Introduction to Ajax

Chris Panayiotou

What is AJAX?

- AJAX = Asynchronous JavaScript and XML
- AJAX is not a new programming language – it is a new way to use existing standards!
- AJAX is the art of exchanging data with a server, and update parts of a web page
  - without reloading the whole page
- AJAX is a catchy name for a programming style made popular in 2005 by Google and other big web developers
- In plain English, Ajax can be thought of JavaScript on steroids

What is AJAX?

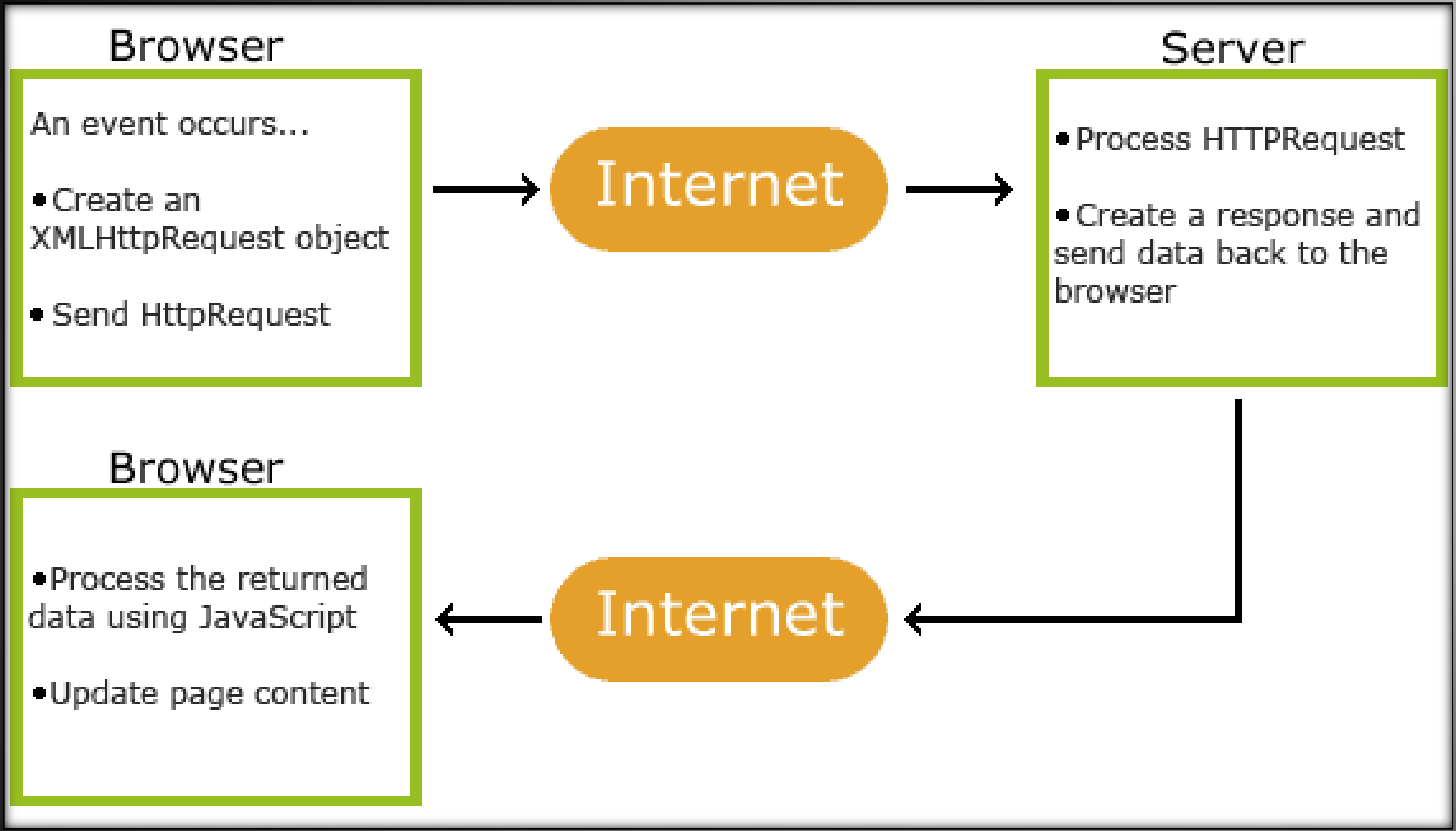
- AJAX is a technique for creating fast and dynamic web pages
- AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes
  - possible to update parts of a web page, without reloading the whole page
  - Classic web pages, (which do not use AJAX) must reload the entire page if the content should change.
- Examples of applications using AJAX: Google Maps, Gmail, Youtube, and Facebook tabs.
- What you should already know to work with AJAX
  - HTML / XHTML
  - CSS
  - JavaScript / DOM

Why AJAX?

