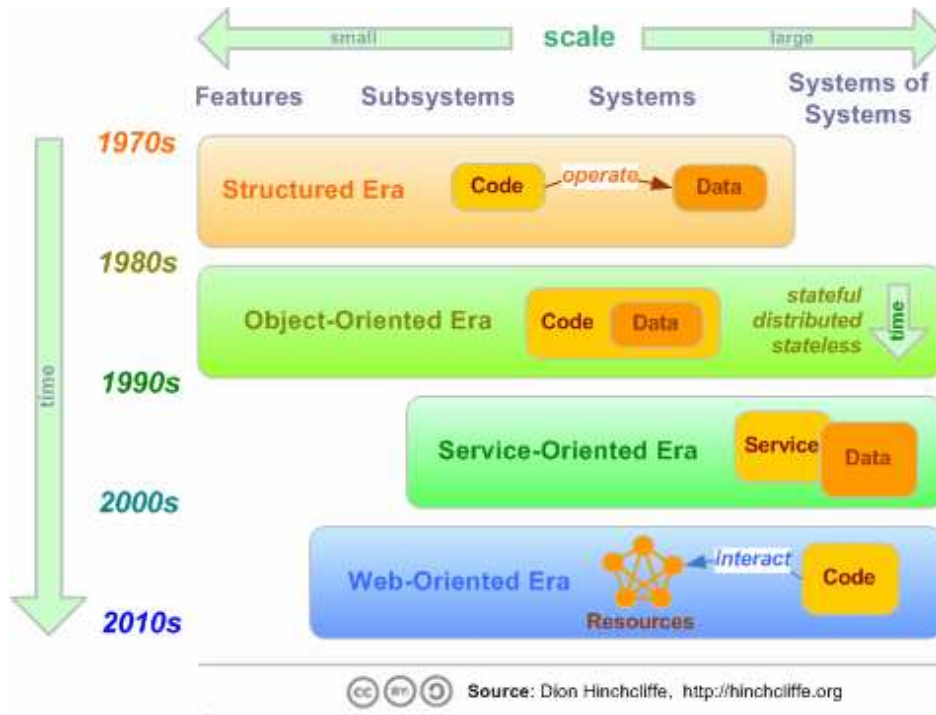


Event-Based Programming Using Top-Down Design and Stepwise Refinement

AJAX Programming for the World Wide Web



By
Dave McGuinness
Urangan State High School

QSITE Conference 2009
IPT Strand

TRADITIONAL WEB APPLICATION PROGRAMMING	3
AJAX – WHAT IS IT?	3
SPA	3
AJAX.....	3
AJAX / SPA – WHY USE IT?	4
AJAX/SPA – WHAT DO YOU NEED?	4
AJAX = DOM + XHTML + CSS + JAVASCRIPT + PHP + MYSQL	5
EXAMPLES	6
MOWING: AN EVENT-BASED WEB PROGRAMMING EXAMPLE.....	6
<i>Top-Down Design</i>	6
<i>Pseudocode</i>	7
<i>HTML</i>	8
<i>Javascript</i>	10
USEFUL TEXTS	11

“Your goal when designing software should be to think about as little as possible at one time. An implementation derived from such a design will tend to be composed of small, manageable "chunks" that

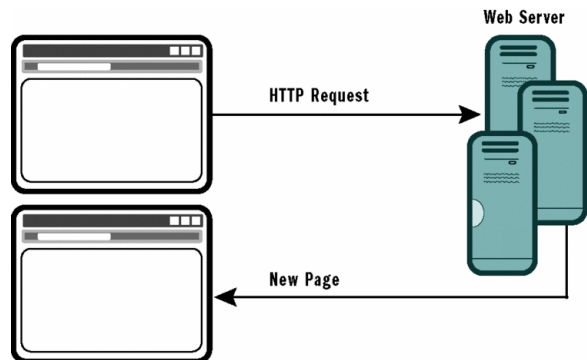
- *relate to problem concepts or tasks*
- *are easier to write*
- *easier to understand*
- *have fewer undesirable dependencies or side-effects on other areas of code*

Terence Parr -Associate professor, Computer Science

University of San Francisco

Traditional Web Application Programming

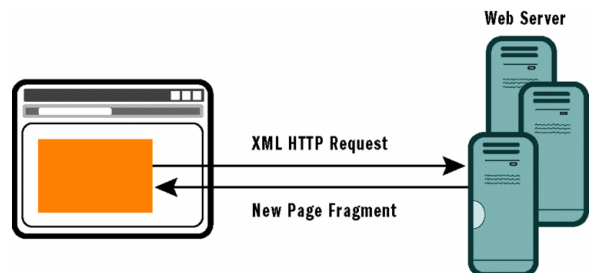
Traditionally, web programming involves presenting something like an input form to the user, which is submitted to a separate page which loads and acts on the data in the input page. This causes the user sit through annoying page refreshes as new pages are loaded. While the page is reloading, no useful work can be done in the browser window. Such applications tend to be a little static and lack useful response to user input. They can be said to be lacking in interactivity.



