timo.kokko@tut.fi
henri.heiskanen@tut.fi

AJAX

Asyncronous Javascript and XML

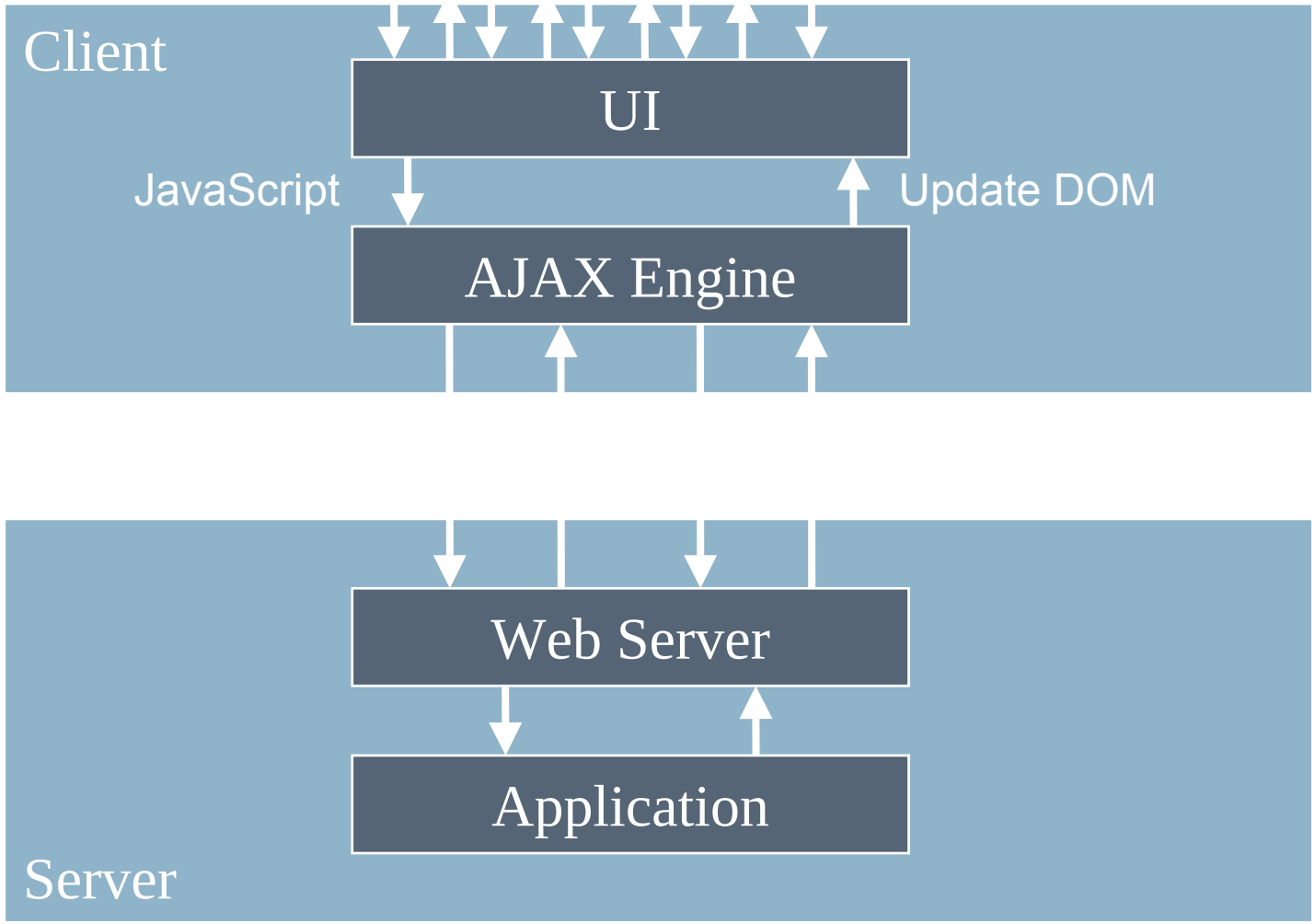
Introduction

- Open standard
- Platform and device independent
- Seamless integration with HTML
- Technologies involved in AJAX applications
  - HTML is used to build Web forms
  - JavaScript code is the core code running AJAX applications
  - DHTML is used to update forms dynamically (div,span)
  - DOM (Document Object Model) used to work with the structure of HTML and sometimes XML returned from server
- XMLHttpRequest Object
  - Handles server communication
  - Bypasses normal application flow
  - JavaScript sends the request behind the scenes
  - Request is asynchronous – Users can continue working

Making a request

- Get data from the Web form
- Build the URL to connect to
- Open a connection to the server
- Set up a callback function
- Send the request

Communication



How does it work in practise?

Creating the XMLHttpRequest object

- Different browsers use different methods to create the XMLHttpRequest object.
- Internet Explorer uses an ActiveXObject, while other browsers use the built-in JavaScript object called XMLHttpRequest.

