



Web Remoting (Part 2)

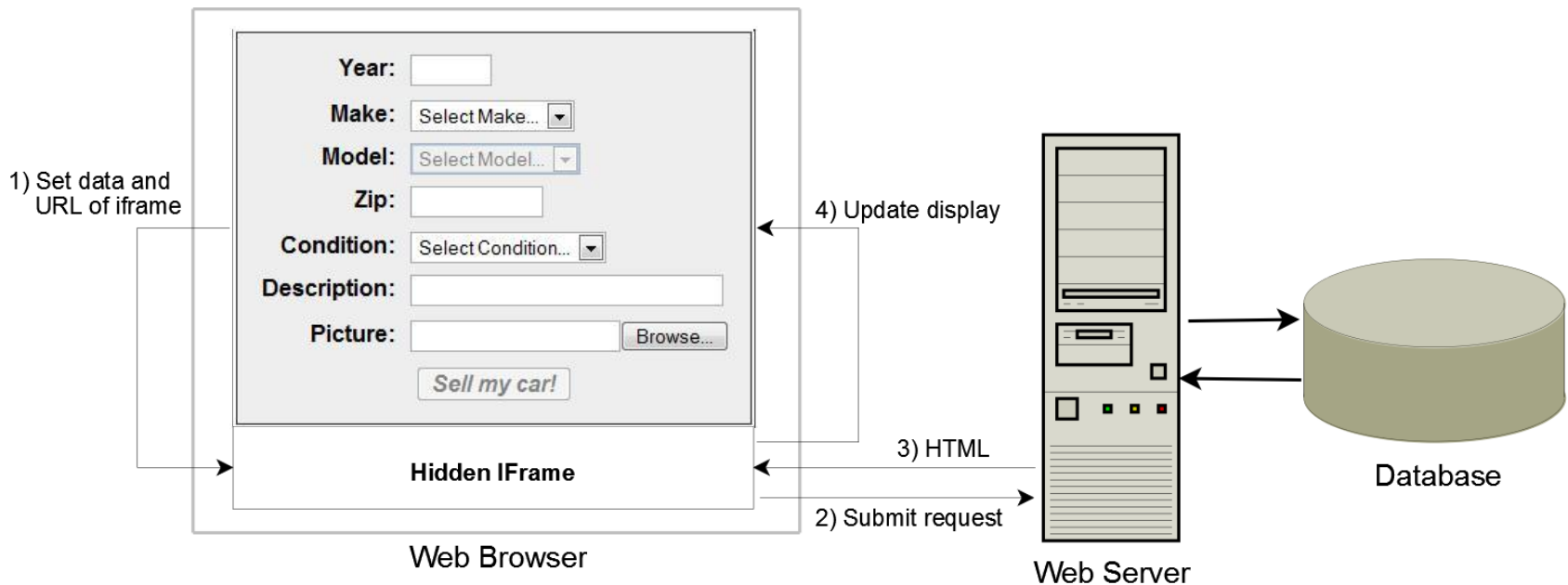
MIS 4530

Dr. Garrett

Hidden IFrame

- Long before there was wide-spread browser support for, or use of, an XMLHttpRequest object, developers were using hidden frames to make requests to the server without reloading the page the user is viewing.
- You can hide a frame using CSS and use JavaScript to dynamically load content into the hidden frame without the user knowing.
- An `<iframe>` can be placed anywhere between the `<body>` tags, and so it becomes embedded in the content of the HTML page.
- A limitation of the XMLHttpRequest object is that you cannot use it to upload a file, so, the only alternative is to use a hidden frame.

Typical Data Flow using a Hidden IFrame



- As an example, suppose we want to create another form for the car sales Web application that allows the user to register a car they want to sell. This registration form will allow the user to enter information about the car and upload a picture of the car, which the application will use to make a listing for the car that other users can browse. The figure in the previous slide shows how the new registration form may look and illustrates the typical data flow when using a hidden `<iframe>`.
- In step 1 the main page uses JavaScript to pass data to the page loaded in the IFrame and sets the source URL of the IFrame. When the IFrame's source URL is changed, it automatically sends a request to the Web server (step 2).
- In step 3 the Web server responds with an HTML page containing any data the client needs.
- In step 4, JavaScript loaded in the IFrame page updates the main page.

