Sharing between LRSs: a collaborative experiment in practical interoperability

There are a number of benefits to sharing statements between LRSs and a number of ways to do so. Three LRS vendors collaborated to create a proof of concept that shared statements between their three LRSs. This whitepaper describes the challenges we met along the way and the lessons that came out of it.

Project Background

Early in 2015, three LRS vendors came together to put the interoperability of their platforms to the test. The goal was to set up a realistic system involving an LMS, several activity providers and three different LRSs (one from each vendor), and then see what happened when we tried to share statements from one LRS to another. The aims of the project were as follows:

- Test and improve the interoperability of the three participating LRSs.
- Test the specification itself; is it specific enough to ensure interoperability?
- Find and prioritise gaps in the Conformance Test Suite; where are there interoperability problems between three LRSs that have used the suite?
- Promote the benefits of sharing statements to the wider community.
- Grow the spirit of collaboration between competing LRS vendors working together to drive the Learning Technologies industry forwards.

Collaborators

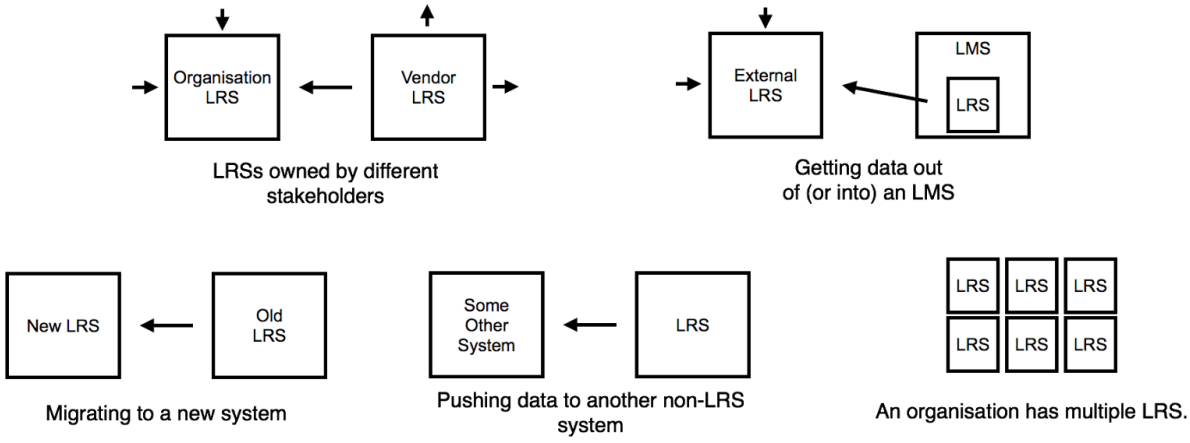
Several people from the three collaborating companies have been involved in the project in one way or another. The points of contact from each collaborating organization, and co-authors of this white-paper are:

- Andrew Downes from Rustici Software with Watershed LRS
- Ali Shahrazad from Saltbox with Wax LRS
- Ryan Smith from HT2 with Learning Locker

These three companies represent three of the most well known LRS vendors.

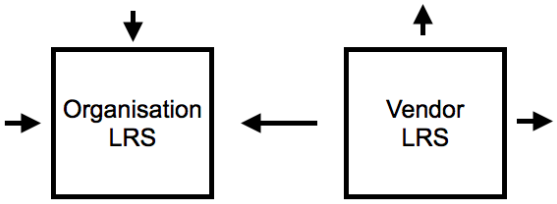
Why share statements?

There are a number of different situations where moving statements from one LRS to another might be useful.



Let's look at each of these in turn.

LRSs owned by different stakeholders

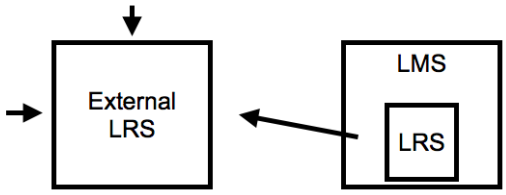


One use case for sharing statements that's already happening in the wild is product vendors maintaining their own LRS and also sending the data to the customer's LRS. This might include authoring tool vendors, content vendors, LMS vendors, indeed any vendor with a product that generates statements.

