TinCan2PROV: Exposing Interoperable Provenance of Learning Processes through Experience API Logs

Tom De Nies  Frank Salliau  Ruben Verborgh  Erik Mannens  Rik Van de Walle
{tom.denies,frank.salliau,ruben.verborgh,erik.mannens,rik.vandewalle}@ugent.be

Ghent University – iMinds
Department of Electronics and Information Systems, Multimedia Lab
Gaston Crommenlaan 8 bus 201
B-9050 Ledeberg-Ghent, Belgium

ABSTRACT
A popular way to log learning processes is by using the Experience API (abbreviated as xAPI), also referred to as Tin Can. While Tin Can is great for developers who need to log learning experiences in their applications, it is more challenging for data processors to interconnect and analyze the resulting data. An interoperable data model is missing to raise Tin Can to its full potential. We argue that, in essence, these learning process logs are provenance. Therefore, the W3C PROV model can provide the much-needed interoperability. In this paper, we introduce a method to expose PROV using Tin Can statements. To achieve this, we made the following contributions: (1) a formal ontology of the xAPI vocabulary, (2) a context document to interpret xAPI statements as JSON-LD, (3) a mapping to convert xAPI JSON-LD statements into PROV, and (4) a tool implementing this mapping. We preliminarily evaluate the approach by converting 20 xAPI statements taken from the public Tin Can Learning Record Store to valid PROV. Where the conversion succeeded, it did so without loss of valid information, therefore suggesting that the conversion process is reversible, as long as the original JSON is valid.

Categories and Subject Descriptors
H.1.m [Models and Principles]: Miscellaneous

1. INTRODUCTION
When a learning process is logged, this log describes which resources, which actions, and which people were involved in producing a certain result. In other words, this log constitutes the provenance of a learning process. Provenance is information about entities, activities, and people involved in producing a piece of data or thing. The PROV family of specifications [14] defines various aspects that are necessary to allow the interoperable interchange of provenance information in heterogeneous environments, such as the Web.

Knowing this, we could investigate all the aspects of logging learning processes, and create a data model based on PROV. However, a significant effort has already been made in this field, namely by the Advanced Distributed Learning (ADL) organization, in the form of the Experience API (xAPI) [20] (also referred to as Tin Can API), a specification to structure experience logs in the JSON format. In its most basic form, an xAPI statement corresponds to the sentence “I did this, and it resulted in that”. In the xAPI, the “I” is modeled as an actor, the “did” as a verb, the “this” as an object, and the “that” as a result. Apart from these basic concepts, various pieces of context information can be added to each xAPI statement. The xAPI is already widely adopted by organizations in the educational field1.

Instead of re-inventing the wheel, this paper specifies a conversion approach between the xAPI and W3C PROV. The approach consists of the following components, each signifying a contribution on their own: (1) an OWL ontology of the xAPI vocabulary, (2) a context document to interpret xAPI statements as JSON-LD [19], (3) a mapping to convert xAPI JSON-LD statements into PROV, and (4) a tool implementing this mapping. This way, developers are offered a choice in technology and serialization when it comes to logging, and the resulting Linked Data is more easily published in a scalable way and made interoperable with other provenance repositories.

The rest of this paper is structured as follows: first, we discuss the context of this paper and its related work. Next, we provide a general overview of our approach, after which we describe each of the aforementioned components in detail and provide a link to an online demonstrator. Finally, we evaluate the approach before concluding with a brief discussion and outlook to future work.

2. CONTEXT & RELATED WORK
The merit of interoperable provenance in the field of education has already been illustrated in literature. For example, it has been shown to help instructors to be more effective and to improve the learning experience [2]. We argue that it can provide teachers and students with an unseen amount of valuable information about the learning process. For example, the speed and continuity at which students complete

1http://tincanapi.com/adopters/
a task – intermittent or all at once – may already indicate a need to revise the task. If information such as that could be linked to the lineage of the study material itself, it would become possible to observe the direct effect of changes in the material on the learning experience. The possibilities become even greater when also taking into account the provenance of the teaching staff (e.g., teachers leaving/joining), the inventory of the IT infrastructure (e.g., the acquisition of a new device), etc. Connections that would never be apparent upon first glance would appear automatically, all because the provenance of all these aspects is made interoperable.

Unfortunately, current models to track learning processes are often designed with one particular use case in mind, and their data is siloed (often for good reasons, such as privacy). For example, Yeh et al. [21] built an e-learning system that keeps learning records such as grades, reading time, login times, and online discussions. The purpose of their system was to measure the effect of blended e-learning. Similarly, the authors of [5] measure patterns in a Web 2.0 learning environment.

