Core SCORM Enhancements

by Gareth Farrington, Advanced Systems Technology

Abstract

In this paper we propose changes to the core of the SCORM specification to improve on its utility in a commercial environment. This includes customers and students running on many different browsers, many kinds of networks, speaking many languages with different disabilities and levels of computer knowledge. In such a Heterogeneous environment we look to the web as an example of a successful technology that has overcome these issues. We seek to apply the lessons and realities of the web to SCORM to enable training delivery in the world at large.

Table of Contents [Hide/Show]

Asynchronous Functions in SCORM

Network requests are subject to widely varying latency. Requests are made for data and after some elapsed time the server returns that data. The SCORM API portends that communication between the SCO and LMS is both synchronous and instantaneous. This leaves implementers of the specification to bridge a gap between these two incompatible worlds in a language that does not support concurrency. Javascript is single threaded and does not have a wait() function, nor can one be implemented.

This leaves developers with uncomfortable implementation choices. The reference implementation uses Java which has threads. If Java is needed to implement SCORM then why not specify that SCORM is a Java standard? There are many good reasons not to require Java or any other plugin on the web. Some platforms do not support these plugins (e.g. iPhone) but are attractive for training. Getting plugins installed can be cumbersome for users and present a barrier to adoption.

To implement SCORM purely in JavaScript the XmlHttpRequest function must be used to communicate with the server. JavaScript does not have another way of sending and receiving data. This function is the basis for AJAX, Asynchronous Javascript And XML. The Asynchronous part here is important. AJAX functions do not return a value directly. Instead a callback function is called when the request has completed and the results are processed there. AJAX is designed to account for an unreliable network with high latency. Because Javascript does not provide a way to wait for results from an AJAX call you cannot use AJAX to implement SCORM functions.

One alternative is the use of 'SJAX' (Synchronous Javascript And XML) but this has its own set of problems. Primarily the browser completely locks up when an SJAX call is in progress. Use of this technique is strongly discouraged by browser vendors and others.

The final, and we believe only reasonable, implementation choice is to lie to the SCO. For functions like Commit, Initialize & Terminate that imply some communication between the browser and the server yet require a return value we can falsify the return. The communication with the LMS can be completed with AJAX after the call has returned. This gives good performance and usability but it introduces a serious problem; what if the AJAX call fails?

If the call fails because the network is disconnected there is no way to inform the SCO that this is the case. At best all successive calls to any SCORM function could be failed with a general error. If the call fails because the client has been attacked and the server spots the illegal data there is an even more serious problem which cannot be surmounted with general errors.

We propose that SCORM functions that imply communication with the server have their functions signatures altered to not return a value and to take a callback function. e.g.

```
API.Commit(callback);
```

Sample Usage:

```
API.Commit(function (result) {
    if (result)
        alert("data committed successfully");
    else
        alert("error committing data. " + API.GetLastError());
});
```

We accept that changing Get() and Set() in this way would be too onerous because the amount of existing usage is quite large. Initialize, Commit & Terminate should be changed.

We further propose that an error be introduced to tell the SCO that the API has failed to communicate with the server due to a network issue and a data model element ('cmi.isConnected') be added so that the API can tell the SCO when the connection has been restored. The LMS can present UI to the user alerting them that there is a connection problem. While the connection problem persists the SCO should stop taking user input and making further API calls (except API.Get('cmi.isConnected')). If the API is operating in a fully disconnected manner then it can set this variable to 'true' and transmit results back to the server when it reconnects.

SCORM 2.0 Reference Implementation Should be Written in Javascript

All good technical specifications have an accompanying reference implementation. In the case of SCORM the spec is so closely tied to Javascript that it seems obvious that the spec should be implemented, in its entirety, in that language. As discussed earlier in this paper, using other languages and plugins to implement SCORM is not desirable. Implementing the specification in Javascript would inform the choices of the specification committee in a way that the current Java based reference implementation has failed to do. This will highlight problems with implementing the spec in Javascript, efficiencies that can be gained from using Javascript (as opposed to treating Javascript as Java) and inconsistencies between the spec document and the test suite.

Plan for Content to Run in a Subdomain

It would greatly simplify the problem of cross domain content for Content Distribution Networks (CDN) if the course change its behavior when it tries to locate the API. CDN's are often implemented as multiple sub domains of a main site. So content may be served from 'cdn1.company.com' and 'cdn2.company.com' while the main LMS is served from the parent domain ('company.com'). Under the Same Origin Policy content loaded in an frame (or iframe) from a subdomain cannot communicate with an API loaded from the parent domain. To get around this the course can set the frame's document.domain property to the parent domain:

```
var parts = document.domain.split('.');
if (parts.length > 2) {
        //remove subdomain(s)
        document.domain = parts[parts.length - 2] + '.' + parts[parts.length - 1];
}
```

We propose that the course search for the API object as normal. If the API object is not found it should then change its document.domain to the parent domain and repeat the search. This has no negative impact on the LMS, courses or SCORM. this problem can only be solved by the courses themselves and the fix is trivial and safe to apply to existing courseware.