The vendor's LRS is normally the initial point of entry for statements, which are then sent on to the customer's LRS and possibly other destinations. The vendor uses their LRS to provide direct reports to the customer and to provide generic reports across their clients that can be used to improve the product or provide deeper insight into the learner taking place.

Although the reports provided by the vendor are often useful, customers also benefit from pulling the data into their own internal LRS. This gives them ownership of the data and allows them to combine the data with that from other sources for more interesting analytics and reporting or supporting various other use cases.

Getting data out of or into an LMS

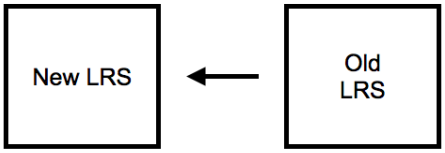


An LRS can either sit inside an LMS or be a separate product outside of it. Looking at products on the market today, LRSs inside LMSs are generally relatively limited stores of data that store data and present reports relating to stuff that happens in, or is launched from, the LMS. Conversely, stand alone LRS products outside the LMS tend to pull in and report on data from a range of sources, sometimes including one or more LMS.

If an LMS contains an LRS, and LMS activity needs to be reported on in an external LRS, it makes sense to allow the LMS LRS to continue to collect all of the LMS data together and then pass that in bulk to the external LRS.

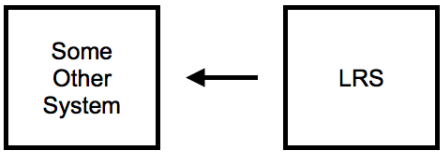
In some cases, organizations already have a lot of reporting, sometimes bespoke, within their LMS and are keen to keep the LMS as the store of all data. In these cases an external LRS can be used to gather up data from a variety of external sources and bring them into the LMS LRS as a single stream.

Migrating to a new system



When choosing a shiney new learning platform, it can be tempting to think it will last forever. All things come to an end though, and having a plan to get your data out when the time comes is vitally important. If you store all your learning records as statements in an LRS, statements can be shared with the new LRS when the time comes, making that part of the data migration incredibly straightforward. You then only have to worry about data not stored in the LRS! This is an important use case of statement sharing.

Pushing statements to a non-LRS system



There are a number of systems other than LRSs that use statements in some way to do some thing. Here's some examples:

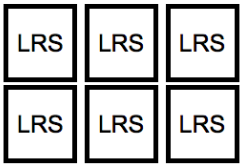
- A business information tool might use information contained in statements alongside other data to deliver business intelligence and provide other functions.
- A Training Delivery System might deliver performance support materials to learners at the point of need based their activity as tracked in statements.
- A credentialing, certification or badging tool might award credentials, certificates or badges based on achievements reported in statements.

All of these tools could retrieve statements by regularly querying the LRS for new Statements. Alternatively, there are benefits for the tool to implement enough of the specification in order to receive statements without incorporating a full LRS. Statements can then be sent to that tool as though it were an LRS and the tool can take action based on those statements. Querying for statements wastes resources when no new statements exist and new statements must wait until the next query before being sent on. Immediately sending statements on from the LRS is the only option when instant action is required.

It's unlikely that you would want to send statements directly to such a tool from an activity provider as these statements would not be stored. Instead, statements would be sent to an LRS configured to send statements on to that specific tool.

We're not aware of any products today that are enabled to receive statements in this way, but it's certainly a possibility for products to consider in the future.

Multiple LRSs within an organization



There are a number of reasons why an organization might have multiple LRSs. For example:

- Following a merger of two companies, where both companies had their own LRS.
- In a large organization with autonomous departments that have their own systems.
- An organization spread across multiple sites and/or countries.
- An organization including sites or operations with limited or intermittent connectivity such as a disaster zone or a ship.

