eclipse rich ajax platform (rap)

Jochen Krause
CEO Innoopract
Member of the Board of Directors Eclipse Foundation
jkrause@innoopract.com
outline

- rich ajax platform
  - project status and background
  - technology overview
The rap project aims to enable developers to build rich, AJAX-enabled web applications by using the eclipse development model, plug-ins and a java-only api.

eclipse plays a significant role in the rich client world

- provides advanced concepts and technologies that can be easily built upon
- "enforces" solid architecture patterns in application development

the goal of the project is to extend the reach of the eclipse platform to ajax

project status: moved from proposal status to "approved, waiting for project lead"

- still no code available in CVS
- eclipse legal process requires "legal clearing" before code can go into CVS
the rap project does not start from scratch, it will receive a code contribution from Innoopract:

- w4t, a widget toolkit that allows development of ajax web ui's in plain java
- technology has proved to work, e.g. with the eclipse download configurator service http://yoxos.com/ondemand - handling 500 concurrent users easily
- the project proposal has received the

The award honours and recognises the most remarkable and outstanding contributions in the world of Java and Eclipse.
current trends in application development

- 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 (with concepts for keeping the client up to date)
  - rich internet application, with a strong focus on ajax technologies
- eclipse has succeeded in delivering a state of the art rich client framework, but the rich client camp is getting pressure from ajax enabled webapps
- the ajax world to date is very colorful, with many very promising technologies and projects. Most of the effort seems to be focused on providing client-side widget toolkits and a communication layer to the server.
ajax suffers from development complexity

- although ajax is a promising vision, the development complexity is very high
- better tools can help
  - e.g. eclipse atf http://eclipse.org/proposal/atf
  - better javascript editors are desperately needed (this can be an area for collaboration)
- frameworks and toolkits can deal with the low level stuff
  - XAP - eXtensible Ajax Platform (Meta-Framework)
  - Kabuki Ajax Toolkit
  - Dojo
  - OpenRico
outline

- rich ajax platform
  - project status and background
  - an eclipse platform strategy
rap leverages and extends the eclipse platform

- rap enables developers to employ the eclipse concepts for developing ajax applications, leveraging the advanced eclipse programming model
- plugin concept – based on osgi, implemented by Eclipse Equinox
- workbench concept – a powerful UI metaphor that facilitates providing a consistent user experience
- a widget toolkit encapsulates all ajax technologies behind Java objects and rendering kits
- only developers who want to create their own widgets need to deal with javascript and ajax
- eclipse as a platform becomes an attractive alternative for ajax development – not only for ajax tooling
a brief example

- webworkbench – look & feel of the eclipse workbench in a browser
  - adding type ahead search

- implementation is not yet based on the eclipse workbench model
  - „hand-coded“ workbench, like creating the workbench look & feel directly in swt
  - showcasing feasibility, performance, look & feel
rap architecture overview

Client side:
- web workbench
- W4T, JFace
- runtime

Server side:
- OSGi

- selection service, action sets, viewparts
- widget toolkit, mvc, handling of distributed environment
- extension points
- modularity, dependency management (bundles / plugins)

based on standard jee technology
eclipse OSGi on the server side

- equinox is providing an two „incubators“ 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 3.2
  - embedding in a servlet container
    - war file to demo is available – starting an eclipse platform server side
    - rap will mainly reuse equinox technology and act as a client
- a new project proposal for eclipse rich server platform has been posted
  - “creation of truly pluggable, componentized, server-side applications“
  - http://eclipse.org/proposals/rsp
eclipse runtime - on the server side

- late bindings
- declarativ
- loose coupling
- contributing
- extending existing implementations

- ONCE per web application
w4t widget toolkit explained

- **Domain models**
  - Viewer components
  - More application specific
  - JFace: Viewers etc.

- **W4T**
  - One programming model: Object-oriented Java

- **SWT**
  - One programming model: Object-oriented Java

---

Everything below this line is encapsulated by the widget library

- **Rendering Kits**
  - for low-level languages: HTML, JS, CSS
  - IE, Netscape 4, Netscape 6, Opera, ...

- **Native libraries**
  - for low-level languages: C / C++
  - dll (Win 32), so (Linux), jni (Mac OS X), ...

---

- **Browser (client)**
- **Low-level diversity**
- **OS (target platform)**
a generic solution for partial UI updates

- send request (client-side)
  - collect form data and submit via XmlHttpRequest
- detect changed components (server-side)
  - hash code based algorithm to trace component state
- apply response (client-side)
  - received HTML fragments are applied to document

- transparent for application developer

- overhead does not affect overall application performance (e.g. WebWorkbench takes 0 – 16 milliseconds to fulfil request)
workbench

- strong coupling between workbench, swt and jface
- need to reimplement core apis to align with widget toolkit (swt api under exploration)

challenges ahead:
- workbench
  - session vs. application scope
  - memory considerations
  - multi user / logins
- layouts
  - absolute positioning, formLayout
- integration with existing web applications
the balance of server and client side

- framework provides client-side handling for workbench parts (open, close, resize)
- widgets can provide client side event processing (e.g. expanding a tree)
- other event processing happens (mostly) server side
  - implementation in java
  - ui changes are calculated on the server side, client will get partial updates
- data binding happens on the server side (jface is the eclipse standard)
- eclipse plugin concept is enabled on the server side inside a web app
  - everything is a plugin (server side) - ui is assembled by contributions (server side) providing a well thought out development model
  - main callenges for running eclipse inside a web app and interacting with a servlet have been solved and are part of the eclipse 3.2 release
  
collaboration with other ajax approaches is important

rap benefits from simple integration of widgets based on common ajax frameworks

- rap's widget toolkit is extensible
  - server side java api (might be moving to swt)
  - rendering kits provide implementation (html, css, js)
  - a canvas can be filled with client side life
  - server side needs info about client state

problems to avoid:
- possible incompatibilities between different libraries
- different versions of libraries

we need to find ways to avoid runtime problems effectively (e.g. ast, classpath, compiler)
how does rap compare to google gwt?

**google gwt is a cool technology**
- running on an emulated java engine in the browser (needs javascript to work)
- javascript is in charge of „drawing“ the user interface
- event handling in GWT is on the client side (+ RPC calls to the server to access data)
- GWT enables a "standalone SWT" comparable approach
- can scale to 100 thousands of concurrent users

**rap is a cool technology, too**
- runs standard html and javascript in the browser (can work with javascript disabled)
- the browsers rendering engine „draws“ the ui, refreshes happen through transfer of html snippets
- RAP relays most client-side events to the server for processing (solely ui related events can be processed on the client).
- running mainly on the server it can access the full java api and make use of OSGi inside a web container, enabling the full usage of the eclipse plugin model
- can scale to thousands of concurrent users
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 about declarative ui development
references

- Eclipse RAP proposal [http://eclipse.org/proposals/rap](http://eclipse.org/proposals/rap)
- Eclipse ATF proposal [http://eclipse.org/proposals/atf](http://eclipse.org/proposals/atf)