- JavaScript enabled cool things with the web browser to make a more user-friendly experience such as
  - Form validation
  - Quirky popup messages
  - Cool web tools and more
- However, JavaScript had no way of sending information between the web browser and the web server
  - If you wanted to get any information from a database on the server, or send user information to a server-side script like PHP, you had to make an HTML form to GET or POST data to the server
  - The user would then have to click "Submit", wait for the server to respond, then a new page would load with the results
    - Problematic when having to wait for especially slow websites!
- AJAX attempts to remedy this problem by letting your JavaScript communicate directly with the server, using a special JavaScript object XMLHttpRequest
  - With this object, your JavaScript can get information from the server without having to load a new page!

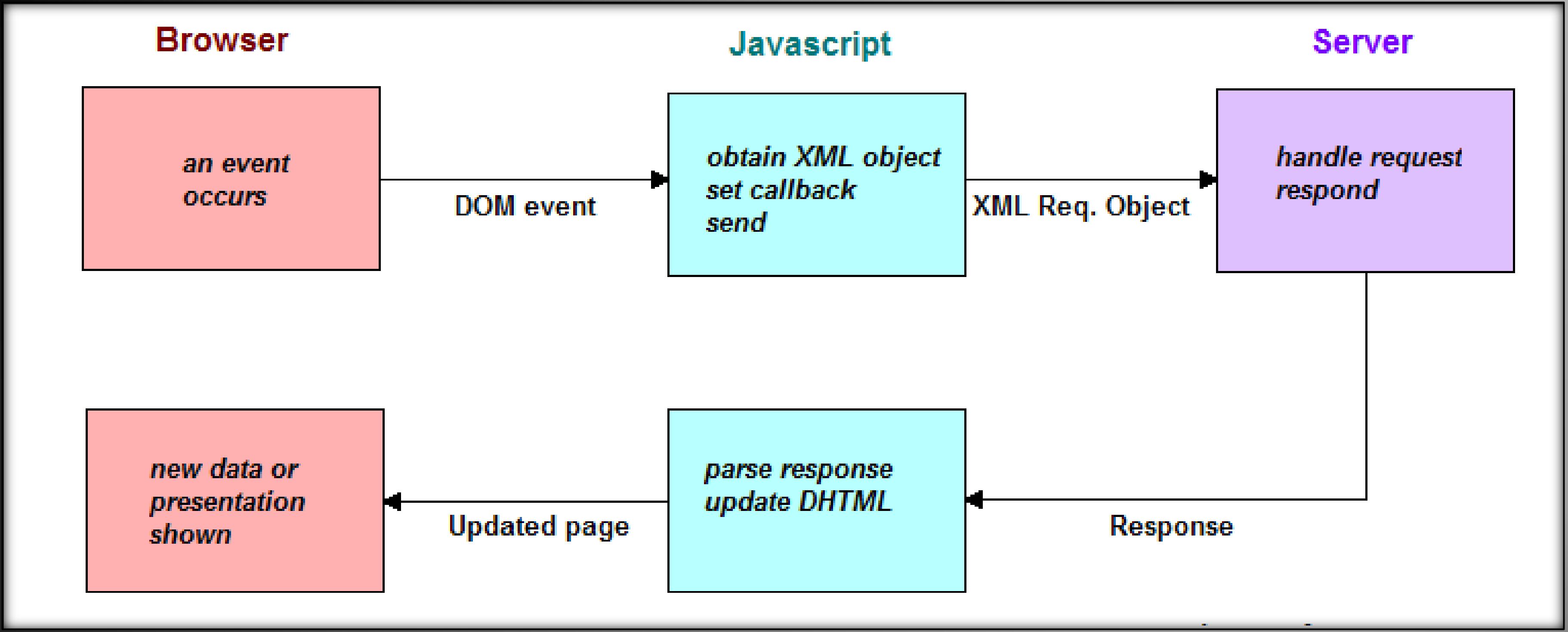
AJAX is Based on Internet Standards

- AJAX uses a combination of:
  - XMLHttpRequest object (to exchange data asynchronously with a server)
  - JavaScript/DOM (to display/interact with the information)
  - CSS (to style the data)
  - XML (often used as the format for transferring data)
- AJAX applications are browser –and platform–independent!


