Opening Legacy Data Silos: Using Experience Data for Educational Impact

Jonathan Poltrack
ADL Initiative Contractor with Problem Solutions
Alexandria, VA
jonathan.poltrack.ctr@adlnet.gov

Tom Creighton
ADL Initiative Contractor with Aquate Corp
Alexandria, VA
tom.creighton.ctr@adlnet.gov

ABSTRACT

The Sharable Content Object Reference Model (SCORM) afforded major benefits to the learning and training industry by creating an environment of interoperability for e-learning content and systems. However, the data that resulted from a learner experiencing SCORM content was often stored in proprietary data stores. As a result, potentially important data was locked away and unable to be used.

Recently, emerging trends in big data, predictive analytics and data visualization renewed interest in accessing massive amounts of learning experience data. Paradata and correlations can be evaluated to provide learner recommendations for relevant content, to present visualizations to teachers so they can see how their content is being used, and to view meaningful analytics that among other things, can be used to refine and improve learning content. But how can this be accomplished when the requisite data is locked in proprietary learning management systems?

This paper will discuss a novel method of intercepting SCORM communications and translating to standard Experience API (xAPI) 'statements'. The xAPI is an emerging technology that allows tracking of experiential data and provides secure access to data once stored. After applying this solution, SCORM run-time data is stored in a learning record store (LRS) allowing secure access for analysis and visualization. It is possible to apply this solution in two distinct ways: content or server-side updates. Both of these are viable, and in some cases almost automatable solutions to exposing vast amounts of SCORM data.

This paper will explore both methods for removing legacy data silos, will discuss the pros and cons of both content and server side updates, will report on the feasibility of these methods by describing software proofs-of-concept, and will illustrate several use cases and examples of the value of leveraging SCORM e-learning data once it is available en masse.

ABOUT THE AUTHORS

Jonathan Poltrack has been involved in education, online learning and the DoD's Advanced Distributed Learning (ADL) Initiative since 1999. At ADL, Jonathan was an early contributor to the Sharable Content Object Reference Model (SCORM), which became the de facto global e-learning specification. He provided expertise and software development to many SCORM software projects including the SCORM Conformance Test Suite, the Sample Run-Time Environment example Learning Management System (LMS), the ADL SCORM RELOAD Editor course packaging tool and a wide variety of content examples.

In 2009, Jonathan rejoined ADL with the goal of modernizing some of ADL's aging technologies. This included the creation of a new learning platform based on emerging technologies and software architecture. Jonathan's previous experience with learning technologies provided the ideal context for him to contribute to the vision of new specification efforts. The first of these specifications, the Experience API (xAPI), updates the SCORM Run-Time and expands types of learning content to be inclusive of native handheld apps, simulations, virtual worlds, sensors, games and other content modalities.

Jonathan is passionate about learning, training and performance support and their intersection with technology. He strives to continue to improve global education initiatives by envisioning, managing and implementing learning solutions.

Tom Creighton is a software engineer supporting the Advanced Distributed Learning Initiative. He has contributed to specifications such as SCORM and the Experience API, and has helped develop and maintain software related to these specifications, such as the SCORM Test Suites and the ADL Learning Record Store. He has also worked on projects aimed at supporting both specifications like the SCORM to TLA Roadmap and the xAPI SCORM Profile. His experience with these learning specifications enables him to create prototypes to integrate emerging technologies, like sensors and beacons, into a learning environment, as well as combine existing specs to demonstrate new features, like a competencies and xAPI prototype.

Opening Legacy Data Silos: Using Experience Data for Educational Impact

Jonathan Poltrack
ADL Initiative Contractor with Problem Solutions
Alexandria, VA
jonathan.poltrack.ctr@adlnet.gov

Tom Creighton
ADL Initiative Contractor with Aquate Corp
Alexandria, VA
tom.creighton.ctr@adlnet.gov

OVERVIEW

Learning management systems (LMSs) store learner data in databases integrated within their internal architecture. Standards for interoperable e-learning content include requirements that indicate how data flows from learning content to the LMS. These standards, like the Advanced Distributed Learning (ADL) Sharable Content Object Reference Model (SCORM) (ADL, 2009) and Aviation Industry Computer Based Training (CBT) Committee (AICC) CMI Guidelines for Interoperability (AICC, 2004), are well established and deeply integrated into most LMSs. The general model specified in these documents consists of a server-side LMS, complete with all of the functionality required to manage a learning enterprise, full of content and distributed around the world. Content is managed by the LMS and displayed to the user client-side after they select a course to be launched. While the course is active and the learner is experiencing it, data is transmitted from the client content to the server-side LMS. This process is shown in Figure 1.

