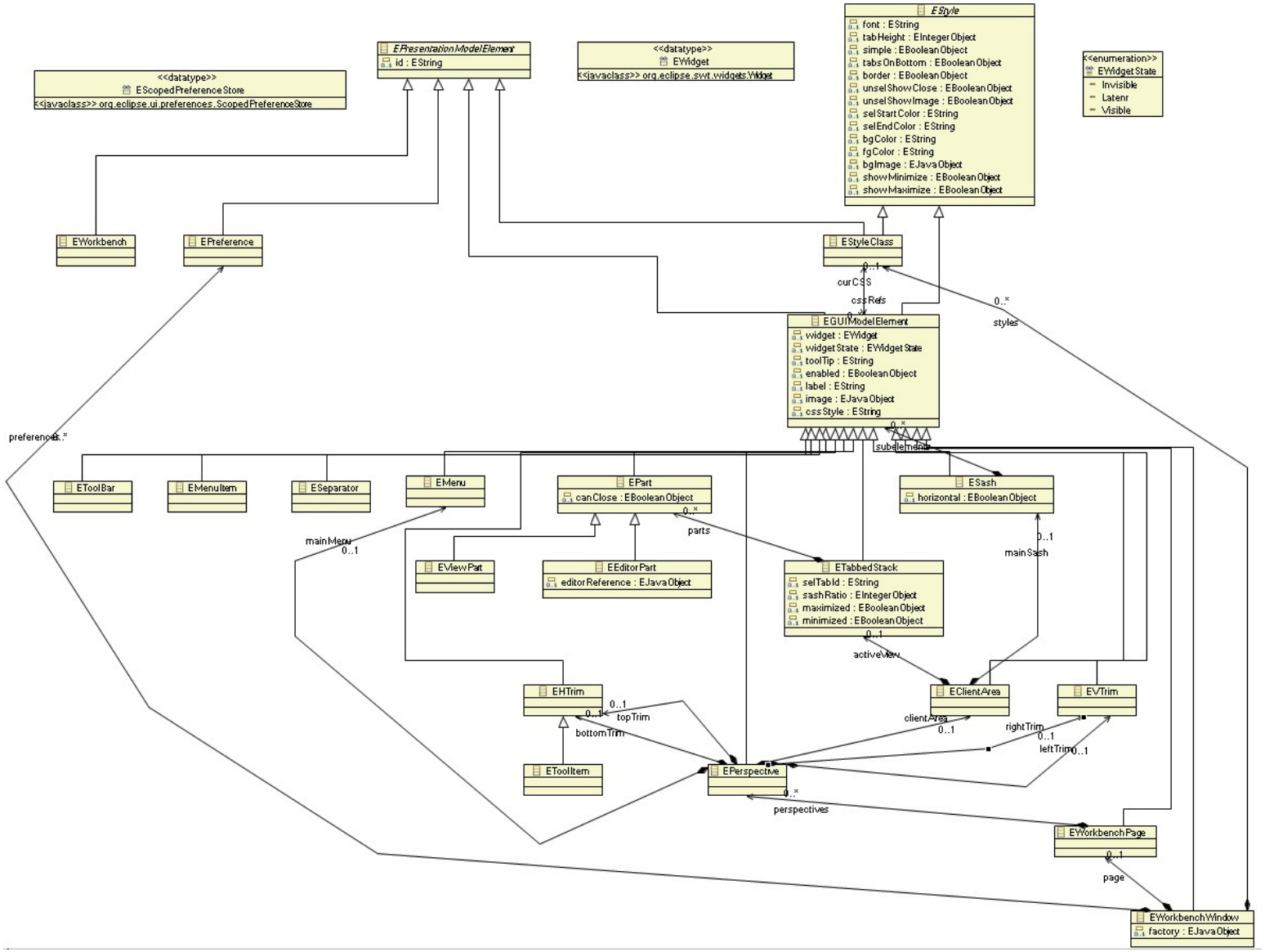
Application model

- Technology: EMF modell
  - Metamodel available
  - Extensions possible
- Contains
  - Application window
  - Views and editors (commonly referred to as *Part*)
    - +layout
  - Commands, menu
- Detailed documentation
  - http://wiki.eclipse.org/E4/UI/Modeled_UI

Application model

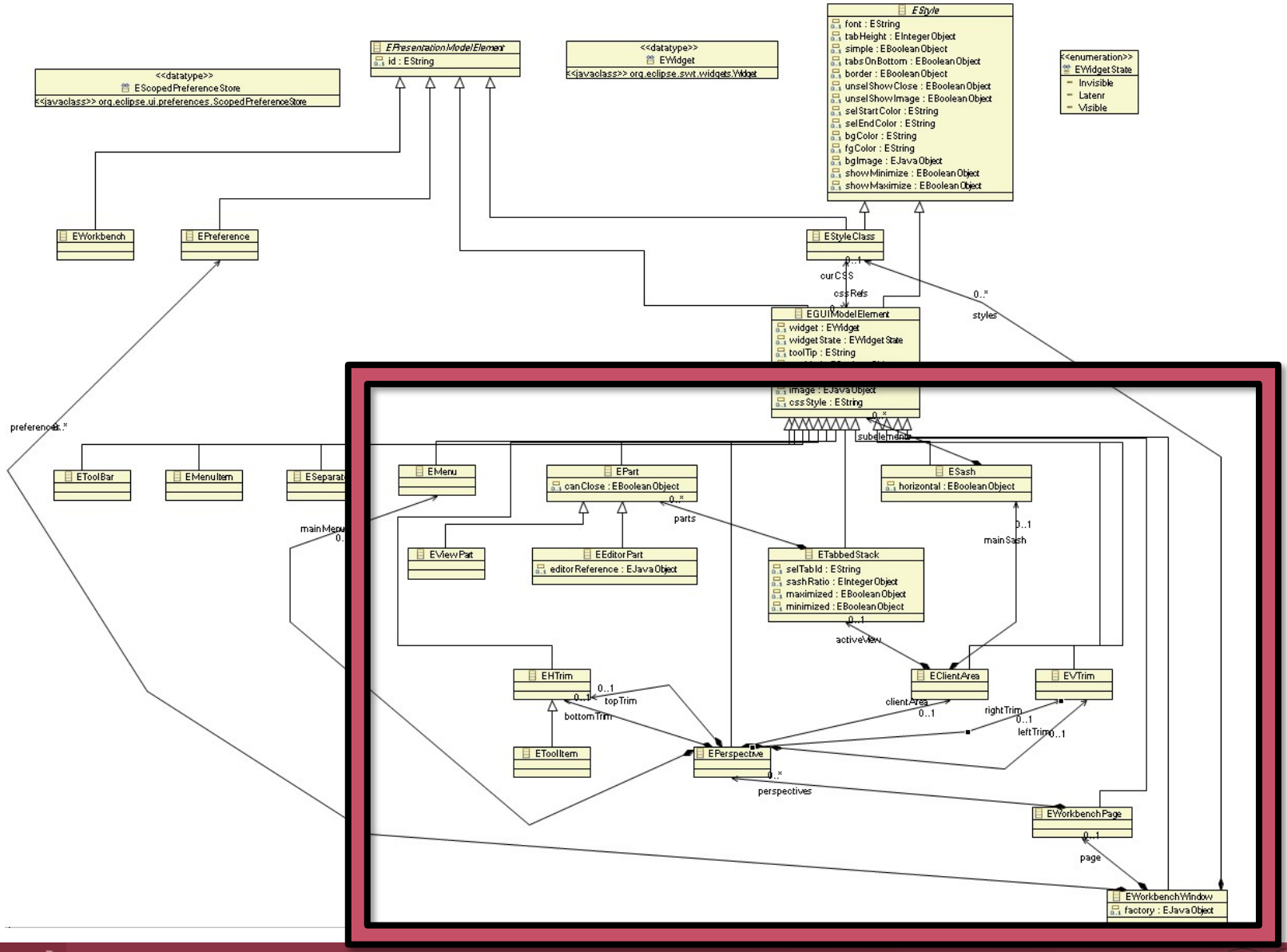
- Live model
  - Synchronized with the GUI
    - User interactions are reflected here
    - Changes update the user interface
      - E.g. programmatic opening of a View
  - Model is serialized on save
    - State restore