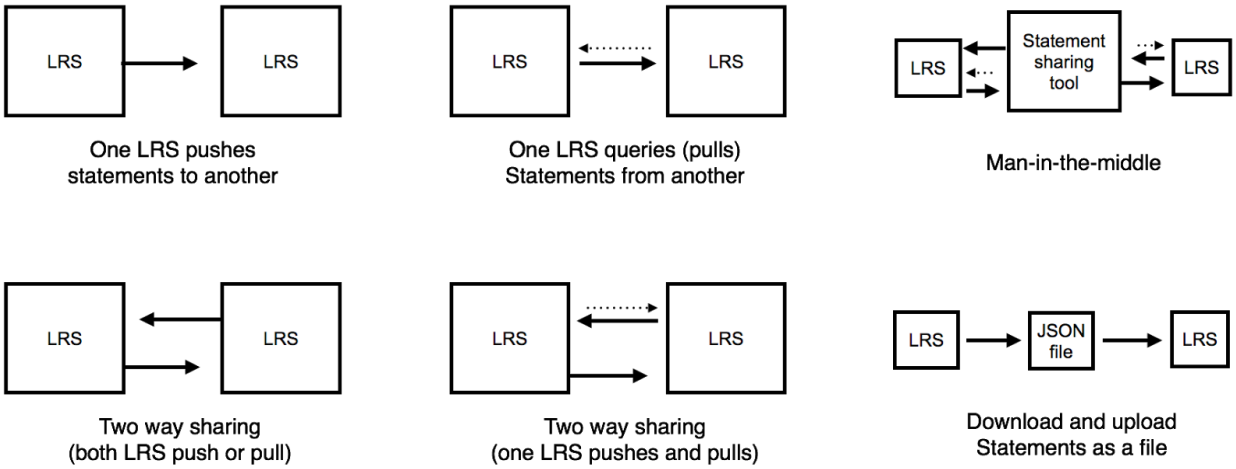
Whatever the reason for multiple LRS, sharing statements between these LRSs allows for overarching reporting of data collated from all these LRSs and ensures that learners can access their data and progress at whatever site they are at, whether that's a ship in the middle of the pacific, an office in Frankfurt or the International Space Station (astronauts are learners too).

Portable learner records

Another use case is the ability of learners to take their learner records with them when they move between and within organizations. This is a relatively complex use case not covered by the the experiment we completed, so the whitepaper collaborators have agreed to leave this particular example out.

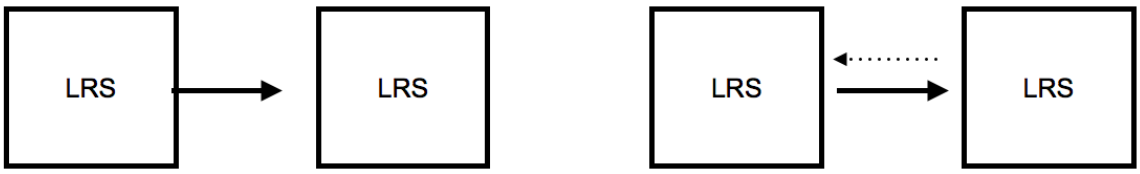
How can I share statements?

There are several different ways that statements can be moved between LRS:



Though we didn't try all of these approaches within the project, it's still helpful to know about them for background. Let's look at each option in more detail.

One way sharing

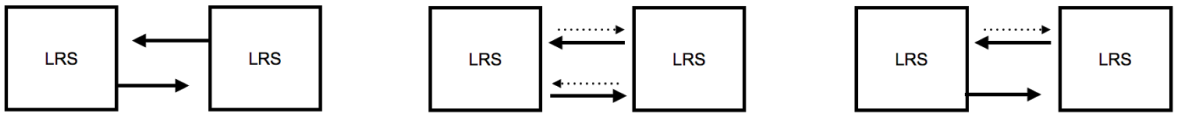


One approach is to have one LRS share its statements with another. This means that all statements in one LRS are transferred to another, but any statements already in the second LRS are not transferred back to the first.

This can either be achieved by one LRS sending statements to another, or by one LRS querying another for statements. Either of these processes follow the data transfer methods outlined in the specification so there is no need for any custom integration between the LRSs. If they are both conformat, it should work (which is the theory we set out to test in practice in this project).

One way sharing is one of the approaches we tested (see 'What we did' below).