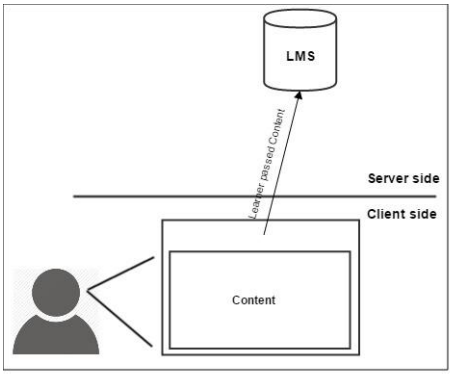

Figure 1, Content tracking to LMS

A new market emerged in the 2000's out of this successful model, so today, many organizations use LMSs as the core of their learning enterprise. This success also resulted in a large number of LMSs distributed around the world on organizational networks. These LMSs contain large amounts of learner data, stored by learning content using the standardized communication protocols. However, this data is often not visible outside of the LMS interfaces and requires expensive customizations to share with other related components like human resource (HR) systems. Although there are some exceptions to this rule, data sharing is an implementation decision, not required by the standards implemented by LMSs.

With the recent popularity of learning analytics and visualizations, the potential importance of this LMS data exploratory investigation increases. How can data be accessed without a common software interface, or worse, manual intervention?

This paper addresses two general solutions to opening legacy LMS data silos. First, Web Service technologies are discussed as the foundation of the solutions. Using this technology, the team developed and tested several prototype implementations and proofs-of-concept. Finally, after evaluating data resulting from test content, several potential case studies are examined that utilize the newly exposed data.

SOLUTION DESCRIPTION

In order to describe the provided solutions, a high level understanding of the Experience API (xAPI) is required. The xAPI, also commonly referred to as "Tin Can API," is a specification from ADL that provides a modern approach to recording learner experiences. It is similar to other learning specification approaches, such as LETSI's Run-time Web Services and IMS' Learning Tools Interoperability and Caliper specifications, in that it defines a Web Service interface and a data format for learning systems.

The xAPI is used for illustration throughout this paper due to our practical experience with the specification. Further, the proofs-of-concept created as part of this project apply xAPI to expose traditional SCORM e-learning data. Any other technology listed above may be a suitable alternative.

Web Services

According to W3C, A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network (W3C, 2004). The use of Web Services allows two separate systems, possibly within different network domains, to communicate and share data.

Digital Government - Building a 21st Century Platform to Better Serve the American People highlights the use of Web Services for public transportation data. These examples serve as an effective analogy for the project defined in this paper. The City of San Francisco funded a project to open their transportation data through a public Web Service interface. Opening the data allowed developers in the public to create various applications with more end-user services than the City of San Francisco could have done by creating an app themselves (Whitehouse, 2011). Figure 2 compares the creation of a single app tied tightly to data and an ecosystem of apps consuming a Web Service.

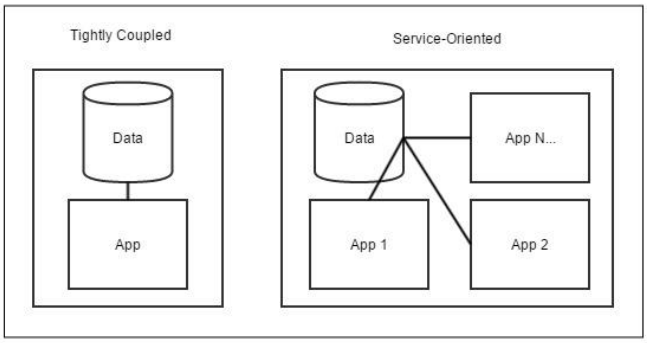

Figure 2, Tightly Coupled vs. Service-Oriented Architecture

In the context of learning systems, Web Services, like xAPI, can have a similar effect. The xAPI defines interfaces to store and retrieve data about learners' experiences with learning content. Such interoperable access to data is an advancement from SCORM and other legacy specifications, which left the retrieval of learner data undefined outside the scope of the content (ADL, 2009). To clarify, data is only accessible via "set" and "get" methods while a learner actively experiences content in SCORM-conformant systems. The interfaces during this active session are well defined and relatively easy to use.

After the learner completes an active session with content, the learning management system (LMS) unloads the content and returns the learner to the LMS internal interface. Here, a subset of data elements may be displayed in a gradebook module, but this is an implementation decision by the LMS vendors and is not required or standardized by the implemented specifications. As a result, large variation exists with regard to data access, visualization and the individual data elements presented to users from system to system.