Workbench metamodel – not up to date!



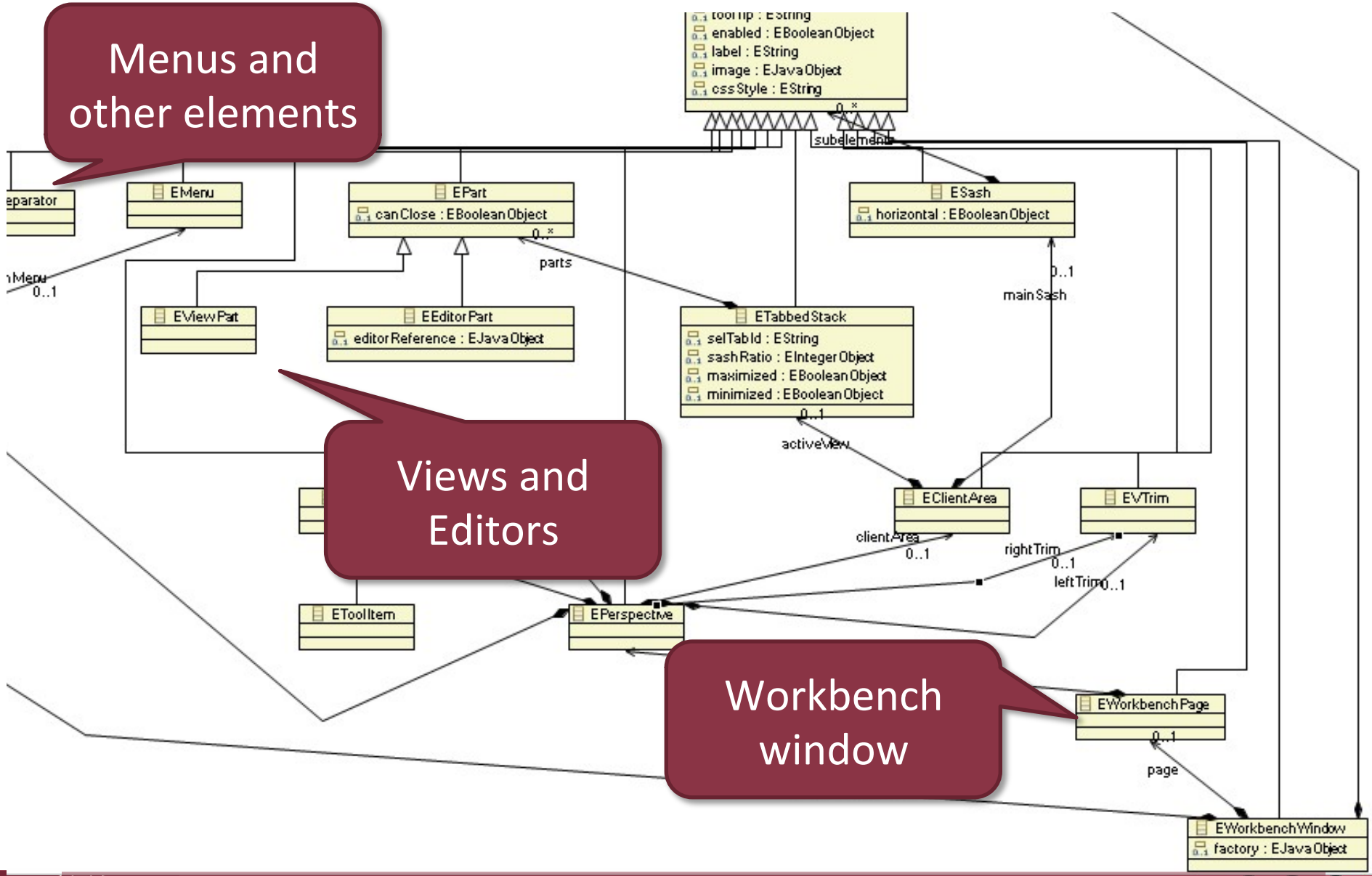
Forrás: http://dev.eclipse.org/mhonarc/lists/eclipse-incubator-e4-dev/msg00222.html

Workbench metamodel – not up to date!