Two way sharing

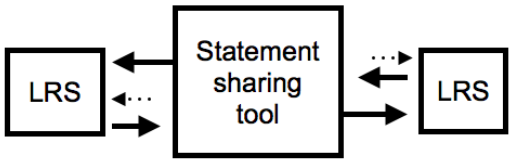


An extension of one way sharing is to additionally share statements in the other direction such that all statements in each LRS are shared with the other. This can be achieved by:

- Both LRSs sending on their statements to the other.
- Both LRSs regularly querying the other LRS to fetch statements.
- One LRS sending it's statements to the other LRS and also querying that LRS for statements.

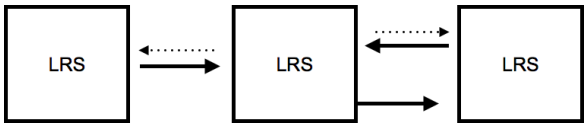
Two way sharing, with a single LRS both forwarding and fetching statements from the other, is one of the approaches we tested (see 'What we did' below).

Man-in-the-middle application



It's also possible to share statements using a 3rd party, man-in-the-middle application that sits outside the LRSs. This kind of application is configured to fetch statements from particular LRSs and send them on to other LRSs. The application doesn't necessarily store the statements itself, it just fetches them and sends them on to their required destination.

We didn't set out to test the man-in-the-middle approach in our project. We did have one LRS fetch statements from another and pass those statements to a third though, which is very similar to the man-in-the-middle approach. In this case an LRS is used as the man-in-the-middle application!



Download and upload

Finally, statements can be between LRSs by downloading the statements as a JSON document from one LRS and uploading it to another. This method is particularly valuable for transferring statements where the LRSs are not able to directly connect to one another due to connectivity issues or security restrictions.

It could also be useful in migrating data between LRSs where there is a time gap between when data can be downloaded from the old LRS and when it is able to be uploaded to the new one, or in situations where a backup of the transfer is required.

This method is clearly manually intensive and therefore much more costly per transfer than automatic methods. It's not suitable for a permanent link between LRSs where regular updates are required. Additionally, the time gap means there's no way for the receiving LRS to return errors to the sending LRS, so any problems with statements would need to be worked out manually.

How does the spec ensure interoperability?

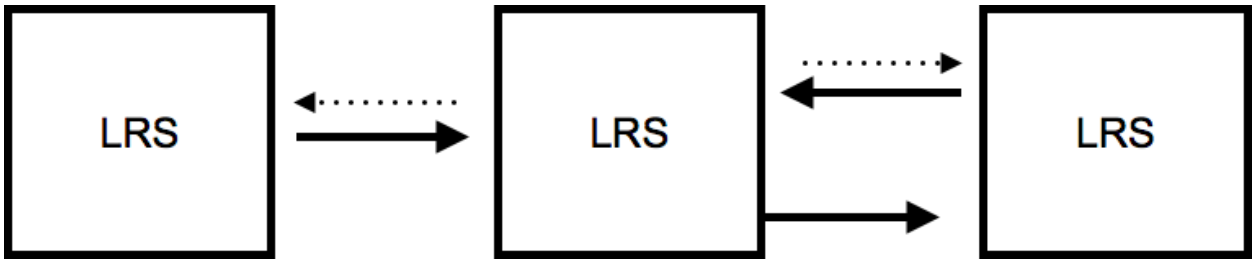
The specification was written with statement sharing in mind. The common structure of statements is useful not just for sending statements to an LRS, but ensures interoperability sharing between LRSs too. Common data transfer mechanisms designed for the flow of data both into and out of the LRS also help to ensure statements can be shared smoothly.

The spec also has rules in place to handle conflicting and duplicate statements. These are important when statements are shared two way between LRSs as statements sent from the first LRS may well come back round from the second. Duplicates need to be avoided without generating unwanted error messages to the end user.

Where validating the original author of a statement that has traversed multiple LRS, the spec outlines a process for digitally signing statements. The final recipient of the statement can validate the signature of a signed statements and confirm the author of the statement is who they say they are.

