Timeline Of Learner Success

As learners engage in a blended eLearning ecosystem, they will build up a history of learning experiences. When that eLearning ecosystem adheres to a framework dedicated to supporting and understanding the learner, such as the Total Learning Architecture (TLA), it becomes possible to retell their story through data. One important aspect of that story is the learner's history of success.

1 Ideal Statements

In order to accurately portray a learner's timeline of success, there are a few requirements of the data produced by a Learning Record Provider (LRP). They are as follows:

- the learner must be uniquely and consistently identified across all LRPs
- learning activities which access a learner's understanding of material should report if the learner was successful or not
  - if the assessment is scored, the grade earned by the learner should be reported
  - if the assessment is scored, the minimum and maximum possible grade should be reported
- The learning activities must be uniquely and consistently identified across all LRPs
- The time at which a learner completed a learning activity must be recorded
  - The timestamp should contain an appropriate level of specificity.
  - ie. Year, Month, Day, Hour, Minute, Second, Timezone

1.1 statement parameters to utilize

The statement parameter locations here are written in JSONPath. This notation is also defined by the Z Schema JSONPath

- $\$.timestamp$
- $\$.result.success$
- $\$.actor$
- $\$.verb.id$

2 TLA Statement problems

The data collected at the TLA pilot run supports the following algorithm.

3 Algorithm

3.1 Summary

1. Query an LRS via a GET request to the statements endpoint using the parameters agent, since and until
2. Filter the results to the set of statements where:
   - $.*verb.id* is one of:
     - http://adlnet.gov/expapi/verbs/passed
     - https://w3id.org/xapi/dod-isd/verbs/answered
     - http://adlnet.gov/expapi/verbs/completed
   - $.*result.success* is true

3.2 Query an LRS via REST

How to query an LRS via a GET request to the Statements Resource [1] [2]

```
Agent = "agent={"account":
                {"homePage": "https://example.homepage",
                 "name": 123456}}"

Since = "since=2018-07-20T12:08:47Z"

Until = "until=2018-07-21T12:08:47Z"

Base = "https://example.endpoint/statements?"

endpoint = Base + Agent + "&" + Since + "&" + Until

Auth = Hash generated from basic auth

S = curl -X GET -H "Authorization: Auth"
         -H "Content-Type: application/json"
         -H "X-Experience-API-Version: 1.0.3"
         Endpoint
```

[1] S is the set of all statements parsed from the statements array within the HTTP response to the Curl request. It may be possible that multiple Curl requests are needed to retrieve all query results. If multiple requests are necessary, S is the result of concatenating the result of each request into a single set

[2] Querying an LRS will not be defined within the following Z specifications but the Results of the Query will be

3.3 xAPI Z Specifications

An xAPI statement(s) is only defined abstractly within the context of Z. A concrete definition for an xAPI statement(s) it outside the scope of this specification.

3.3.1 Basic Types

$IFI ::= mbox \mid mbox\_sha1sum \mid openid \mid account$

- Type unique to Agents and Groups, The concrete definition of the listed values is outside the scope of this specification

$OBJECTTYPE ::= Agent \mid Group \mid SubStatement \mid StatementRef \mid Activity$

- A type which can be present in all activities as defined by the xAPI specification

$INTERACTIONTYPE ::= true-false \mid choice \mid fill-in \mid long-fill-in \mid matching \mid performance \mid sequencing \mid likert \mid numeric \mid other$

- A type which represents the possible interactionTypes as defined within the xAPI specification

$INTERACTIONCOMPONENT ::= choices \mid scale \mid source \mid target \mid steps$

- A type which represents the possible interaction components as defined within the xAPI specification
- the concrete definition of the listed values is outside the scope of this specification

$CONTEXTTYPES ::= parent \mid grouping \mid category \mid other$

- A type which represents the possible context types as defined within the xAPI specification

