# Module 13

Cool (!?) Stuff

(JavaScript, AJAX, Greasemonkey)

# Overview

- You have learnt how things should be
  - XML, Namespaces, XML Schema
  - Web Services / REST
  - XPath/XQuery/XSLT, XUpdate, XQueryP
  - Fulltext
- Next: Reality
  - JavaScript, AJAX, ...
  - perceived as cool because (so far) no alternative
- Can we turn XML/XQuery into reality?
  - if not you, who?

# References

- N. Zakas: Professional JavaScript :-)
- D. Crane et al.:  Ajax in Action  :-(
- M. Pilgrim: Greasemonkey Hacks
- Some articles on the Web + demos
- (There are tons of books and articles; these are just the ones I used.)

# JavaScript

- Invented by Netscape in 1995
  - syntax and concepts from Java (thus the name)
  - regular expressions from Perl
- Goal: validate form input at the client
  - server roundtrip was long and expensive
  - waiting half a minute for an error msg. is annoying
- Microsoft followed: JScript (avoid „Java“ in name)
- JavaScript is composed of three parts
  - ECMAScript: First standardized in 1997
  - DOM: W3C Standard (1998-2004)
  - BOM: browser-specific model (windows, cookies, etc.)
- Implementations / compliance differ highly
  - a whole chapter in my book on „browser detection“

# ECMAScript

- First standard in 1997 based on Netscape proposal
- Variables are dynamically typed (vs. XML: optional)

```
var test = „hi“;   // test is a string
test = 55;  // legal! test is turned into a number
```

- Very forgiving syntax and semantics
  - e.g., semicolons are optional
- Runs in a browser („hello world“ in browser)

```
<script type="text/javascript">
   alert("Hello World!");
</script>
```

# Typing in ECMAScript

- Variable contain two kinds of values
  - primitive values (e.g., number, string, ...)
  - reference to an object (just like in Java)
  - (functions are objects; so var can refer to a funct.)
- typeOf function: determine type of value of var
  - boolean, number, string
  - null: reference to an object (indpendent of class)
  - undefined: special value if uninitialized
- (Anekdote: There was a bug in Netscape concerning the typeOf function. Standard was created around that bug.)

# Numbers

- Support for integers and float
  - infinity, - infinity, NaN
  - octal and hex also supported (in fact any base)

```
var i = 10;
alert(i.toString(16));  // outputs „A"
```

- Parsing of numbers:

```
parseInt(„1234blue");  // 1234
parseInt(„blue");      // NaN
parseInt(„22.5");      // 22
parseInt(„A", 16);     // 10
parseInt(„0xA");       // 10
parseFloat(„0xA");     // NaN
parseFloat(„22.34.5"); // 22.34
parseFloat(„22blue");  // 22.0
```

# Other built-in types

- Strings
  - Unicode (UTF-8) encoded
  - C conventions for eol, etc.  (e.g., „\n“)
  - Java conventions for concatenation etc.
    - warning: performance!!! (object creation)
- Boolean
  - careful: fineprint for casts and „BEV“
  - (similar complexity as in XQuery)

# Array

- Syntax as in Java

  var aValues = new Array();

  aValues[3] = „Wednesday“;

- Never out of bounds
  - grows implicitly and dynamically
  - (sets uninitialized values to „undefined“)
- Additional functionality
  - push, pop: use array to implement stack
  - slice: select sub-sequences
  - (some of the XQuery functionality on sequences)

# Operators, Statements, Syntax

- Whereever possible, borrow from Java
- Operators (all as in Java)
  - arithmetic, Boolean (and, or, not), Bit operators
- Statements
  - if-then-else, do-while, while, for -> as in Java
  - for-in statement to iterator through enumerations
  - labels, break, continue, switch, return -> Java
  - function definition + call -> ~ Java
- eval(program) Function
  - takes a JavaScript program as input (string)
  - executes the program
  - eval(„alert(‚Hello, CS 345b');“); // watch quotes!