Browser
An event occurs...
• Create an XMLHttpRequest object
• Send HttpRequest
Internet
Server
• Process HTTPRequest
• Create a response and send data back to the browser
Browser
• Process the returned data using JavaScript
• Update page content
Internet

How AJAX Works

AJAX's basic architecture



How AJAX Works

1. Capture an HTML event that you want to respond to by calling a JavaScript
2. In the JavaScript function
    1. Create an XMLHttpRequest object
    2. Set the callback function that will handle the server's response
    3. Send the XML request object to the server
3. Get the server's response and process it through the set callback function
    1. The onreadystatechange event is triggered and the set callback function is called
    2. Get the server's response from the created XMLHttpRequest object
    3. Update the page using the received information

AJAX – Create an XMLHttpRequest Object

- The keystone of AJAX is the XMLHttpRequest object
  - All modern browsers support the XMLHttpRequest object (IE5 and IE6 use an ActiveXObject).
  - It is used to exchange data with a server behind the scenes
    - Thus enabling updating parts of a web page, without reloading the whole page
- Syntax for creating an XMLHttpRequest object:

```
variable=new XMLHttpRequest();
```

- For old versions of Internet Explorer (IE5 and IE6) use an ActiveX Object:

```
variable=new ActiveXObject("Microsoft.XMLHTTP");
```

- To handle all browsers first check if the browser supports the XMLHttpRequest object. If it does, create an XMLHttpRequest object, if not, create an ActiveXObject:

```
var xmlhttp;
if (window.XMLHttpRequest)  {
  // code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
} else  {
  // code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
```

AJAX – Send a Request to a Server

- To send a request to a server, we use the *open()* and *send()* methods of the XMLHttpRequest object:

```
xmlhttp.open("GET","ajax_info.txt",true);
xmlhttp.send();
```

Method	Description
open(method,url,async)	Specifies the type of request, the URL, and if the request should be handled asynchronously or not. **method:** the type of request: GET or POST **url:** the location of the file on the server **async:** true (asynchronous) or false (synchronous)
send(string)	Sends the request off to the server. **string:** Only used for POST requests

AJAX – Send a Request to a Server

- GET or POST?
- GET is simpler and faster than POST, and can be used in most cases
- However, always use POST requests when:
  - A cached file is not an option (update a file or database on the server)
  - Sending a large amount of data to the server (POST has no size limitations)
  - Sending user input (which can contain unknown characters), POST is more robust and secure than GET

AJAX – Send a Request to a Server

- GET Requests – A simple GET request:

```
xmlhttp.open("GET","demo_get.asp",true);
xmlhttp.send();
```

- In the example above, you may get a cached result. To avoid this, add a unique ID to the URL:

```
xmlhttp.open("GET","demo_get.asp?t=" + Math.random(),true);
xmlhttp.send();
```

- If you want to send information with the GET method, add the information to the URL:

```
xmlhttp.open("GET","demo_get2.asp?fn=Henry&ln=Ford",true);
xmlhttp.send();
```

AJAX – Send a Request to a Server

- POST Requests – A simple POST request:

```
xmlhttp.open("POST","demo_post.asp",true);
xmlhttp.send();
```

- To POST data like an HTML form, add an HTTP header with *setRequestHeader()*. Specify the data to send in the *send()* method:

```
xmlhttp.open("POST","ajax_test.asp",true);
xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");
xmlhttp.send("fn=Henry&ln=Ford");
```

Method	Description
setRequestHeader(*header,value*)	Adds HTTP headers to the request. **header:** specifies the header name **value:** specifies the header value

AJAX – Send a Request to a Server

- The url parameter of the *open()* method, is an address to a file on a server:

```
xmlhttp.open("GET","ajax_test.asp",true);
```

- The file can be any kind of file, like .txt and .xml, or server scripting files like .asp and .php (which can perform actions on the server before sending the response back)