$[STATEMENTS, STATEMENT]$

- Basic types for the results of querying an LRS

$[ACTOR, VERB, OBJECT, RESULT, CONTEXT, TIMESTAMP, STORED, ATTACHEMNTS]$

- Basic types for the top level properties of a statement

$[ID, AGENT, GROUP, EXTENSIONS, OBJECTTYPE, NAME, DISPLAY, DESCRIPTION]$

- Basic types for paramters which can exists within multiple types of top level properties

$[MEMBER]$

- Basic type unique to Groups

$[DEFINITION, TYPE, MOREINFO]$

- Basic types unique to object

$[SCORE, SUCCESS, COMPLETION, RESPONSE, DURATION]$

- Basic types unique to result

$[REGISTRATION, INSTRUCTOR, CONTEXTACTIVITIES, REVISIOIN, PLATFORM, LANGUAGE]$

- Basic types unique to context

$[USAGETYPE, CONTENTTYPE, SHA2, FILEURL]$

- Basic types unique to attachment

3.3.2 Id Schema

$$
\begin{array}{|l}
\hline
Id \\
id : \mathbb{F}_1 \#1 \\
\hline
\end{array}
$$

- the schema Id introduces the component id which is a non-empty finite set of 1 value

3.3.3 Schemas for Agents and Groups

$$
\begin{array}{|l}
\hline
Agent \\
agent : AGENT \\
objectType : OBJECTTYPE \\
name : \mathbb{F}_1 \#1 \\
ifi : IFI \\
\hline
objectType = Agent \\
agent = \{ifi\} \cup \mathbb{P}\{name, objectType\} \\
\hline
\end{array}
$$

- The schema $Agent$ introduces the component $agent$ which is a set consisting of an ifi and optionally an $objectType$ and/or $name$

$$
\begin{array}{|l}
\hline
Member \\
Agent \\
member : \mathbb{F}_1 \\
\hline
member = \{a : AGENT \mid \forall a : a_0..a_n \bullet a = agent\} \\
\hline
\end{array}
$$

- The schema $Member$ introduces the component $member$ which is a set of objects a, where for every a within $a_0..a_n$, a is an $agent$

$$
\begin{array}{|l}
\hline
Group \\
\hline
Member \\
group : GROUP \\
objectType : OBJECTTYPE \\
ifi : IFI \\
\hline
name : \mathbb{F}_1 \#1 \\
objectType = Group \\
group = \{objectType, name, member\} \lor \{objectType, member\} \lor \\
\qquad \{objectType, ifi\} \cup \mathbb{P}\{name, member\} \\
\hline
\end{array}
$$

- The schema *Group* introduces the component *group* which is of type $GROUP$ and is a set of either *objectType* and *member* with optionaly *name* or *objectType* and ifi with optionally *name* and/or *member*

$$
\begin{array}{|l}
\hline
Actor \\
\hline
Agent \\
Group \\
actor : AGENT \lor GROUP \\
\hline
actor = agent \lor group \\
\hline
\end{array}
$$

- The schema *Actor* introduces the component *actor* which is either an *agent* or *group*

3.3.4 Verb Schema

$$
\begin{array}{|l}
\hline
Verb \\
\hline
Id \\
display, verb : \mathbb{F}_1 \\
\hline
verb = \{id, display\} \lor \{id\} \\
\hline
\end{array}
$$

- The schema $Verb$ introduces the component *verb* which is a set that consists of either *id* and the finite set *display* or just *id*

3.3.5 Object Schema

$$
\begin{array}{|l}
\hline
Extensions \\
\hline
extensions, extensionVal : \mathbb{F}_1 \\
extensionId : \mathbb{F}_1 \#1 \\
\hline
extensions = \{e : (extensionId, extensionVal) \mid \forall i, j : e_i..e_j \bullet \\
\qquad (extensionId_i, extensionVal_i) \lor (extensionId_i, extensionVal_j) \land \\
\qquad (extensionId_j, extensionVal_i) \lor (extensionId_j, extensionVal_j) \land \\
\qquad extensionId_i \neq extensionId_j\} \\
\hline
\end{array}
$$

