eclipse rich ajax platform (RAP)

Jochen Krause
Project lead Eclipse RAP project
jkrause@innoopract.com

eclipse rich ajax platform project

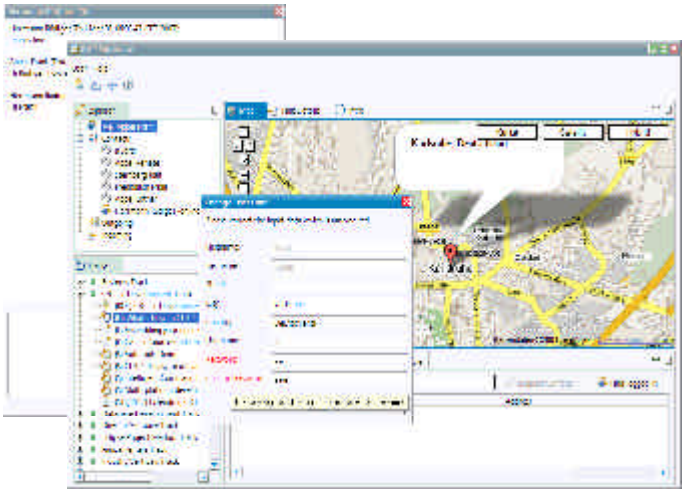
rap aims to enable developers to build rich, AJAX-enabled web applications by using the eclipse development model, plug-ins and a java-only api

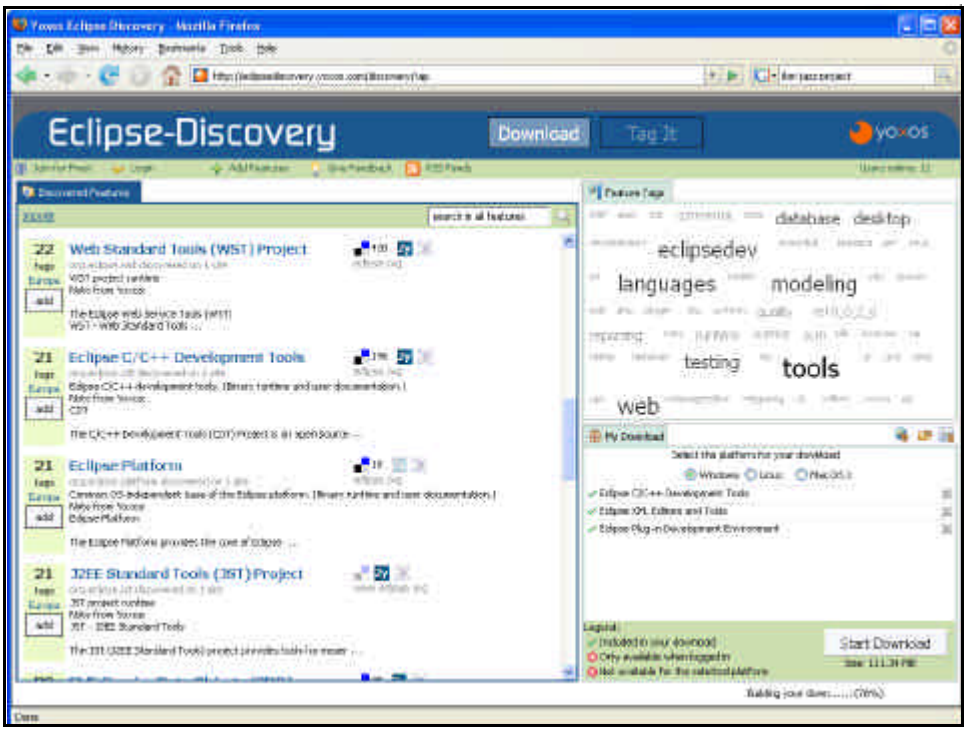
project status: M4 released, release 1.0 will be on September 28, 2007