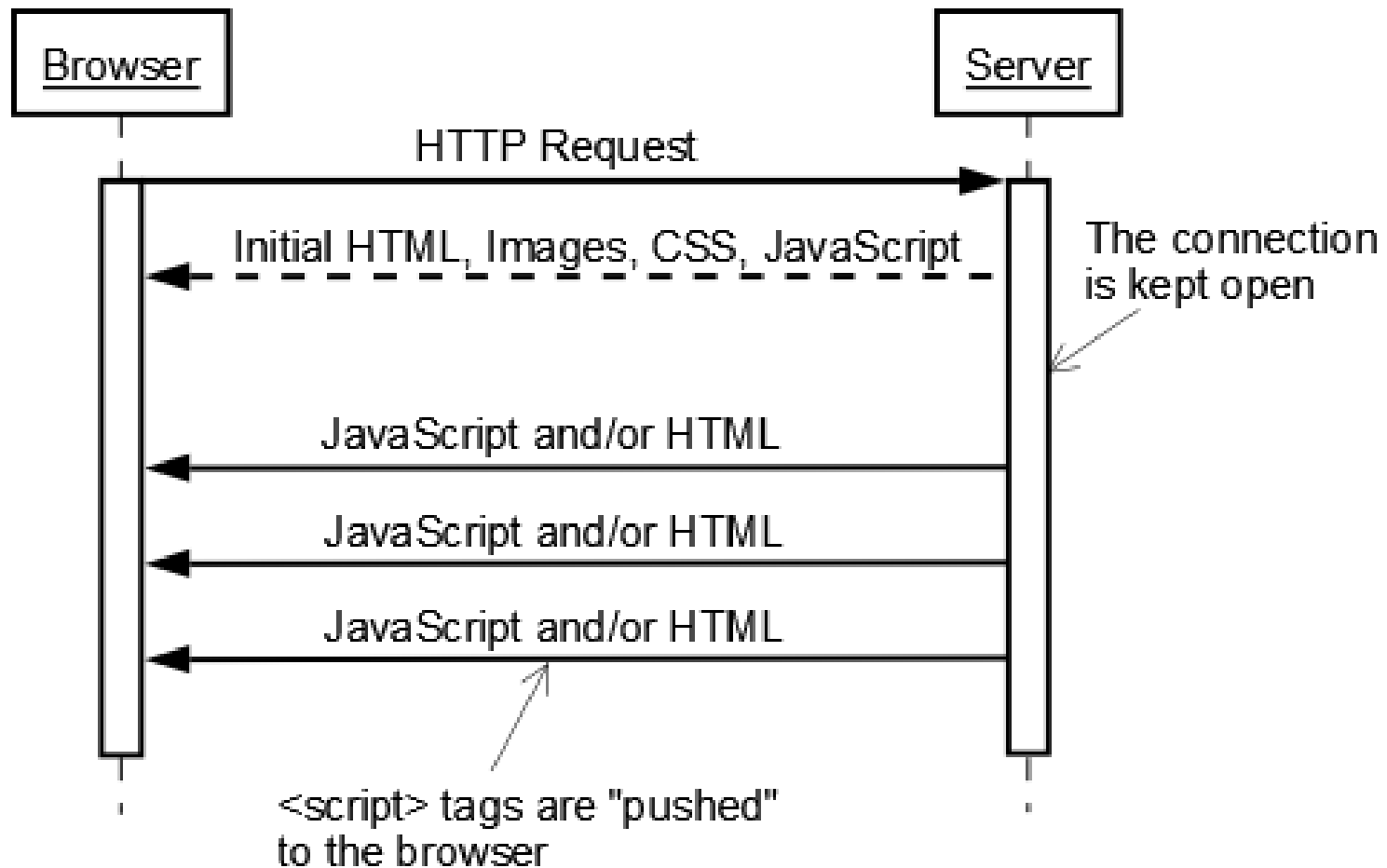
```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    ...
    <script type="text/javascript" src="upload.js"></script>
  </head>
  <body>
    <form id="carEntryForm" name="carEntryForm" method="POST"
      enctype="multipart/form-data" action="registerCar.php"
      target="uploadFrame">
      ...
      <label for="picture">Picture:</label>
      <input id="picture" name="picture" type="file" /><br />
      <div id="buttonRow">
        <button id="submitBtn" type="submit">
          Sell my car!
        </button>
      </div>
    </form>
    <iframe src="about:blank"
      id="uploadFrame" name="uploadFrame"></iframe>
  </body>
</html>
```

```
function init() {
    var submitBtn = document.getElementById("submitBtn");
    var origBtnText = submitBtn.firstChild.nodeValue;
    var form = document.getElementById("carEntryForm");
    form.onsubmit = function() {
        submitBtn.disabled = true;
        submitBtn.firstChild.nodeValue =
            "Uploading Information...";
        return true;
    };
    var iframe = document.getElementById("uploadFrame");
    iframe.onload = function() {
        if (iframe.href != "about:blank") {
            submitBtn.firstChild.nodeValue = origBtnText;
            submitBtn.disabled = false;
        }
        return true;
    };
}
// Execute init() after the page loads.
window.onload = init;
```

