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 compatable with the xAPI Z notation

- $.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

$IFI ::= mbox \,|\, mbox\_sha1sum \,|\, openid \,|\, account$

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

$OBJECTTYPE ::= Agent \,|\, Group \,|\, SubStatement \,|\, StatementRef \,|\, Activity$

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

$INTERACTIONTYPE ::= true{-}false \,|\, choice \,|\, fill{-}in \,|\, long{-}fill{-}in \,|\, matching \,|\, performance \,|\, sequencing \,|\, likert \,|\, numeric \,|\, other$

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

$INTERACTIONCOMPONENT ::= choices \,|\, scale \,|\, source \,|\, target \,|\, steps$

- 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

$CONTEXTTYPES ::= parent \,|\, grouping \,|\, category \,|\, other$

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

$[STATEMENT]$

- Basic types for the results of querying an LRS

$[AGENT, GROUP]$

- Basic types for Agents and collections of Agents

3.3.2 Id Schema

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

- 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

$$
\begin{array}{l}
\underline{\hspace{0.3cm}Agent\hspace{0.3cm}} \\
agent : AGENT \\
objectType : OBJECTTYPE \\
name : \mathbb{F}_1 \, \#1 \\
ifi : IFI \\
\hline
objectType = Agent \\
agent = \{ifi\} \cup \mathbb{P}\{name, objectType\}
\end{array}
$$

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

$$
\begin{array}{l}
\underline{\hspace{0.3cm}Member\hspace{0.3cm}} \\
Agent \\
member : \mathbb{F}_1 \\
\hline
member = \{a : AGENT \mid \forall a : a_0..a_n \bullet a = agent\}
\end{array}
$$

- 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*

$$
\begin{array}{l}
\underline{\hspace{0.3cm}Group\hspace{0.3cm}} \\
Member \\
group : GROUP \\
objectType : OBJECTTYPE \\
ifi : IFI \\
\hline
name : \mathbb{F}_1 \, \#1 \\
objectType = Group \\
group = \{objectType, name, member\} \vee \{objectType, member\} \vee \\
\qquad \{objectType, ifi\} \cup \mathbb{P}\{name, member\}
\end{array}
$$

- 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*

$$
\begin{array}{l}
\underline{\hspace{0.3cm}Actor\hspace{0.3cm}} \\
Agent \\
Group \\
actor : AGENT \vee GROUP \\
\hline
actor = agent \vee group
\end{array}
$$

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

3.3.4 Verb Schema

```
┌─ Verb ─────────────────────────────────────────────
│ Id
│ display, verb : $\mathbb{F}_1$
├────────────────────────────────────────────────────
│ verb = {id, display} ∨ {id}
└────────────────────────────────────────────────────
```

- 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

```
┌─ Extensions ──────────────────────────────────────────────
│ extensions, extensionVal : $\mathbb{F}_1$
│ extensionId : $\mathbb{F}_1 \#1$
├───────────────────────────────────────────────────────────
│ extensions = {e : (extensionId, extensionVal) | ∀i, j : e_i..e_j •
│              (extensionId_i, extensionVal_i) ∨ (extensionId_i, extensionVal_j)∧
│              (extensionId_j, extensionVal_i) ∨ (extensionId_j, extensionVal_j)∧
│              extensionId_i ≠ extensionId_j}
└───────────────────────────────────────────────────────────
```

$$extensions = \{e : (extensionId, extensionVal) \mid \forall i, j : e_i..e_j \bullet$$
$$(extensionId_i, extensionVal_i) \vee (extensionId_i, extensionVal_j) \wedge$$
$$(extensionId_j, extensionVal_i) \vee (extensionId_j, extensionVal_j) \wedge$$
$$extensionId_i \neq extensionId_j\}$$

- 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

```
┌─ InteractionActivity ──────────────────────────────────────────────
│ interactionType : INTERACTIONTYPE
│ correctResponsePattern : $seq_1$
│ interactionComponent : INTERACTIONCOMPONENT
├────────────────────────────────────────────────────────────────────
│ interactionActivity = {interactionType, correctReponsePattern, interactionComponent}∨
│                        {interactionType, correctResponsePattern}
└────────────────────────────────────────────────────────────────────
```

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

```
┌─ Definition ─────────────────────────────────────────────────
│ InteractionActivity
│ Extensions
│ definition, name, description : 𝔽₁
│ type, moreInfo : 𝔽₁ #1
├──────────────────────────────────────────────────────────────
│ definition = ℙ₁{name, description, type, moreInfo, extensions, interactionActivity}
└──────────────────────────────────────────────────────────────
```

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

```
┌─ Object ─────────────────────────────────────────────────────
│ Id
│ Definition
│ Agent
│ Group
│ Statement
│ objectTypeA, objectTypeS, objectTypeSub, objectType : OBJECTTYPE
│ substatement : STATEMENT
│ object : 𝔽₁
├──────────────────────────────────────────────────────────────
│ substatement = statement
│ objectTypeA = Activity
│ objectTypeS = StatementRef
│ objectTypeSub = SubStatement
│ objectType = objectTypeA ∨ objectTypeS
│ object = {id} ∨ {id, objectType} ∨ {id, objectTypeA, definition}
│          ∨{id, definition} ∨ {agent} ∨ {group} ∨ {objectTypeSub, substatement}
│          ∨{id, objectTypeA}
└──────────────────────────────────────────────────────────────
```

