Greater Expectations

AJAX Based Web Applications

Steffen Meschkat
Google

28. Dezember 2005

Overview

- The name of the game
- The rules, or what really happens
- The consequences, or what's essential beyond the hype
- Technological Components
- Engineering Practice
- Conclusion, or the usual prophecies

AJAX

What's in the name

- Asynchronous
- JavaScript
- XML

... but:

- all of the parts are either nonessential or redundant.
- boils down to client side scripting, but the name CSS is taken.
- a bad name, however, is better than no name.

What happens in AJAX applications

Classic Web Application

1. click on a link / submit of a form
2. HTTP request
3. HTTP response **replaces** the document.

- *generic event–action mapping*

Modern Web Application

1. user event
2. invokes a scripted event handler
3. event handler *may* initiate data transfer
4. user event handler or transfer callback **updates** the document

- *specific event–action mapping*

Consequences

Sophisticated User Interaction

- display can be partially updated, modified, animated
- complex manipulations are possible
- *user interaction like in 1990*

Client Side Session State

- transient session state is managed on the client
- persistent user state maintained on the server
- *corrects a long standing architectural aberration*

Intermission

The bad thing about doing something right the first time is that nobody appreciates how difficult it was.

Web technologies give us plenty of opportunity to appreciate how difficult it was.

Successful technologies are used for things they were never intended for, and people complain how inadequate they are.

Web technologies are very successful indeed.

Technological Components

CSS – Cascading Style Sheets

- defines visual layout properties, etc.

DOM – Document Object Model

- API for structured text

JavaScript

- nice scripting language with an undeserved bad reputation

HTTP

- transport for background data transfer
- IFRAME/onload or XMLHttpRequest/onreadystatechange

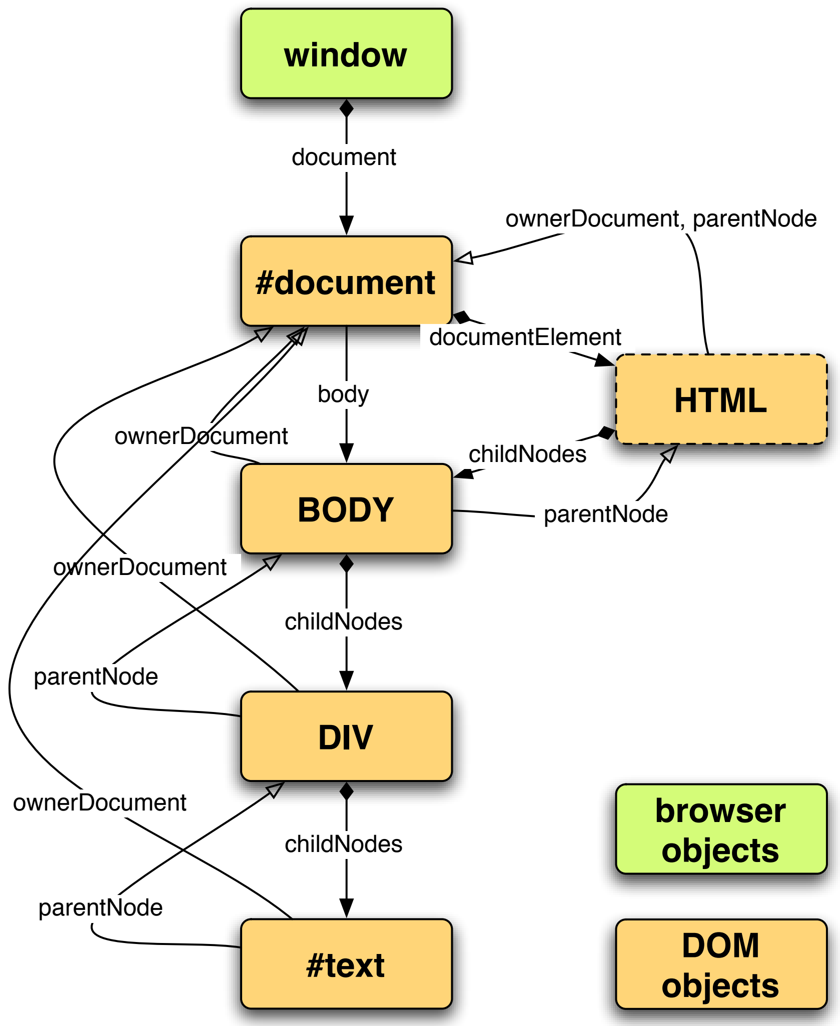
Transfer Data Format

- XML or JavaScript object literals (JSON)