Add Required Plugins To The Manifest

Courses require different plugins and even different versions of those plugins to function correctly. Currently a large problem of us is ensuring that the students browser is equipped to handle the course ware published by several vendors. We would like to see the SCO report, via the manifest, what plugins and what versions of those plugins are required to view the content.

With that information we can verify that the users browser is capable of viewing the content in all SCO's before they launch the course. This is something that has not been a major issue in the DoD world where these requirements are spelled out in a contract. We find that in servicing the US Army Reserves and our commercial customers that their systems vary widely and this is a frequent cause of customer support issues. User experience could be greatly improved if this was incorporated.

Often users must contact IT support at their organization to get permission or access to install plugins. With this we could check entire courses of study against the users browser and get the correct minimal set of plugins installed with a single visit from IT.

```
<plugins>
    <plugin name="flash" version="8.0" />
    <plugin name="QuickTime" version="8.0" />
    <plugin name="flash" version="8.0" />
</plugins>
```

Secure Test Questions

Because the SCO contains the answers to test questions and because the API is embedded in the browser SCORM courses can always be spoofed. The user of the browser is the User's Agent for accessing content. The User Agent acts in the

users interest, not in the interest of SCORM or the SCO. User Scripts (Bookmarklets, Greasemonkey) can be used to modify scripts and data on any web page. A core false assumption in the SCORM specification is that the browser will somehow secure the SCO from attack. In fact the opposite is true, it is the browsers duty to *enable* attacks on the SCO and API.

Two factors have saved SCORM from a serious attack. First it is a very obscure standard used by a tiny number of vendors largely for DoD projects. Second, the value of the targets has been low. There is little incentive to cheat on Alcohol awareness training when it is easy to click through the course.

As SCORM becomes more ubiquitous and the value of targets secured by SCORM courseware increases attacks will proliferate.

A comprehensive solution for secure testing is non trivial. The problem is systemic, the user can fake not only the score but also answers to individual questions, the time taken on the exam etc. The potential exists for capture & replay attacks on the network and on the client itself. Correct responses could be recorded from the SCO and played back by the attacker.

Ultimately, information on the correct answer must not be embedded in the SCO. If the attacker is denied access to the answer key then the best they can do is capture and replay what they hope is a successful session.

Support from the LMS is needed to evaluate submitted answers against the answer key. Also the LMS needs to withhold final scores until all attackers that might collude (e.g. students in the same class or school) have completed the examination. This prevents the attacker from knowing if a captured session is useful for replay.

Successive uses of the same course can be extended by using sequencing to supply a random subset of questions from a large question bank. This would render a replay attack less valuable although any repeated questions could be matched up from a recorded session and spoofed.

Finally the same questions could be re-used by re-generating the answer key and questions with new unique identifiers so that all recorded sessions would become invalid.

We propose that SCO's operating in secure mode have an answer key file, separate from the manifest file. Several current LMS implementations allow the SCO access to the manifest file because it is part of the content package. The answer key shall not be accessible to the SCO. Further the score and timing functions will become read only and the score shall always return 0. The answers are evaluated against the answer key by the server and the score is provided through the LMS to the student.

- "JavaScript Pseudo-URL Exploit in SCORM" - Daniel Pupek.

Web Proxy for Cross Site Requests

We propose adding a function to enable the SCORM content to make cross site requests. This has a number of uses, like gathering real time content and communicating with external systems. The function would conform to the best practices for AJAX, such as providing callback for success and failure. A good place to start would be Prototype.js and it's Ajax.Request functionality which has become a de facto standard.

Additionally as a security measure the Manifest should include the list of domains the SCO will communicate with. The client might fall under a cross site scripting attack and this function would be a particularly useful target. All target URL's would be checked against the domain list in the manifest before they were allowed to proceed. Providing functionality without this critical safeguard would be a mistake.

Further the list can be presented to the LMS Administrator when the course is imported so that the domains are checked and approved.

Wire Protocol

Without providing a specific specification we generally support the creation of a wire protocol equivalent to SCORM. This would allow for the API in the browser or a game or simulation somewhere else on the network to communicate SCORM data back to the LMS. Certainly Web Services based on SOAP are well established at this point. These are also newer technologies that provide improved efficiency (JSON) and ease of use (REST). We look forward to working with others to develop a useful protocol for both the browser (via AJAX) and non browser clients of the LMS.