Wishlist for Web Programming

Peter Thiemann

Universität Freiburg

Links Meeting, Edinburgh, Scotland, 6 April 2005

The Structure of Modern Web Sites

kinds of content	static	generated
passive	images downloads stylesheets	contents of files data bases
executable	traditional scripts	generated scripts

- Usually a mix
- About 50% of all sites have executable content

Thesis

- More than 90% generated content for some sites (search engines, news services, blogs, . . .)
- Much of it programmed in an ad-hoc way (CGI, Perl, PHP, . . .)
- Appropriate programming technology sorely needed

From the Structure . . .

static/passive $\Rightarrow$ web server

generated/passive • access to input

- database and file access
- computation
- output generation: templates, transformations

static/executable $\Rightarrow$ web server

- does it fit with the static parts?

generated/executable computation $\Rightarrow$ meta programming

Wishlist

- programming model
  - session concept
  - callback concept
  - composition of functional components (parameterization)
  - quality assurance (type safety)
  - support for programming in the large (abstraction, parameterization)
- features
  - XML generation
  - database access
  - support for transactions
  - XML processing (mostly for Web services)
  - email, instant messaging
  - other APIs (Java based?)

Subjective Reflection

- some systems, *e.g.*, BigWig, JWig, WASH, PLT-Scheme, . . .
  - deliver on the programming model
  - do not score highly on features

  ⇒ consequently, they are not widely used
- PHP (Perl, Python)
  - score badly on the programming model/maintenance/. . .
    - ∗ unchecked string references (href and action attributes) between pages
    - ∗ retrieval of input fields through unchecked strings
    - ∗ input delivered in terms of strings
  - feature-laden; easy access to Java APIs
  - leading deliverator of dynamic content on the web today
- JSP scores better in all respects, but is much less frequently used

What Seems to Make a Web Programming Technology **Successful** . . .

Features, **Features**, **Features** plus

- Familiar concepts (kills WASH)
- Low learning curve (kills WASH, *Wig, JSP)
- Seamless integration (kills BigWig)
- Ease of development and deployment (kills JSP)

How to sell technology like WASH?

- keep the features but change the host language to JavaScript
  - fix up quirks of the language
  - add static typing; nominal types (classes); constrained polymorphism
- integrate server-side scripting with client-side scripting
  - less diversity in application development
  - interaction between client and server part of application checkable by compiler
- migration path: untyped $\Rightarrow$ typed islands $\Rightarrow$ fully typed

On JavaScript

- industry standard (EcmaScript)
- right visibility and apparent familiarity (it has objects)
- low learning curve
- rich feature set
- libraries available
- client-side applications abundant
- server-side: existing application servers as backend (`whitebeam.org`, `helma.org`, `cocoon.apache.org`)
- but a weak dynamic type system

Example Web Script

- Display a time-dependent greeting
- Read in a name and echo a personalized greeting
- Two styles
    1. Presentation and application logic muddled up
    2. Clean separation between presentation (skin) and application

  $\Rightarrow$ Observe that skins are pure HTML

  $\Rightarrow$ Designers need not know about programming technology

```
function main () {
   var today = getDate ();
   ask <html><head><title>Greeting</title></head>
         <body><p>Today is {today}
                  <input type="submit" name="{daytime (today)}" /></p>
               <p>Enter your name <input type="text" name="{who}" />
                  <input type="submit" name="{greet (who)}" /></p>
         </body>
       </html>
}
function daytime (date) {
   var currentTime = getTime ();
   var what = phrase (currentTime);
   ask <html><head><title>Daytime</title></head>
         <body>It’s {what} of {date}!
         </body>
       </html>
}
function greet (who) {
  ask <html><head><title>Greeting</title></head>
        <body>Hello, {who}!
        </body>
      </html>
}
```

```
function main () {
   var today = getDate ();
   ask (mainSkin (today))
}
```

```
function mainSkin (today) {
  <html><head><title>Greeting</title></head><body>
    <p>Today is {today}
    <input type="submit" name="{daytime (today)}" /></p>
    <p>Enter your name <input type="text" name="{who}" />
    <input type="submit" name="{greet (who)}" /></p>
    </body>
  </html>
}
```

```
function daytime (date) {
   var curTime = getTime ();
   var what = phrase (curTime);
   ask (daySkin (what, date))
}
```

```
function daySkin (what, date) {
  <html><head><title>Daytime</title></head>
    <body>It's {what} of {date}!
    </body>
  </html>
}
```

```
function greet (who) {
  ask (greetSkin (who))
}
```

```
function greetSkin (who) {
  <html><head><title>Greeting</title></head>
    <body>Hello, {who}!
    </body>
  </html>
}
```

From JavaScript to WASH/JS

- JavaScript is untyped

  ⇒ create type system and/or static analysis

  ⇒ leads to “better JavaScript”

  ⇒ helps discover errors in existing programs

  ⇒ see paper @ ESOP’05

- JavaScript is interpreted

  ⇒ create compiler for suitable subset

  ⇒ can exploit analysis results

- JavaScript is weird

  ⇒ No, the browsers’ object hierarchy differs between vendors

  ⇒ Well, see the ESOP paper