Web Service Foundations

Instructor: Peter Baumann

email: p.baumann@jacobs-university.de

tel: -3178

office: room 88, Research 1

```
#titanic {
   float: none;
}
```

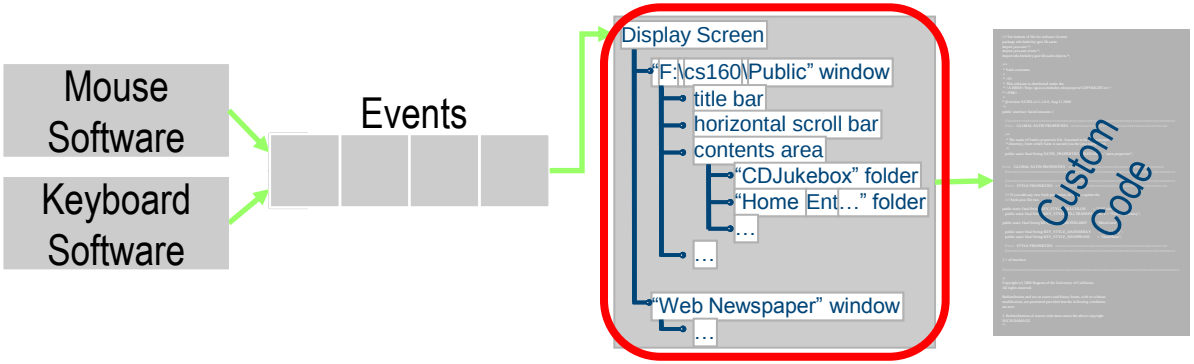
Overview

- Information modelling → DOM
- Communication modelling → AJAX

Document Object Model (DOM)

From General GUI to Web DOM

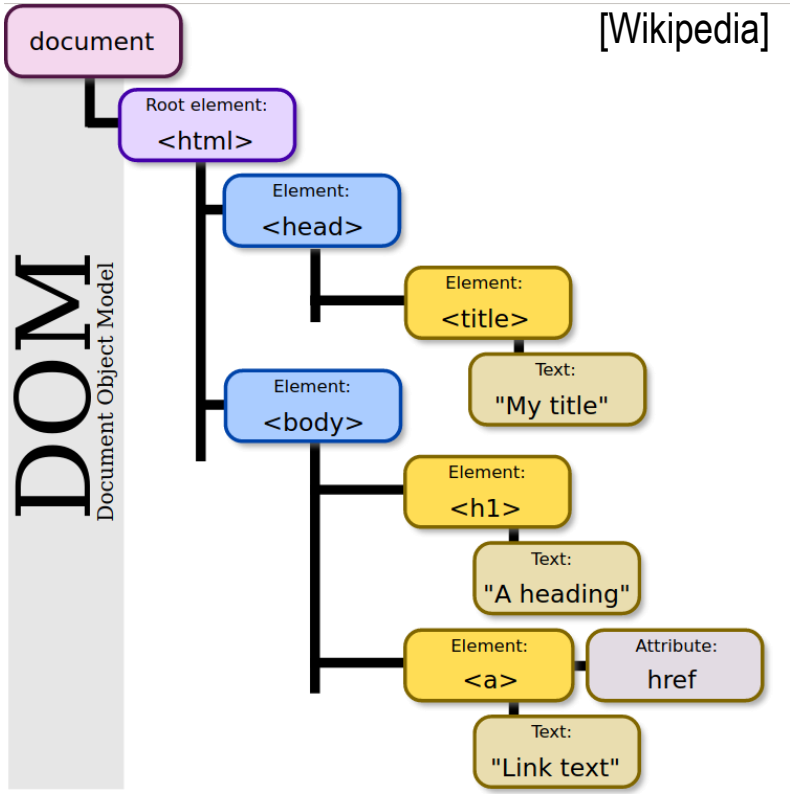
- Remember GUI architecture: Interactor Tree



- Web browser = aka specialized Mini-GUI
  - GUI Interactor → Document Object Model (DOM)
  - HTML (with CSS) = encoding for DOM trees
  - „Custom Code“ = JavaScript, DOM manipulation + AJAX

DOM Definition

- DOM = tree representing browser window contents
  - Element (“node”) = object + attributes + text
  - Document: root node
- Manipulation through JS methods, ex:
  - document.createElement("div")
  - document.getElementById("table");
  - document.getElementById(“div”).innerHTML
  - body.onload



CSS

- CSS = simple language for pre-setting of DOM attributes
- Factors out appearance (CSS file) from contents (HTML file)
- Examples:
  - H1 { font-size: 15pt; }
  - .larger { font-size: larger; }
    - *here is my text*
  - pre { overflow: auto; }

AJAX
(Asynchronous Javascript and XML)