# Dialogues

- Alert: `alert(„hello world“);`
  - display box with „hello world“ and ok button
- Confirm: `confirm(„Are you sure?“);`
  - display box with ok and cancel button
  - returns a Boolean value

  `if (confirm(„Are you sure?“)) { ... } else { ... }`
- Prompt: `prompt(„Who are you?“, „Nobody“);`
  - display box with input field and ok button
  - returns a String value

# Functions

- At first glance, straightforward (Java without types)

```
function show(msg) { alert(msg); }
show(„Hello, CS 345b“);  // works as expected
```

- Every function has implicit `arguments` param (~main)
- Fineprint:  Functions are Objects themselves

```
var show = new Function(„msg“, „alert(msg);“);
show(„Hello, CS 345b“);  // does the same as before
```

- Some implications
  - higher-order functions possible
  - no overloading, polymorphism (last def. counts)
  - function definition can be changed at runtime
  - functions can have other properties (e.g., functions)

# Objects and Classes

- Officially „classes“ do not exist, but de-facto they do
  - objects are instances of classes
  - classes define properties of objects
  - properties: values, references, functions
- As in Java, Object is the „root of hierarchy“

```
var o = new Object();
o.hasOwnProperty(„isPrototypeOf“);  // true
o.hasOwnProperty(„name“); // false
o.propertyIsEnumerable(„name“); // false
o.toString();  // serialize the object
```

- instanceOf ~ typeOf
  - detect dynamically the type of an object

# Flexibility

- As in XML, possible to add and delete (user-def) properties to individual instances

```
var o = new Object();
o.name = „Tic“;  // implicit insert of property
alert(o.name);    // Tic
delete(o.name); // destroy property „name“
alert(o.name);    // undefined value (not error!)
delete(o.toString);  // error; class property!
```

- Garbage collection (as in Java)
  - delete destroys property; i.e., reference
  - objects destroyed when no references to them

# What is „this“?

- Functions are bound to objects dynamically
  - need a mechanism to refer to calling object

```
function showColor() { alert(this.color); }
var oCar1 = new Object(); oCar1.color = „red“;
var oCar2 = new Object(); oCar2.color = „blue“;
oCar1.showColor = showColor;
oCar2.showColor = showColor;
oCar1.showColor();   oCar2.showColor();
```

- What does this function do?

```
function showColor() { alert(color); }
```

  - (looks for global variable color. If exists, prints its value (i.e., calls „toString“). If not, displays „null“.)

# Constructors

- Since classes do not exist, need work-arounds
  - factory function (problematic!)
  - constructor function (problematic!)
  - prototype definition (problematic!)
  - hybrid constructor / prototype (recommended!)
  - dynamic prototype (ugly, but okay)
- Hybrid constructor / prototype

```
function Car(color) { this.color = color; }
Car.prototype.showColor = function()
  { alert(this.color); };
var oCar1 = new Car(„red“);  oCar1.showColor();
var oCar2 = new Car(„blue“); oCar2.showColor();
```

# Prototypes

- Predefined property of every Object
  - in example: use prototype of „Car“
- All instances of a class reference the same prototype
  - modifying the prototype of one affects all
- Properties of prototype are inherited by instances
  - in example: all cars inherit the „showColor“ property
- (Can also be used to override properties of built-in classes.)