Example

```
function GetXmlHttpObject() {
        var xmlHttp=null;
         try {
         // Firefox, Opera 8.0+, Safari
         xmlHttp=new XMLHttpRequest();
         }
         catch (e) {
         // Internet Explorer
                  try {
                 xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
                 }
                 catch (e) {
      xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
      }
        }
         return xmlHttp;
}
```

XMLHttpRequest: important properties

- onreadystatechange
  - defines the function that will process the server response
- readyState
  - holds the status of the server's response. Each time the readyState changes, the onreadystatechange function will be executed
  - possible values range from 0 to 4, where 4 denotes a complete request
- responseText

Sending a request to the server

- To send off a request to the server, we use the open() method and the send() method of the XMLHttpRequest
  - The open() method takes three arguments. The first argument defines which method to use when sending the request (GET or POST). The second argument specifies the URL of the server-side script. The third argument specifies whether the request should be handled asynchronously.
  - The send() method sends the request off to the server.

Example

```
var fieldValue = document.getElementById(“field”).value;
var url = “/cgi-local/script?field=” + escape(fieldValue);
request.onreadystatechange = updatePage();
xmlHttp.open("GET", url, true);
xmlHttp.send(null);
```

Hooking in the Web form

```
<form>
 <p> City: <input type=”text” id=”city” size=”25”
    onChange=”callServer();” /></p>
</form>
```

Handling server responces

- HTTP ready states
  - 0: The request is not initialized (Before calling open)
  - 1: The request has been set up (Before calling send)
  - 2: The request has been sent (Content headers are usually available)
  - 3: The request is in process (Partial data available)
  - 4: The request is complete (Server response ready)
- HTTP status codes
  - Server-side code has only traditional Web-specific methods of reporting information
  - Status code 200 → everything ok

Example

```
xmlHttp.onreadystatechange=function() {
      if(xmlHttp.readyState==4)
       {
      if (request.status == 200)
      {
        // do something with xmlHttp.responseText
      }
      else if (request.status == 404)
        // request URL does not exist
      else
        alert(“Error : status code is ” + request.status);
       }
}
```

Is there an easier way?

- Yes. Using an AJAX-enabled JavaScript library facilitates the development process. In addition:
  - JavaScript support is unfortunately browser-dependent, thus there is no standard way to create an XMLHttpRequest object, but a proper library hides browser-specific anomalies (or at least it should...)
  - There are many JS libraries available, one of which is Prototype, maybe the widest-spread and best-known of all AJAX-libraries. It also provides some useful ways to manipulate the DOM tree.

Prototype example

- new Ajax.Request('/foo/bar', {asynchronous:true, evalScripts: true, onSuccess: function(t) { alert(t.responseText); }});
  - Options may include request parameters, method, synchrony/script evaluation information, handler functions for successful or unsuccessful requests etc.
- Prototype also features other interesting objects, like Ajax.Updater and Ajax.PeriodicalUpdater

Other options

- jQuery
- Yahoo! Connection Manager
- Scriptaculous (based on Prototype)
- Rico (based on Prototype)
- DWR
- JSON RPC
- Google Web Toolkit
- OpenLaszlo
- Dojo

Reverse AJAX

- Reverse Ajax is different from Ajax, as Reverse Ajax is a suite of technologies for pushing data from a server to a client. These technologies are:
  - AJAX for handling the data on the client side in a smooth and interactive way, and passing data between server and client.
  - A technology for pushing server data to a browser. There are three options for that purpose: Comet, Piggyback and Polling.

Reverse AJAX – ”server push” techniques

- Comet: a connection between a server and client is kept open, by slowly loading a page in a hidden frame.
- Piggyback: extra data is added (piggybacked) onto a normal client-server interaction.
- Polling: the client repetitively queries (polls) the server.

Reverse AJAX implementations

- For Java there are a few libraries:
  - DWR 2.0
  - JSON RPC
- For PHP, there is the Xaja framework

AJAX advantages

- No more futile page reloading -> improved usability, less client-server traffic
- AJAX + Reverse AJAX enable real-time web applications
- Separation of data, format, style, and function

AJAX issues

- Browser integration
  - ”Back” button doesn't work as possibly presumed
  - Bookmarking a particular state of the application might be difficult
  - Printing problems
- Response-time concerns
- Hard to learn (see newsgroup ajax.web.technology)
- Search engine optimization
- Reliance on JavaScript
  - Too much JavaScript code slows down the browser
  - Has to be enabled
- Cross-browser incompatibility
- Remote-site integration (JavaScript sandbox)
- Server communication
  - XML/HTML/JSON/plain text
- Layout problems + DOM access

Potential places to use AJAX

- Form driven interaction
  - Forms are slow
- Deep hierarchical tree navigation
  - Discussion threads...
- Rapid user-to-user communication
  - Immediate discussions...
- Voting, Rating submissions
  - Users don’t want to wait longer than 1 second
- Filtering and involved data manipulation
  - Applying filter, sorting by date...

Where AJAX should not be used

- Simple forms
  - Simple comment forms, submit order form
- Search
  - LiveSearch on blogs
- Basic navigation
  - Why write code to emulate the browser behavior
- Replacing large amount of text
  - Small pieces of page can be more dynamically updated
- Display manipulation
  - Use AJAX in data synchronization, manipulation and transfer
  - Don’t use for maintainable and clean Web applications
- Useless widgets
  - Sliders, drag and drops, bouncies, mouse gestures
  - For example a slider to select price

Future

- What will become of AJAX?
  - AJAX-enabled web applications will gradually supersede traditional web applications
  - Then again, not all web pages need AJAX- functionality
  - Still, AJAX takeover is non-contingent in areas where usability, interactivity and responsiveness are deemed critical and a smooth user interface is desired

Questions???