AJAX – Send a Request to a Server

- Asynchronous - True or False?
- For the XMLHttpRequest object to behave as AJAX, the async parameter of the *open()* method has to be set to true:
- Sending asynchronous requests is a huge improvement
  - Many of the tasks performed on the server are very time consuming
  - Before AJAX, this operation could cause the application to hang or stop
- With AJAX, the JavaScript does not have to wait for the server response
  - Can execute other scripts while waiting for server response
  - Deals with the response when the response ready
- When using async=true, specify a function to execute when the response is ready in the onreadystatechange event:

```
xmlhttp.onreadystatechange=function()  {
    if (xmlhttp.readyState==4 && xmlhttp.status==200) {
        document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
    }
}
xmlhttp.open("GET","ajax_info.txt",true);
xmlhttp.send();
```

AJAX – Send a Request to a Server

- Using async=false is not recommended, but for a few small requests this can be ok
  - Remember that the JavaScript will **NOT continue to execute**, until the server response is ready
  - If the server is busy or slow, the application will hang or stop
- When you use async=false, **do NOT write an onreadystatechange function**
  - Just put the code after the *send()* statement:

```
xmlhttp.open("GET","ajax_info.txt",false);
xmlhttp.send();
document.getElementById("myDiv").innerHTML=
                                          xmlhttp.responseText;
```

AJAX – Server Response

- To get the response from a server, use the *responseText* or *responseXML* property of the XMLHttpRequest object.

Property	Description
responseText	get the response data as a string
responseXML	get the response data as XML data

- If the response from the server is not XML, use the *responseText* property
- The *responseText* property returns the response as a string, and you can use it accordingly:

```
document.getElementById("myDiv").innerHTML=
                                        xmlhttp.responseText;
```

AJAX – Server Response

- If the response from the server is XML, and you want to parse it as an XML object, use the *responseXML* property
- Request the file cd_catalog.xml and parse the response:

```
xmlDoc=xmlhttp.responseXML;
txt="";
x=xmlDoc.getElementsByTagName("ARTIST");
for (i=0;i<x.length;i++)  {
  txt=txt + x[i].childNodes[0].nodeValue + "<br />";
}
document.getElementById("myDiv").innerHTML=txt;
```

AJAX – The onreadystatechange Event

- When a request to a server is sent, we want to perform some actions based on the response
- The onreadystatechange event is triggered every time the *readyState* changes
- The *readyState* property holds the status of the XMLHttpRequest

Property	Description
onreadystatechange	Stores a function (or the name of a function) to be called each time the **readyState** property changes
readyState	Holds the status of the **XMLHttpRequest** (0 to 4): 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 404: Page not found

AJAX – The onreadystatechange Event

- In the onreadystatechange event, we specify what will happen when the server response is ready to be processed
- When *readyState* is 4 and status is 200, the response is ready:

```
xmlhttp.onreadystatechange=function()  {
    if (xmlhttp.readyState==4 && xmlhttp.status==200)  {
        document.getElementById("myDiv").innerHTML=
                                                        xmlhttp.responseText;
    }
}
```

- The onreadystatechange event is triggered four times, one time for each change in *readyState*

Example
The *showHint()* Function

- When a user types a character in the input field the function *"showHint()"* is executed. The function is triggered by the "onkeyup" event. JavaScript code:

```
function showHint(str) {
    var xmlhttp;
    if (str.length==0)  {
        document.getElementById("txtHint").innerHTML="";
        return;
    }
    if (window.XMLHttpRequest)  {          // code for IE7+, Firefox, Chrome, Opera, Safari
        xmlhttp=new XMLHttpRequest();
    } else  {                              // code for IE6, IE5
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.onreadystatechange=function()  {
        if (xmlhttp.readyState==4 && xmlhttp.status==200) {
            document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
        }
    }
    xmlhttp.open("GET","gethint.asp?q="+str,true);
    xmlhttp.send();
}
```

- HTML code

```
<form action="">
Name: <input type="text" id="txt1" onkeyup="showHint(this.value)“ />
</form>
<p>Suggestions: <span id="txtHint"></span></p>
```

A bit About AJAX Security

- Ajax has a sandbox security model
  - As a result, your AJAX code (and specifically, the XMLHttpRequest object) can only make requests to the same domain on which it's running
  - That is, if you have AJAX code running on www.foo.com, it must make requests to scripts and pages that run on www.foo.com