- 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

6

```
┌─ Score ─────────────────────────────────────────────
│ score : 𝔽₁
│ scaled, min, max, raw : ℤ
├─────────────────────────────────────────────────────
│ scaled = {n : ℤ | −1.0 ≤ n ≤ 1.0}
│ min = n < max
│ max = n > min
│ raw = {n : ℤ | min ≤ n ≤ max}
│ score = ℙ₁{scaled, raw, min, max}
└─────────────────────────────────────────────────────
```

- 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 : 𝔽₁ #1
│ result : 𝔽₁
├─────────────────────────────────────────────────────
│ success = {true} ∨ {false}
│ completion = {true} ∨ {false}
│ result = ℙ₁{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 ∨ GROUP
├─────────────────────────────────────────────────────
│ instructor = agent ∨ 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*

$\boxed{\begin{array}{l} \underline{\textit{Context}} \rule{300pt}{0.4pt} \\ \textit{Instructor} \\ \textit{Team} \\ \textit{Object} \\ \textit{Extensions} \\ registration, revision, platform, language : \mathbb{F}_1 \,\#1 \\ parentT, groupingT, categoryT, otherT : CONTEXTTYPES \\ contextActivities, statement : \mathbb{F}_1 \\ \rule{200pt}{0.4pt} \\ statement = object \setminus (id, objectType, agent, group, definition) \\ parentT = parent \\ groupingT = grouping \\ categoryT = category \\ otherT = other \\ contextActivity = \{ca : object \setminus (agent, group, objectType, objectTypeSub, substatement)\} \\ contextActivityParent = (parentT, contextActivity) \\ contextActivityCategory = (categoryT, contextActivity) \\ contextActivityGrouping = (groupingT, contextActivity) \\ contextActivityOther = (otherT, contextActivity) \\ contextActivities = \mathbb{P}_1\{contextActivityParent, contextActivityCategory, \\ \qquad\qquad\qquad contextActivityGrouping, contextActivityOther\} \\ context = \mathbb{P}_1\{registration, instructor, team, contextActivities, revision, \\ \qquad\qquad\quad platform, language, statement, extensions\} \end{array}}$

- 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

$\boxed{\begin{array}{l} \underline{\textit{Timestamp}} \rule{180pt}{0.4pt} \\ timestamp : \mathbb{F}_1 \,\#1 \end{array}}$

$\boxed{\begin{array}{l} \underline{\textit{Stored}} \rule{180pt}{0.4pt} \\ stored : \mathbb{F}_1 \,\#1 \end{array}}$

- 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

```
┌─ Attachments ─────────────────────────────────────────────
│ display, description, attachment, attachments : 𝔽₁
│ usageType, sha2, fileUrl, contexntType : 𝔽₁ #1
│ length : ℕ
├──────────────────────────────────────────────────────────
│ attachment = {usageType, display, contentType, length, sha2} ∪ ℙ{description, fileUrl}
│ attachments = {a : attachment}
└──────────────────────────────────────────────────────────
```

3.3.10 Statement and Statements Schema

```
┌─ Statement ───────────────────────────────────────────────
│ Id
│ Actor
│ Verb
│ Object
│ Result
│ Context
│ Timestamp
│ Stored
│ Attachements
│ statement, $ : STATEMENT
├──────────────────────────────────────────────────────────
│ statement = {actor, verb, object, stored}∪
│             ℙ{id, result, context, timestamp, attachments}
│ $ ↝ statement
└──────────────────────────────────────────────────────────
```

- The schema *Statement* introduces the component *statement* which consists of the components *actor*, *verb*, *object* and *stored* and the optional components *id*, *result*, *context*, *timestamp*, and/or *attachments*

- The schema *Statement* also binds the component *statement* to the variable $ so that JSONPath can be used within Operation schemas which require reaching into a *statement*. This is accomplished by using the . (select) notation starting at $ (root) and navigating into subsequent components of the *statement*

```
┌─ Statements ──────────────────────────────────────────────
│ Statement
│ statements : 𝔽
├──────────────────────────────────────────────────────────
│ statements = {s : statement}
└──────────────────────────────────────────────────────────
```

- The schema *Statements* introduces the component *statements* which is a non-empty finite set of components *statement*

3.4 Timeline Learner Success Z Specifications

The following Z Schemas define the system state, initialization and operations necessary to perform the Timeline Learner Success Algorithm

3.4.1 Timeline Leaner Success Basic Types

$COMPLETION :==$
$\{http : //adlnet.gov/expapi/verbs/passed\} \mid$
$\{https : //w3id.org/xapi/dod - isd/verbs/answered\} \mid$
$\{http : //adlnet.gov/expapi/verbs/completed\}$