The xAPI addresses this variation by providing standardized and secure access to data at any time without regard to the state of the learning content. Availability of data allows for third-party systems to consume the data promoting possibilities like:

- A reporting app summarizing a classroom's progress through a course
- An app that determines a learner's knowledge based on his or her past learner experiences
- An app that compares one learner's progress through a subject compared to the progress of an entire class

Flexibility and Interoperability of an Extensible Data Format

The xAPI encodes these learning experience data sets using a format largely influenced by the Activity Streams format (Learning Solutions, 2013). This format represents simple statements about some activity within a certain context (W3C Wiki, 2015). The xAPI defines the properties and values of these statements, such as the actor - generally the learner, the verb - the statement action, and the object - the event, course or activity, along with optional properties for results, learning context, and date of the experience. Examples of some xAPI statements could be:

- Learner A passed Course X
- Learner A answered Question 2 with Choice A
- Learner A joined CS 101 Forum
- Learner A posted "*How to build a Java jar file*" in CS 101 Forum on March 14, 2015
- Team B completed Simulation X on July 2, 2014 16:00:23

This format allows developers to track more granular and diverse information about a learner's interaction with learning content. It is still possible to track SCORM-like events such as starting a lesson or passing a course but it is now also possible to track individual events as well. The activity stream paradigm supports things that were undefined in SCORM, like:

- the learner's answer for question 1 on the first attempt compared to the answer on his or her second attempt,
- the number of times a learner changed their answer before submitting a quiz, or
- the percentage of all learners who answered a specific question correctly.

The xAPI specification, however, focuses on defining the format of the statements, not the data encoded in that format. That flexibility enables the xAPI to be used in various learning environments from traditional SCORM content to virtual world scenarios; however, flexibility reduces the level of interoperability. Although artificial intelligence algorithms and ontologies muddy the waters on a definitive statement, Figure 3 depicts the general effect of data rigidity on interoperability.

Figure 3, Effect of Data Format Rigidity on System Interoperability

For example, imagine that a developer wants to create an app to show students their progress in their courses based on xAPI statements. The format of each statement is consistent - *Actor Verb Object* - but what about the values? Does a statement such as, "*Learner A finished Course X with success*" from one course mean the same as, "*Learner A passed Course Y*" in another course? Or, do the statements - "*Learner A passed Checkpoint 1*" by one vendor and "*Learner A passed Question 1*" from another vendor - mean the same thing?

Even when encoding data from well defined data models, such as SCORM, it is necessary for systems that retrieve that data to "*understand*" how the system that recorded those statements decided to encode the SCORM data. For example, which xAPI statement represents the SCORM success status (pass/fail) element?

- "Learner A passed SCO 1" or
- "Learner A finished SCO 1"

Finding a way to deal with the flexibility of the data format is necessary to support collection and interpretation of data across various organizations or domains.

Solving Data Inconsistencies through Communities of Practice

Communities of Practice (CoPs) formed in an effort to get organizations and developers within a similar domain or topic to work together to define a common set of vocabulary and practices when using the xAPI. For instance, what verbs should be used for content reporting xAPI statements about a learner's experiences in a simulation? Is there a verb to indicate that the simulation has started? Are there any other rules around a statement that represents the start of the simulation?

As these communities grow and develop rules, organizations use artifacts produced by the CoPs to report xAPI statements consistently. For example, reporting systems for a particular domain or topic consume data in a

consistent format and understand its meaning even when the statements were generated by a variety of sources. A "profile" contains the collection of these CoP rules with the intent to support a specific community, domain or use case.

xAPI SCORM Profile

A group of individuals experienced with e-learning and web-based courses came together to create the *xAPI SCORM Profile*. The document provides guidance on how to translate SCORM run-time behaviors and data into xAPI statements (ADL SCORM Profile, 2015). It address issues such as,

- what data needs provided to content at launch,
- how to represent the SCORM temporal model in xAPI statements, and
- how to represent a SCORM data model elements in xAPI JavaScript Object Notation (JSON).

These guidelines resolve the issues regarding xAPI flexibility by constricting the usage to a finite set of representations for SCORM data. All systems that use the xAPI SCORM Profile report a start or end of a learning session the same way. The SCORM data is encoded as xAPI statements consistently and consumers of that data, such as reporting systems or LMSs, can easily interpret the meaning.