A more comprehensive approach was proposed by Mazza et al. [11] in the form of MOCLog, a tool to analyze and present log data on a server running Moodle, an open-source PHP-based learning management system. While the rationale behind their approach is similar to ours – namely that all data that can be logged has potential value for analysis –, their system is catered towards one specific technology. This prevents other sources of external information to be interlinked with the logged data. In fact, mapping the MOCLog data to PROV might be an interesting case for future research efforts.

For a more extensive review of current student monitoring technologies, we refer to Corbi & Burgos [1], who provide insights on standards such as the Caliper framework by IMS [9], IEEE standard 1484.11.1/2 [8], JSON Activity Streams [18], and the xAPI. Of all the learning process monitoring technologies mentioned above, Tin Can seems to be the most developer-friendly, which explains its wide adoption by the industry. Therefore, exposing its data in a complementary way, by mapping it reversibly to an interoperable model is a logical step. The inspiration for this comes from previous conceptual mapping efforts to W3C PROV, driven by the same philosophy. Examples of such mappings include our own Git2PROV [3] mapping for the version control system Git, the W3C Provenance WG’s Dublin Core mapping [4], and a mapping to Datalog [12].

3. APPROACH

Figure 1 provides a high-level overview of our approach. The workflow starts with a Tin Can statement in the JSON format, which needs to be converted to PROV. We could just map every Tin Can property to a corresponding PROV concept. However, to allow for the mapping process to be reversed (i.e., making it possible to convert the provenance back to Tin Can), this would require an annotation in each PROV statement, indicating the original Tin Can property. While this is easily achieved by introducing an optional attribute (e.g., tinCan2prov:property="actor"), there is a more elegant solution.

This solution consists of first converting the Tin Can statement into proper Linked Data. The most straightforward approach to do this is by providing a JSON-LD context as explained in Section 5, mapping each term in the Tin Can statement to a IRI (Internationalized Resource Identifier) describing that term. This allows the original JSON object representing the Tin Can statement to remain unchanged, while providing us with the identifiers (IRIs) necessary to map the statement to PROV. This way, an xAPI actor object mapped to a PROV Agent can be associated with both types, with no need to introduce extra attributes.

Unfortunately, the IRIs provided by the ADL organization for the basic Tin Can terms point to PDF and GitHub URIs, making them not machine-interpretable. Ideally, the IRIs should be dereferenceable to a human-readable (e.g., HTML) or machine-interpretable (e.g., OWL) representation, depending on which type is requested. As this is currently not the case, we created our own instance of the xAPI ontology created by ADL, to be referred to from the JSON-LD context. This is described in detail in Section 4. If ADL would host its own instance of such an ontology in the future, the IRIs could easily be adapted.

Once the JSON-LD context is in place, each concept in the xAPI ontology is then mapped to its corresponding PROV representation, as explained in Section 6. Finally, this representation is serialized in one of the PROV serializations, as described in Section 7.

![Figure 1: High-level overview of TinCan2PROV.](image)

4. XAPI ONTOLOGY

At the time of this paper, the official specification of the xAPI is hosted in two places: one PDF document [20] specifying version 1.0.1 and one GitHub repository³ where the ongoing development is managed. Unfortunately, neither of these provide a machine-interpretable version of the xAPI, leaving their IRIs unsuitable to be used as Linked Data.

The verbs and activities vocabularies are specified in a better way. All possible values for the term verb are listed at http://www.adlnet.gov/expapi/verbs/. Analogously, all possible values for activity are listed at http://www.adlnet.gov/expapi/activities/. Each verb and activity has its own IRI, dereferenceable to a (human-readable) description of the concept. No machine-interpretable description is provided at this IRI at the time this paper was written, but the overall structure of the vocabulary suggests that this might be planned for the near future.

³http://www.w3.org/TR/json-ld/#the-context
³http://GitHub.com/adlnet/xAPI-Spec/
To allow for our proposed workflow to be executed, we created a formal version of the xAPI ontology as specified by ADL. Specifically, we hosted our own version of the specification, in a human- and machine-interpretable way.