HTTP Streaming

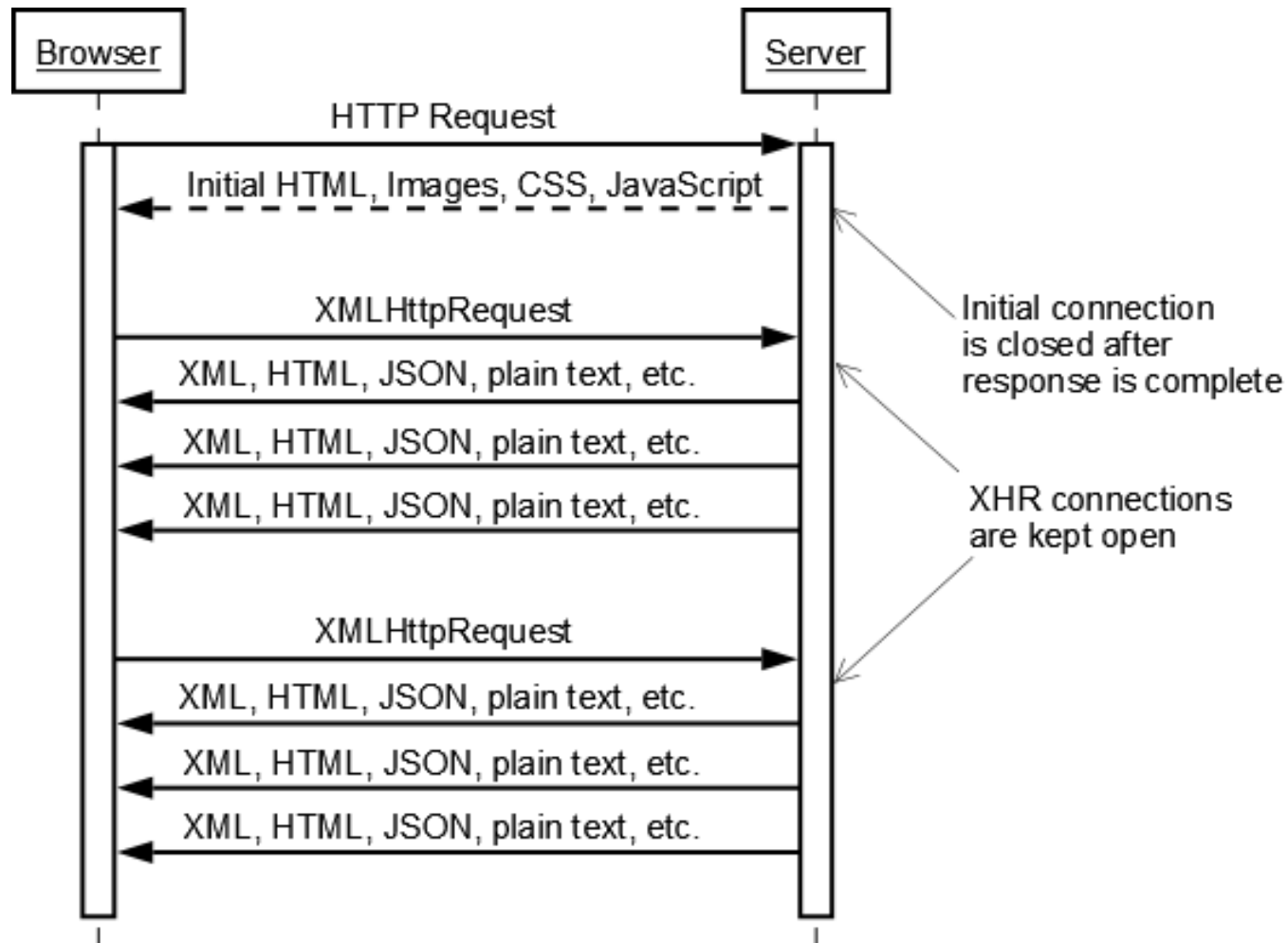
- The standard HTTP model is based on the Web browser *pulling* data from the Web server.
- To achieve a *push* model where the server pushes data to the browser, one of two techniques is typically used, although a combination can also be done. The first technique involves the normal request from the browser, followed by multiple responses from the server using the same long-lived HTTP connection. This technique is sometimes called *page streaming*.