This includes the added benefit of allowing non-SCORM content to be recognized by systems that understand this profile. For example, a companion app to some SCORM content can now report learner experiences in the same format as the SCORM content. When an LMS pulls statements from the learning record store (LRS), it also receives the statements about the companion app. These statements can be included as progress toward a course or as additional supplemental information.

The SCORM profile fills a specific gap allowing interoperability of traditional e-learning content in an xAPI enabled environment. We applied this profile using two separate methods to begin to expose data locked in distributed LMSs.

PROOFS-OF-CONCEPT

As discussed in previous sections, LMSs typically store data in private databases and limit access to data after initial storage. As a result, a large set of educational data is inaccessible and unusable. In order to provide interoperable access to learning data, we proposed storing copies of data elements in an LRS. The LRS, by definition, exposes learning data through a standard Web Service interface and thus solves our primary problem. We prototyped several solutions that move data to an LRS. Each solution is valid and results in somewhat similar functionality; however, distinct pros and cons exist for each project.

Proof-of-Concept – Updating Content

Many organizations do not have the capability to update their LMS environment. This can be due to lack of open source code, budget to perform updates, and/or relevant development skills on staff. To account for these common limitations, our first solution addresses data silos by updating content, or specifically in this case, SCORM courses. SCORM courses consist of several web files packaged into a single compressed file. Usually, these files are text-based so they can be updated by a large variety of text editors.

SCORM courses contain sharable content objects (SCOs), which track information about learners. A data model instance is scoped to an attempt on a SCO. For example, if a course includes ten (10) SCOs, then there is a data model associated with each SCO.

SCOs use the SCORM Run-Time Environment to get and set data in an LMS (ADL, 2009). Although the run-time environment uses standard JavaScript technology, it can be tedious to find the LMS interface and make the appropriate method calls. To simplify, in the early 2000's, ADL created and released a wrapper file, APIWrapper.js, to abstract the complexity of the SCORM Run-Time Environment. Many authoring tools, content developers and contractors began using this file for communication. Assuming the existence of the APIWrapper.js file, a simple solution arises to track SCORM data to an xAPI LRS. This provides an opportunity to augment a single area in a course and result in tracking to an LRS.

Dual Track

Our content-focused solution nicknamed "Dual Track", continues to use the SCORM Run-Time, but also tracks to an LRS via the xAPI. The solution must continue to track via SCORM while copying the data to an LRS. This is due to the monolithic nature of SCORM. In SCORM, the LMS manages the runtime, houses content in a repository, manages users, provides search features, sequences content and much more. There are dependencies between some of these features.

Figure 4 illustrates an example content launch sequence. Take note that decisions are made based on a learner's score on a quiz.

The LMS evaluates this rule based on the value of a SCORM data model element. If our silo solution intercepts the run-time communications and redirects to an external LRS, then the value is never set from the LMS point of view and the course sequence breaks. So, our solutions must "*dual track"* data in the LMS **and** LRS to maintain predictable LMS behavior.

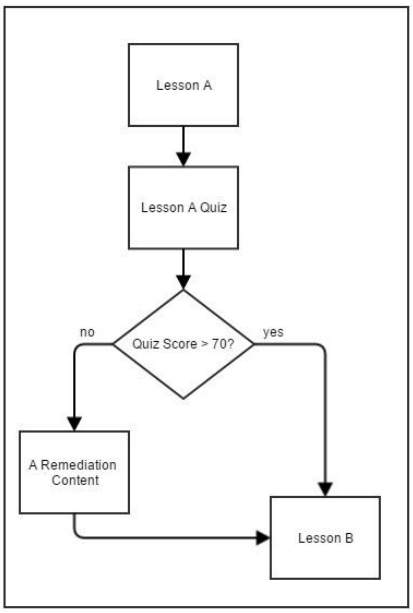

Figure 4, Content Sequence Based on a Score Saved in the LMS

To apply the wrapper solution, a new APIWrapper.js file is swapped into the course, a few additional files are added that include the xAPI-SCORM mapping (xapiwrapper.min.js and SCORMToXAPIFunctions.js) and identifiers are assigned to each SCO (ADL 2014). These identifiers uniquely distinguish SCOs outside of the managed LMS environment.

After the solution is applied, users taking the course get records in their LMS as they did historically, but now also get copies in an LRS.

No Wrapper, No Problem

The solution described above requires the APIWrapper.js file. What if APIWrapper.js was not used in the SCORM course?

The solution is structured in such a way that there are only three integration points for xAPI calls. These integration points correspond to SCORM functions, specifically initialization, termination, and set value.