AJAX – What is it?

SPA

Single Page Application is a term used to describe browser-based programming that contains the complete user interface within one page, without the need for page refreshes. This approach provides the user with a more *desktop-centric* experience which is more immediate. While Firefox and IE7/8 refresh much faster than IE6, there is still an annoying delay at virtually every step of the traditional form-based approach to web-programming. The SPA application should allow the user to forget that they are remote from the experience and just get on with their business. This is particular true of game applications.



AJAX

Asynchronous Javascript And XML most commonly uses a range of reasonably mature technologies to provide a useful client-side programmable interface within an internet browser. Careful selection of the relevant technologies can provide a rich multimedia experience on a 2-D level. The term, **AJAX**, has generally been broadened to include any type of asynchronous programming technique.

AJAX is best used with the **XHTML** 1.0 standard which extends **HTML** 4.01 to standardise on an **XML**(e**X**tensible **M**arkup **L**anguage) compliant document model.

- **XHTML** is used to outline the basic structure of the document
- **CSS** (**C**ascading **S**tyle **S**heets) standard is used to outline the appearance of the page
- **Javascript** programming language (standardised by W3C as **ECMAScript**) is used to manipulate the page via the **DOM** (**D**ocument **O**bject **M**odel).

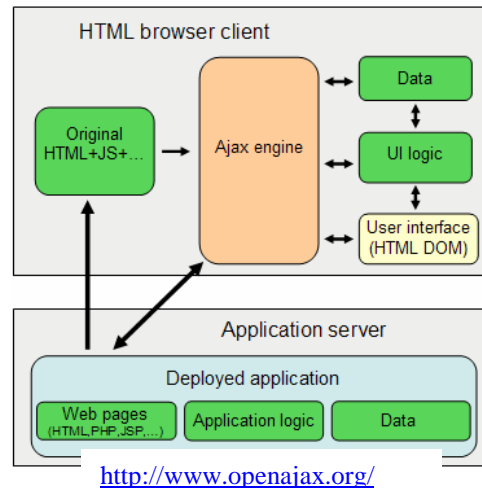
AJAX relies on an un-planned use of the XMLHttpRequest object of the browser available in IE, but part of all browsers. This allows a browser page to request data from the server and be notified when it becomes available. Then the application parses the data to provide output to page via Javascript. This allows the user to go on using the application while the program updates as necessary in the background. While the original intent of the AJAX request was to return XML data, but in fact, the technology is not limited to just this kind of text document.

AJAX/PHP

AJAX can be used to call **PHP** scripts. The HTML output of the PHP scripts, including any data retrieved from a web server-based DBMS (such as MySQL), can be inserted into the page to create new content or replace existing content. If the developer is already familiar with, and indeed has already developed a substantial code-base in, PHP then progress to useful AJAX development is very smooth.

AJAX / SPA – Why use it?

- TCO – Total Cost of Ownership = zero dollars. Mozilla Firefox is free. Many suitable text editors, starting with Notepad.exe are free. No more software is required. MS Visual Web Developer – Express Edition which is roughly equivalent to Dreamweaver CS3 is available for free download from Microsoft.com
- Clear and easily accessible standards; all readily available from websites. The HTML, CSS, XHTML, XML, DOM, and ECMA standards are all maintained by CERN at www.w3c.org. Downloadable references are available. The Mozilla Project also provides very useful references for Javascript and DOM.
- PHP and MySQL are open-source , free products with high levels of on-line support.
- Narrow frame of reference and limited environment make AJAX more accessible for students. It does not overwhelm with gadgets like VB or Delphi. Students switching from the Delphi IDE immediately were more comfortable and "felt" that the work was easier. This is only a perception, but a very important one when considering the benefits of scaffolding student work. Everything has to be done within the bounds of a normal web page. Surprisingly, very useful applications and entertaining games can be created in this way. The environment is definitely more than sufficient for a secondary school IT program.
- Clearer separation of interface design from program code, e.g. concepts such as syntax and layout of code can be learned in the simpler and more forgiving environment of HTML/CSS, then enhanced by upgrading to XHTML, before venturing into the more daunting area of coding where students tend to focus on functionality and the important issues of meaningful identifiers and readable code tends to get lost in the confusion.