What we did

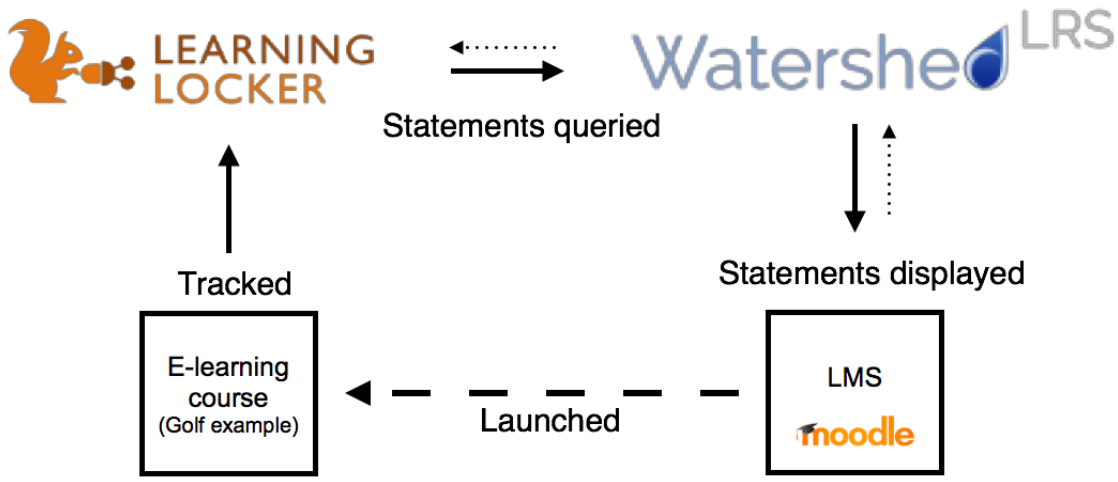
For this project we had three LRS in a chain, with the middle LRS providing a connection between the other two. One connection was set up as a one way link pulling in statements, whilst the other was a two way link both pushing and pulling statements. See the diagram below:



Initial investigation

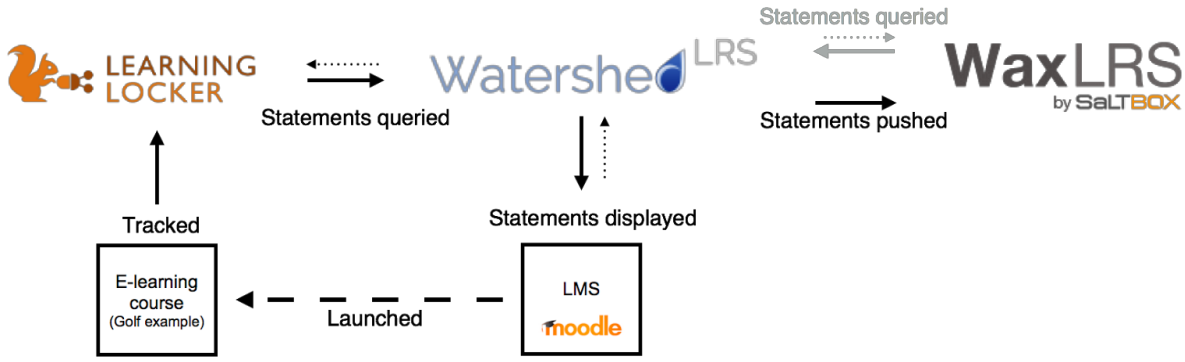
The first step in our investigation was to have a go at hooking the three LRSs together to see what worked and what didn't.

We set up a Moodle LMS with Tin Can plugins installed and launched the Golf Prototype (as an example of formal e-learning) from Moodle, tracking statements into Learning Locker. We then pulled these statements into Watershed as an example of sending statements in one direction from one LRS to another. Moodle was configured to fetch the statements from Watershed to display to the learner, completing the loop.