- The schema $Extensions$ introduces the component $extensions$ which is a non-empty finite set that consists of ordered pairs of $extensionId$ and $extensionVal$. Different $extensionId$s can have the same $extensionVal$ but there can not be two identical $extensionId$ values
- $extensionId$ is a non-empty finite set with one value
- $extensionVal$ is a non-empty finite set

$$
\begin{array}{|l}
\hline
InteractionActivity \\
interactionType : INTERACTIONTYPE \\
correctResponsePattern : \mathrm{seq}_1 \\
interactionComponent : INTERACTIONCOMPONENT \\
\hline
interactionActivity = \{interactionType, correctReponsePattern, interactionComponent\} \lor \\
\qquad\qquad \{interactionType, correctResponsePattern\} \\
\hline
\end{array}
$$

- The schema $InteractionActivity$ introduces the component $interactionActivity$ which is a set of either $interactionType$ and $correctResponsePattern$ or $interactionType$ and $correctResponsePattern$ and $interactionComponent$

$$
\begin{array}{|l}
\hline
Definition \\
InteractionActivity \\
Extensions \\
definition, name, description : \mathbb{F}_1 \\
type, moreInfo : \mathbb{F}_1 \#1 \\
\hline
definition = \mathbb{P}_1\{name, description, type, moreInfo, extensions, interactionActivity\} \\
\hline
\end{array}
$$

- The schema $Definition$ introduces the component $definition$ which is the non-empty, finite power set of $name$, $description$, $type$, $moreInfo$ and $extensions$

$$
\begin{array}{|l}
\hline
Object \\
\hline
Id \\
Definition \\
Agent \\
Group \\
Statement \\
objectTypeA, objectTypeS, objectTypeSub, objectType : OBJECTTYPE \\
substatement : STATEMENT \\
object : \mathbb{F}_1 \\
\hline
substatement = statement \\
objectTypeA = Activity \\
objectTypeS = StatementRef \\
objectTypeSub = SubStatement \\
objectType = objectTypeA \lor objectTypeS \\
object = \{id\} \lor \{id, objectType\} \lor \{id, objectTypeA, definition\} \\
\qquad \lor \{id, definition\} \lor \{agent\} \lor \{group\} \lor \{objectTypeSub, substatement\} \\
\qquad \lor \{id, objectTypeA\} \\
\hline
\end{array}
$$

- The schema *Object* introduces the component *object* which is a non-empty finite set of either *id*, *id* and *objectType*, *id* and *objectTypeA* and *definition*, *agent*, *group*, or *substatement*
- The schema *Statement* and the corresponding component *statement* will be defined later on in this specification

3.3.6 Result Schema

$$
\begin{array}{|l}
\hline
Score \\
\hline
score : \mathbb{F}_1 \\
scaled, min, max, raw : \mathbb{Z} \\
\hline
scaled = \{n : \mathbb{Z} \mid -1.0 \leq n \leq 1.0\} \\
min = n < max \\
max = n > min \\
raw = raw = \{n : \mathbb{Z} \mid min \leq n \leq max\} \\
score = \mathbb{P}_1\{scaled, raw, min, max\} \\
\hline
\end{array}
$$

- The schema *Score* introduces the component *score* which is the non-empty powerset of *min*, *max*, *raw* and *scaled*

Result

$Score$
$Extensions$
$success, completion, response, duration : \mathbb{F}_1 \#1$
$result : \mathbb{F}_1$

$success = true \lor false$
$completion = true \lor false$
$result = \mathbb{P}_1\{score, success, completion, response, duration, extensions\}$

- The schema *Result* introduces the component *result* which is the non-empty power set of *score*, *success*, *completion*, *response*, *duration* and *extensions*

3.3.7 Context Schema

Instructor

$Agent$
$Group$
$instructor : AGENT \lor GROUP$

$instructor = agent \lor group$

- The schema $Instructor$ introduces the component *instructor* which can be ether an *agent* or a *group*