AJAX/SPA – What do you need?

1. A text editor, most preferably with syntax highlighting and a multiple document interface. ConTEXT (<http://www.context.cx>) is the best one I have seen so far. It is freeware and readily available. Alternatively, Adobe Dreamweaver or Microsoft Visual Web Developer are useful syntax-highlighted IDEs, especially the MS product which is available free. VWD does not natively support PHP at this stage, although it does a very nice job of HTML/CSS and Javascript (which it still calls Jscript). VWD has very nice code-completion in the script editor, which Dreamweaver does not.
2. A standards-compliant browser, like **Mozilla Firefox**, definitely not IE, not even version 7, which still does not implement the CSS standard correctly and drives students mad with paranoid defence mechanisms. **Firefox 3.5** is very compliant and includes some very useful features, including the **DOM Inspector** and a very helpful **Javascript Error Console**. **Firefox 3.5** also supports some

elements of HTML 5 including the very useful <audio> tag which allows native playback of wav files. This, in particular, is very useful for student projects.

3. An extension of the **Firefox** interface called the **Web Developers Toolbar** adds some brilliant functionality to **Firefox** that makes it the browser of choice for web development.

AJAX = DOM + XHTML + CSS + Javascript + PHP + MySQL

AJAX can be organised logically by separating code into relevant files. Because the interface is built on the Document Object Model within the browser window, the DOM is the foundation code.

The layout of the application in the browser window is built-in **HTML**. Students will need a working knowledge of HTML structures before attempting AJAX. Latest versions of Javascript require dynamically created code to meet the XHTML standard or errors may be thrown by the interpreter. It follows that it is a good idea to insist that students follow the more stringent syntax of XHTML from the beginning to avoid such problems later. Understanding of the TABLE, DIV and SPAN elements is important as these give complete control over location and movement on the page.

Once students have mastered HTML, **CSS** presentation should be added in, in-line, and in a separate .css file. While in-line stylesheets work fine, the clear organisational separation of a css file can be a useful concept the flows on to further work.

Javascript code should also be added mainly in a separate file for clear separation of function from layout. Where necessary it is useful to place some code in-line, but a much more useful construct is to use a top-down design approach to link Javascript functions in the .js file to events triggered by the interface.

Once students have mastered the basis of the three coding elements, you can then move on to developing basic applications to demonstrate the use of sequence, selection, and iteration.

Beyond that, application development should be taught via **top-down design** and **step-wise refinement** approach. I recommend using a design tool like **Inspiration** or **Dia** to develop a block diagram for the top-down design and **Pseudocode** the step-wise refinement. Examples of these are available from the workshop resources at <http://www.mcgoo.com.au>.