History

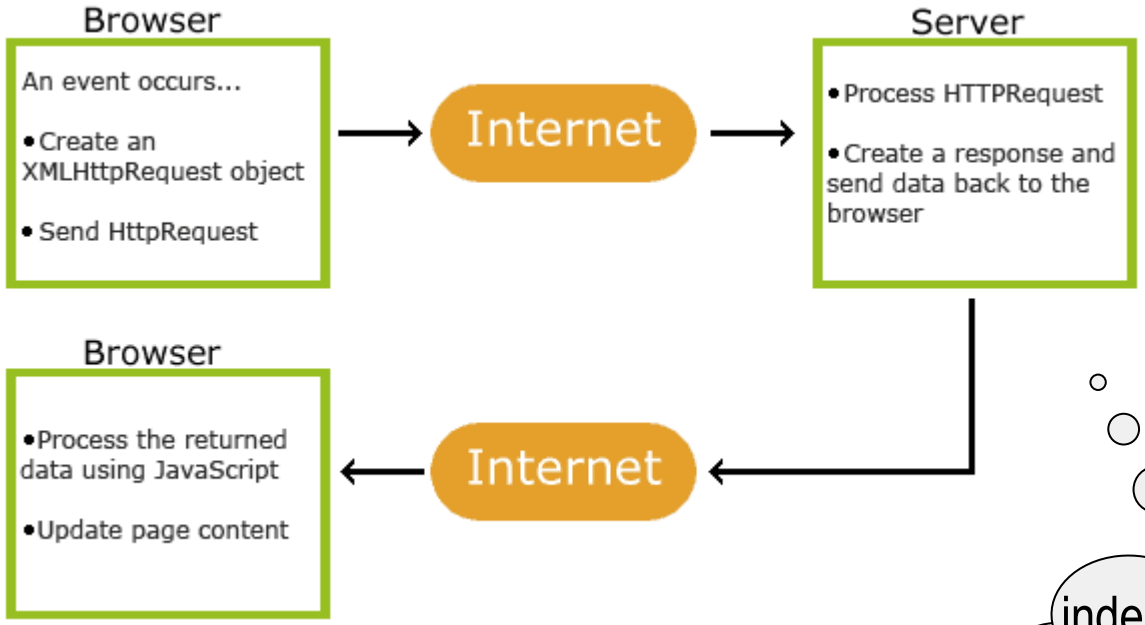
- Challenge: want more interactivity than "click link / reload *complete* page“
  - HTML's `iframes`
- Microsoft IE5 XMLHttpRequest object
  - Outlook Web Access, supplied with Exchange Server 2000
- 2005: term "AJAX" coined by Jesse James Garnett
- made popular in 2005 by Google Suggest
  - start typing into Google's search box → list of suggestions

AJAX

- AJAX = Asynchronous Javascript and XML
- web development technique for creating more interactive web applications
  - Goal: increase interactivity, speed, functionality, usability
  - not complete page reload → small data loads → more responsive
- asynchronous: c/s communication independent from normal page loading
  - JavaScript + ~~XML~~ + any server-side PL

JSON

AJAX Client/Server Communication



independent from server implementation

communication decoupled from user interaction

[w3schools.com]

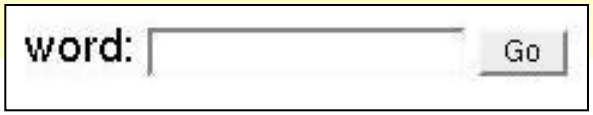
AJAX Constituent Technologies

- The core: JavaScript XMLHttpRequest object
  - Sends data, waits for response via event handler
  - Replaces <FORM> and HTTP GET / POST
- Client DOM manipulated to dynamically display & interact
  - Inject response into any place(s) of DOM tree
  - client-side scripting language: JavaScript, Jscript, ...
- Some data format
  - XML, JSON, HTML, text, ...
- Some server agent
  - Servlet, script, ...

Ajax Example: Traditional Style

- Client:

```
<form method='GET' action='http://.../ajax-ex.php'>
   word:
   <input name='wordKey' type='text'>
   <input type='submit' value='Go'>
</form>
```



- Server:

```
<?
   echo 'You have entered ' . $_GET['wordKey']
      . ' and your IP is: ' . $_SERVER['REMOTE_ADDR'];
?>
```

- Client, after page reload: You have entered Moribundus, and your IP is: 127.0.0.1

Step 1: Avoid Complete Page Reload

```
<form name='wordForm'>
   word:
   <input name='wordKey' type='text'>
   <input type='button' value='Go' onClick='JavaScript:callBack()'>
   <div id='result'></div>
</form>
```

```
function callBack()
{  var SERVICE = 'http://.../ajax-ex.php';
   var req = new XMLHttpRequest();
   var val = document.forms['wordForm'].wordKey.value;
   req.open( 'GET', SERVICE+'?wordKey='+val, true );
   req.setRequestHeader( 'Content-Type',
                         'application/x-www-form-urlencoded' );
   req.send( null );
   req.onreadystatechange = function()
   {  if (req.readyState == 4)
         document.forms['wordForm'].result.innerHtml =
            req.responseText;
   }
}
```