# Inheritance

- Again, must be simulated. Several options:
  - masquerading
  - prototype chaining
  - hybrid
- Prototype chaining

```
function ClassA() {}
ClassA.prototype.color = „red";
ClassA.prototype.show = function() {alert(this.color);}
function ClassB() {}
ClassB.prototype = new ClassA();
ClassB.prototype.name = „";
ClassB.prototype.show = function() {alert(this.name);}
```

# Built-in Objects

- Carry the system-defined functionality
- Properties of Global object
  - undefined, NaN, infinity, Object, ...
  - isNaN(), alert, isFinite(), parseInt(), eval(), ...
- Properties of Math object
  - E, SQRT1_2, SQRT2, PI, ...
  - min(), max(), abs(), ceil(), ...
- Built-in vs. host objects
  - built-in (Global, Math): defined by system environ.
  - host (DOM, BOM): defined by user, program

# BOM (Browser Object Model)

- In browser, there is a pre-defined *window* obj.
  - frames with their names (a frame is a window)
  - document (including images, links, location, ...)
  - history
  - navigator (type of browser)
  - sceen
  - cookies
- BOM allows
  - opening new windows (e.g., pop-ups), resize, ...
  - manipulation of histroy, status bar, ...
- Warning: Again, highly browser specific!

# DOM (Document Object Model)

- W3C standard API (non-declarative)
  - navigate through documents
  - manipulate documents
  - equivalent to XML: InfoSet - not XDM!!!
  - (resembles CODASYL data model of the 60‘s)
- DOM is not JavaScript specific
  - but integral part of JavaScript
- All browsers use DOM as internal store
  - parse HTML; read and manipulate HTML via DOM
  - non of the browsers implements full DOM standard
  - (outside browser, DOM is losing mindshare - too clumsy and expensive)

# DOM

- Navigation
  - getElementById, getElementByName
  - parentNode, childNodes, ...
- Observers of a node
  - nodeName, nodeType, nodeValue, ...
- Constructors
  - createAttribute, createElement, ...
- Manipulation
  - insertBefore, ...

# DOM Example (JS vs. XQuery)

```
var allDivs, newElement;
allDivs = document.evaluate("//*[contains(., 'Beate')]",
   document, null,
   XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE, null);
if (allDivs.snapshotLength > 0) {
   newElement = document.createElement('img');
   newElement.src = 'http://.../Heart-clr-Web.gif';
   document.body.insertBefore(newElement,
                              document.body.firstChild);
}
```

- Equivalent XQuery

```
if (/body ft:contains(„Beate“))
   insert <img src=„http://...“/> into /body;
```

# XML Support in JavaScript

- XML ~ DOM (equivalent)
  - browsers support XML because they support DOM
  - IE (Active-X), Mozilla have JS library functions to load XML into DOM (browser specific!)
- AJAX is based on this observation!

# XPath Support (Mozilla)

```
var oEval = new XPathEvaluator();
var oRes = oEval.evaluate(„XPath“, context node,
                          namespace resolver, resulttype, null);
if (oRes != null) {
   var oElem = o.Res.iterateNext();
   while (oElem) {
     alert(o.Elem.tagName);
     oElem = oRes.iterateNext();
   }
}
```

- XSLT: XSLTEvaluator + transformNode() funct.
- (Obviously, all this is just one line in XQuery!)

# E4X (ECMA-357)

- Simplify access and manipulation of XML
  - DOM conceived as too clumsy
- XML is a primitive (like String, Boolean, …)

```
var x = new XML();
x = <BankAccount>
        <owner id=„4711“>D. Duck</owner>
        <balance curr=„EUR“>123.54</balance>
    </BankAccount>
```

# E4X

- Access to elements
  - Child access: „.“

```
x.balance
```

  - Attribute axis: „.@“

```
x.balance.@curr
```

  - Iteration

```
var total = 0;
for each (x in allBankAccounts.BankAccount) {
   total += x.balance }
```

- Updates
  - Delete nodes

```
delete x.comment
```

  - Insert nodes

```
x.comment += <comment>blabla</comment>
```

# Events

- GUI reacts on events (from user + server)
- JavaScript is event-based language!
  - basis for everything: AJAX, drag&drop, ...
- JavaScript events standardized in DOM Level 3
  - events are associated to DOM nodes
  - unfortunately, no browser implements standard
- Two sides of the coin
  - specify: which kind of event on which object
  - specify: handler to process the event