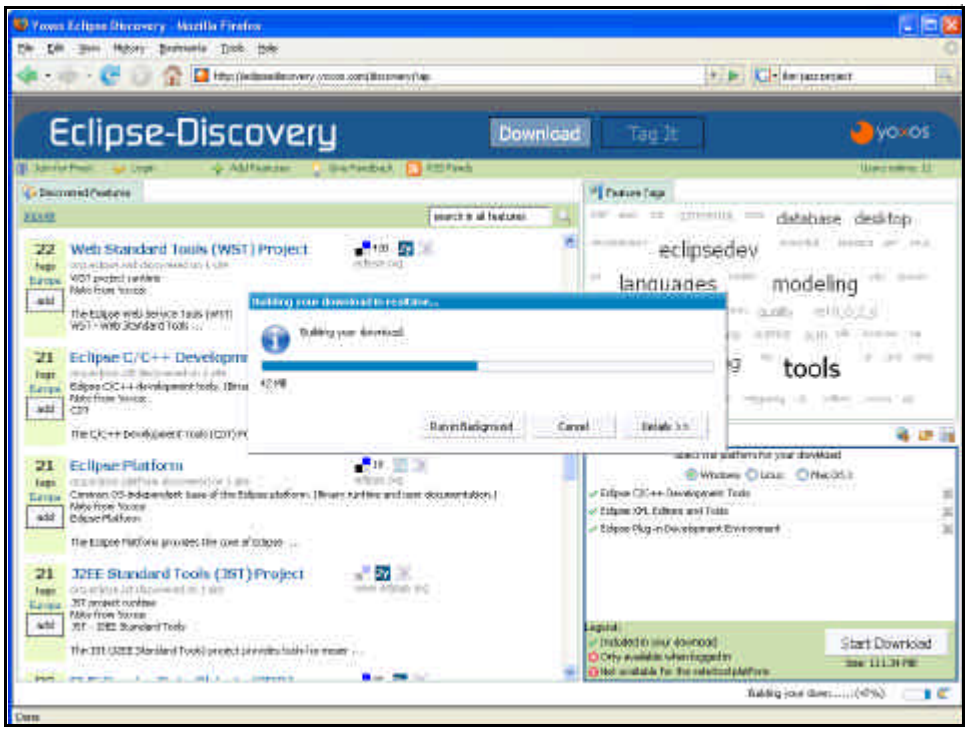
- started with a large code contribution from Innoopract
- client side javascript framework qooxdoo available under dual license (EPL, LGPL)
- demo .war archive and M4 builds ready for download
- the project has received the

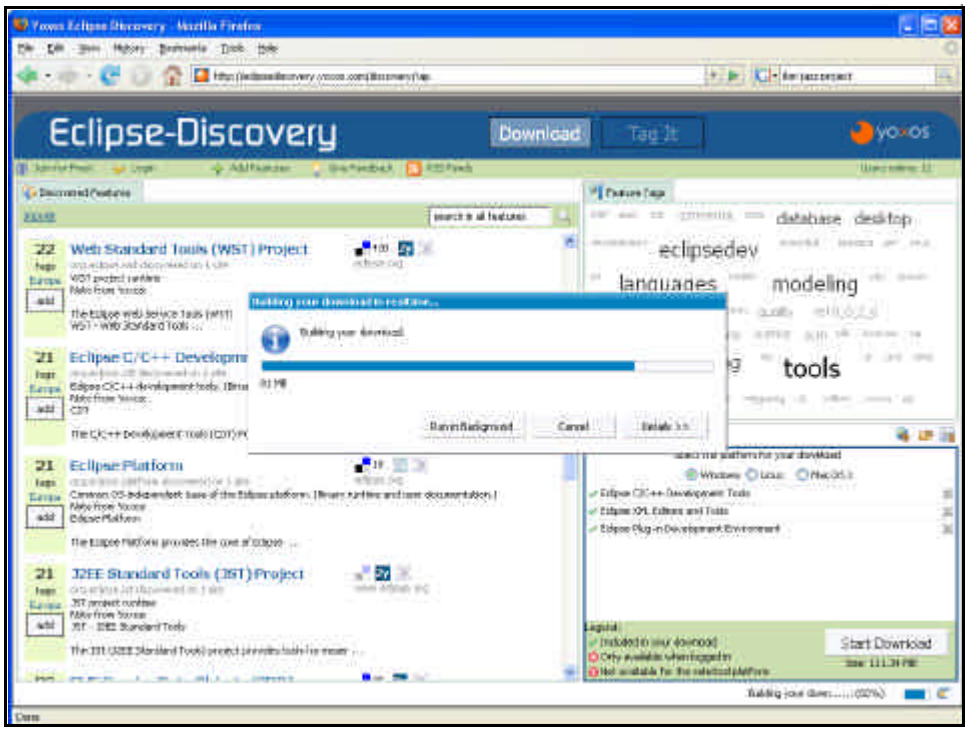
- the award honours and recognises the most remarkable and outstanding european contributions in the world of Java and Eclipse.

some RAP demo applications






Eclipse-Discovery
Download
Tag It
yoxos
Web Standard Tools (WST) Project
Eclipse C/C++ Developme
Eclipse Platform
J2EE Standard Tools (JST) Project
database
desktop
eclipsedev
languages
modeling
tools
Start Download


Eclipse-Discovery
Download
Tag It
yoxos
Web Standard Tools (WST) Project
Eclipse C/C++ Developme
Eclipse Platform
J2EE Standard Tools (JST) Project
database
desktop
eclipsedev
languages
modeling
tools
Start Download

what does rap has to offer?