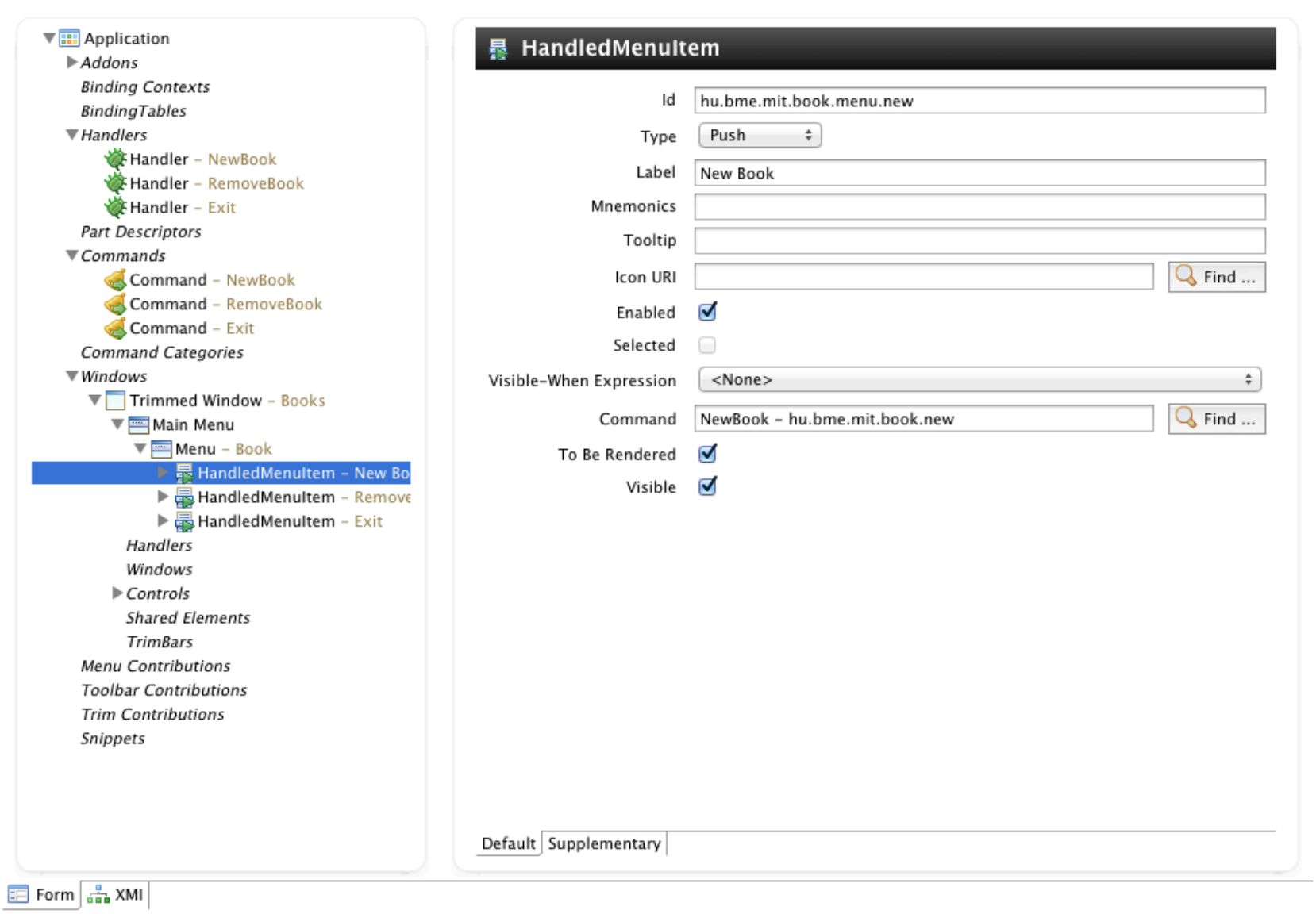
Forrás: http://dev.eclipse.org/mhonarc/lists/eclipse-incubator-e4-dev/msg00222.html

Workbench metamodel – not up to date!



Forrás: http://dev.eclipse.org/mhonarc/lists/eclipse-incubator-e4-dev/msg00222.html

Application model editor (e4 tools)



Annotation based API

- Implementing classes
  - Free inheritance hierarchy
  - Lifecycle events via annotations
  - JSR 250: Common Annotations for the Java™ Platform

View example – 3.x

```
public class SampleView extends ViewPart {

	@Override
	public void createPartControl(Composite parent) {
		// TODO Auto-generated method stub
	}

	@Override
	public void setFocus() {
		// TODO Auto-generated method stub
	}

	@Override
	public void dispose() {
		// TODO Auto-generated method stub
	}


}
```

View example – 4.x

```
public class SampleView extends ViewPart {

	@Override@PostConstruct
	public void createPartControl(Composite p
		// TODO Auto-generated method stub
	}

	@Override@Focus
	public void setFocus() {
		// TODO Auto-generated method stub
	}

	@Override@PreDestroy
	public void dispose() {
		// TODO Auto-generated method stub
	}


}
```

No inheritance

Semantic annotations

What are the benefits?

Annotation based API

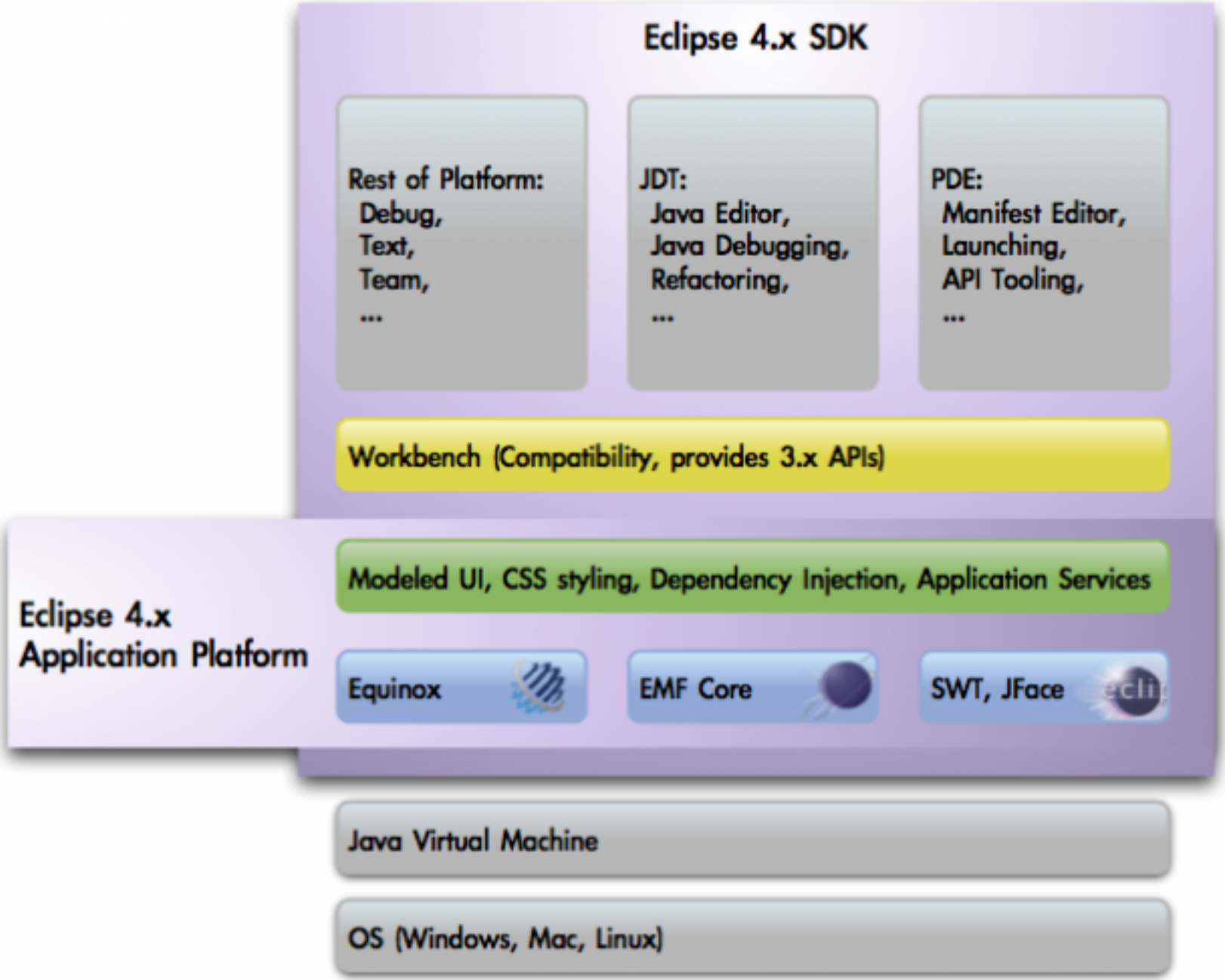
- What have we gained?
  - Free inheritance hierarchy
    - Any use is possible
  - Container idependent code
    - Class is free to reuse
      - Even outside Eclipse
      - Injection needs to be done
- What have we lost?
  - Compatibility with previous platform
    - If plug-ins not working, nobody will update

