SCORM 2.0:
Break it up, make it easier, and encourage accessibility.

A white paper by Philip Hutchison

August 15, 2008

http://pipwerks.com
philip@pipwerks.com

Break it up

Face it: SCORM 2.0 will never please *everyone*. As with SCORM 1.x, some people will find SCORM 2.0 lacks important functionality, while a majority will feel it is overly complicated.

I believe the best approach is to break SCORM into clearly defined modules or levels, of which only the first level should be mandatory. LMS vendors should have the option to implement the higher levels based on client demand. SCORM should also include an open-source plugin framework that allows developers to enable custom functionality for handling pedagogical and/or technological approaches not covered by the base SCORM spec.

Proposed levels:

Bear with me, as these are just rough concepts and not fully formed at this point; I would love it if someone could think of catchier names!

SCORM 2.0 Level 1A: Basic ECMAScript API for Course Interoperability
SCORM 2.0 Level 1B: Basic Web Services for Course Interoperability
SCORM 2.0 Level 2: Quizzes and Interactions Module
SCORM 2.0 Level 3: Sequencing and Navigation for Aggregated Content

Level 1A: Basic ECMAScript API for Course Interoperability

The most commonly used feature of SCORM is the API that enables communication between a course and an LMS. The other SCORM features don't even come close in terms of popularity and acceptance.

The majority of e-learning developers simply want to know that their course will work on any LMS without customization; they aren't interested in working with content aggregation or sequencing, and they tend to use commercial software that packages their courses as self-contained standalone SCOs.

In my mind, these developers are the working class of the e-learning industry. Many of them don't have the time, funding, or technical skills to dig into content aggregation theory; all they know is that they bought a 'rapid e-learning' development program for a few hundred bucks so that they can deliver Course X by next Friday. All their boss cares about is that the course is delivered on-time and that it works in their LMS.

Is this a pessimistic view of the industry? Perhaps. However, it's also a realistic view based on the exploding market for affordable e-learning development tools. I find it an exciting example of democratization in progress: people with no formal background in training or education are suddenly empowered enough to build an entire e-learning course in an afternoon. Online education is no longer the sole domain of vendors who charge $10,000 for a recycled cookie-cutter course.

From a SCORM point of view, the needs of this audience are simple: ensure there is a standardized method of communication between courses and LMSs. That's it – no content aggregation or sequencing, just storage and retrieval of course tracking data. Meeting the needs of this largest demographic should be a top priority, well above the implementation of shareable content objects.

As far as which technology should be utilized for this task, ECMAScript/JavaScript makes sense; SCORM 1.x required the LMS to expose the API in ECMAScript, and making implementation pretty straightforward for commercial e-learning development software vendors. Likewise, all modern web browsers support ECMAScript, making this task one of the 'low hanging fruits.'

Having said that, an ECMAScript API is not a perfect solution, and has a few major drawbacks. This brings us to a second option for enabling course interoperability: Web services.

SCORM 2.0 Level 1B: Basic Web Services for Course Interoperability

Less layers and constriction.

One of the louder complaints about SCORM comes from course developers who develop Adobe Flash-based courses or other non-HTML-based courseware; they argue that a ECMAScript-based API is too constricting. They have valid points.

For starters, an ECMAScript-based API requires a web browser (with JavaScript enabled) to act as a liaison between the course and LMS. When developing with Adobe Flash, ExternalInterface is required to convert Flash's ActionScript commands to JavaScript the browser can understand. This gets complicated, requiring more code (with more potential for errors), and could slow down

course response times. It would be easier for these developers if they could communicate with a server directly, eliminating the JavaScript middleman.

Some developers would like to take this a step further, eliminating the need for a browser altogether; they'd like to be able to run courseware directly from a desktop application. SCORM as a web service would allow developers to tap directly into a server via a communication protocol such as HTTP, Adobe's AMF, or even something akin to IRQ and instant messaging. They would be free to use any programming language they like in their courseware, as long as it supports the whatever communication protocol the SCORM web service uses.

Personally, I think these are great ideas, but I have a completely different motive for supporting SCORM as a web service: accessibility.

More accessible.
Assistive browsing technology are devices or software such as screen readers, braille displays, and alternative input devices. These devices enable people with specific physical disabilities (such as blindness, low vision, deafness, and full or partial paralysis) to access web-based content and services.

It's common knowledge that most assistive browsing technology has spotty JavaScript support. Even for assistive technology devices that have good JavaScript support, users encounter problematic uses of JavaScript, such as reliance on mouse or keyboard events (mouseover, mouseout, onclick, onkeypress); users who don't have a standard mouse or keyboard can have great difficulty trying to use these web pages. Thus when courseware relies on JavaScript, it is potentially shutting out a significant number of people who can't use the same devices sighted or fully mobile people use.