# Event Example

- Alert click on specific  element
  - embed into HTML

```
<div onclick= „alert(‚I was clicked‘)“>Hi</div>
```

  - specify in separate JavaScript

```
<script> var oDiv = document.getElemById(„div1“);
            oDiv.onclick = function() { alert(„...“); }
</script>
... <div id=„div1“>Hi</div>
```

- Both versions are equivalent

# Kind of Events

- Mouse Events
  - click, dbclick, mousedown, mouseup, mousemove, mouseout, mouseover, dragstart, drag, dragend, ...
- Keyboard Events
  - keypress, keydown, keyup, ...
  - particular events for „alt“ key (dictates UI features!)
- HTML Events: track changes in BOM
  - load, unload, abort, error, resize, focus, blur, scroll
- DOM Events: track changes in DOM
  - DOMSubTreeModified, DOMNodeInserted, ...

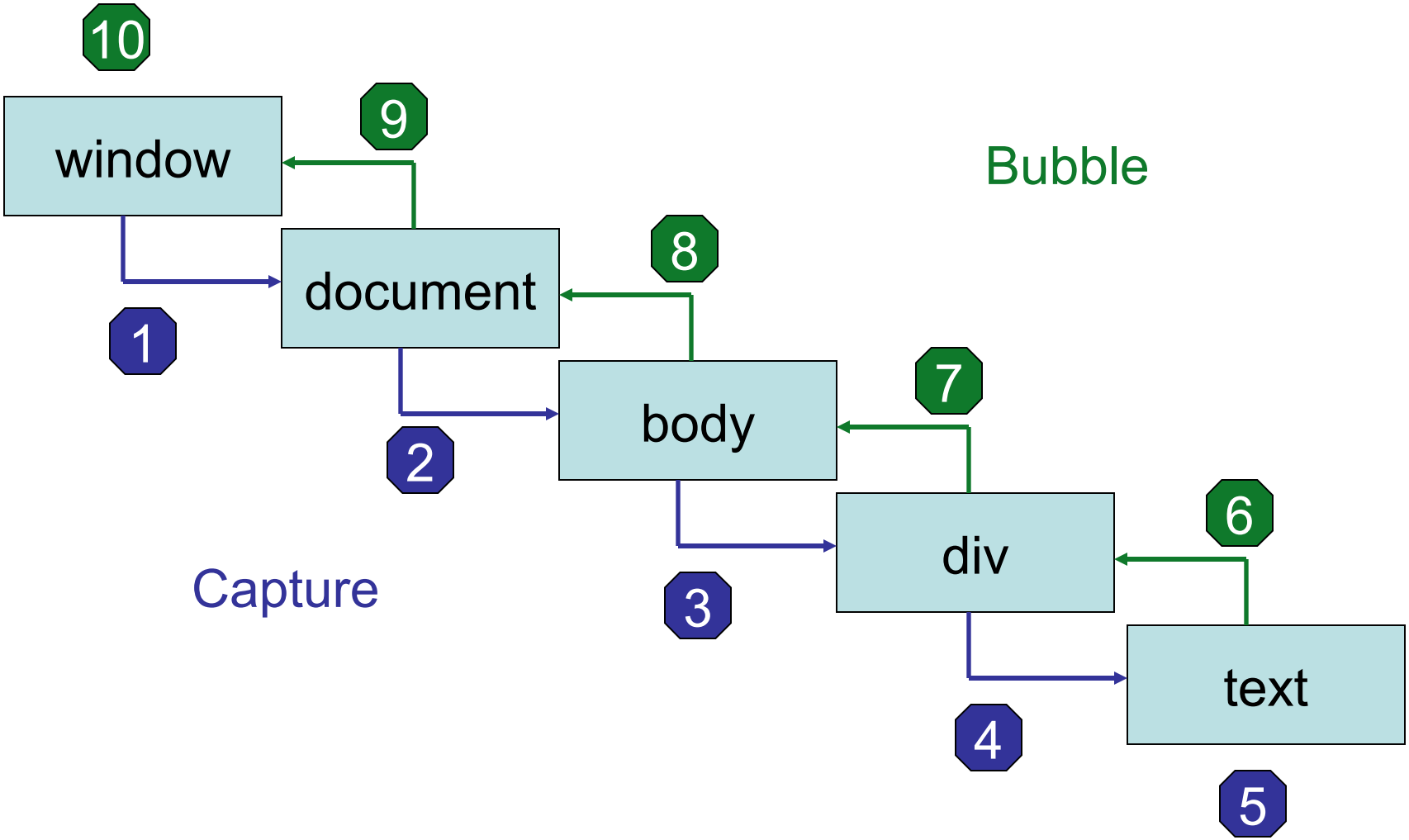
# Event Object

- Created by browser when event occurs
  - implicitly destroyed when event fully processed
- Accessible by event handler (function associated to event)
  - Internet Explorer: „window.event“ object
  - Mozilla: implicit first parameter to event handler
- Contains the following information
  - object (i.e., DOM node) that triggered the event
  - mouse info: x,y coordinates, status of buttons
  - keyboard info: e.g., is alt-key pressed?
  - type of event

# Capturing and Bubbling

- Events occur at nodes in a (DOM) tree
  - does event at child trigger event at parent?
  - in which order are child and parent events proc.?
- Capturing and Bubbling
  - Capture: event walks down (not supported in IE)
  - Bubbling: event walks up
- Controlling event processing
  - specify in which phase event should be processed
  - possibly stop capturing or bubbling („break“ stmt.)

# Capturing and Bubbling



# Multiple Event Handlers

- The same object can have several event handlers for the same event (IE)

```
oDiv.attachEvent(„onclick“, fnclick1);
oDiv.attachEvent(„onclick“, fnclick2);
```