Annotation based API

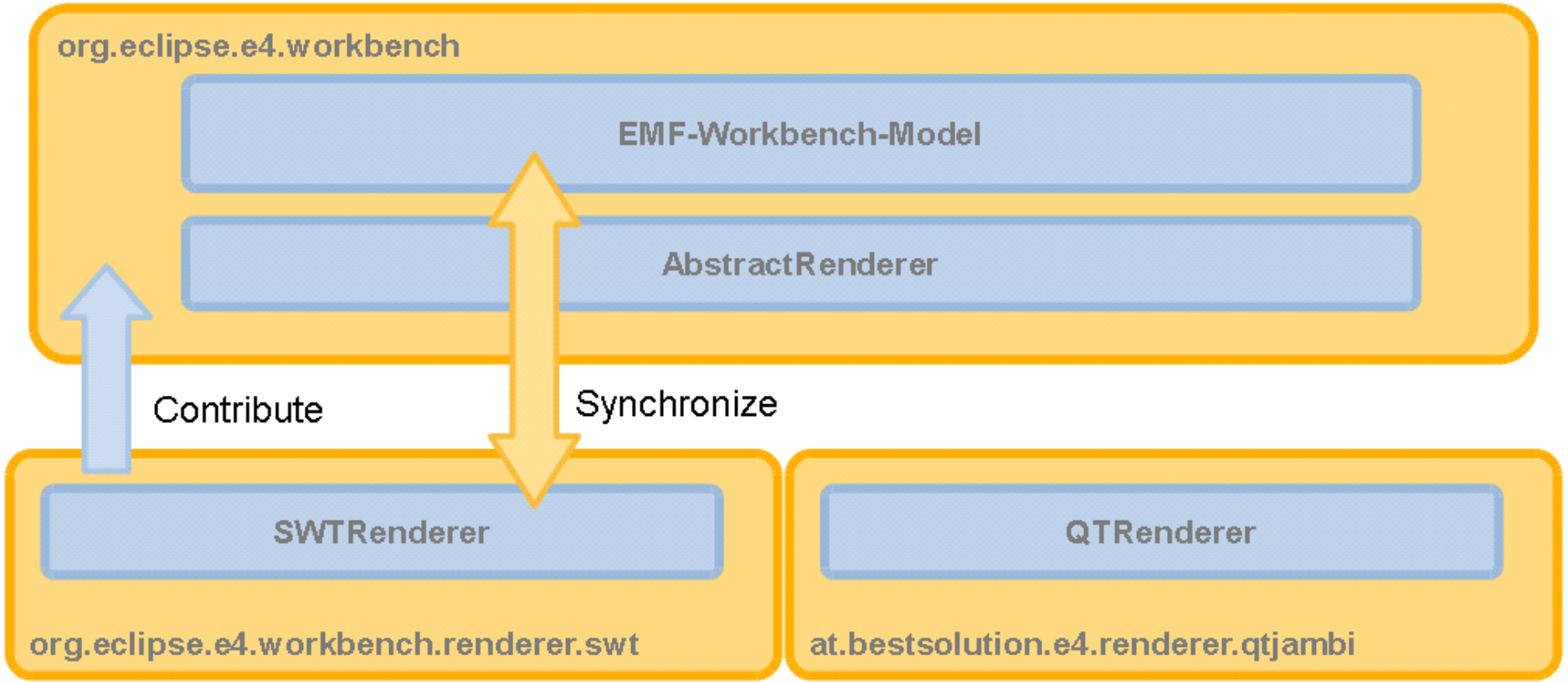
- What have we gained?
  - Free inheritance hierarchy
    - Any use is possible
  - Container idependent code
    - Class is free to reuse
      - Even outside Eclipse
      - Injection needs to be done
- What have we lost?
  - Compatibility with previous platform
    - If plug-ins not working, nobody will update

Compatibility with 3.x API

- Re-implement the base classes required by 3.x
  - Use the new API
- Not a perfect clone
  - Smaller issues
  - API compatibility achieved
- Using 3.x and 4.x GUI parallel
  - Not supported officially (in 2012)
  - BUT: based on a EclipseCon ‘12 talk possible