Team

$Group$
$team : GROUP$

$team = group$

- The schema $Team$ introduces the component *team* which is a *group*

$$
\begin{array}{|l}
\hline
Context \\
\hline
Instructor \\
Team \\
Object \\
Extensions \\
registration, revision, platform, language : \mathbb{F}_1 \#1 \\
parentT, groupingT, categoryT, otherT : CONTEXTTYPES \\
contextActivities, statement : \mathbb{F}_1 \\
\hline
statement = object \setminus (id, objectType, agent, group, definition) \\
parentT = parent \\
groupingT = grouping \\
categoryT = category \\
otherT = other \\
contextActivity = \{ca : object \setminus (agent, group, objectType, objectTypeSub, substatement)\} \\
contextActivityParent = (parentT, contextActivity) \\
contextActivityCategory = (categoryT, contextActivity) \\
contextActivityGrouping = (groupingT, contextActivity) \\
contextActivityOther = (otherT, contextActivity) \\
contextActivities = \mathbb{P}_1\{contextActivityParent, contextActivityCategory, \\
\qquad\qquad contextActivityGrouping, contextActivityOther\} \\
context = \mathbb{P}_1\{registration, instructor, team, contextActivities, revision, \\
\qquad platform, language, statement, extensions\} \\
\hline
\end{array}
$$

- The schema *Context* introduces the component *context* which is the non-empty powerset of *registration*, *instructor*, *team*, *contextActivities*, *revision*, *platform*, *language*, *statement* and *extensions*

3.3.8 Timestamp and Stored Schema

$$
\begin{array}{|l}
\hline
Timestamp \\
\hline
timestamp : \mathbb{F}_1 \#1 \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
Stored \\
\hline
stored : \mathbb{F}_1 \#1 \\
\hline
\end{array}
$$

- The schema *Timestamp* and *stored* introduce the components *timestamp* and *stored* respectively. Each are non-empty finite sets containing one value

3.3.9 Attachements Schema

$$
\begin{array}{|l}
\hline
\textit{Attachments} \\
\hline
display, description, attachment, attachments : \mathbb{F}_1 \\
usageType, sha2, fileUrl, contexntType : \mathbb{F}_1 \, \#1 \\
length : \mathbb{N} \\
\hline
attachment = \{usageType, display, contentType, length, sha2\} \cup \mathbb{P}\{description, fileUrl\} \\
\hline
\end{array}
$$

3.3.10 Root Level Dependent Global Variables

$$
\begin{array}{|l}
\hline
\textit{Statement} \\
\hline
s : Statement \\
\hline
s = \{Actor, Verb, Object, Timestamp\} \mid \\
\quad \{Actor, Verb, Object, Timestamp, Context\} \mid \\
\quad \{Actor, Verb, Object, Timestamp, Result\} \mid \\
\quad \{Actor, Verb, Object, Timestamp, Result, Context\} \\
\hline
\end{array}
$$

- The variable s is of type $Statement$ and consists of an $Actor$, $Verb$, $Object$, $Timestamp$ and optionally $Context$ and $Result$

$$
\begin{array}{|l}
\hline
\textit{Statements} \\
\hline
S : Statements \\
\hline
S = \{ \, s : Statement \mid S \neg \emptyset \} \\
\\
\hline
\end{array}
$$

- The variable S is of type $Statements$ and is a set of objects s, each of type $Statement$
- The variable S is a non empty set

3.3.11 Global Variables

Actor Verb Object Result Context Timestamp Stored Attachments

3.3.12 Timeline Leaner Success System State

$$
\begin{array}{|l}
\hline
TimelineLearnerSuccess \\
\hline
S_{extra}, S_{completion}, S_{success}, S_{failure} : \mathbb{P}\, S \\
\hline
S_{extra} \cup S_{completion} = S \\
S_{extra} \cap S_{completion} = \{\} \\
S_{success} \cup S_{failure} = S_{completion} \\
S_{success} \cap S_{failure} = \{\} \\
\hline
\end{array}
$$

- The sets $S_{extra}, S_{completion}, S_{success}, S_{failure}$ are the powerset of S