- **proven technology: osgi and rcp – now for Ajax apps**
  - programmers use Java to build Ajax applications
  - enables code reuse between rich client and Ajax applications
  - provides proven eclipse component model
  - scalable UI and workbench style "mashups"
  - advances the use of equinox on the server side and as an application platform
- **skill preservation: a common platform for rich client and Ajax in Java**
  - allows organization to leverage investment in skills and technology
  - improves interoperability
  - common integrated development tools
  - simplifies Ajax development – implementation completely in Java

why? modularization for web applications

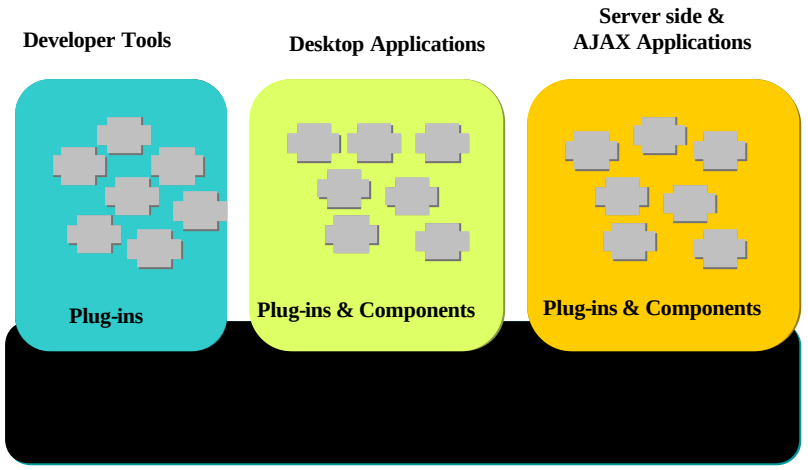
- the most commonly applied technology for developing user interfaces in the past decade, templating for (simple) HTML, is getting replaced by two new major trends:
  - **rich client applications**
  - **Ajax applications**
- eclipse has succeeded in delivering a state of the art rich client framework, offering an **extensible component model** based on an industry standard (osgi) - this has often been tried, but now it seems to be working for the first time
- eclipse technology is becoming rapidly adopted on the server side – the benefit of pluggable components is very appealing

modularization (plug-in approach) will lead help to improve productivity and allow higher levels of reuse

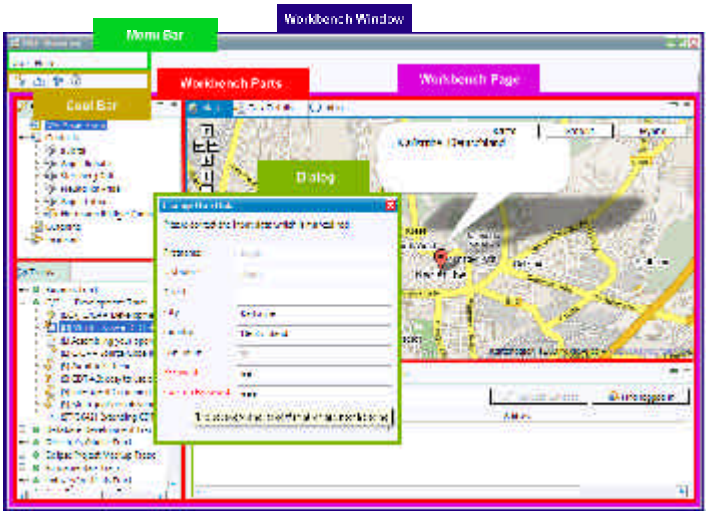
why? encapsulation

- although ajax is a promising vision, the development complexity is very high
- **better tools are under way**
  - e.g. eclipse atf http://eclipse.org/atf
  - better javascript editors are desperately needed
- **frameworks and toolkits deal with the low level stuff (js)**
  - **qooxdoo js framework**
  - **zimbra Toolkit**
  - **Dojo**
  - **OpenRico**
- **developing real world applications remains a tedious task**
  - browers behave differently, browser performance, browser memory leaks