Over the last few years, the professional web development community has rallied strongly around the issue, promoting the concept of 'progressive enhancement,' which in its simplest terms means that the content of every web page should be functional and accessible if JavaScript, Cascading Style Sheets or a browser plugin is unavailable. (In the case of multimedia such as a video, the web page should provide quality text-based equivalents.)

This has become an even bigger *cause célèbre* with the advent of Web 2.0 and its reliance on *ajax* (JavaScript, xmlhttprequest, and remote content). The basic premise of ajax is that it prevents the browser from having to perform a postback for every action taken by a user. It's meant to make a website site feel quicker and more responsive. While ajax has often added a welcome slickness to a web page's user interface, it has also had unintended consequences on people who use assistive browsing devices.

Here's a simple example of how ajax affects a blind web surfer:

> **Before ajax**: A blind user with a screen reader loads a web page containing a form. The user submits the form, and is redirected to a confirmation page the screen reader can read. There is no confusion about what has transpired.
>
> **After ajax**: A blind user with a screen reader loads a web page containing a form. The user submits the form, and ajax is used to intercept the form submission. Instead of being redirected to a confirmation page, the current page simply hides the form and displays a

> "success" message in a bright yellow box. Because the screen reader doesn't know the content of the page has been changed by JavaScript, the user is left wondering if the form submission was successful.

As you can see from this example (which is a very common scenario), the use of JavaScript can break the traditional web browsing model, which has trickle-down effects on devices such as screen reader.

For this example, progressive enhancement proponents would suggest designing the web page to work the old-fashioned way (using a redirect on successful form submission), then use JavaScript to enhance the page for sighted users by disabling the page redirect and replacing it with the yellow box technique. This provides the blind user the opportunity to interact with a fully functioning site simply by disabling JavaScript.

Sadly, e-learning developers have been excruciatingly slow to adopt the principles of progressive enhancement. I believe a big reason for this — apart from lack of discussion — is that most e-learning courseware is delivered in Adobe Flash format or HTML files that depend heavily on JavaScript for even the simplest of tasks.

This is where the concept of web services for SCORM comes in: if an e-learning course developer could use a web service that uses a standard HTTP protocol (or similar) for all course-server interactions, course content pages could be served as standard HTML without requiring JavaScript. In theory, SCORM could use a stateful web service and eliminate dependence on framesets and iframes, too. This would be a huge boost to accessibility in e-learning courseware.

SCORM 2.0 Level 2: Quizzes and Interactions Module

One of the other loud complaints about SCORM is its spotty quiz and interaction tracking abilities. I believe SCORM's current quiz and interaction functionality is underused, largely because it can be difficult to implement, and the interaction data is often not reportable in an LMS.

There is a big demand for better quizzing and interaction tracking in courseware. It seems obvious that if SCORM 2.0 offers a well-designed, easy-to-use quiz system, it will be an instant hit.

There is no need to forcibly tie a quiz and interactions module to any content aggregation or sequencing module; it would be best suited as a standalone module that can work on top of the base SCORM framework.

SCORM 2.0 Level 3: Sequencing and Navigation for Aggregated Content

While content aggregation, sequencing, and navigation get most of the buzz in discussions about SCORM 2.0, I believe they will still wind up being the least-used features, and therefore should be an optional module.

Don't get me wrong — I believe in the possibilities of aggregated content as much as anyone — it's just that the topic is incredibly complicated, and it's obvious there's no easy answer.

I do feel strongly that regardless of the model used for aggregation and sequencing, it is of utmost importance that SCORM remains as flexible as possible, including being neutral regarding programming languages and file formats.

I don't profess to have any answers, but here are some scattered thoughts on the topic:

- Content aggregation should be handled by the course developer's code (or a service in an LMS), *not* SCORM.
- Sequencing should be handled by a simple XML-based manifest. SCORM should ensure these manifests are standardized.
- Navigation should be handled by the course developer's code, based on information contained in the manifest. SCORM doesn't need to specify how navigation works or provide code. Keep SCORM simple!
- SCORM's content aggregation standards should be programming language-neutral and not be too concerned with how the content is aggregated.
    - I'm thinking of all the server-side languages that can be used to generate web pages (ASP, PHP, Ruby, ColdFusion, JSP, etc.); regardless of server-side process, all the finished products output standard HTML.
    - SCORM should limit itself to ensuring LMS/CMS systems use standardized protocols for delivering the content on-demand, and should NOT have an active role in the aggregation.
        - HTTP(S) is already a standard communication protocol; do we really need anything else?
- Content aggregated for use in a SCO should be treated as 'dumb' content, with no expectation of scripting that connects with the LMS, unless the content uses the previously mentioned Quizzes and Interactions Module.
    - An exception might be the aggregation of whole SCOs; in these cases, can a content aggregation manifest include a nested content aggregation manifest?
    - The types of content that can be aggregated are expanding every day, and utmost flexibility should be given to the developer. Locking them into a scripting language reduces flexibility.

Make it easier

Standards that are difficult to use don't tend to last very long; people will look for easier alternatives, be they hacks or competing standards.