Our formal ontology corresponds for the most part to the official xAPI specification. We constructed it by going through sections 4.0 and 5.0 of the xAPI document on ADL’s GitHub repository, and creating an OWL ontology following two simple rules. First, whenever an objectType was encountered, a corresponding owl:Class was created and – if applicable – linked to its superclass by rdfs:subClassOf. Second, whenever a property was encountered, a corresponding owl:ObjectProperty was created. In both cases the value of the rdfs:isDefinedBy property was set to the IRI of the xAPI.md document on GitHub (followed by a #), and rdfs:label was set to the name of the objectType. Finally, the possible instances of the :Verb class were enumerated as every verb listed at http://www.adlnet.gov/expapi/verbs/, and every activity listed at http://www.adlnet.gov/expapi/activities/ was made a subclass of :ActivityType. An example of a simple xAPI statement, modeled in the ontology is illustrated in Figure 2.

Figure 2: Example of a simple statement in the xAPI ontology.

In a number of cases, an extra class was created to support the modeling of more complex objects that aren’t supported by default in OWL or RDF Schema. For example, the range of the :display, :name and :description properties includes a :LanguageMap. A similar approach was used to model :extensions.

A full description of the ontology is available at http://semweb.mmlab.be/ns/tincan2prov/, which is abbreviated using the prefix xapi: throughout the rest of this paper. When navigated to this ontology with a browser, an HTML representation of the ontology will be shown. However, when an RDF media type4 is specified in the Accept header of the

HTTP request for the same IRI, an RDF (OWL) description will be returned. Ideally, such an ontology should be hosted at the ADL organization itself in the future, for example at http://www.adlnet.gov/expapi/.

5. ADDING JSON-LD CONTEXT

In order to convert a JSON document to JSON-LD, we have to design and specify a JSON-LD context (@context). Such a context document maps all terms that may occur in a document to their corresponding IRIs in the ontology. Our JSON-LD context document is available at http://semweb.mmlab.be/ns/tincan2prov/tincan2prov.jsonld. To convert a Tin Can JSON statement to Linked Data, a @context entry referencing this document is added to the root of the JSON, an @type entry with value xapi:Statement, as well as the following snippet to every :verb and :object property: @context: { "id": "@id" }. This is illustrated in Example 1.

A few additional conventions are necessary to ensure a smooth conversion, the first of which regarding language. The xAPI conforms to RFC 5646 [17] language tags for internationalization, while JSON-LD conforms to the older RFC 4646 [16]. In other words, upon conversion all language tags must be changed (if necessary) to comply with RFC 4646. The default language in our context document is set to “en”.

The second convention concerns extensions and attachments. The xAPI allows the addition of extra JSON maps as extension to the vocabulary. However, since the keys of these maps are unknown, it is impossible for us to define a proper JSON-LD context for them. Therefore, when extensions are used, developers wishing to convert their xAPI statements to JSON-LD must provide this context themselves. In our ontology, we provided the generic :Extension class, described by the properties :key and :value, which could be used in such a context document.

Example 1: xAPI Statement in JSON-LD.

```json
{  "@context": "http://semweb.mmlab.be/ns/tincan2prov/tincan2prov.jsonld",  "@type": "http://semweb.mmlab.be/ns/tincan2prov/Statement",  "actor": {    "@id": "http://www.example.org/exercise1",    "mbox": "mailto:tom.denies@ugent.be",    "name": "Tom De Nies",    "objectType": "Agent"  },  "verb": {    "@id": "xapi-verbs:completed",    "display": {      "en": "completed",      "nl": "afgewerkt"    }  },  "object": {    "@id": "http://www.example.org/exercise1",    "objectType": "Activity",    "definition": {      "name": "Example Activity"  }  }}
```

4http://www.w3.org/2008/01/rdf-media-types
Example 2 shows what happens when this statement is converted to an RDF notation such as Turtle.

Example 2: the same xAPI Statement in Turtle.

```turtle
[] xapi:actor [
  a xapi:Agent;
  xapi:name "Tom De Nies"@en;
  foaf:mbox <mailto:tom.denies@ugent.be> 
];
  xapi:verb xapi-verbs:completed ;
```

6. MAPPING XAPI TO PROV

In this section, we describe a mapping between our formal instance of the xAPI ontology, and the PROV Ontology (PROV-O) [10]. By doing this, we are effectively mapping every Tin Can concept to a PROV concept.

We start from an RDF representation of an xAPI statement, obtained by following the steps described in Section 5. For each :Statement, a bundle is created using TriG notation, as specified in PROV-Links [13]. This bundle will contain all triples for this statement, including those created during the JSON-LD conversion.