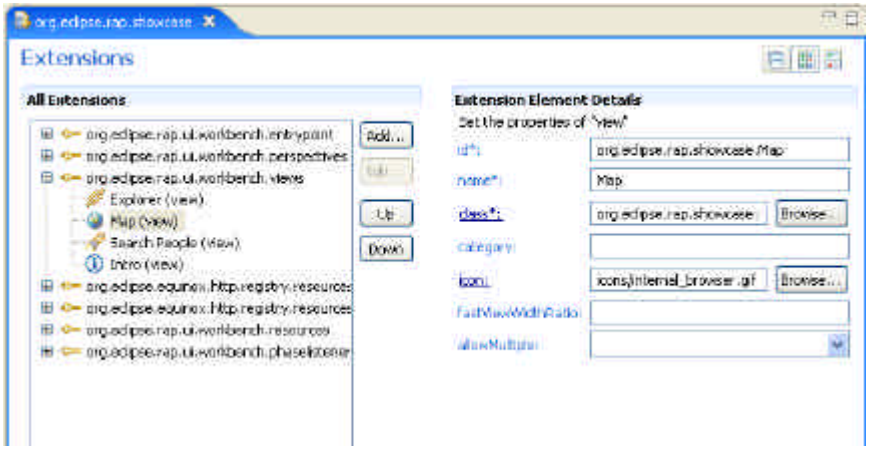
Continuous Evolution of the Eclipse Platform



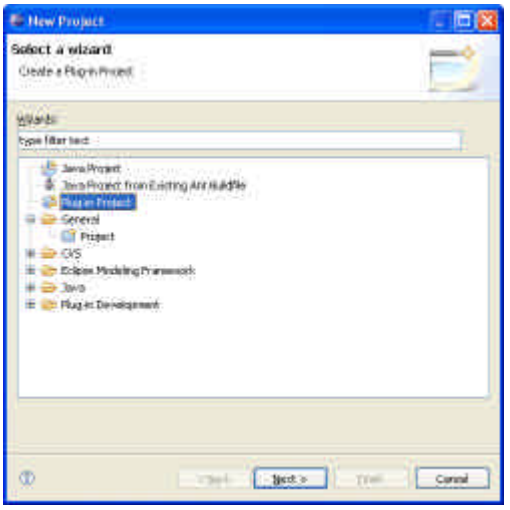
the anatomy of a RAP application



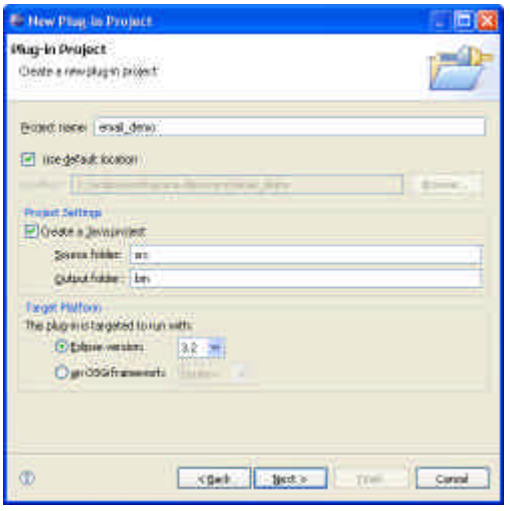
developers view of a web workbench



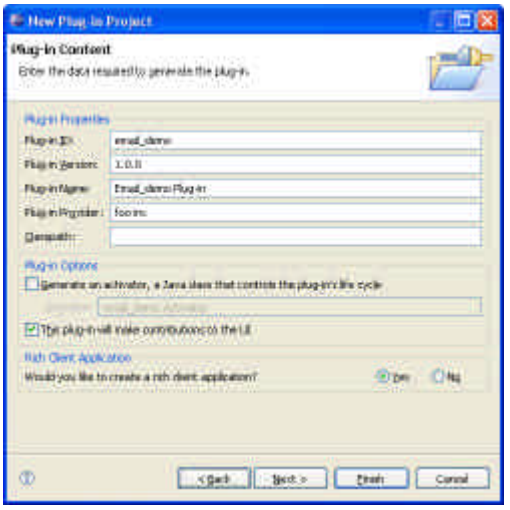
create your first rap application (1)



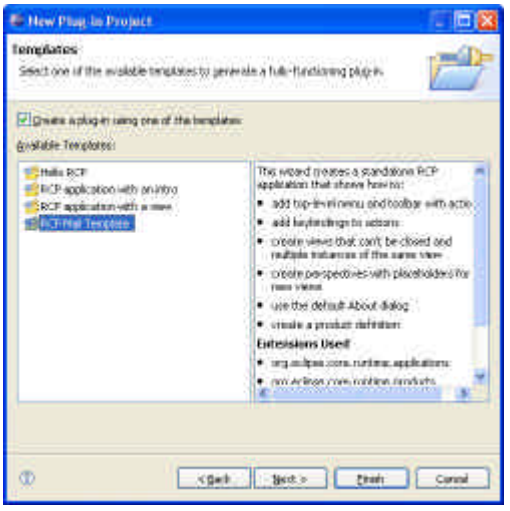
create your first rap application (2)