The proof-of-concept converts both SCORM Version 1.2 and SCORM 2004 (2nd, 3rd, 4th Ed) to xAPI tracking. Specifically, our dual track solution integrates in the standardized SCORM run-time methods detailed in Table 1.

Table 1, Wrapper Integration Points

Integration Point	SCORM Version 1.2	SCORM 2004
Content Initializes	LMSInitialize()	Initialize()
Content Set Value	LMSSetValue()	SetValue()
Content Terminates	LMSFinish()	Terminate()

If the APIWrapper.js file was not used, the following code can be manually entered resulting in a complete *dual track* integration:

Initialize Content

After the SCORM run-time successfully initializes, add the following code:

```
var config = {<Specific LRS Configuration Values>};
xapi.setConfig(config);
xapi.initializeAttempt();
```

Set Values
After the SCORM run-time performs a valid set value:

```
xapi.saveDataValue(name, value);
```

Initialize
During the SCORM run-time terminate, add the following code:

```
xapi.terminateAttempt();
```

Validity
This approach mostly automates conversion to xAPI but also contains several cons. Let's first examine the positives:

- Does not require LMS updates
- Can be relatively simple depending on source course complexity
- Normalizes data across SCORM versions

An LRS must be installed or available, but LMS code requires no updates. This solution is comprised of just a few steps and doesn't necessarily require a software engineer. If there are several courses undergoing conversion and some conform to different versions of SCORM, this solution can normalize the data.

For example, in SCORM version 1.2, the "score" data model element is "cmi.core.score.raw". This represents percentage correct and is bound by an integer between 0 and 100. In SCORM 2004, the "score" data model element is "cmi.score.scaled". This value represents a normalized score and is bound by a decimal number between 0 and 1.

So, if we mix SCORM versions, and we view the raw data, we might see the data indicated in Table 2:

Table 2, Course raw data example

Course 1	Score 90	Pass
Course 2	Score 1	Pass
Course 3	Score 1	Fail

In this example, you might conclude that there is an error in Course 2 or Course 3 since a score of 1 both passes and fails. However, if Course 2 conforms to SCORM 2004 and Course 3 conforms to SCORM Version 1.2, then the values are actually 100% and 1% respectively. In LMS gradebook modules, this conversion is automatically handled, but examining the raw data, it is not.

Our dual track solution normalizes these values to a scaled score between 0 and 1, regardless of SCORM version. This is also applied to several other data model elements that differ in SCORM versions, thus requiring less sense-making when using data in reporting and analytic applications.

Although a good approach for those without access to their LMS code, this solution contains some drawbacks as well.

- Every SCO in every course must be updated
- Data is only converted to xAPI and stored in an LRS when a learner experiences the course

In order to completely update every course, every SCO must be updated with this wrapper solution. Depending on the number of courses, this task may be large. However, content can be phased in over time during regular updates.

More importantly, an LRS receives dual-tracked data only when a user experiences a course. This leaves out previously stored data. Historical data is not available for analytics or reporting unless additional considerations are made. See the *Updating the LMS* section for details on enabling historical data.

Proof-of-Concept – Updating the LMS

An alternative approach to updating every piece of content focuses on the LMS. In SCORM, the LMS launches content and provides the content with an interface to retrieve and store data (ADL, 2009). Because of these responsibilities, the LMS "knows" when content is launched, when the content reports interactions and scores, and when the content is completed. Additionally, the LMS often hosts the interface used to display a learner's progress within courses and the results. With such a central role in the learning architecture, it is possible for an updated LMS to serve as a complete solution for opening up the legacy learning data. The following sections discuss several options and our proofs-of-concept using the ADL Sample Run-Time Environment (RTE) example LMS implementation.

Updating the LMS Content Player

When an LMS sends content to the client, it usually provides a wrapper around the content to contain the SCORM API and navigation controls. This wrapper, or content player, acts as the conduit between the content and the LMS.

For example, when the content is ready to begin communication with the LMS, it issues an Initialize request to the SCORM API. The updated content player proof-of-concept listens for the Initialize call and sends an associated xAPI statement to the LRS. This statement is formatted according to the rules in the xAPI SCORM Profile for a consistent data format. This same pattern is duplicated for the learner's progress and further interactions within the course. Figure 5 illustrates an LMS player while making associated xAPI statements.

We determined early on in our proof-of-concept the need to update the Sample RTE content player in order to evaluate the effectiveness of this approach. The Sample RTE contains Java software no longer valid due to security updates. While replacing this Java code with a new JavaScript approach, the team added xAPI reporting via the new JavaScript module.