Standards that are difficult to understand are even worse; people will fail to see the point of the standard and freely ignore its existence.

Some people in the SCORM community make a good point when they say that over-reliance on existing standards might become a tripping point for SCORM 2.0. Heaven knows the standards produced by the IMS Global Consortium are absolutely dreadful to implement; the imsmanifest.xml standard is one of the most dreaded elements of SCORM 1.x.

In some areas, there may not be many standards for the SCORM workgroup to choose from, especially if when looking for royalty-free open standards. Locking the SCORM workgroup into a hard and fast "only use existing standards" rule may end up forcing SCORM to adopt less-than-ideal standards rather than creating their own simpler, targeted, and easier-to-use standards.

Regardless, whichever standards are adopted, they should be simplified as much as possible, made as easy to use as possible, and documented in clear, easy-to-understand language.

The manifest's destiny

The SCORM user community has clearly declared the imsmanifest.xml file a loser; it's confusing, it's hard to create without the aid of software, and the documentation for it is practically nonexistent. Simply put, the manifest as we know it has got to go!

If SCORM 2.0 is modularized, folks who don't use the content aggregation features should be able to upload a single SCO to an LMS without a manifest. The developer would only need to specify a few small bits of data in the LMS, such as the name and location of the launch file, the mastery score percentage (if any), and a few other odds and ends. 90% of the xml in the imsmanifest is useless for non-aggregated SCOs.

If the consensus is that SCORM should continue to use a manifest file for simple SCOs, I suggest creating a new manifest XML schema that simplifies the process for all involved.

Standardized API wrapper

Courses that use the SCORM ECMAScript API normally use a 'wrapper' that abstracts the API functions and adds error-checking and conditional logic.

The ADL never standardized the wrapper file itself. Over the years, untold numbers of custom wrappers have been implemented. This is bad news, because it means each course winds up using non-standardized function names, variable names and conditional logic in its JavaScript. This jeopardizes the interoperability of courses, and makes updating or editing someone else's scripts that much more difficult.

I propose the creation of a standardized wrapper that follows modern best practices for JavaScript, including (pseudo) name-spacing, the avoidance of global variables and functions, consistent error-checking, and legible/sensible naming conventions for variables and functions.

I created my own interpretation of the ADL wrapper using these techniques (available at http://pipwerks.com/lab/scorm/wrapper/), which, while certainly not perfect, serves as an adequate example.

Keep it simple

There is a lot of excitement in the e-learning community about the prospects of integrating Web 2.0 technology and concepts into e-learning. Some even discuss abandoning the traditional 'course' model that SCORM normally supports in favor of blended, socialized learning. These are great discussions, but I strongly warn against letting these discussions overwhelm SCORM 2.0's development; I suggest keeping the SCORM 2.0 spec as simple as possible, and introduce the use of server-side SCORM plug-ins to handle whatever new technology e-learning developers wish to incorporate.

I would hate to see SCORM 2.0 development pushed out by two or three years because of problems implementing a spec for what may wind up being a rarely used SCORM element. The more complex the base SCORM standard gets, the more difficult it will be for end users to implement it. If we ensure the base standard is a simple communication API, the majority of users

should be able to use it with little difficulty, and will be satisfied with the result. *Then* we can discuss adding additional functionality via modules or plugins.

Following this approach ensures SCORM base will remain lightweight and flexible, and will provide LMS vendors the option to add whichever plugins their customer base demands. It also provides developers with an opportunity to design third-party plugins that work with the SCORM standard.

This is the same modular approach used successfully by JavaScript frameworks, object-oriented programming languages, and even computer operating systems.

Encourage accessibility

I won't spend much time on this topic since I've already expressed my opinion, but I would like to take a moment to reiterate that the e-learning development community has generally done a *terrible* job at making online courses accessible. From the LMS all the way down to the courses, accessibility is a big problem.

While SCORM isn't an accessibility standard, I would like to encourage everyone involved with SCORM 2.0 to keep accessibility in mind when making decisions about SCORM 2.0 features. If existing standards are adopted for use in SCORM 2.0, try to select standards that are accessibility-friendly.

Much like 'going green' to help the environment, it's often the little changes you make that have the most lasting impact. It may mean more work, but in the end it's worth it.

Final thoughts

My main goal is to ensure SCORM becomes easier to understand, implement and use. Breaking SCORM into modules is a great way to accomplish this goal; LMS vendors and course developers can choose to only use as much as they need without the overhead and confusion of a bloated spec such as the imsmanifest spec.

Providing a plugin architecture or framework would invite greater participation from the development community, and would prevent the SCORM workgroup from feeling like they need to have all the answers.

Web services would allow more flexibility for developers, and possibly improve accessibility for online courseware.

In parting, I would like to thank everyone at LETSI for taking on this project — you're heroes to many people. It's wonderful to know that the public has input on the future of SCORM.

I'm very excited about SCORM 2.0 and can't wait to see how this movie ends!