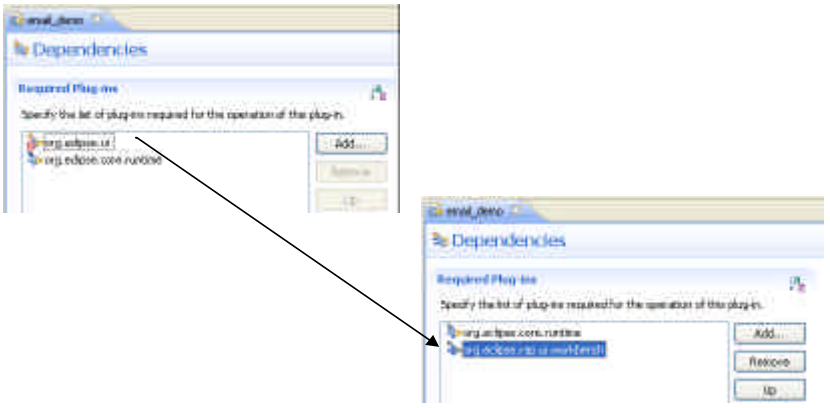
create your first rap application (3)



create your first rap application (4)



(5) Replace dependency on org.eclipse.ui with org.eclipse.rap.ui.workbench



you need to have the rap target runtime in place

resolve compile errors (4 in total)

- dialogs have slightly different API (for callback handlers)
  - additional parameter needed
  - can be null if no callback is necessary

```
public void run() {
    MessageDialog.openInformation(window.getShell(),
    "Open",
    "Open Message Dialog!",
    null);
}
```

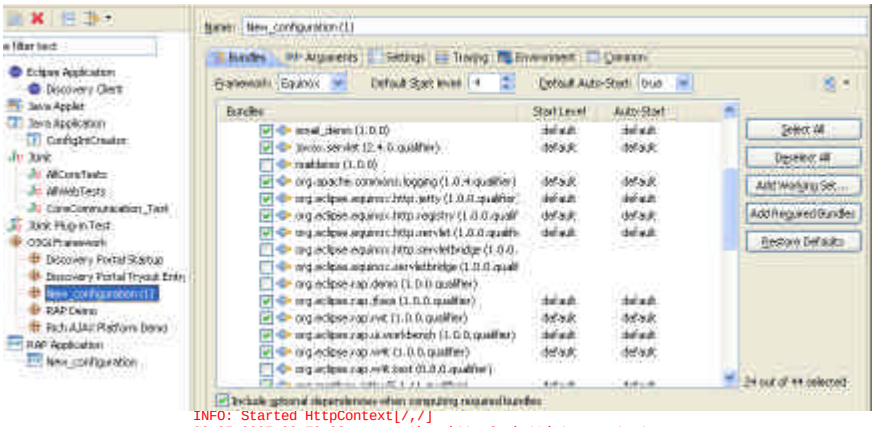
- remove OPEN_NEW_WINDOW (not yet implemented)

```
//  newWindowAction = ActionFactory.OPEN_NEW_WINDOW.create(window);
//  register(newWindowAction);
```

Implement IEntryPoint and define extension

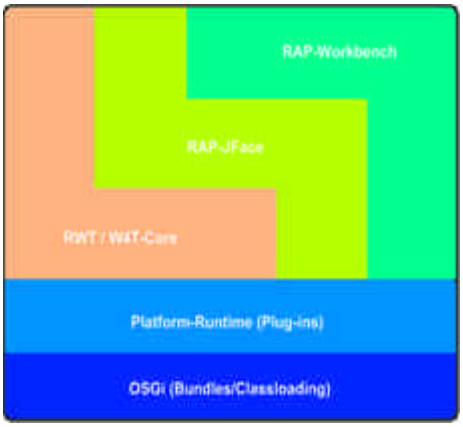
```
/**
 * This class controls all aspects of the application's execution
 */
public class Application implements IPlatformRunnable, IEntryPoint {

  public Display createUI() {
    Display display = PlatformUI.createDisplay();
    PlatformUI.createAndRunWorkbench( display,
                                      new ApplicationWorkbenchAdvisor() );
    return display;
  }
```

run it




```
INFO: Started HttpContext[/,/]
03.07.2007 23:59:23 org.mortbay.http.SocketListener start
INFO: Started SocketListener on 0.0.0.0:8008
03.07.2007 23:59:23 org.mortbay.util.Container start
INFO: Started org.mortbay.http.HttpServer@7736bd
```

