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 evaluate a learner's understanding of material must report if the learner was successful or not
  - the grade earned by the learner must be reported
  - the minimum and maximum possible grade must 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

- $\$.timestamp$
- $\$.result.success$
- $\$.result.score.raw$
- $\$.result.score.min$
- $\$.result.score.max$
- $\$.verb.id$

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

$[STATEMENT]$

- Basic types for the results of querying an LRS

$[AGENT, GROUP]$

- Basic types for Agents and collections of Agents

3.3.2 Id Schema

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

- 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

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

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

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

- 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*

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

- 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*

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

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

3.3.4 Verb Schema

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\;Verb\;}}\\
Id\\
display, verb : \mathbb{F}_1\\
\hline
verb = \{id, display\} \vee \{id\}
\end{array}
$$

- 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

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

- 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

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\;InteractionActivity\;}}\\
interactionType : INTERACTIONTYPE\\
correctResponsePattern : \mathrm{seq}_1\\
interactionComponent : INTERACTIONCOMPONENT\\
\hline
interactionActivity = \{interactionType, correctReponsePattern, interactionComponent\} \vee\\
\quad \{interactionType, correctResponsePattern\}
\end{array}
$$

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

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

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

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\ Object\ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
Id \\
Definition \\
Agent \\
Group \\
Statement \\
objectTypeA, objectTypeS, objectTypeSub, objectType : OBJECTTYPE \\
substatement : STATEMENT \\
object : \mathbb{F}_1 \\
\hline
substatement = statement \\
objectTypeA = Activity \\
objectTypeS = StatementRef \\
objectTypeSub = SubStatement \\
objectType = objectTypeA \vee objectTypeS \\
object = \{id\} \vee \{id, objectType\} \vee \{id, objectTypeA, definition\} \\
\qquad\qquad \vee\{id, definition\} \vee \{agent\} \vee \{group\} \vee \{objectTypeSub, substatement\} \\
\qquad\qquad \vee\{id, objectTypeA\} \\
\hline
\end{array}
$$

- 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

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

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

$$
\begin{array}{|l}
\hline
\textit{Result} \\
Score \\
Extensions \\
success, completion, response, duration : \mathbb{F}_1 \#1 \\
result : \mathbb{F}_1 \\
\hline
success = \{true\} \vee \{false\} \\
completion = \{true\} \vee \{false\} \\
result = \mathbb{P}_1\{score, success, completion, response, duration, extensions\} \\
\hline
\end{array}
$$

- 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

$$
\begin{array}{|l}
\hline
\textit{Instructor} \\
Agent \\
Group \\
instructor : AGENT \vee GROUP \\
\hline
instructor = agent \vee group \\
\hline
\end{array}
$$

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

$$
\begin{array}{|l}
\hline
\textit{Team} \\
Group \\
team : GROUP \\
\hline
team = group \\
\hline
\end{array}
$$

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

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ Context\ \hspace{25em}}} \\
Instructor \\
Team \\
Object \\
Extensions \\
registration, revision, platform, language : \mathbb{F}_1\ \#1 \\
parentT, groupingT, categoryT, otherT : CONTEXTTYPES \\
contextActivities, statement : \mathbb{F}_1 \\
\hline
statement = object \setminus (id, objectType, agent, group, definition) \\
parentT = parent \\
groupingT = grouping \\
categoryT = category \\
otherT = other \\
contextActivity = \{ca : object \setminus (agent, group, objectType, objectTypeSub, substatement)\} \\
contextActivityParent = (parentT, contextActivity) \\
contextActivityCategory = (categoryT, contextActivity) \\
contextActivityGrouping = (groupingT, contextActivity) \\
contextActivityOther = (otherT, contextActivity) \\
contextActivities = \mathbb{P}_1\{contextActivityParent, contextActivityCategory, \\
\qquad\qquad contextActivityGrouping, contextActivityOther\} \\
context = \mathbb{P}_1\{registration, instructor, team, contextActivities, revision, \\
\qquad\qquad platform, language, statement, extensions\} \\
\hline
\end{array}
$$

- 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

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ Timestamp\ \hspace{25em}}} \\
timestamp : \mathbb{F}_1\ \#1 \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ Stored\ \hspace{25em}}} \\
stored : \mathbb{F}_1\ \#1 \\
\hline
\end{array}
$$