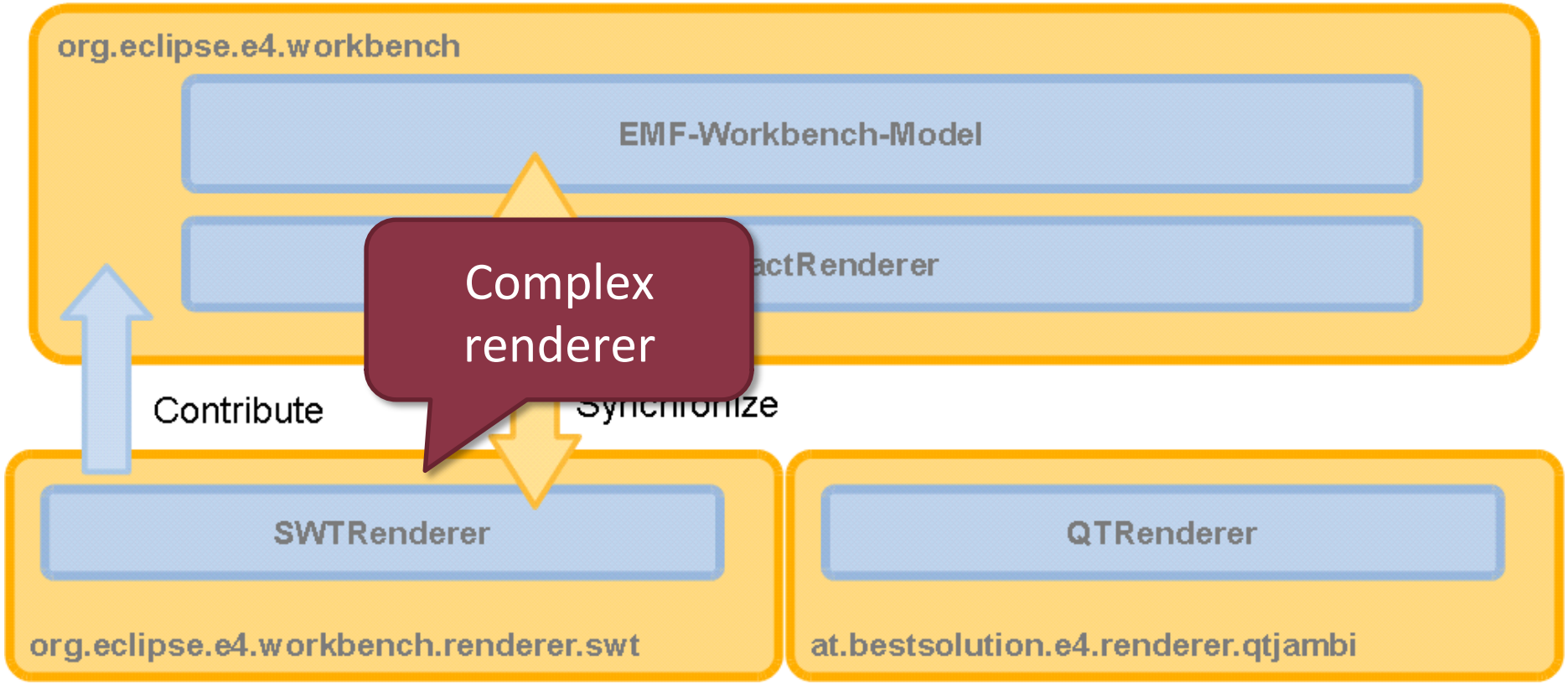

Eclipse 4.x SDK
Rest of Platform:
Debug,
Text,
Team,
...
JDT:
Java Editor,
Java Debugging,
Refactoring,
...
PDE:
Manifest Editor,
Launching,
API Tooling,
...
Workbench (Compatibility, provides 3.x APIs)
Eclipse 4.x
Application Platform
Modeled UI, CSS styling, Dependency Injection, Application Services
Equinox
EMF Core
SWT, JFace
Java Virtual Machine
OS (Windows, Mac, Linux)

Model processing during runtime

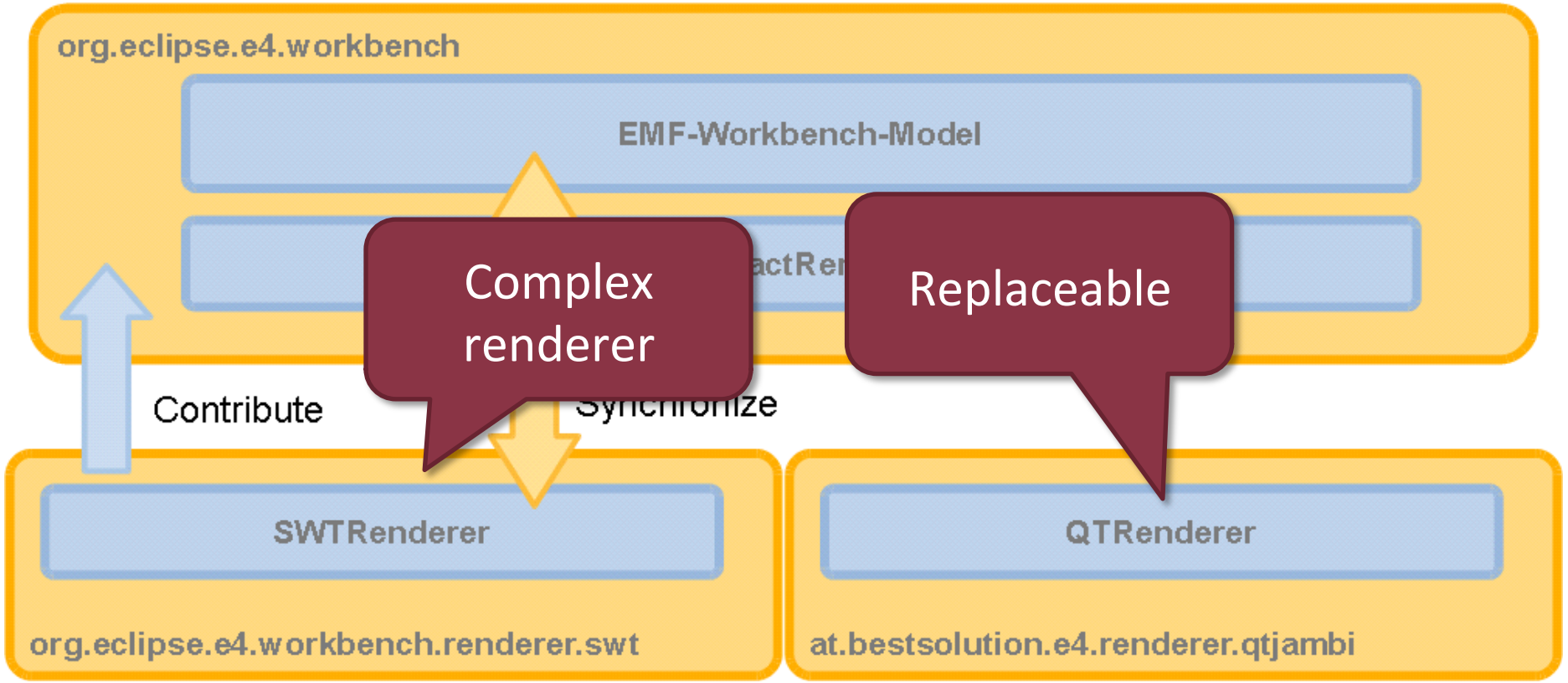


Forrás: Tom Schindl, Lars Vogel, Kai Tödter: Eclipse e4 tutorial, EclipseCon 2010