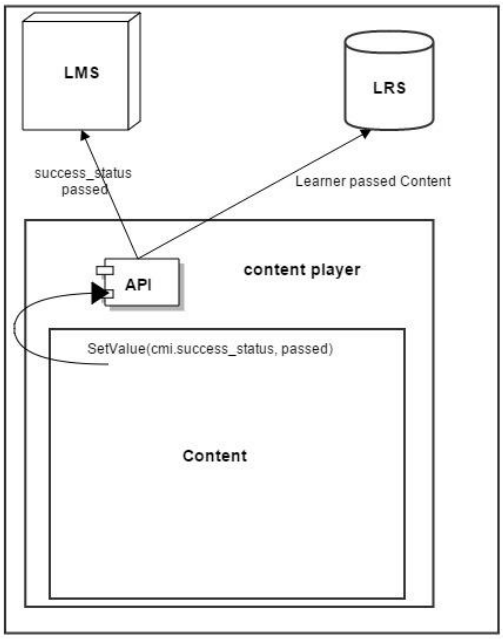

Figure 5, Content Player Sending xAPI Statements

For example, per the SCORM specification, the SCORM API provides a SetValue() function. We updated that function to pass the SetValue() values, the data model element and value, to an xAPI JavaScript library. The xAPI library takes the data model element, like cmi.success_status, and creates the appropriate xAPI statement assigning it the appropriate value sent by the SCO. Essentially this is the same dual-track process discussed in the *Updating Content* section, except that this solution requires only one update on the LMS instead of one for each piece of content.

Updating the LMS to Import and Export Data

One of the drawbacks to updating the LMS content player as described in the previous section is that it does not expose previously stored data. LMSs likely stored data before the player update and contain a history of learner progress and interactions with courses. Historical activity, such as previous year's performance in a course, or previous learners' attempts on content, are still locked within the LMS even after application of the previously mentioned solutions. Additionally, even if accessible, the data might not be in a normalized or in a consistent format.

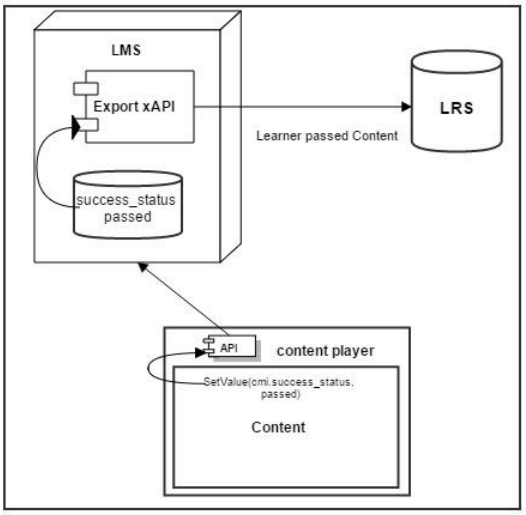

Figure 6, LMS Exporting xAPI Statements to an LRS

Export
To address these issues, an additional proof-of-concept was created with a feature to export existing data. This process, like the update to the content player, normalizes the data in the form of xAPI statements conforming to the xAPI SCORM Profile and sends the data to an LRS. The update initially converted all historical data into statements; however, this same process could be used to update the LRS with recent interactions as well, possibly replacing the need to update the content player. Figure 6 shows an LMS export scenario.

This approach enables the system to export any data within the LMS providing the organization standardized access to historical records and past attempts on content. Also, the translation and submission of data to the LRS doesn't need to happen during the course run-time. This makes it easier to control when the LMS sends the data to the LRS and how it does it. For instance, the LMS could have an hourly process collecting new SCORM data and batch-submitting it to an LRS.

Import
Additionally, we focused on implementation of an import feature. The goal was to pull learner progress and interaction data from an xAPI LRS to satisfy content that was tracked by the LMS. We started by updating the same Sample RTE system so that an administrator creates a course structure without uploading any content. The connection to content is created by associating the external content's xAPI Activity URI (unique identifier) to its representation in the course structure stored in the LMS. At some later point in time, perhaps by some scheduled process, or by manual triggers - as was the case in our example, the LMS queries the LRS for xAPI statements that are associated with the specific activity. Those statements are then used to set the progress of the external course within the LMS as seen in Figure 7.