- 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

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

3.3.10 Statement and Statements Schema

$$
\begin{array}{|l}
\hline
\textit{Statement} \\
Id \\
Actor \\
Verb \\
Object \\
Result \\
Context \\
Timestamp \\
Stored \\
Attachements \\
statement, \$: STATEMENT \\
\hline
statement = \{actor, verb, object, stored\} \cup \\
\qquad \mathbb{P}\{\mathrm{id}, result, context, timestamp, attachments\} \\
\hline
\end{array}
$$

- 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* allows for subcomponent of *statement* to refrenced via the . (selection) operator

$$
\begin{array}{|l}
\hline
\textit{Statements} \\
Statement \\
statements : \mathbb{F} \\
\hline
statements = \{s : statement\} \\
\hline
\end{array}
$$

- 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

$$
\begin{array}{|l}
\multicolumn{1}{l}{TimelineLearnerSuccess} \\
\hline
Statements \\
S_{all} : \mathbb{F}_1 \\
S_{completion}, S_{success}, S_{processed} : \mathbb{F} \\
\hline
S_{all} = statements \\
S_{completion} \subseteq S_{all} \\
S_{success} \subseteq S_{completion} \\
S_{processed} = \{pair : (statement.timestamp, \mathbb{N}\#1)\} \\
\hline
\end{array}
$$

- 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 $S_{completion}$
- the set $S_{processed}$ is a finite set of pairs where each contains a *statement.timestamp* and a natural number

3.4.3 Initial State of Timeline Learner Success System

$$
\begin{array}{|l}
\multicolumn{1}{l}{InitTimelineLearnerSuccess} \\
\hline
TimelineLearnerSuccess \\
\hline
S_{all} \neq \emptyset \\
S_{completion} = \emptyset \\
S_{success} = \emptyset \\
S_{processed} = \emptyset \\
\hline
\end{array}
$$

- The set S_{all} is a non-empty set
- The sets $S_{completion}, S_{success}$ and $S_{processed}$ are all initially empty

3.4.4 Filter for Completion

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Completion}} \\
\hline
Statement \\
completion : STATEMENT \rightarrowtail \mathbb{F} \\
s? : STATEMENT \\
s! : \mathbb{F} \\
\hline
s? = statement \\
s! = completion(s?) \\
completion(s?) = \textbf{if } s?.verb.id : COMPLETION \textbf{ then } s! = s? \textbf{ else } s! = \emptyset \\
\hline
\end{array}
$$

- 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

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{FilterForCompletion}} \\
\hline
\Delta TimelineLearnerSuccess \\
Completion \\
completions : \mathbb{F} \\
\hline
completions = S_{all} \\
completions' = \{s : STATEMENT \mid completion(s) \neq \emptyset\} \\
S'_{completion} = S_{completion} \cup completions' \\
\hline
\end{array}
$$

- 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

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Success}} \\
\hline
Statement \\
success : STATEMENT \rightarrowtail \mathbb{F} \\
s? : STATEMENT \\
s! : \mathbb{F} \\
\hline
s? = statement \\
s! = success(s?) \\
success(s?) = \textbf{if } s?.result.success : SUCCESS \textbf{ then } s! = s? \textbf{ else } s! = \emptyset \\
\hline
\end{array}
$$

- 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

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

- 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*

$FilterStatements \mathrel{\widehat=} FilterForCompletion \mathbin{;} FilterForSuccess$

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

3.4.6 Processes Results

$$
\begin{array}{|l}
\textit{Scale} \\
\hline
\textit{scaled}! : \mathbb{N} \\
\textit{raw}?, \textit{min}?, \textit{max}? : \mathbb{Z} \\
\textit{scale} : \mathbb{Z} \rightarrow \mathbb{N} \\
\hline
\textit{scaled}! = \textit{scale}(\textit{raw}?, \textit{min}?, \textit{max}?) \\
\textit{scale}(\textit{raw}?, \textit{min}?, \textit{max}?) = (\textit{raw}? * ((0.0 - 100.0)\ \textit{div}\ (\textit{min}? - \textit{max}?))) + \\
\qquad (0.0 - (\textit{min}? * ((0.0 - 100.0)\ \textit{div}\ (\textit{min}? - \textit{max}?)))) \\
\hline
\end{array}
$$