Model processing during runtime



Model processing during runtime

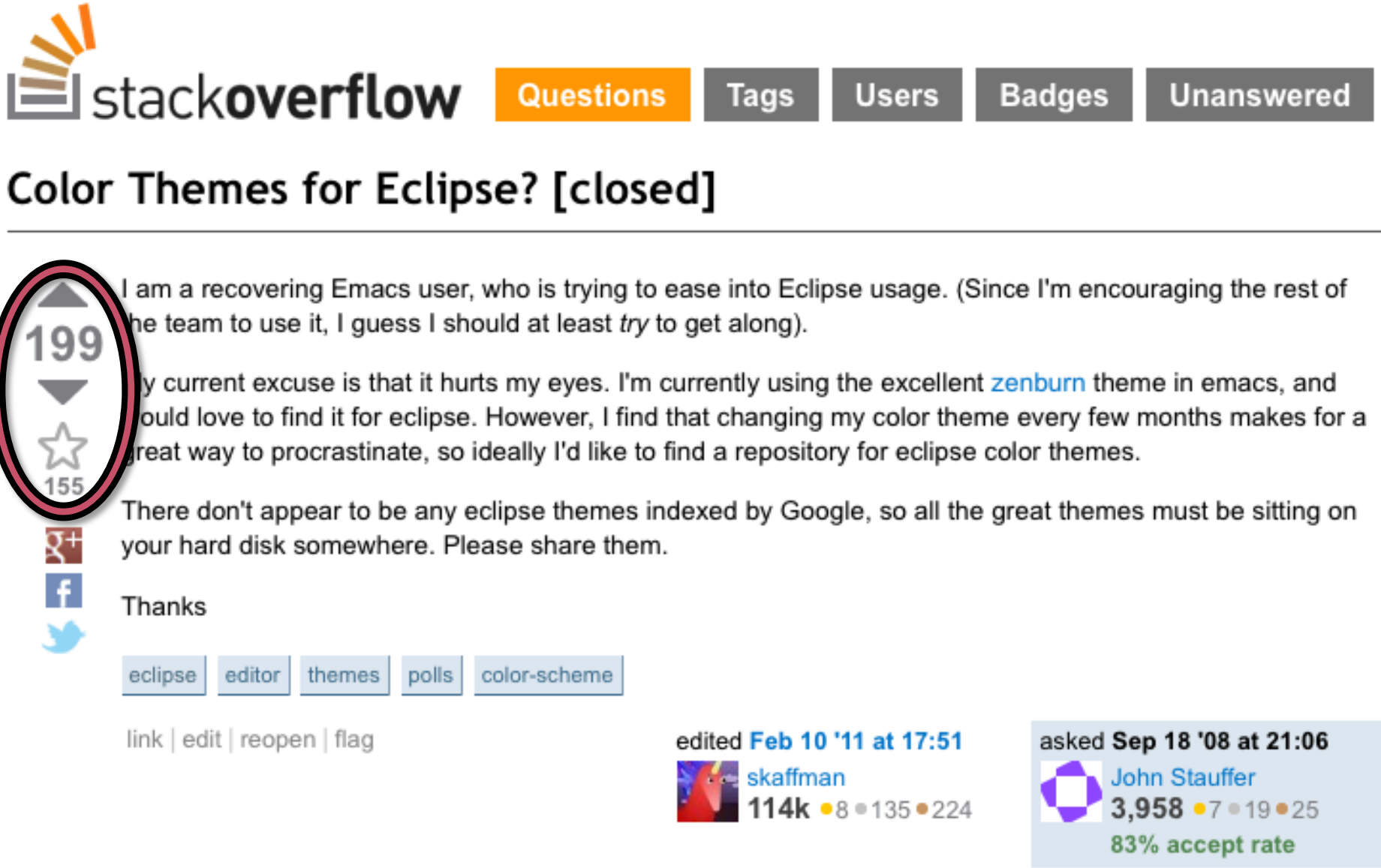


CSS based theming

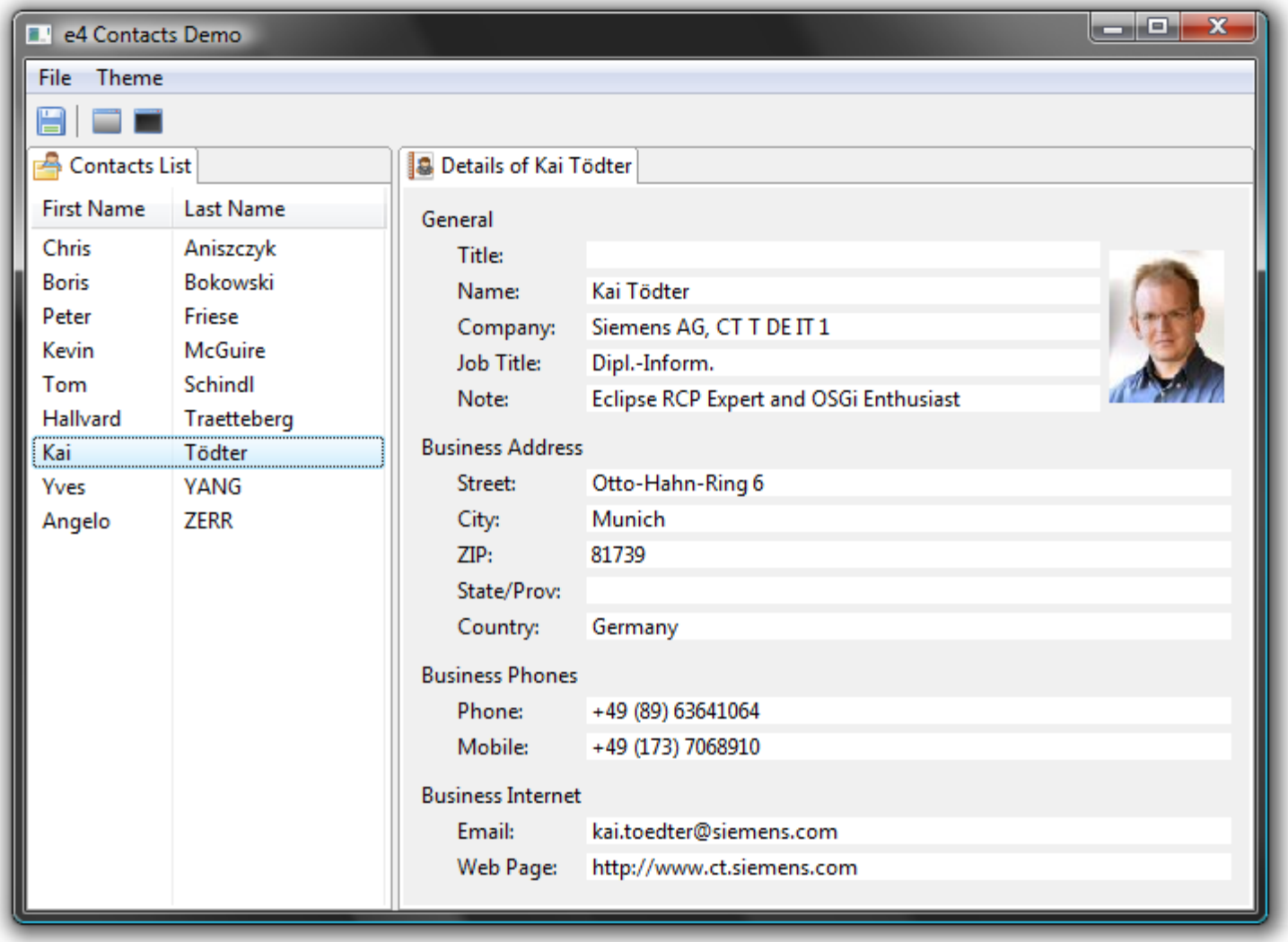
Application theming

- Eclipse 3.x
  - Partially possible
    - Colors, formatting stored and queried
    - Basically key-value pairs
      - Not flexible enough
  - How serious is this problem?
    - IDE
      - Good question
    - RCP application
      - “Must not look like the IDE!”