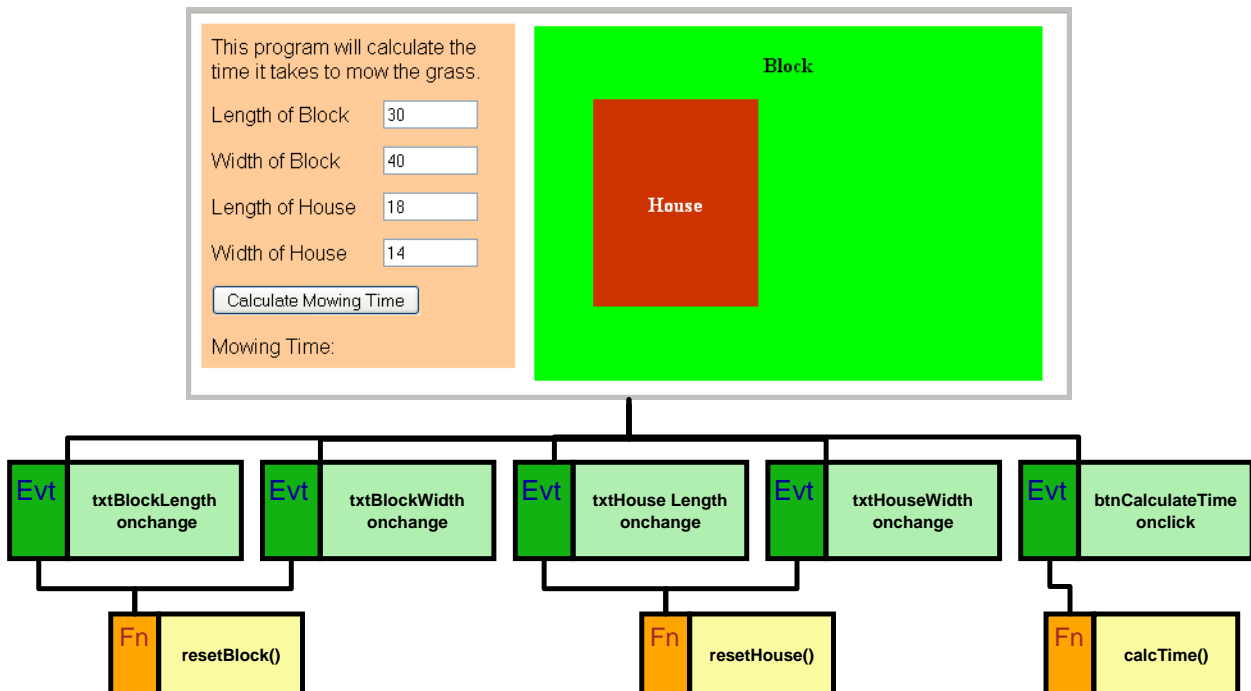
Examples

Mowing: An Event-Based Web Programming Example

This example is based on an exercise in the textbook, *Simple Program Design* (Robertson). This text focuses on algorithm design theory with no specific programming language used. The approach to algorithm design fits very well with the SPA/AJAX web programming environment.

In this program, the user inputs dimensions for a house block and then is able to calculate how long it will take to mow.

Top-Down Design



This top-down structure diagram has been developed in [Dia](#) which is an excellent application for diagramming structural ideas. [Inspiration](#) can also be used, but this is not free.

Students are taught to develop a mock-up of the interface in **HTML** and **CSS** before attempting algorithm design. This requires them to think about **HCI** (Human Computer Interface). What will the game look like to the user? How will the user interact with the game?

Events that represent all possible user interactions are linked to the interface, and then appropriate functions to handle these events are linked to the events.

Pseudocode

The next step is to start stepwise refinement of the algorithm by using pseudocode. Inspiration allows the use to add notes to each node in the design tree. We use this note window to enter the pseudocode steps for the program. Later this pseudocode is copied as comments in to the **Javascript** code editor from which to develop the actual program code. In Outline View, **Inspiration** shows all of the notes as an outline of the design.

Mowing Area Program

txtBlockLength onchange

resetBlock()

Set divBlock width = txtBlockWidth value

Set divBlock length = txtBlockLength value

txtBlockWidth onchange

txtHouseLength onchange

resetHouse()

Set divHouse width = txtHouseWidth value

Set divHouse length = txtHouseLength value

txtHouseWidth onchange

btnCalculateTime onclick

calcTime()

set numBlockWidth = txtBlockWidth value as float

set numBlockLength = txtBlockLength value as float

set numHouseWidth = txtHouseWidth value as float

set numHouseLength = txtHouseLength value as float

set numBlockArea = numBlockWidth * numBlockLength

set numHouseArea = numHouseWidth * numHouseLength

set numMowingArea = numBlockArea - numHouseArea

set numMowingTime = numMowingArea / 2

set strOutput = "The mowing time will be " + numMowingTime + " minutes."

spnOutput innerHTML = strOutput

HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Mowing Area Program</title>
<script type="text/javascript" src="mowingArea.js" ></script>
<style type="text/css">
<!--
.style1 {font-family: Arial, Helvetica, sans-serif}
#divBlock {
    position:absolute;
    width:357px;
    height:237px;
    z-index:1;
    left: 268px;
    top: 10px;
    background-color: #00FF00;
    padding:20px;
}
#divHouse {
    position:absolute;
    width:89px;
    height:122px;
    z-index:2;
    left: 314px;
    top: 67px;
    background-color: #CC3300;
    padding:20px;
}
.style2 {
    color: #FFFFFF;
    font-weight: bold;
}
-->
</style>
</head>

<body>
<table width="245" border="0" cellpadding="6" cellspacing="2" bgcolor="#FFCC99">
<tr>
<td colspan="2"><span class="style1">This program will calculate the time it
takes to mow the grass.</span></td>
</tr>
<tr>
<td width="120"><span class="style1">Length of Block</span></td>
<td width="95"><span class="style1">
<label>
<input name="txtBlockLength" title="Block Length in metres" type="text"
id="txtBlockLength" size="8" onchange="resetBlock()"
"/>
</label>
</span></td>
</tr>
<tr>
<td><span class="style1">Width of Block</span></td>
<td><span class="style1">
<input name="txtBlockWidth" title="Block Width in metres" type="text"
id="txtBlockWidth" size="8" onchange="resetBlock()"/>
</span></td>
</tr>
<tr>
```