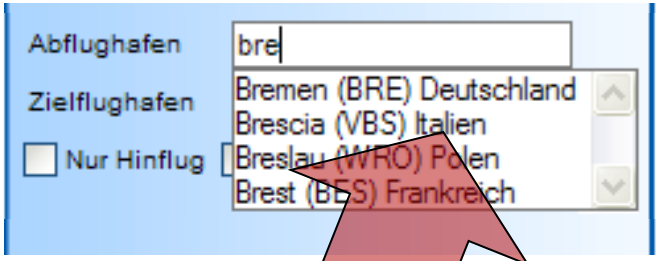
0	request not initialized
1	request set up
2	request sent
3	request in process
4	request complete

word: _____

You have entered Moribundus, and your IP is: 127.0.0.1

Step 2: Avoid SUBMIT Button

- Before: just re-implemented submit; now: allow c/s activity at any time
  - Event handlers
- Ex: suggest keywords with every char typed
  - No submit button!




```
<input name='wordKey' onKeyUp='JavaScript:callBack()'>
```

```
<? ...
   $query = "select entry from Airports
             where entry like '" . $_GET['wordKey'] . "%'";
   $result = mysql_query( $query );
   while ($row = mysql_fetch_array( $result ) )
   {
       print $row[ 'entry' ] . ",";
   }
?>
```

How to ship back & inject data?

Step 3: Selective Page Update

- response parsing code:

```
req.onreadystatechange=function()
{   if(req.readyState==4)
    {   var p = eval( "(" + req.responseText + ")" );
        document.myForm.firstName.value = p.firstName;
    }
}
```

- JSON string sent from server:

```
{   "firstName": "John",
    "lastName": "Smith",
    "address":
    {   "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [ "212 732-1234", "646 123-4567" ]
 }
```

- Server sends:

```
<?  echo '{' + '"firstName":' + obj.firstName + ','
             + '"lastName":'  + obj.lastName  + ','
            …
       + '}'
?>
```

JSON Security Concerns

- JavaScript `eval()`
  - most JSON-formatted text is also syntactically legal JavaScript code!
  - built-in JavaScript `eval()` function executes code received
- Invitation to hack:
  - embed rogue JavaScript code (server-side attack)
  - intercept JSON data evaluation (client-side attack)
  - Safe alternative: `parseJSON()`
- Cross-site request forgery (XSS attack):
  - malicious page can request & obtain JSON data belonging to another site

AJAX / JSON Portability

- AJAX uses standardized components, supported by all major browsers:
  - JavaScript, [XML | JSON], HTML, CSS
- **`XMLHttpRequest`** object part of std DOM
  - Windows: ActiveX control Msxml2.XMLHTTP (IE5), Microsoft.XMLHTTP (IE6)
- Code needs to detect at runtime: feature detection („sniffing“)
  - JavaScript: `if (testElem.style.flex !== undefined && testElem.style.flexFlow !== undefined) {…}`
  - CSS: `@support { … }`
  - Modernizr
- Frameworks provide abstraction layers
  - Angular, react.js, jquery, d3, ...

Sample Tool Support: jQuery

- JavaScript library, http://jquery.com
- Code examples:

```
$( "button.continue" ).html( "Next Step..." )
```

```
$.ajax({
  url: "/api/getWeather",
  data: {
    zipcode: 97201
  },
  success: function( data ) {
    $( "#weather-temp" ).html( "<b>" + data + "</b> degrees" );
  }
});
```

Demo: jquery

Appraisal: AJAX Advantages

- Reduced bandwidth usage
  - No complete reload/redraw
  - HTML generated locally
  - only actual data transferred
  - $\rightarrow$ payload coming down much smaller in size
- Deferred loading
- Separation of data, format, style, and function

Appraisal: AJAX Disadvantages

- **Browser integration**
  - not in browser history, bookmarks
- **Search engine optimization**
  - Indexing?
- **Web analytics**
  - Tracking of accessing page
    vs portion of page
    vs click?
- **Response time**
  - effects from delays
    sometimes difficult to understand for users
- **Reliance on JavaScript**
  - Large (!) script files → delayed loading
  - IDE support...emerging
  - Users can disable JavaScript
- **Security**
  - Malicious code
  - XSS

Summary

- DOM = tree structure representing content of browser window
  - Nodes („elements“) with attributes
  - Predefined attributes for appearance; extensible with own attributes
- CSS = settings for attributes
- JavaScript = dynamic manipulation of DOM tree
- AJAX allows to add desktop flavour to web apps
- Web programming paradigm based on existing, available standards
- Issues: browser compatibility, security, web dynamics
- Manifold usages – see modern Web pages
  - real-time form data validation; autocompletion; bg load on demand; responsive design; sophisticated user interface controls and effects (trees, menus, data tables, rich text editors, calendars, progress bars, ...); partial submit; mashups (app mixing); desktop-like web app

In a Nutshell

- DOM = tree structure for Web content
- HTML = contents & structure → DOM elements
- CSS = styling & handling → DOM attributes
- JavaScript = dynamics → functions on DOM
- JSON = JavaScript subset → DOM data