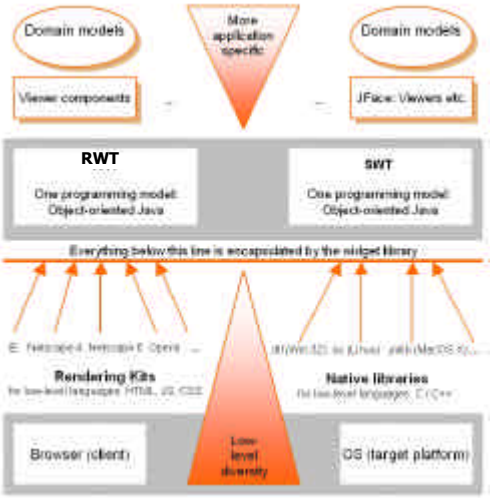
RAP architecture overview



Server Side

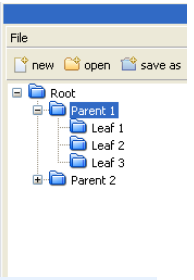
Client Side (Browser)

widget toolkit for www explained



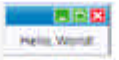
wigdet toolkit – rap windowing toolkit (rwt)

- SWT api (no binary compatiblity)
- composition of widgets into a component tree
- event driven ui
- lifecycle management of the request
- AJAX engine – based on qooxdoo
- rendering kits (life cycle adapter - LCA)
- userdefined components




```
TreeViewer viewer = new TreeViewer( left );
viewer.setContentProvider( new TreeViewerContentProvider() );
viewer.setInput( this );
Tree tree = viewer.getTree();
FormData treeData = new FormData();
    tree.addSelectionListener( new SelectionListener() {

      public void widgetSelected( final SelectionEvent event ) {
        txtGroupNameTab1.setText( "treeItem selected: " + event.item.getText() );
      }
    } );
```


a RAP hello world




```
public class RWTHello implements IEntryPoint {

  public Display createUI() {
    Display result = new Display();
    final Shell shell = new Shell( result );
    RowLayout layout = new RowLayout();
    layout.justify = true;
    layout.pack = true;
    shell.setLayout( layout );
    Label label = new Label( shell, SWT.CENTER );
    label.setText( "Hello, World!" );
    shell.pack();
    shell.open();
    return result;
  }
}
```

```
<extension
    id="org.eclipse.rap.demo.democntrypoint"
    point="org.eclipse.rap.ui.workbench.entrypoint">
  <entrypoint
      class="org.eclipse.rap.tutorial.HelloWorld"
      parameter="default"/>
</extension>
```


```
public class SWTHello {

  public static void main( String[] args ) {
    Display display = new Display();
    final Shell shell = new Shell( display );
    RowLayout layout = new RowLayout();
    layout.justify = true;
    layout.pack = true;
    shell.setLayout( layout );
    Label label = new Label( shell, SWT.CENTER );
    label.setText( "Hello, World!" );
    shell.pack();
    shell.open();
    while( !shell.isDisposed() ) {
      if( !display.readAndDispatch() )
        display.sleep();
    }
    display.dispose();
  }
}
```

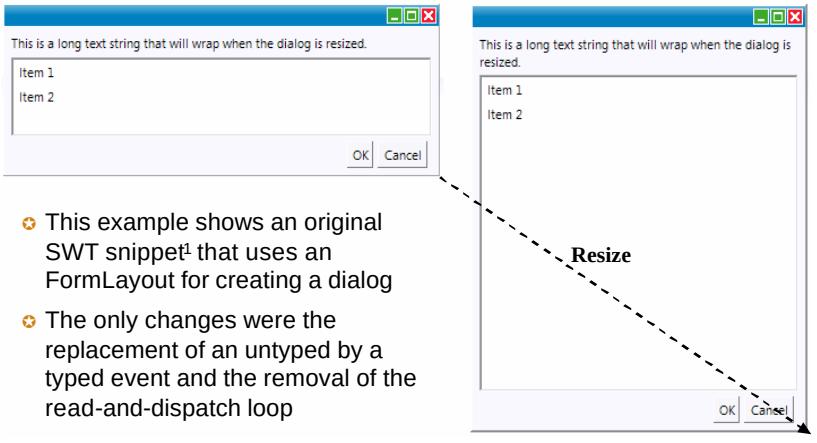
RWT: layout

- Layout calculations of RWT UIs are done on the server-side
- Layouting takes place after the user has resized for example a s hell and therefore a server-side turn-around is needed
- The advantage of this approach is the reuse of the powerful SWT layout implementations, even custom layouts used in RCP applications can easily reused