- Event handlers are executed in order
  - fnclick1 is called before fnclick2
- Detach event handlers (IE)

```
oDiv.detachEvent(„onclick“, fnclick1);
```

- Again, all this is browser dependent (DOM L3)

```
oDiv.addEventListener(event, fct, capture?);
oDiv.removeEventListener(event, fct, capture?);
```

# AJAX

Asynchronous JavaScript And XML

# AJAX and Rich Clients

- Remember: JavaScript invented for forms
- AJAX brings it to the next level
  - Web page becomes whole application
- Rich Client:
  - EXCEL:
    - **rich** application with powerful user interface
    - but **not client** because it is closed on the PC
  - www.amazon.com
    - **client** application because data comes from server
    - but **not rich** because the UI is weak (little flexibility)
- AJAX:
  - client executes application; takes over control
  - server delivers „data“ not „content“

# AJAX Goals

- No installation necessary, existing technology
  - AJAX=JavaScript (DOM) + CSS + XMLHttpRequest
- Very good performance / no „glittering“
  - fine-grained update of content (no complete reload)
  - keep state at client (e.g., cache, personalization)
- Asynchronous interaction: do not block user
  - wake up your child, but do not wait until it gets up
- AJAX Examples
  - Google Maps: scroll to the left; reload one column
  - Web-based spreadsheet: reload only affected cells

# XMLHttpRequest

- JavaScripts way to initiate an HTTP Request
  - (yes, you guessed correctly): not standardized
  - browser-specific ways to get XMLHttpRequest obj.
- Sending a request (oReq is XMLHttpReq obj.)

```
oReq.open(„post“, „http://wutz.com“, true);
oReq.setRequestHeader(„...“, „...“);
oReq.send(„name=Donald&password=12345“);
```

- parameters of „open“ function
  - HttpMethod: get, put, post, head, delete
  - URL: URL of service to call
  - Boolean: should the call be asynchronous?
- „send“ function: encoding of parameters of call