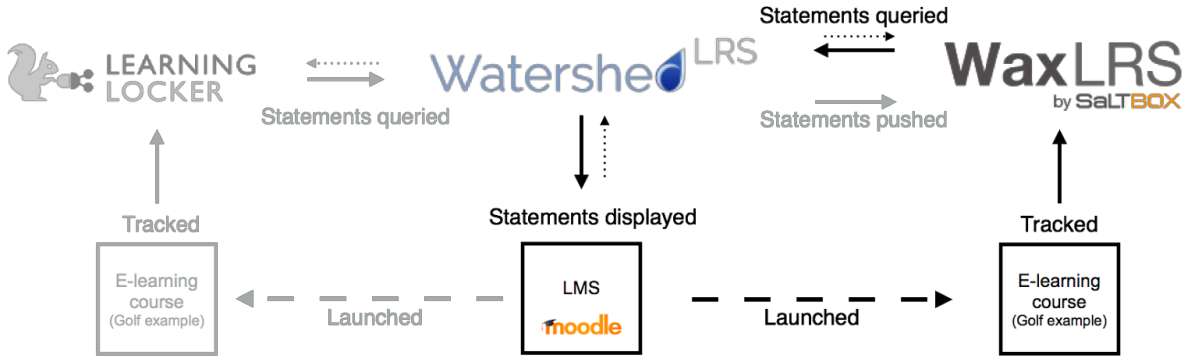


Next, we brought Wax LRS into the mix, setting up a two way link between Wax and Watershed. Statements from Wax were pulled into Watershed and statements from Watershed were pushed to Wax. This meant statements from the Golf Example were going through Learning Locker, into Watershed and on to Wax.

At this stage, Wax did not have any new statements to send to Watershed that it didn't already have. The pull from Wax to Watershed only served to test how the LRSs handled duplicate statements which had originally come from Watershed into Wax and were now being pulled back to Watershed from Wax.



We then pointed the Golf example at Wax in order to test the flow of new statements from Wax into Watershed and through to the display on Moodle. This is essentially the same flow as we originally tested with Learning Locker, but using Wax instead.



This initial investigation was broadly a success: we managed to get statements going to all the places we wanted them. This is a significant achievement, given that the three different products from different vendors had never before been tested together. Following the specification and using the conformance suite had set us on a solid foundation.

The test did reveal a number of bugs and issues though. Certain statements wouldn't work at all, and whatever statements we sent, the whole process was fragile; it would stop working

almost immediately when a link was created and a few statements had been sent. The two-way link did work in both directions, but not at the same time. It was clear there was work to be done.

Fixing bugs

The next step was to document the bugs and pass them to the developers working on the relevant LRS. All of the issues discovered through this process have now been resolved. What's interesting though is that many of the issues uncovered related to launching the Golf Prototype from Moodle and getting data out of that, rather than sending statements between LRSs. We had to make some fixes to the Golf Prototype and to the Moodle plugins too. The changes to the Golf Prototype are currently being peer reviewed before they are published; you can follow the pull request (and even download the unpublished versions) here.

Summary of issues

For the technically inclined, here's a summary of the issues we hit.

Areas unrelated to statement Forwarding where we had difficulties were:

- The LRS was rejecting Documents sent to the Document APIs due to problems with eTags.
- The LRS was rejecting the format of the Correct Responses Pattern for numeric questions. The specification was ambiguous on whether the LRS or Golf Prototype were right.
- The format of some of the Choice Ids in the Golf Prototype was wrong.
- The Golf Prototype was passing an additional query string parameter to the LRS that it shouldn't have done.

Issues relating to sharing statements between LRSs included:

- An incorrectly included 'more' link with statement results when there were, in fact, no more statements.
- With two-way statement forwarding where a statement was going out and then coming back in, comparision of the statements was failing such that the receiving LRS believed it was receiving a different statement with the same id, rather than a duplicate of an existing statement. This was causing the link to stop working entirely.
- There were issues with timestamp comparison when fetching statements such that the same statements were being retrieved again and again in a loop.

The development teams of each LRS worked hard to address these issues alongside other work priorities until we were ready to try again!