Then, PROV concepts are inferred and asserted for each property of the statement. The details of all the inferred PROV concepts are listed in Table 1. Note that during the JSON-LD conversion, class instances are created as the values for the properties :actor, :verb, :object, :result, :context, :attachments, and :contextActivities, respectively. The remaining properties that don’t map to any PROV concepts are kept as they are, and will be asserted as attribute-value pairs in serializations other than RDF.

The result is an RDF document of mixed PROV-O and xAPI ontology concepts, which conforms to the PROV Data model. As explained in Section 7, it is now possible to translate this document into one of the other PROV serializations. Figure 3 shows a simplified version of such a provenance graph, representing the same xAPI statement as in Figure 2.

7. SERIALIZATION

There are 3 official W3C PROV serializations: PROV-N [15] and PROV-O were published as recommendations, and PROV-XML [6] was published as a note. Others have created their own serializations, such as PROV-JSON [7] and SVG.

As described in Section 6, we restrict our implementation of the mapping to the RDF (PROV-O) serialization. For the other serializations, we refer to the excellent ProvTranslator by the University of Southampton, which – at the time of this paper – supports PROV-N, PROV-O, PROV-JSON, PROV-XML, Turtle, TriG, and SVG.

8. TRY IT YOURSELF

An online demonstrator of the workflow described above is available at http://tincan2prov.org. The demonstrator, as illustrated in Figure 4, provides a form where a user can enter a Tin Can statement in JSON format, which – upon submission – is then converted to JSON-LD, RDF, and other formats.
and PROV-O. As the JSON-LD to RDF conversion process was not our primary focus, we relied on the jsonld\(^8\) and n3\(^9\) libraries for Node JS for this step. At the time of writing, advanced features such as extensions and attachments are not yet fully supported due to the arbitrary nature of their properties. This remains as a challenge for future work. All updates regarding the ongoing development and improvements are published at the same URI.

9. EVALUATION

A mapping can be deemed successful if it converts data from one representation to another, without losing any information. In this paper, we introduced two separate mappings. On the one hand, we introduced a workflow to convert Tin Can statements to Linked Data using the xAPI ontology. On the other hand, we created a mapping between this xAPI ontology and W3C PROV. It’s important to keep this distinction in mind when interpreting the evaluation results.

A formal proof of completeness between either of these representations is beyond the scope of this workshop paper. However, we do evaluate the mapping demonstrator by performing a limited empirical evaluation. We copied 20 diverse statements\(^10\) from the Tin Can Public LRS\(^11\), and converted them first to JSON-LD, and then to PROV using the online demonstrator provided. Upon successful conversion, we then manually inspected each of the representations for loss of information. By ‘loss of information’, we mean that data present in one representation, can no longer be found in another representation.

The detailed evaluation is provided at the following URL: [http://tincan2prov.org/evaluation.html](http://tincan2prov.org/evaluation.html). On this page, the original Tin Can statements are listed as they were copied from the public LRS, as well as their JSON-LD form and their PROV-O form. Additionally, the PROV graph of successful results can be viewed, courtesy of the Prov-Translator. We provide a summary of the most important observations here.

During this preliminary evaluation, we discovered a number of technical challenges with regard to robustness to user error. For example, one statement did not convert from JSON to RDF, due to incorrect URL-encoding of an identifier in the original JSON statement, which means the mapping tool was not at fault. In another statement, the key "ar-SA@calendar=gregorian" was used in an attempt for internationalization. However, this does not result in a valid

\(^8\)[https://www.npmjs.com/package/jsonld](https://www.npmjs.com/package/jsonld)
\(^9\)[https://www.npmjs.com/package/n3](https://www.npmjs.com/package/n3)

---

Table 1: Actions taken and PROV concepts asserted for each observed property of a xAPI statement. In all cases, any remaining properties are kept as attribute-value pairs to the corresponding PROV concept.