- The union of sets S_{extra} and $S_{completion}$ is equal to the complete set of statements S
- No values are shared between the sets S_{extra} and $S_{completion}$
- The union of sets $S_{success}$ and $S_{failure}$ is equal to the set $S_{completion}$
- No values are shared between the sets $S_{success}$ and $S_{failure}$

3.3.13 Initial State of Timeline Learner Success System

$$
\begin{array}{|l}
\hline
InitTimelineLearnerSuccess \\
TimelineLearnerSuccess \\
\hline
S_{extra} = \{\} \\
S_{completion} = \{\} \\
S_{success} = \{\} \\
S_{failure} = \{\} \\
\hline
\end{array}
$$

- The sets $S_{extra}, S_{completion}, S_{success}, S_{failure}$ are all initially empty

3.3.14 Filter for Completion

$$
\begin{array}{|l}
\hline
VerbIdCompletion \\
V_{completion} : Verb \\
\hline
V_{completion} = http://adlnet.gov/expapi/verbs/passed \mid \\
\qquad\qquad\quad https://w3id.org/xapi/dod-isd/verbs/answered \mid \\
\qquad\qquad\quad http://adlnet.gov/expapi/verbs/completed \\
\hline
\end{array}
$$

- The var $V_{completion}$ has a value of one of the above IRIs and is of type $Verb$

$$
\begin{array}{|l}
\hline
FilterForCompletion \\
\Delta TimelineLearnerSuccess \\
\hline
S'_{completion} = \{\, s : Statement \mid V_{completion} \in s \land s \in S\} \\
S'_{extra} = \{\, s : Statement \mid V_{completion} \notin s \land s \in S\} \\
\hline
\end{array}
$$

- The updated set $S'_{completion}$ is the set of all statements s where $V_{completion}$ is in s and s is in S
- the updated set S'_{extra} is the set of all statements s where $V_{completion}$ is not in s and s is in s

3.3.15 Filter for Success

$$
\begin{array}{|l}
\hline
ResultSuccessTrue \\
\hline
R_{successful} : Result \\
\hline
R_{successful} = true \\
R_{successful} \neq false \\
\hline
\end{array}
$$

- The var $R_{successful}$ has a value of $true$ but not $false$ and is of the type $Result$

$$
\begin{array}{|l}
\hline
FilterForSuccess \\
\hline
\Delta TimelineLearnerSuccess \\
s_{completion} : Statement \\
\hline
s_{completion} \in S_{completion} \\
S'_{success} = \{ s_{completion} : Statement \mid R_{successful} \in s_{completion} \} \\
S'_{failure} = \{ s_{completion} : Statement \mid R \notin s_{completion} \} \\
\hline
\end{array}
$$

- The set $s_{completion}$ is of type $Statement$ and is in the set $S_{completion}$
- The updated set $S'_{success}$ is the set of all statements $s_{completion}$ where $R_{successful}$ is in $s_{completion}$
- The updated set $S'_{failure}$ is the set of all statements $s_{completion}$ where $R_{successful}$ is not in $s_{completion}$

3.3.16 Return

$$
\begin{array}{|l}
\hline
Return \\
\hline
\Xi TimelineLearnerSuccess \\
S_{success}! : Statements \\
\hline
S_{success}! = S_{success} \\
\hline
\end{array}
$$

- The returned variable $S_{success}!$ is equal to the current state of variable $S_{success}$

3.4 Pseudocode

Algorithm 1: Timeline of Learner Success

Input: S
Result: $S_{success}$
while S *is not empty* **do**
  for each Statement s in S
  if $s.verb.id = V_{completion}$ **then**
    add s to $S_{completion}$
  else
    add s to S_{extra}
  end
end
while $S_{completion}$ *is not empty* **do**
  for each Statement $s_{completion}$ in $S_{completion}$
  if $s_{completion}.result.success = R_{success}$ **then**
    add $s_{completion}$ to $S_{success}$
  else
    add $s_{completion}$ to $S_{failure}$
  end
end

3.5 Result JSON Schema

JSON schema describing the returned data structure

3.6 Visualization Description

description of the associated visualization in english

3.7 Visualization prototype

This section will be updated to a prototype viz