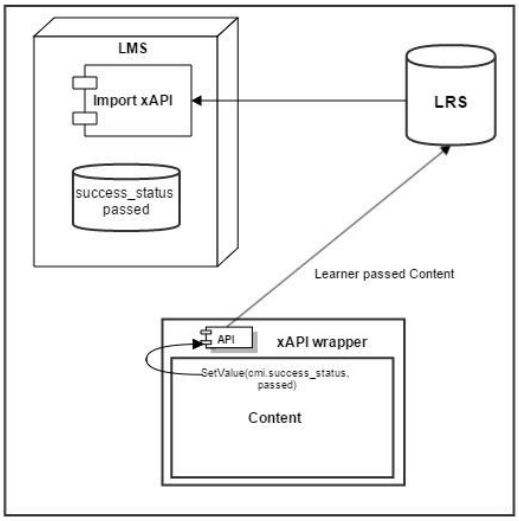

Figure 7, LMS Updating Internal Records from LRS xAPI Data

This process is useful in the case where an instructor wants to include some external content, such as Khan Academy lessons, mobile courses, or simulations, as part of their online lessons. With SCORM there is no way to include external content or courses. However, by connecting identifiers to an LMS course structure, the LMS is able to query the LRS for relevant statements and update its internal status for the external course.

Validity
The updated LMS proofs-of-concept allow the development effort to be focused to one location. Unlike the approach of updating the content, the changes using this approach are all centralized to the LMS. The team determined the following list of pros and cons.

Pros of this approach:

- Updating the content player only requires an update to a single component rather than potentially updating a large number of courses. This significantly reduces the overall time and effort required.
- LRS receives learner experience data from content. Many times various development shops create content. By centralizing the xAPI tracking to the LMS, the organizations building the content are not required to understand the xAPI or make changes to the content.
- The *xAPI SCORM Profile* data mapping normalizes data between SCORM versions.
- LMS updates enable the organization to export historical records and past attempts on content. This is one major differentiator from the previous content-based solutions.
- The translation and submission of data to the LRS does not need to happen during course run-time. This makes it easier to control when the LMS sends the data to the LRS.

- Organizations can use lessons not necessarily managed in or hosted by the LMS. Simulations, mobile activities, and externally hosted content can now be accumulated into a lesson or course and tracked by the LMS the same way as managed SCORM content.

Cons of this approach:

- There are expenses associated with the LMS updates, whether by an LMS vendor or via open source code. Time, resources and significant expertise are required.
- The initial export of historical and archived data may be large. Converting years worth of past learner performance data will likely be resource and network intensive. Some commercial LRSs charge based on network activity or number of statements, which might make an exhaustive export expensive.

USING DATA

Previously, we described how SCORM data can be copied to a specialized database called an LRS, where it is accessible via a standardized and secure interface. Important questions remain about the use of the data. Our proof-of-concept worked with traditional e-learning course to examine any potential differences in LMS data vs. LRS data.

Changes in State

During the first few executions of the e-learning test course by test users, differences in data began to appear. One type of difference tagged "state changes", involves changes in data over time. Learning management systems generally store a single atomic value for any given data model element. For example, when taking a quiz, a learner changes their answer several times before selecting a final value for submission. In the LMS environment, only the last value input is saved and used later for calculations like grading.

After automatically converting SCORM statements to xAPI, we noticed several answers for a quiz. The conversion code was examined for errors to explain the multiple answers for each question. At this point, it was determined that these were all valid values and that the tester had changed their answer several times before moving on to the next module

Data for Personalization

Although student data is stored in LMSs, it is mainly used to provide learners and instructors with simple, rolled-up information like scores, success (pass/fail) and completion (complete/incomplete). Data displayed in gradebook LMS interfaces is infrequently used to provide individualized instruction. Learners enter content with different degrees of engagement, existing knowledge and interest and these conditions are related to their success (Clark, Dede, 2006). Personalized content can be used to keep those already engaged, engaged in the content while providing different information to those that require motivation in the subject.

The learning data created while a user experiences content is more comprehensive than the data available in an LMS. Looking at a simple example, LMSs are responsible for maintaining the last attempt of the learner (ADL, 2009). They may store additional information, but this is often not the case, as the standards supported by LMSs do not require the maintenance of old attempt data. However, historical attempts provide valuable insights into learners' prior knowledge.

The solutions described in the *Proofs-of-Concept* section maintain data associated with an attempt. Each attempt is accessible via a standard API and can be queried by the content, or even a different authorized system, in order to use the data to adapt learners' future learning experience. This is not the case in traditional learning solutions like SCORM LMSs for two main reasons:

1. SCORM requires that only the current or last attempt data is stored
2. SCORM provides the current attempt data only to the SCO that set the data

The proofs-of-concepts maintain all attempts and the granular data associated with them. In addition, any content, in the original course or not, can access and use the data once it is available via an LRS.