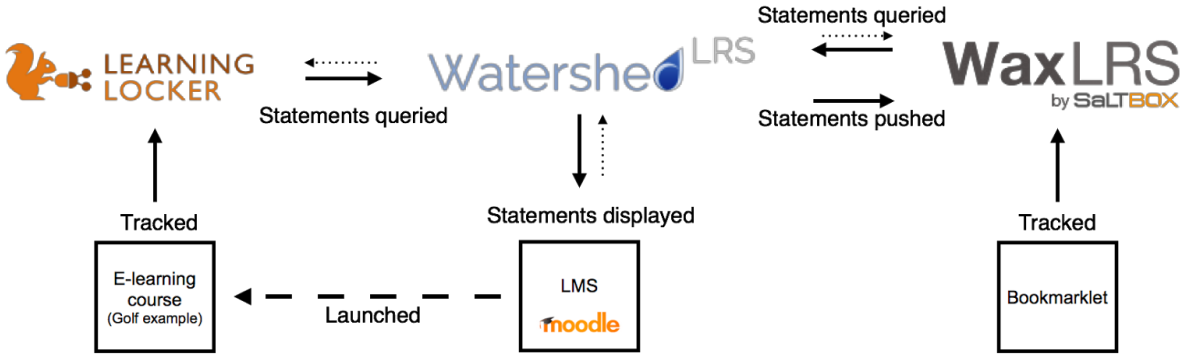
Reviewing the spec and conformance test

As the development teams worked on the issues, we raised an issue against the specification itself to clarify the correct format of the Correct Responses Pattern for numeric questions. This was quickly resolved in the specification. We also raised issues for all the problems we hit against the Conformance Test Suite. Once these tests are built into conformance testing, it will help to ensure that other LRSs do not run into the same problems that we hit. That said, we recommend that all LRSs carry out testing against other LRSs as we have done here.

Our Proof of Concept

The final proof of concept was relatively similar to the set up we tried in the initial investigation except that we used a different data source to feed into Wax - a bookmarklet - instead of re-using the Golf Prototype. We wanted to show a variety of data sources taking different routes through different LRSs being pulled together and displayed on the dashboard in Moodle.

Our final system looked liked this:

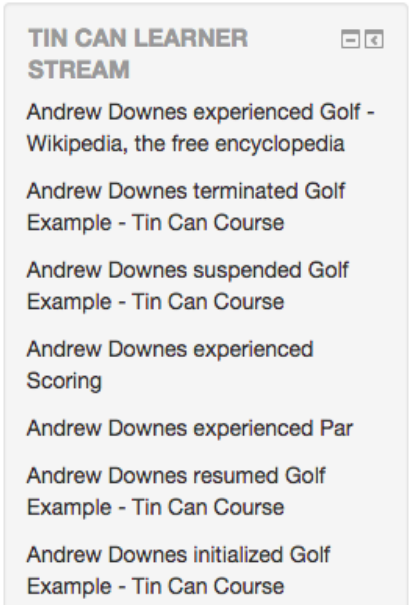


In this example, Learning Locker has been integrated into Moodle such that Moodle launches the Golf Example prototype and statements from Moodle and the Golf Example are sent to Learning Locker. These are forwarded on to Watershed.

At the same time, statements from the bookmarklet are sent to Wax LRS. As Wax and Watershed are linked, statements from the Golf Example are passed on to Wax and statements from Bookmarklet are passed to Watershed.

Moodle then pulls **all** statements from Watershed (including those originally from Learning Locker and Wax) and displays them to the learner as an activity stream *(pictured right)*.

Unlike the initial test, in the final proof of concept, all of these links are live at the same time. In theory, the learner could use the bookmarklet and example e-learning course at the same time and the statements from both would be fed through and displayed back to them on Moodle.



Screen capture and sharing

The final step in our project was to share the results of our experiment. We recorded a screencast demoing the final proof of concept system which will be published alongside this white paper. You can watch the screencast here.

As we recorded the screen capture, we realized that there was a short delay between a statement arriving in Wax and Learning Locker and it being pulled into Watershed. This is because when pulling statements, we have to wait for Watershed to make another check for new statements. This issue does not affect pushing statements from Watershed as incoming statements can be forwarded on instantly.

As a result of this, we had to pause screen recording in a couple of places for continuity. Do bear in mind as you watch the screencast that in reality the statements may have taken a minute or so to arrive at their final destinations.

We'll be presenting this project and whitepaper at xAPI Camp on 24th March 2015 and in a webinar at the end of April/early May.