Application theming

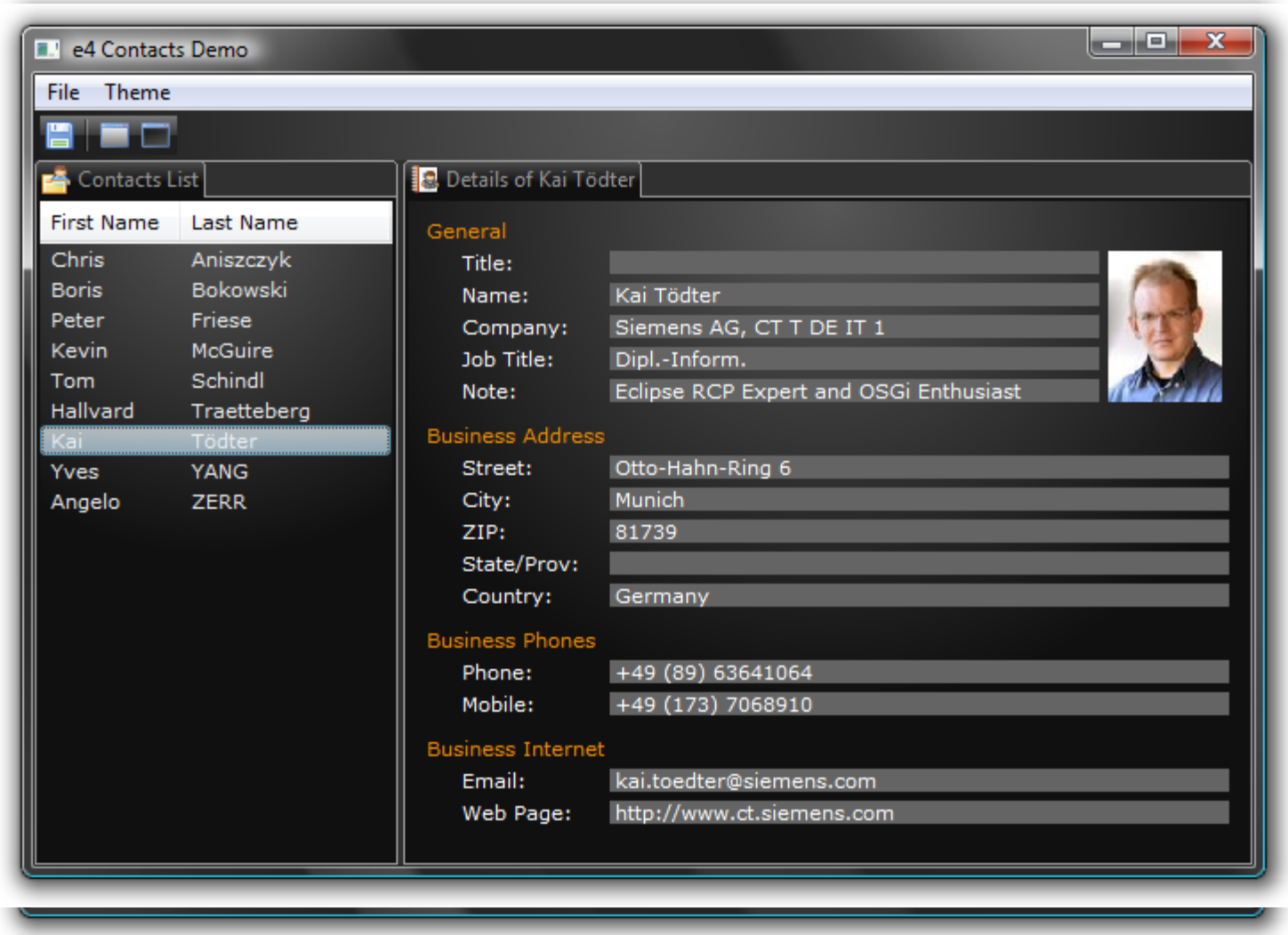


Theming – Contacts demo



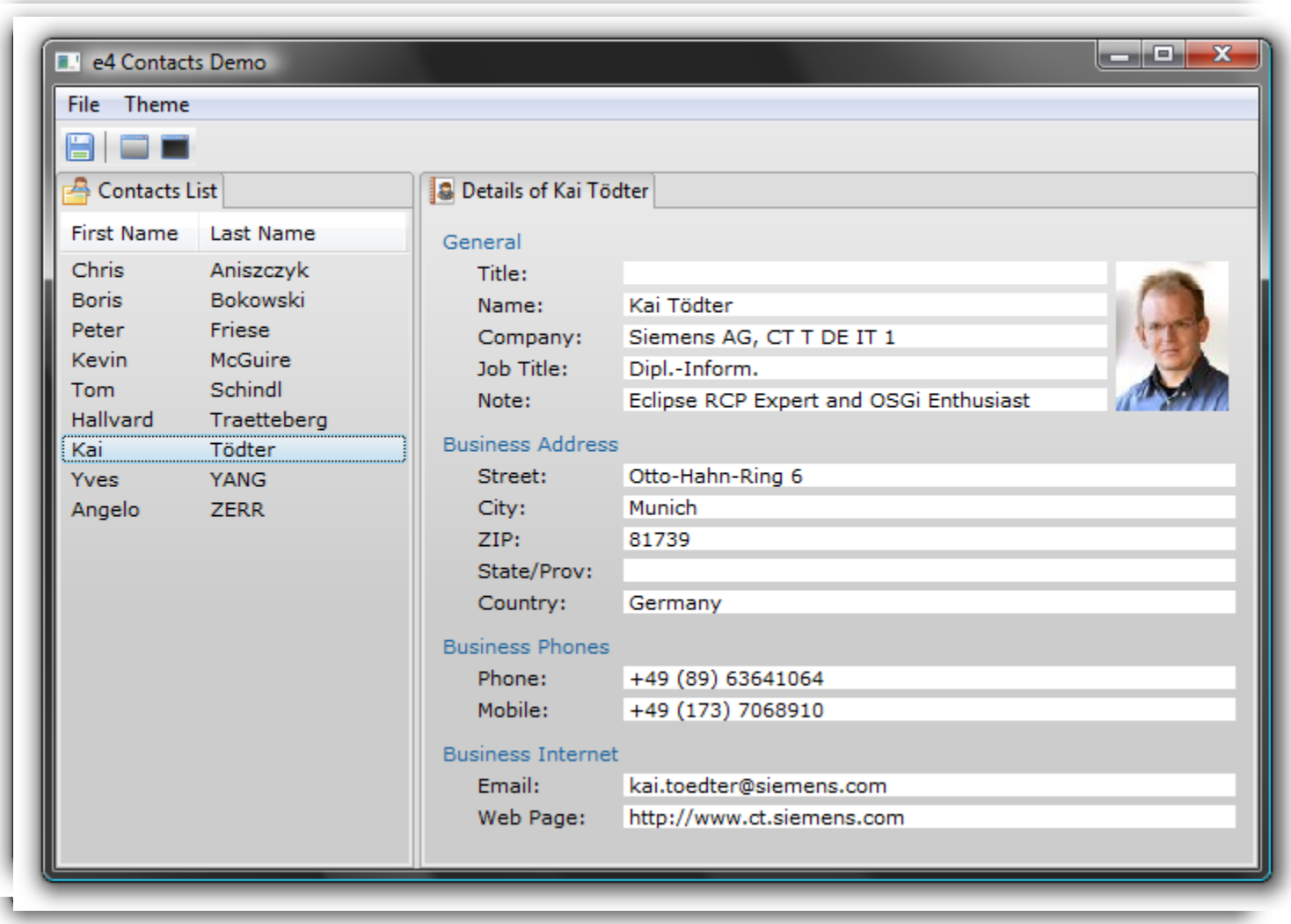
Forrás: Tom Schindl, Lars Vogel, Kai Tödter: Eclipse e4 tutorial, EclipseCon 2010