- The schema *Scale* introduces the function *scale* which takes 3 arguments, $raw?$, $min?$ and $max?$. The function converts $raw?$ from the range $min? .. max?$ to 0.0..100.0

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad ProcessStatements \quad}} \\
\Delta TimelineLearnerSuccess \\
Scale \\
FilterStatements \\
processed : \mathbb{F} \\
\hline
processed = S_{success} \\
processed' = \{p : (\mathbb{F}_1 \#1, \mathbb{N}\#1) \mid \\
\qquad \textbf{let } \{processed_i..processed_j\} == \{s_i..s_j\} \bullet \\
\qquad \forall i, j : s_i..s_j \bullet \exists p_i..p_j \bullet \\
\qquad first\, p_i = s_i.timestamp \land \\
\qquad second\, p_i = scale(s_i.result.score.raw, s_i.result.score.min, s_i.result.score.max) \land \\
\qquad first\, p_j = s_j.timestamp \land \\
\qquad second\, p_j = scale(s_j.result.score.raw, s_j.result.score.min, s_j.result.score.max)\} \\
S'_{processed} = S_{processed} \cup processed' \\
\hline
\end{array}
$$

- The operation $ProcessStatements$ introduces the variable $processed$ which is equalivant to the set $S_{success}$ which is the result of the operation $FilterStatements$
- The operation defines the variable $processsed'$ which is a set of objects p which are each an ordered pair of 1) a finite set containing one value and 2) a single positive number.
- The first component of every object p, is the timestamp from the associated *statement* within *processed* ie. $s.timestamp$
- The second component of every object p is the result of the function $scale$. The score values contained within the associated *statement* s are the arugments passed to $scale$. ie $scale(s.result.score.raw, s.result.score.min, s.result.score.max)$
- The result of the operation $ProcessStatements$ is to updated the set $S_{processed}$ with the values contained within $processed'$

3.4.7 Sequence of Operations

$ProcessedStatements \mathrel{\widehat=} FilterStatements \mathbin{⨾} ProcessStatements$

- The schema $ProcessedStatements$ is the sequential composition of operation schemas $FilterStatements$ and $ProcessStatements$
- $FilterStatements$ happens before $ProcessStatements$

3.4.8 Return

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad Return \quad}} \\
\Xi TimelineLearnerSuccess \\
ProcessedStatements \\
S_{processed}! : \mathbb{F} \\
\hline
S_{processed}! = S_{processed} \\
\hline
\end{array}
$$

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

3.5 Pseudocode

Algorithm 1: Timeline of Learner Success

```
Input: S_all
Result: coll
coll := []
while S_all is not empty do
    for each statement s in S_all
    if s.verb.id = COMPLETION then
        add s to S_completion
    else
        noop
    end
end
while S_completion is not empty do
    for each statement sc in S_completion
    if sc.result.success = SUCCESS then
        add sc to S_success
    else
        noop
    end
end
while S_success is not empty do
    for each statement ss in S_success
        let ss.result.score.raw = raw?
            ss.result.score.max = max?
            ss.result.score.min = min?
            scaled = scale(raw?, min?, max?)
        concat coll [ss.timestamp, scaled]
end
```

- The Z schemas are used within this pseudocode
- The return value coll is an array of arrays, each containing a timestamp and a scaled score.

3.6 JSON Schema

```
{"type":"array",
   "items":{"type":"array",
      "items":[{"type":"string"}, {"type":"number"}]}
}
```

3.7 Visualization Description

The Timeline of Learner Success visualization will be a line chart where the domain is time and the range is score on a scale of 0.0 to 100.0. Every array within the array returned by the algorithm will be a point on the chart. The domain of the graph should be in chronological order.

Aditional features may be implemented on top of this base specification but would require adding aditional values to each sub-array returned by the algorithm. These values would only be limtied by the fields contained within the xAPI statements being used to populate the visualization. ie.

- A tooltip containing the name of an activity when hovering over a specific point on the chart

  - this would require adding the value of $\$.object.definition.name$ to each subarray
- A tooltip containing the device on which the activity was experienced

  - this would require adding the value from $\$.context.platform$ to each subarray
- A tooltip containing the instructor associated with a particular data point

  - this would require adding the value from $\$.context.instructor$ to each subarray

3.8 Visualization prototype