<table>
<thead>
<tr>
<th>Statement property</th>
<th>condition/property</th>
<th>Action taken</th>
<th>PROV concept asserted</th>
</tr>
</thead>
<tbody>
<tr>
<td>:actor</td>
<td>:name</td>
<td>prov:Agent</td>
<td>&lt;value of :verb&gt; prov:wasAssociatedWith &lt;this prov:Agent&gt;</td>
</tr>
<tr>
<td></td>
<td>:member</td>
<td></td>
<td>prov:label</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>prov:hadMember</td>
</tr>
<tr>
<td>:verb</td>
<td>:display</td>
<td>prov:Activity</td>
<td>&lt;this prov:Activity&gt; prov:used &lt;value of :object&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>prov:label with value for every language</td>
</tr>
<tr>
<td>:object</td>
<td>:name</td>
<td>prov:Entity</td>
<td>&lt;value of :verb&gt; prov:wasGeneratedBy &lt;value of :verb&gt;</td>
</tr>
<tr>
<td></td>
<td>:type</td>
<td></td>
<td>prov:Entity</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>prov:label with value for every language</td>
</tr>
<tr>
<td>:result</td>
<td></td>
<td>prov:Entity</td>
<td>&lt;value of :verb&gt; prov:wasGeneratedBy &lt;value of :verb&gt;</td>
</tr>
<tr>
<td>:score</td>
<td></td>
<td>prov:Entity</td>
<td>&lt;value of :verb&gt; prov:used &lt;value of :verb&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;root statement id&gt; prov:wasInfluencedBy &lt;this :Statement&gt;</td>
</tr>
<tr>
<td>:context</td>
<td>:statement</td>
<td>prov:Entity</td>
<td>&lt;value of :verb&gt; prov:used &lt;value of :Context&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;root statement id&gt; prov:wasInfluencedBy &lt;this :Statement&gt;</td>
</tr>
<tr>
<td>:contextActivities</td>
<td>:parent</td>
<td>prov:Collection with all :Activity objects below as prov:hadMember.</td>
<td>with prov:label=&quot;Parent&quot;</td>
</tr>
<tr>
<td></td>
<td>:grouping</td>
<td></td>
<td>&lt;value of :context&gt; prov:wasInfluencedBy &lt;this :Activity&gt;,</td>
</tr>
<tr>
<td></td>
<td>:category</td>
<td></td>
<td>with prov:label=&quot;Grouping&quot;</td>
</tr>
<tr>
<td></td>
<td>:other</td>
<td></td>
<td>&lt;value of :context&gt; prov:wasInfluencedBy &lt;this :Activity&gt;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>with prov:label=&quot;Other&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>prov:Entity</td>
<td>&lt;value of :verb&gt; prov:qualifiedStart &lt;prov:Start with same time&gt;</td>
</tr>
<tr>
<td>:timestamp</td>
<td></td>
<td>prov:wasGeneratedBy</td>
<td></td>
</tr>
<tr>
<td>:authority</td>
<td></td>
<td></td>
<td>&lt;this value&gt; rdf:type prov:Agent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;statement id&gt; prov:wasAttributedTo &lt;this prov:Agent&gt;</td>
<td></td>
</tr>
<tr>
<td>:attachments</td>
<td>for each :Attachment</td>
<td>prov:Entity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>:display</td>
<td></td>
<td>prov:label with value for every language</td>
</tr>
</tbody>
</table>

\(^10\)After 20 statements, it became increasingly difficult to find more statements with enough diversity on the public LRS.

Language Map when converted to RDF. Therefore, these keys were filtered out during conversion to JSON-LD.

Converting the Tin Can RDF representation to PROV went smoothly. For the 19 statements that did successfully convert to N3, we observed no loss of valid information in the PROV-O representation. This suggests that the mapping in these cases is fully reversible, with the exception of invalid elements such as the aforementioned internationalization tags.

10. DISCUSSION & FUTURE WORK

By providing a reversible mapping workflow, we have increased the interoperability of Tin Can, without sacrificing its information content. Even apart from the inferred PROV, the JSON-LD conversion step had merit on its own: after this step, Tin Can data can now be (anonymized and) exposed as Linked Data. Adaptation by the ADL organization of a formal ontology such as ours in the future would improve the situation even better.

As for our own future work, we will continue the evaluation and development of the mapping tool to increase robustness. Currently, extensions and attachments are not fully supported. We mean to improve this by providing a JSON-LD context for commonly used extensions and attachments, allowing them to fit into our proposed workflow. Furthermore, our proposed workflow will be adopted in the context of the Flemish project EduTablet\textsuperscript{12}, furthering innovation in digital learning with mobile devices. This will result in a large corpus of learning log data, which will allow us to perform an evaluation in terms of new knowledge learned by exposing the learning logs as PROV.

Acknowledgments

The research activities in this paper were funded by Ghent University, iMinds (a research institute founded by the Flemish Government), the Institute for Promotion of Innovation by Science and Technology in Flanders (IWT), the FWO-Flanders, and the European Union, in the context of the EduTablet project.

11. REFERENCES


\textsuperscript{12}http://www.iminds.be/en/projects/2014/03/20/edutab