  So far the following Layouts are successfully adopted: BorderFillLayout, CBannerLayout, CTabFolderLayout, EnhancedFillLayout, FillLayout, FormLayout, GridLayout, RowLayout, SashFormLayout, ScrolledCompositeLayout, StackLayout, TrimLayout, ViewFormLayout, WrapperLayout

- Due to the Ajax-Rendering Engine which only sends the status delta the UI updates are fast enough

example: FormLayout



- This example shows an original SWT snippet[1] that uses an FormLayout for creating a dialog
- The only changes were the replacement of an untyped by a typed event and the removal of the read-and-dispatch loop

1) http://dev.eclipse.org/viewcvs/index.cgi/org.eclipse.swt.snippets/src/org/eclipse/swt/snippets/Snippet65.java?view=co

RWT: Events

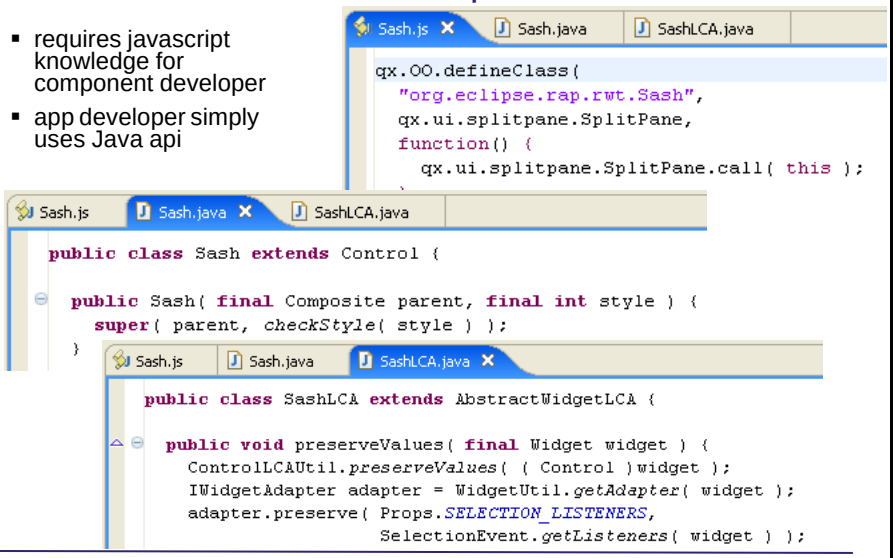
- RWT now supports typed and untyped events
- due to the limitations of the distributed environment it will not be possible to provide all SWT event types
- key strokes or mouse-down, -move etc. events would cause too much network traffic and would suffer because of the network latency
- modify events behave slightly different, since they collect consecutive key-strokes and submit the whole set

contributing custom RWT widgets

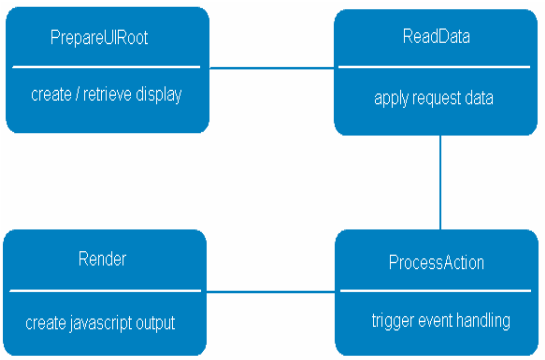
- create the widget class by Composite extension
- create an appropriate widget LCA (LifeCycleAdapter)
- create the Javascript libraries that are needed by the custom widget
- register the Javascript libraries

extensible – user defined components

- requires javascript knowledge for component developer
- app developer simply uses Java api



the RWT lifecycle phases



- at the end of the *ReadData* phase all widget attributes are in sync with the client
- these attributes are preserved for later comparison
- during the *ProcessAction phase* attribute changes may occur
- the *Render* phase compares the widget attributes with their preserved values and submits only the delta to the client

JFace

- implementations can be “reused” as long as they do not require functionality that is not provided by RWT
- subset of JFace api

currently available:

- TreeViewer
- TableViewer
- Actions, ToolBarManager, MenuManager
- Dialogs
- Window, ApplicationWindow
- ImageDescriptor

taking plugins to web applications

- eclipse plugin concept is enabled on the server side inside a web app
- alternatively a web service (e.g. jetty) can be started as an eclipse service
- everything is a plugin (server side)
  - late bindings
  - declarativ
  - loose coupling
  - contributions
- can be integrated with standard jee servers – eclipse runtime runs once per web application
  - core plugins can be reused if they are stateless
  - rwt can handle user sessions (based on servlet standard)

eclipse on the server side – osgi by equinox

- equinox is provides bundles for running eclipse inside a web app and interacting with a servlet
  - server side integration - main problems have been solved and are part of Eclipse 3.3, projects have graduated from the incubator to the “orbit” subproject of equinox http://www.eclipse.org/equinox/server/
  - embedding in a servlet container
    - war file to demo is available – starting an eclipse platform server side
  - rap is mainly reusing equinox technology and acts as a client for this project

invoking the web workbench

```
public class DemoWorkbench implements IEntryPoint {

  public Display createUI() {
    final Display result = PlatformUI.createDisplay();
    PlatformUI.createAndRunWorkbench( result, new DemoWorbenchAdvisor() );
    return result;
  }
}
```