# XMLHttpRequest

- Handle Response: Wait for event on oReq

```
oReq.onreadystatechange = function () {
   if (oReq.readyState == 4) {
     alert(oReq.responseText); }
   else { alert(„still loading...“); }
}
```

- Answer from server triggers event in oReq
  – comes in different stages: 1, 2, 3, 4 (4 ~ complete)
- responseText: answer as plain text
- responseXML: answer as DOM (if possible)
  – returned DOM integrated into DOM of page

# JSON (JavaScript Object Notation)

- First uses: Yahoo! in 12/05; Google in 12/06
- Serialization of JavaScript Objects
  - makes it possible to ship data
  - direct competition to XML
- Example (Wikipedia):

```
{    "firstName": "John", "lastName": "Smith",
      "address": { "city": "New York, NY", "zipCode": 10021 },
      "phoneNumbers": ["212 732-1234","646 123-4567"]     }
```

- Deserialilze: „eval“ function (or JSON parser)

```
var john = eval (msg);
```

  - eval function considered not secure!!!

# AJAX at Work

- Replace search box with search result
- Double-combo script
  - two pulldown menus: country and city
  - available cities depend on selected country
  - (34 pages of description in my AJAX book)
- Type-ahead suggestions
  - while typing, do an auto-complete
  - e.g., E-Mail client
- Portal
  - e.g., Google personalization
- All examples really complicated
  - Don‘t try to build *whole* app with AJAX!

# AJAX Libraries

- Hype + complexity has triggered many „tools“
  - gwt: compile Java to JavaScript
  - libraries: e.g, Scriptaculous, Rico, Backbase, Dojo, Echo2, Zimbra, Yui
- Problems with libraries
  - need to load the whole shabang (no „DLL“)
  - cannot mix libraries because no „namespaces“
- Open AJAX Alliance
  - conventions for AJAX libraries
  - players: BEA, Google, IBM, Microsoft
  - Is that still cool?  AJAX is victim of its own success.

# Other Cool Stuff

# Greasemonkey

- The Old Days
  - Web page comes with JavaScript (GUI)
  - every application defines its own GUI
- Greasemonkey
  - Users define their own JavaScript (GUI)
  - Users fix / customize the Web to their needs
  - Decouple GUI from app; third party vendors for GUI
- This is going to change everything!?
  - end of the battle to „own“ the GUI???

# Greasemonkey Examples

- Remove ads from all Web pages
- Add a link to Wikipedia for every word on a Web page
- Display the bn price in addition to the amazon price on amazon.com
- Draw a heart page that contains „Beate“
- Mark Pilgrim: Greasemonkey Hacks
- http://userscripts.org

```
// ==UserScript==
// @name          WIKI X
// @namespace    http://dbis.ethz.ch
// @description Put „X“ on Database entry
// @include  http://en.wikipedia.org/wiki/Database
// ==/UserScript==
var main, newElement;
navbar = document.getElementById('contentSub');
if (navbar) {
   newElement = document.createElement('p');
   newElement.innerHTML = '<p>See also X </p>';
   navbar.parentNode.insertBefore(newElement,
                                  navbar.nextSibling);
}
```

# Greasemonkey Demo

- Search for „Beate“
- Go to Wikipedia History
- Platypus

# Summary

- Why is all this cool?
  - you can do things you could not do before
  - programming the Web!
- Why is it complicated?
  - no standard -> factor of (2+x) in complexity
  - no standardization -> lack of tools (e.g., debuggers)
  - its success -> added features that do not compose
  - designed by hackers for their own purpose (that is a strenght, but also a weakness)
  - limited to browser, client-side computing („C/S Jojo“)
  - impedance mismatch: JavaScript Objects and DOM
  - many ways to do the same thing (e.g., OO)
  - re-invents the XML wheel - there is no JS magic!
- Will XML and „XQueryP“ win the battle in the browser?