Embracing Eclipse Orion

Andy Clement, Staff Engineer, aclement@vmware.com  @andy_clement
Who am I?

- Based in Vancouver, lead the language/framework tooling team
  - Background in hacking Eclipse JDT
  - Team delivers:
    - AspectJ
    - AJDT
    - Groovy-Eclipse
    - Grails IDE (for Eclipse)
    - Gradle integration for Eclipse
  - Contributes to the Spring Tool Suite, Groovy/Grails Tool Suite

- Most recently created the Scripted project
What is this talk about?

- Using pieces of Orion to build something new
- Why we built something new
- What we built and the technology used
  - Demos demos demos
- Feeding that back into Orion
- What next?
Why get into the JS space

1. **JavaScript was coming onto the radar**
   - Spring users
   - Node.js
   - Tooling less than ideal

2. **Cloud development rising in popularity**
   - Cloud9/Codenvy/etc

3. **Rising opposition to IDEs?**
   - Some developers preferring textmate/sublime over an IDE
   - Many new developers, especially JS devs, wouldn’t dream of using an IDE

4. **Getting a bit bored of writing eclipse plugins!**
Interviews

- Interviewed JS developers
  - What is good/bad about your tools?

- No real surprises
  - Source code awareness invaluable, early error indication
  - Current tools content assist and code navigation not great
  - Debugger integration would be nice (and better debugging for JS callback hell)

- What they didn’t say…
  - No users (yet) demanding cloud based development
  - No-one asking for collaborative code editing
  - No-one asking for ‘social coding’ facilities
  - No real request for one uber tool that does everything…
Prototypes…

- **Prototype #1**
  - Eclipse Orion + extras (content assist, cloud foundry integration)
  - Very little interest internally
  - Benefits (better JS dev experience) didn’t outweigh the costs (cloud workspace)

- **Learned a lot!**
  - JS development can be painful

- **Prototype #2**
  - Browser based editor experience was good – eclipse orion very familiar
    - Doesn’t mean server has to be remote
  - Build something to solve *our* problems
  - Orion self hosted not quite the flow we wanted
Use Orion in self hosted mode?

- Not quite the flow flow we wanted
  - Wanted a more traditional old-school editor feeling
  - Wanted a single-page-app structure

- Provided more facilities than we needed (git integration/security/etc)

- Content assist quite limited
  - needed server side component to understand a project

- Self hosting mode a little cumbersome to setup

- Wanted to explore light table like features
  - UI flexibility didn’t feel quite there in default Orion

- *Orionode addresses some of these issues*
Orion structure

Eclipse Orion Client (Multi-page JavaScript/HTML/CSS)

- Navigator
- Git integration
- Site hosting
- Sign-on

- Editor
  - Syntax highlighting
  - JSLint
  - Content assist

Eclipse Orion Server (Java/OSGi)
Orion structure

Eclipse Orion Client (Multi-page JavaScript/HTML/CSS)

- Navigator
- Git integration
- Site hosting
- Sign-on
- Editor
  - Syntax highlighting
  - JSLint
  - Content assist
Scripted structure

Scripted Client (Single-page JavaScript/HTML/CSS)

- Sidebar Navigator
- Dialogs
  - Outline/OpenFile/Search
- Syntax highlighting
- JSLint > JSHint
- Enhanced Content assist

Scripted Server (Node.js)

- Dependency Analyzer
The technology in Scripted

- Orion Editor
  - Some reuse issues: messy dependencies, ordering issues for listeners/stylers
- Dependency injection with wire.js (see cujo.js for more info…)
- Doctrine for jsdoc parsing
- And…
The technology in Scripted - Esprima

- Esprima

- Made recoverable for our purposes
  - Nice to see all your errors, not just the first
  - Code in an editor is typically a work in progress, often ‘unfinished’
    - Still want an Ast: for navigation, content assist, outline, etc
  - Usual recovery techniques:
    - Recognizing missing tokens
    - Rewinding and trying alternate parse routes
    - Adding new parsing rules allow for ‘typical’ errors
  - Demo

- Key component of the inferencing system
Inferencing system

- Builds on the recoverable parser
- Control flow analysis
  - Walks the AST and decorates it as information is learned:
    ```javascript
    function doSomething() {
      return { val: 9 };
    }
    var x = doSomething().val;
    x.
    ```
  - Only single pass, fast, a good enough starting point
  - Doesn’t see runtime properties
- Quite limited if only works on a single file...
Inferencing system: dependency analyzer

- Inferencer and dependency analyzer co-operate
  - For content assist and navigation

- Various module systems exist for JS

- Picked common module systems initially:
  - AMD (used in orion JS code)
  - CommonJS (used by Node)

- Quite simple:
  - Inferencing system asks server to compute transitive dependency graph for a file
  - Inferencer runs over dependencies
  - Easy for CommonJS, harder for AMD
Dependency analyzer: AMD modules

- **Typical loader, like RequireJS is highly configurable**
  - Supports plugins
  - Paths can be remapped
  - Package declarations

- **Scripted policy**
  - Standard module configs just work, anything funky supported through extra configuration

- **Implementation**
  - Find html file with data-main, find configuration side

```
index.html
<script data-main="setup.js"
   src="lib/require.js"></script>
```

```
setup.js
requirejs.config({
  packages: [
    { name: 'probes',
      location: '../components/probes',
      main: 'probe'},
    { name: 'when',
      location: '../components/when',
      main: 'when'},
    { name: 'wire',
      location: '../components/wire',
      main: 'wire'},
  ],
  paths: {
    "wire/domReady": 'lib/domReady',
    jquery: 'lib/jquery-1.7.2.min',
    sockjs: 'lib/sockjs-0.3.1.min',
  }
});
```
Demo time

Scripted Demo
Results of prototype 2

- **Much more interest**
- **Some JS hackers in VMware have switched to it**
  - They would like: more refactoring support, better debugger integration

**What next?**
- More code comprehension – more module systems
- More hints to inferencing – typescript/vjet defs
  - Improving library support for content assist
- Explore some ‘light table’-like behaviours
- Other languages: Java?
Feeding back into orion

- **Same interfaces maintained for content assist**
  - Advanced content assist just drops back in
  - No cross file support but exploring integration of dependency analyzer into orionode

- **Other features worth moving/porting back:**
  - Improved tooltips: Jsdocs and inferred signature formatting
  - Editor auto-indent/unindent
  - Refactoring
The End

Any questions?

- **Resources**
  - Scripted Project
    - [https://github.com/scripted-editor/scripted](https://github.com/scripted-editor/scripted)
  - New and noteworthy:
  - SpringSource blog: various articles on Scripted
  - Eclipse Orion

**Twitter:** @andy_clement

**email:** aclement@vmware.com
Give Feedback on the Sessions

1. Sign In: www.eclipsecon.org

2. Select Session Evaluate

3. Vote +1 0 -1