This solution saves a potentially large set of actionable data about the learner, the learner's experiences, the content and contextual information.

Tying in External Content

Our *LMS Updates* solution yielded potential data usages not possible with our *Updating Content* solution. For example, LMSs generally support web-based content only, as the standards they implement require content rendered in a web browser. This creates an environment where, without significant updates, it is difficult or even impossible to integrate something like a mobile app data into an LMS data store (Sramek, 2013).

When an LMS pulls data from an LRS, it enables integration of data from any content that communicates with an LRS. This solution permits the data integration from wearables, mobile apps, desktop simulations, virtual worlds, games and more. Further, the content must only track via Web Services to the LRS, not requiring a persistent connection to an LMS.

While testing our solutions, we integrated a native Android application containing performance support information into our Sample RTE example LMS. The performance support application augmented a course on the same subject matter also available in our sample LMS. Before the LMS Updates solution, the LMS gradebook was not able to include data from the Android app. After updates, the LMS pulled the Android app data from the LRS to the LMS internal database and displayed it as it would any SCORM course.

SUMMARY

Traditional e-learning data is stored in LMS databases, distributed globally. These systems often do not expose standardized interfaces to access data once it is stored. However, many LMSs have been in use and collecting data for over a decade. The data may contain valuable insights for learning professionals and learners alike. Data can be used to tailor content to the individual, to provide real-time feedback on an exam or assist curriculum designers in refining content.

Our project applied two proofs-of-concept: content and LMS updates, in order to attempt to unlock data previously in proprietary LMS silos. Our tests indicate that both solutions can be used to move data from a proprietary LMS database to a standardized LRS, where data is accessible via a standard API. Once stored in the LRS, the data can be used for several use cases described in this document and more. Additional testing should be implemented to verify our testing results and to garner additional information on other use cases enabled by access to LMS data.

REFERENCES

Advanced Distributed Learning (ADL) Initiative (2015), *Experience API SCORM Profile*. Retrieved June 17, 2015 from https://github.com/adlnet/xAPI-SCORM-Profile/blob/master/xapi-scorm-profile.md

Advanced Distributed Learning (ADL) Initiative (2014), *SCORM-to-xAPI-Wrapper Readme*. Retrieved June 10 ,2015 from https://github.com/adlnet/SCORM-to-xAPI-Wrapper/blob/master/README.md

Advanced Distributed Learning (ADL) Initiative (2009). *Sharable Content Object Reference Model (SCORM) 2004 4th Edition Version 1.1*.

Aviation Industry Computer Based Training (CBT) Committee (AICC) (2004), *CMI Guidelines for Interoperability Revision 4.0*, Retrieved June 2, 2015 from http://www.immagic.com/eLibrary/ARCHIVES/TECH/AICC_US/A040816G.pdf

Clark, Jody, Dede, Chris (2006). *Robust Designs for Scalability*.

Elias, Tanya (2011). *Learning Analytics: Definitions, Processes and Potential*. Retrieved May 15 from http://learninganalytics.net/LearningAnalyticsDefinitionsProcessesPotential.pdf

IMS Global (2013), *IMS Global Common Cartridge Profile*, Retrieved June 17, 2015 from http://www.imsglobal.org/cc/ccv1p3/imscc_Overview-v1p3.html

Learning Solutions Magazine (2013), *The xAPI and the LMS: What Does the Future Hold?*. Retrieved June 17, 2015 from http://www.learningsolutionsmag.com/articles/1271/the-xapi-and-the-lms-what-does-the-future-hold

Sramek, Jan (2013), *Apps, Courses or Neither?*. Retrieved May 25, 2015 from http://better.io/resources/articles/tj-apps-courses-or-neither.pdf

Tseng, Judy C.R., Chu, Hui-Chun, Hwang, Gwo-Jen, Tsai, Chin-Chung (2008). *Development of an Adaptive Learning System with Two Sources of Personalized Information*.

Whitehouse (2011), *Digital Government: Building a 21st Century Platform to Better Serve the American People*. Retrieved May 20, 2015 from https://www.whitehouse.gov/sites/default/files/omb/egov/digital-government/digital-government.html

W3C (2004), *Web Services Glossary*. Retrieved June 17, 2015 from http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice

W3C Wiki (2015), *Activity Streams*. Retrieved June 17, 2015 from http://www.w3.org/wiki/Activity_Streams