$SUCCESS :== \{true\}$

3.4.2 Timeline Leaner Success System State

$$
\begin{array}{|l}
\hline\ TimelineLearnerSuccess \underline{} \\
\ Statements \\
\ S_{all} : \mathbb{F}_1 \\
\ S_{completion}, S_{success} : \mathbb{F} \\
\hline
\ S_{all} = statements \\
\ S_{completion} \subseteq S_{all} \\
\ S_{success} \subseteq S_{completion} \\
\hline
\end{array}
$$

- The set S_{all} is a non-empty finite set and is the component *statements*

- The sets $S_{completion}$ and $S_{success}$ are both finite sets

- the set $S_{completion}$ is a subset of S_{all}

- the set $S_{success}$ is a subset of $S_{completion}$

3.4.3 Initial State of Timeline Learner Success System

$$
\begin{array}{|l}
\hline\ InitTimelineLearnerSuccess \underline{} \\
\ TimelineLearnerSuccess \\
\hline
\ S_{all} \neq \emptyset \\
\ S_{completion} = \emptyset \\
\ S_{success} = \emptyset \\
\hline
\end{array}
$$

- The set S_{all} is a non-empty set

- The sets $S_{completion}, S_{success}$ are both initially empty

3.4.4 Filter for Completion

```
┌─ Completion ─────────────────────────────────────────────────┐
│ Statement                                                    │
│ completion : STATEMENT ⇸ F                                  │
│ s? : STATEMENT                                               │
│ s! : F                                                       │
├──────────────────                                            │
│ s? = statement                                               │
│ s! = completion(s?)                                          │
│ completion(s?) = if s?.verb.id : COMPLETION then s! = s? else s! = ∅ │
└──────────────────────────────────────────────────────────────┘
```

- The schema *Completion* inroduces the function *completion* which takes in the variable $s?$ and returns the variable $s!$

- The variable $s?$ is the component *statement*

- $s!$ is equal to $s?$ if \$.*verb.id* is of the type *COMPLETION* otherwise $s!$ is an empty set

```
┌─ FilterForCompletion ────────────────────────────────────────┐
│ ΔTimelineLearnerSuccess                                      │
│ Completion                                                   │
│ completions : F                                              │
├──────────────────                                            │
│ completions = S_all                                          │
│ completions' = {s : STATEMENT | completion(s) ≠ ∅}          │
│ S'_completion = S_completion ∪ completions'                 │
└──────────────────────────────────────────────────────────────┘
```

- the set *completions* is the set S_{all}

- The set *completions'* is the set of all statements s where the result of $completion(s)$ is not an empty set

- the updated set $S'_{completion}$ is the union of the previous state of set $S_{completion}$ and the set *completions*

3.4.5 Filter for Success

```
┌─ Success ────────────────────────────────────────────────────┐
│ Statement                                                    │
│ success : STATEMENT ⇸ F                                     │
│ s? : STATEMENT                                               │
│ s! : F                                                       │
├──────────────────                                            │
│ s? = statement                                               │
│ s! = success(s?)                                             │
│ success(s?) = if s?.result.success : SUCCESS then s! = s? else s! = ∅ │
└──────────────────────────────────────────────────────────────┘
```

- the schema *Success* introduces the function *success* which takes in the variable $s?$ and returns the variable $s!$

- the variable $s?$ is the component *statement*

- $s!$ is equal to $s?$ if $\$.result.success$ is of the type $SUCCESS$ otherwise $s!$ is an empty set

$$
\begin{array}{|l}
\hline
\_FilterForSuccess _____ \\
\Delta TimelineLearnerSuccess \\
Success \\
successes : \mathbb{F} \\
\hline
successes = S_{completion} \\
successes' = \{s : STATEMENT \mid success(s) \neq \emptyset\} \\
S'_{success} = S_{success} \cup successes \\
\hline
\end{array}
$$

- the set *successes* is the set $S_{completion}$

- The set $successes'$ contains elements s of type $STATEMENT$ where $success(s)$ is not an empty set

- The updated set $S'_{success}$ is the union of the previous state of $S_{success}$ and *successes*

3.4.6 Sequence of Operations

$FilterStatements \mathrel{\widehat{=}} FilterForCompletion \fatsemi FilterForSuccess$

- The schema *FilterStatements* is the sequential composition of operation schemas *FilterForCompletion* and *FilterForSuccess*

- *FilterForCompletion* happens before *FilterForSuccess*

3.4.7 Return

$$
\begin{array}{|l}
\hline
\_Return _____ \\
\Xi TimelineLearnerSuccess \\
FilterStatements \\
S_{success}! : \mathbb{F} \\
\hline
S_{success}! = S_{success} \\
\hline
\end{array}
$$

- The returned variable $S_{success}!$ is equal to the current state of variable $S_{success}$ after the operations *FilterForCompletion* and *FilterForSuccess*

12

3.5 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

3.6 Result JSON Schema

JSON schema describing the returned data structure

3.7 Visualization Description

description of the associated visualization in english

3.8 Visualization prototype

This section will be updated to a prototype viz