Extension example: text formatting

Most email clients turn this:

This text should be _underlined_
This text should be /italic/
This text should be *bold*
Extension example: text formatting

Most email clients turn this:  
This text should be _underlined_
This text should be /italic/
This text should be *bold*

into this:  
This text should be **underlined**
This text should be *italic*
This text should be **bold**
Extension example: text formatting

Most email clients turn this: 

This text should be _underlined_
This text should be /italic/
This text should be *bold*

into this:

This text should be **underlined**
This text should be *italic*
This text should be **bold**

Gmail doesn’t – how can we add this feature?
Motivation: Extending the Web

- Extending Web Applications
  - “User scripts”
  - Bookmarklets

“When Gmail loads, run this script to preprocess and reformat the messages”
Motivation: Extending the Web

- Extending Web Applications
  - “User scripts”
  - Bookmarklets
- Extending the Browser
  - Firefox extensions
  - Chrome extensions, Opera widgets…

“When I open a new tab, show me thumbnails of my ten most recently visited websites”
Motivation: Extending the Web

- Extending Web Applications
  - “User scripts” - 40,000 scripts
  - Bookmarklets

- Extending the Browser
  - Firefox extensions - 6,000 extensions
  - Chrome extensions, Opera widgets…

- Very popular

- Unknown numbers of problems…
Userscript: Formatting Gmail messages

Most email clients turn this: This text should be _underlined_ into this: This text should be **underlined**

2. Replace the function with a new one...

```javascript
var oldP = unsafeWindow.P;
unsafeWindow.P = function(iframe, data) {
    if (data[0] == "mb")
        data[1] = format(data[1]);
    return oldP.apply(iframe, arguments);
}
```

1. Save the original function

3. ... that transforms punctuation to HTML

4. ... and then calls the original function
Extension: “SpeedDial” Firefox tabs
Extension: “SpeedDial” Firefox tabs
Extension: “SpeedDial” Firefox tabs

1. Find the function
2. Retrieve its source code...
3. ...edit that string to include changes...
4. And replace the function with the modified version

```
SpeedDial.init = function () {
...
  eval("getBrowser().removeTab = "+
    getBrowser().removeTab
    .toString()
    .replace(
      'this.addTab("about:blank");',
      'if (SpeedDial.loadInLastTab) {
        this.addTab("chrome://speeddial" +
        '/content/speeddial.xul"' +
      } else {$(!)}
  ));
...
}
```
How are extensions written?

- Mostly in JavaScript, HTML, and CSS
- No overarching software engineering guidelines
- “Just get it to work”
  - HTML & CSS: Overlays
  - JS: Wrapping
  - JS: Monkey-patching
Drawbacks of these approaches

- Wrapping is inflexible
  - can’t be “inside” the function

- Monkey patching is brittle
  - Patch may silently fail, may introduce syntax errors

- Both are incorrect for closures
  - They are new closures that have the wrong environment

- Both are incorrect for aliases
  - All other aliases are unmodified

- What to do?
Aspects

- An aspect defines what new code to run, and when to run it.
- Advice defines what new code to run.
- Pointcuts define when to trigger it.

```java
at pointcut(callee(Math.sin))
before(x) {
    print("x is ", x);
}
```
Advising names versus closures

Global object

window

Window object

P

Closure for P
Advising names versus closures

Suppose `preprocess == window.P`
Userscript does not correctly advise preprocess!

Suppose `preprocess == window.P`

Advising names versus closures
Advising names versus closures

Suppose `preprocess == window.P`

Advice is visible to all aliases to `window.P`
Aspects for functions

- Support *before*, *after*, and *around* advice
  at pointcut(callee(launchMissiles)) around(x) {
    if (!authorized(x))
      print("WARNING!!!");
    else if (proceed() == false) {
      print("Launch failed");
    }
    return retval;
  }
Revisiting the Gmail userscript

- Use before callee advice:

```javascript
var oldP = unsafeWindow.P;
unsafeWindow.P = function(iframe, data) {
    if (data[0] == "mb")
        data[1] = format(data[1]);
    return oldP.apply(iframe, arguments);
};
```

```javascript
at pointcut(callee(unsafeWindow.P)) before(iframe, data) {
    if (data[0] == "mb")
        data[1] = format(data[1]);
}
```
Revisiting the SpeedDial extension

- **Use** before `statementContaining` advice:

  ```javascript
  eval("getBrowser().removeTab = "+
    getBrowser().removeTab
    .toString().replace(
      'this.addTab("about:blank");',
      'if (SpeedDial.loadInLastTab) ' +
      'this.addTab("chrome://speeddial/content/speeddial.xul"); else $!'
    ));
  
  at pointcut(statementContaining(this.addTab("about:blank")) &&
  within(getBrowser().removeTab)
  before {
    if (SpeedDial.loadInLastTab)
      this.addTab("chrome://speeddial/content/speeddial.xul")
    else
      this.addTab("about:blank")
  }
  ```
“Full Disclosure”: Aspects with a twist

- We want to support an aspect language for JS that:
  - Has standard before/after/around function advice
  - Can access local variables within functions
  - Can syntactically revise existing functions
  - Is as easy as monkey-patching

- ...and:
  - Introduces minimal runtime overhead
  - Permits static analysis of aspects

- Technique: *open up closures and modify them*
Implementing Aspects for JS

- Targeting MSR JScript Compiler
- High-level relevant features:
  - Entirely managed code
    - Runtime environment, generated code are standard .NET
  - JIT compilation
    - Can implement weaving as we JIT the code
    - Makes dynamic weaving “almost free”
Future work
Future work
Future work
Future work

- Given aspects that can replace most monkey-patches, efficiently and more correctly...
- Re-implement this scheme for other JS engines
- Manually translate more extensions to use aspects, and see how often they’re useful
- Develop static analyses to detect conflicts among extensions before runtime.