Page Streaming Communication



- Some drawbacks to page streaming are: (1) The browser accumulates objects, which uses up memory and could eventually cause the interface to bog down; (2) HTTP connections will inevitably fail, so you must have some plan to recover; (3) Most servers aren't designed to handle multiple simultaneous long-lived connections.
- The second technique, sometimes called *service streaming*, uses one or more long-lived XMLHttpRequest calls. In this technique, you can make the long-lived HTTP connection(s) anytime after the page is loaded and close and reopen the connection whenever you need, for example, to recover from a failed connection. The HTTP connection is kept open and data is pushed to the client from the server in the same manner as page streaming. An advantage to service streaming, however, is that you can push just about any data you want as long as you can process it in JavaScript (like JSON). With page streaming, you can only push HTML tags.

Service Streaming Communication



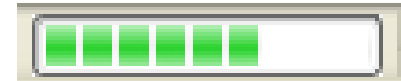
Web Remoting Pitfalls

- The Web has been around long enough that users are accustomed to certain conventions.
 - You expect to get a visual progress indicator from the browser when waiting for a page to load
 - You expect to be able to click the back button and see the last Web page you were viewing
 - You expect to be able to bookmark pages so you can return directly to them at a later time
- Ajax introduces forms of interaction that break these conventions.

Visual Feedback

- One of the easiest problems to fix that Ajax introduces is a lack of visual feedback. With the traditional use of HTTP, when a user clicks on a hyperlink the browser provides some form of visual feedback that work is being done, like a spinning image or a progress bar. When you use the XMLHttpRequest object the browser does not provide any indication to the user that something is happening.
- A simple fix for this is to display a text message or animated image while the XMLHttpRequest is being completed and hide the indicator when the request is complete.
- Progress indicator examples:

Loading...



Browsing History

- As a user surfs the Web, the browser stores the URLs for all the pages the user has visited in a cache, commonly called the history. The browser then enables the Back and Forward buttons to allow the user to navigate back and forward through the browsing history.
- When a request is made using XMLHttpRequest, the URL is *not* stored in the browser's history.
- You can avoid this pitfall by designing your application such that elements that look like hyperlinks *are* traditional hyperlinks that do a page reload and get entered into the browser's history. Elements of your application that use XMLHttpRequest don't have to fool the user into thinking they should behave like a traditional hyperlink.
- JavaScript libraries like YUI and Dojo also have solutions.

Bookmarking

- With static pages, the URL in your browser's navigation bar can be bookmarked and you can return to that page by simply clicking on the bookmark. Therefore, users will expect this of your Ajax Web application as well. However, an XMLHttpRequest could be made and the response used to change a significant section of the page but the URL in the browser's navigation bar will not change.
- You can reduce your user's expectations of bookmarking-ability by only changing content on the page that strictly has an "application" feel to it, but this will only take you so far. It's important that you design for the ability to bookmark the state of your Web application even though you may have to provide a non-traditional way for the user to bookmark it (i.e., Google Maps "Link to this page" hyperlink).
- JavaScript libraries like YUI and Dojo also provide solutions.

Same-Origin Policy

- This policy prevents JavaScript code from reading or setting the properties of windows and documents that have a different *origin* than the document containing the script. The origin of a document includes the protocol, the domain name (host), and port of the URL from which the document was loaded. Documents belong to the same origin if and only if those three values are the same.
- In other words, if the Web page in which your JavaScript code is loaded was pulled from `http://www.ajaxbook.com` and you wanted to use an XMLHttpRequest to directly grab some data from an Amazon Web service you would be out of luck.
- The same-origin policy was implemented to prevent malicious scripts in one frame or window from accessing personal content in another frame or window.

- The only current, cross-browser exception to the same-origin policy applies to windows and frames. The `domain` property of the `Document` object, by default, contains the hostname of the server from which the document was loaded. JavaScript in two different windows or frames can change their `domain` property to the same domain to interact with each other. However, the domain property can only be set to a valid domain suffix of the default value. For example, if the default value is “`www.ajaxbook.com`” then you can only change the value to “`ajaxbook.com`”. The domain set must have at least one dot in it so it cannot be set to a top-level domain, like “`com`”.
- Another exception to the same-origin policy has been drafted by the W3C and implemented by Firefox 3, called W3C’s Access Control for Cross-site Requests.