Theming – Contacts demo



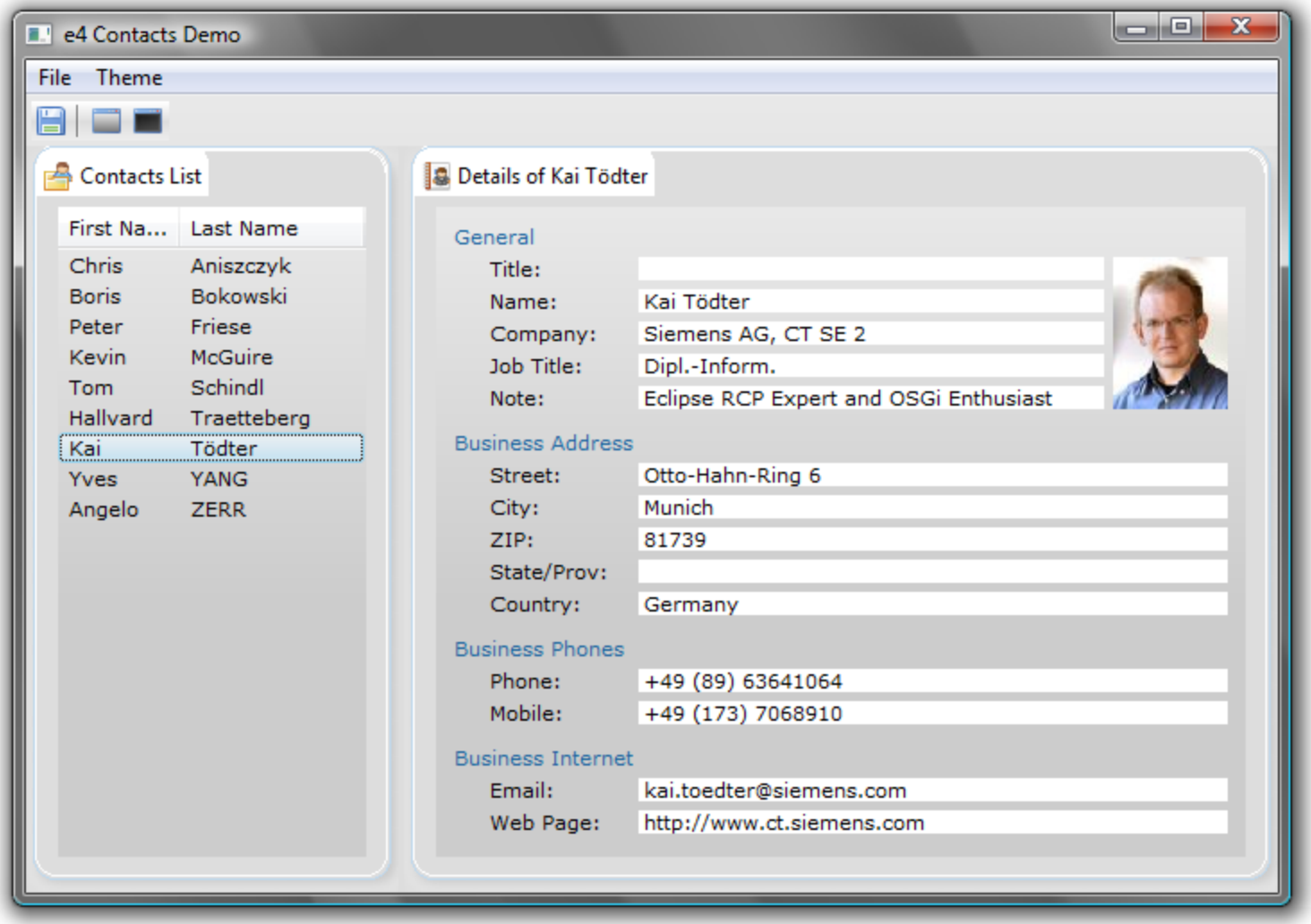
Forrás: Tom Schindl, Lars Vogel, Kai Tödter: Eclipse e4 tutorial, EclipseCon 2010

Theming – Contacts demo



Forrás: Tom Schindl, Lars Vogel, Kai Tödter: Eclipse e4 tutorial, EclipseCon 2010

Theming – Contacts demo



Forrás: Tom Schindl, Lars Vogel, Kai Tödter: Eclipse e4 tutorial, EclipseCon 2010

Theming – Eclipse IDE

- Styles
  - Based on products
  - More possible
    - Changes possible during runtime
- Unique attributes attached to widgets:

```
Label label = new Label(parent,
  SWT.NONE);
label.setData("org.eclipse.e4.ui.cs
  s.id", "SeparatorLabel");
```

Theming in platform 4.x

- Reasonably wide possibilities
- Report from EclipseCon ’12:
  - CSS on E4
  - http://holistictendencies.wordpress.com/2012/03/31/css-on-e4-eclipsecon/
-

New problems

New Platform – New problems

New Platform – New problems

- Are there any?

- Performance problems
  - In some cases noticeable slowdowns
    - Platform team asks: report it (reproducibly)! ☺
- Service list is dynamic
  - Exploration based learning
    - “How can I get the current selection?”
  - Debugging
    - “Why do I get a null from the injector?”

New Platform – New problems

- Are there any?

- Performance problems
  - In some cases noticeable slowdowns
    - Platform team asks: report it (reproducibly)! ☺
- Service list is dynamic
  - Exploration based learning
    - “How can I get the current selection?”
  - Debugging
    - “Why do I get a null from the injector?”

Summary

Eclipse Application Platform 4.x

- Platform advances
  - Dependency injection
  - OSGi services as first-class citizens
  - New services
    - Theming
    - Better event handling
  - Sadly: new bugs
    - Sometimes performance issues
- But this is the future
  - Worth learning it

Sources, additional materials

- e4 wiki
  - http://wiki.eclipse.org/E4
- Eclipse 4 wiki
  - http://wiki.eclipse.org/Eclipse4
- John Arthorne, Paul Webster, Boris Bokowski, Oleg Besedin: *What's the context?*
  - *http://www.eclipse.org/e4/resources/contexts.pdf*
- Wim Jongman: Why Eclipse 4? (the Egg Laying WoolMilkPig)
  - http://industrial-tsi-wim.blogspot.hu/2012/10/why-eclipse-e4-egg-laying-woolmilkpig.html