Illustration: W3C DOM

```
interface Node {
  readonly attribute DOMString        nodeName;
           attribute DOMString        nodeValue;
  readonly attribute unsigned short   nodeType;
  readonly attribute Node             parentNode;
  readonly attribute NodeList         childNodes;
  readonly attribute NamedNodeMap     attributes;
  // ...
}
```

Illustration: W3C DOM



```
<html>
  <body>
    <div>hello world</div>
  </body>
</html>
```

JavaScript

The language has been ridiculed for its name because it its semantics are so far from Java. However ...

- it was invented when Java was a client side technology,
- used to tie Java applets into their pages, and control Java objects, called **LiveConnect**,
- but Java failed as CS technology (*"compile once, debug everywhere"*),
- as did, btw., JavaScript as a server side technology,
- hence java objects are no longer there; script DOM objects instead.

JavaScript

Better than its reputation:

- custom dynamic objects, constructors,
- prototypes and delegation,
- functions are objects,
- dynamic functions, closure,
- rich literals,
- exceptions,
- lisp with C syntax.

However:

- insane scope rules,
- semicolon insertion,
- dynamic (i.e., no) typing.

JavaScript and HTML

Lexical, syntactic, and semantic conflicts with HTML

```
<script><!--
//<![CDATA[
if (a < b) ...;
document.write('</' + 'script>');
//]]>
//-->
</script>
```

- Markup characters <, >, & in the script
- end tags `</script>` in the script
- Script is displayed as text if script element is not recognized

Illustration: XMLHttpRequest Document Transport

```
function get(url, callback) {
  var transport = new XMLHttpRequest;

  transport.onreadystatechange = function() {
    if (transport.readyState == 4) {
      callback(transport.responseXML);
    }
  }

  transport.open('GET', url, true);
  transport.send(null);
}
```

details missing: cross browser transport instantiation

Illustration: IFRAME Document Transport

```
function get(url, callback) {
  var transport = document.createElement('IFRAME');

  transport.onload = function() {
    callback(transport.contentDocument);
  }

  transport.src = url;
}
```

details missing: response data format details, iframe life cycle

Illustration: XML vs. JSON – Text Format

XML

```
<data>
 <location lon="13.4156" lat="52.5206"/>
</data>
```

JSON

```
var data = {
  location: {
    lon: 13.4156,
    lat: 52.5206
  }
};
```

Illustration: XML vs. JSON – API

XML

```
var lon = datanode.firstChild.getAttribute('lon') - 0;
```

JSON

```
var lon = data.location.lon;
```

Intermission

FAUST

Who then art thou?

MEPHISTOPHELES

Part of that power which still
Produceth good, whilst ever scheming ill.

(*Goethe: Faust.*)

Practical Consequences

Cross Browser Compatibility

- different implementations of all mentioned technologies
- different (but always many) bugs
- enforces good libraries

Separation of interaction logic and application logic

- implemented in different languages
- separated by flexible and extensible protocol
- provides containment of idiosyncrasies

Illustration: Event Handler Registration, I

W3C DOM

```
node.addEventLister('click', function() {
  alert('clicked ' + this.nodeName);
}, true);
```

IE

```
node.attachEvent('click', function() {
  alert('click on ' + this.nodeName);
});
```

old style

```
node.onclick = function() {
  alert('click on ' + this.nodeName);
};
```

Illustration: Event Handler Registration, II

```
Event.listen = function(node, event, handler) {
  if (browser.type == SAFARI && event == 'dblclick') {
    node['on' + event] = handler;

  } else if (node.addEventListener) {
    node.addEventListener(event, handler, false);

  } else if (instance.attachEvent) {
    node.attachEvent('on' + event, handler);

  } else {
    node['on' + eventName] = handler;
  }
}
```

Practical Challenges

Deployment

- not yet mature
- e.g., cf. compilation, modularization, cache control

Bookmarking and History

- essential web interaction patterns
- don't work in most applications with server side session state

Frameworks

- or rather, to resist the temptation to build one
- because there already is one

The Usual Prophecies

Mobile code

- holds the promises that java made; yet one more reason to call it JavaScript

Application state migration

- bookmarking, if it works, does exactly that

Software as a service

- as close as it currently gets

Client side application integration

- limited by cross site scripting and transclusion restrictions
- cf. Google Maps API