What else could we have done

This example was deliberately simple for the purposes of demonstration and in practice the scenarios where statements will be shared between LRS may be more complex. The following could have easily been included in the demo instead of or as well as the elements we showed:

- Two way sync of statements between Learning Locker and Watershed. We included this as a one way sync as a way of contrast between the two way sync used between Wax and Watershed; we could easily have done this the other way round.
- Displaying statements in an external report outside of Moodle.
- Transferring existing statements. Our example showed statements being transferred as they were generated, but this approach also works for existing statements.
- One or more of the other statement sharing methods described in 'How can I share statements?' above.

Lessons learnt

The most important part of this project is the lessons learnt for LRS vendors and our customers. These are outlined below:

The spec is robust

The vast majority of issues we hit were bugs with one LRS or another. We found no ambiguities or gaps in the specification relating to sharing statements between LRS.

We did find one very minor spec ambiguity around the format of the Correct Response Pattern in some types of questions that had been carried over from SCORM. The xAPI Working Group are working to reduce dependency on SCORM within the spec and this issue, along with several others, will be clarified in the next version of the specification, 1.0.3.

Conformance testing is important

A significant number of tests were added to ADL's Conformance Test Suite sometime prior to this project. This paved the way for the project and ensured that areas that might have been issues for this project were handled ahead of time. We already know that the test suite is not complete and more work is needed by the community. That some issues still had to be fixed even after using the test suite proves this point. Passing the test suite doesn't guarantee conformance on its own.

All the issues we uncovered during this project have been raised on the Conformance Test Suite Github to be fixed.

Interoperability is hard

One key take away from this exercise is that even with an agreed specification to follow, interoperability is hard. We chose three of the most well known, high profile LRSs and they failed to interoperate fully on the first attempt. Once we had fixed the issues, the second attempt went extremely smoothly. This illustrates the importance to LRS vendors of going beyond conformance testing and trying their products with one another, as we have done (and possibly copying our method), then fixing issues that arise.

Sharing statements is not the hard bit!

Many of the issues we hit during this project were not directly related to sharing statements between LRS, but rather related to getting the statements into the LRS in the first place. (See the section titled 'Fixing Bugs' above).

This illustrates the importance on validation of statements by LRS; by ensuring invalid statements are not accepted into the LRS in the first place, there are less likely to be issues when moving these statements on into another LRS.

This also illustrates the importance of LRS vendors testing with multiple Activity Providers and Activity Providers testing with multiple LRS. Any adopter will make some errors implementing the specification and there are still some minor ambiguities in the specification that the xAPI Working Group are working on. Each successive patch release of the specification improves this, but there will always be a need for practical testing with real world implementations.

LRSs could do more

Comparing the ‘Benefits of Sharing statements’ section above to the features of our various LRSs was a reminder that we’ve all got work to do in order to fully take advantage of this feature of the specification. We all have statement sharing features somewhere on our roadmap so expect to see further development in this area in future and ask your vendor if you have any specific requirements.

Pushing is better than Pulling

As we described above, when pulling statements, the LRS periodically queries for new statements. This can lead to a delay in statements reaching their final destination as each batch of statements has to wait for the next query cycle.

In scenarios where it’s important statements are available in the final LRS almost immediately after being received by the first, we recommend that statements are pushed instead. Pushing statements also has the advantages of reducing the load on the LRS being queried regularly when no new statements exist and allowing statements to be sent to other statement receiving systems which are not full LRS.

Find out more

The screencast of our proof of concept can be found here. Also watch out for details of a webinar on this topic at the end of April!

The contact details of each of the whitepaper collaborators and links to their LRS websites are below.

Andrew Downes: andrew.downes@tincanapi.com
Watershed LRS: watershedlrs.com

Ali Shahrazad: ali.shahrazad@saltbox.com
Wax LRS: saltbox.com

Ryan Smith: ryan.smith@ht2.co.uk
Learning Locker: learninglocker.net

QR codes

If you're reading a printed copy of this whitepaper, scan the QR codes below to access digital versions.

This whitepaper: http://goo.gl/gWnrXL

The screencast: http://goo.gl/yGcT3h