```

    <td><span class="style1">Length of House</span></td>
    <td><span class="style1">
      <input name="txtHouseLength" title="House Length in metres" type="text"
id="txtHouseLength" size="8" onchange="resetHouse()"/>
    </span></td>
  </tr>
  <tr>
    <td><span class="style1">Width of House</span></td>
    <td><span class="style1">
      <input name="txtHouseWidth" title="House Width in metres" type="text"
id="txtHouseWidth" size="8" onchange="resetHouse()"/>
    </span></td>
  </tr>
  <tr>
    <td colspan="2">
      <input type="submit" name="btnCalculateTime" id="btnCalculateTime"
value="Calculate Mowing Time" onclick="calTime()" />
    </td>
  </tr>
  <tr>
    <td colspan="2"><span id="spnOutput" class="style1">
      <label></label>
      Mowing Time: </span></td>
  </tr>
</table>
<div id="divBlock">
  <div align="center"><strong>Block</strong></div>
</div>
<div class="style2" id="divHouse">
  <div align="center">
    <p>&nbsp;</p>
    <p>House</p>
  </div>
</div>
</body>
</html>

```

Javascript

```
// JavaScript Document
// Mowing Area Program

function getElement(strId)
{
    var eltToGet = document.getElementById(strId);
    return eltToGet;
}

// function resetBlock()
function resetBlock()
{
    // set divBlock = getElement 'divBlock'
    var divBlock = getElement('divBlock');
    // Set divBlock width = txtBlockWidth value
    divBlock.style.width = document.getElementById("txtBlockWidth").value *10 + 'px';

    // Set divBlock length = txtBlockLength value
    divBlock.style.height = document.getElementById("txtBlockLength").value *10 + 'px';
}

// function resetHouse()
function resetHouse()
{
    // set divHouse = getElement 'divHouse'
    var divHouse = getElement('divHouse');
    // Set divHouse width = txtHouseWidth value
    divHouse.style.width = document.getElementById("txtHouseWidth").value *10 + 'px';

    // Set divHouse length = txtHouseLength value
    divHouse.style.height = document.getElementById("txtHouseLength").value *10 + 'px';
}

// function calcTime()
function calcTime()
{
    // set numBlockWidth = txtBlockWidth value as float
    var numBlockWidth = parseFloat(document.getElementById("txtBlockWidth").value);
    // set numBlockLength = txtBlockLength value as float
    var numBlockLength = parseFloat(document.getElementById("txtBlockLength").value);

    // set numHouseWidth = txtHouseWidth value as float
    var numHouseWidth = parseFloat(document.getElementById("txtHouseWidth").value);
    // set numHouseLength = txtHouseLength value as float
    var numHouseLength = parseFloat(document.getElementById("txtHouseLength").value);

    // set numBlockArea = numBlockWidth * numBlockLength
    var numBlockArea = numBlockWidth * numBlockLength;
    // set numHouseArea = numHouseWidth * numHouseLength
    var numHouseArea = numHouseWidth * numHouseLength;
    // set numMowingArea = numBlockArea - numHouseArea
    var numMowingArea = numBlockArea - numHouseArea;
    // set numMowingTime = numMowingArea / 2
    var numMowingTime = numMowingArea / 2;
    // set strOutput = "The mowing time will be " + numMowingTime + " minutes."
    var strOutput = ' The mowing time will be ';
    var strOutput += numMowingTime.toFixed(2) + ' minutes.';
    // spnOutput innerHTML = strOutput
    var spnOutput = getElement('spnOutput');
    spnOutput.innerHTML = strOutput;
}
}
```

Useful Texts

Simple Program Design, Lesley Anne Robertson , Nelson

Javascript: A Programmer's Companion from the Basics through DHTML, CSS, and DOM, Stefan Koch, Wiley.

Javascript and AJAX, Tom Negrino and Dori Smith, Peachpit Press.

Ajax Bible, Steven Holzner, Wiley.

Spring into HTML and CSS, Molly E. Holzschlag, Addison-Wesley

DHTML Utopia: Modern Web Design Using Javascript and DOM, Stuart Langridge, Sitepoint.

HTML Utopia: Designing Without Tables Using CSS, Dan Shafer, Sitepoint.

Information and Intelligent Systems, Kevin Savage, www.edit.net.au