- Startup involves extensions or implementations of the following types:
  - **WorkbenchAdvisor**
  - **WorkbenchWindowAdvisor**
  - **ActionBarAdvisor**
  - **IPerspectiveFactory**
- These types are quite familiar for RCP developers and serve the same purpose as their RCP equivalents

differences between RAP and RCP

- RAP applications are client-server applications with a browser as the client (Thin Client) and the OSGi infrastructure on the server
- OSGi-Bundles are shared between sessions (in general there are 3 scopes on the server: application-, session- and request-scope)
- restrictions of the distributed environment do not allow to build the API of RWT as a complete one-to-one mapping to SWT (-> no binary compatibility – RCP code cannot be reused unrestrictedly)
- RWT will become a subset of SWT functionality

Singletons with Sessionscope

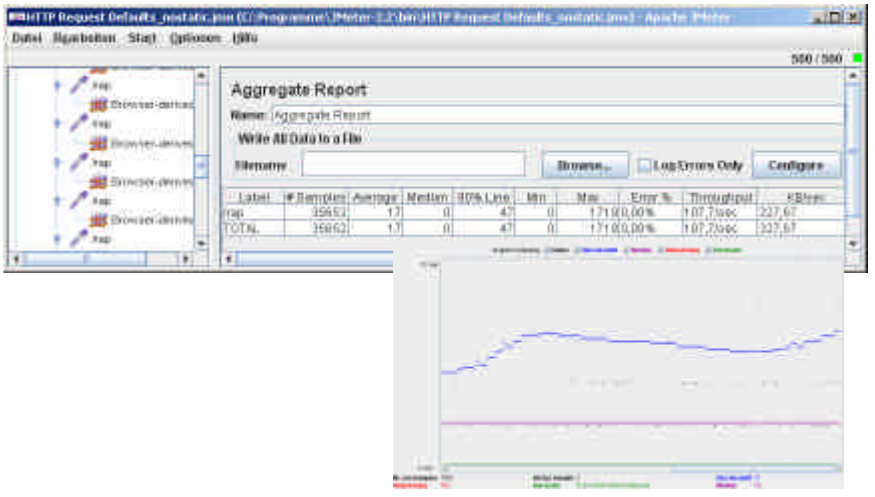
```
public class Workbench extends SessionSingletonBase implements IWorkbench {

  private Workbench() {
  }

  public static Workbench getInstance() {
    return ( Workbench )getInstance( Workbench.class );
  }
```

- Implementation of RCP workbench is a classical singleton
- Within RAP this would cause the workbench to run in application scope
- The RAP workbench holds state-information which belongs to the session scope
- W4Toolkit provides *SessionSingletonBase* to create easily singletons with session scope

but how does it perform?

- RAP does has a per session memory requirement on the server
- we have extensive experience with performance and memory optimization from more than 5 years of W4T
- web workbench performance tests have been conducted
  - Core Duo CPU, 500 MB of heap, demo web workbench
  - 500 concurrent users, one request every five seconds
  - average response times below 20 ms
  - approx. 100 MB heap space
- load balancing possible with standard apache mod_jk / mod_proxy (requires some work)
- W4T based Yoxos on Demand serves several thousand users every day – and has received 5000 ratings of 9 out of 10

sample performance metrics



conclusion

- ajax is here to stay, but it has yet to overcome some obstacles
- ajax does not need to be in contradiction with rich clients – the technologies can complement each other
- shielding ajax complexities is one of the hottest topics today – a java api (swt) has proved to work in rich ui development, but there is also a strong movement to build javascript libraries

- give rap a try - http://eclipse.org/rap/

references

- Eclipse RAP project http://eclipse.org/rap/
- Eclipse Equinox project http://www.eclipse.org/equinox/
- Eclipse Rich Client platform http://eclipse.org/rcp/
- Eclipse ATF project http://eclipse.org/atf
- Google Web Toolkit http://code.google.com/webtoolkit/
- qooxdoo – JavaScript GUI framework http://qooxdoo.org