DAVE Framework Learning Analytics Algorithms

Yet Analytics

July 26, 2018

Introduction

This document introduces the initial learning analytics algorithms, **timeline of learner success** and **which assessment questions are the most difficult** of the DAVE framework. This document will be updated to include the remaining learning analytics questions defined within the 2018 TLA Data Requirements document in addition to other learning analytics algorithms which have yet to be defined. Updates may also address refinment of these algorithms and this document should be understood to be an example of algorithm presentation and not the final state of any defined algorithm.

The structure of this documents is as follows:

1. A formal specification for the data standard xAPI written in Z and referenced within the formal specifications of learning analytics algorithms
2. An algorithm definition which will consist of:
    (a) an introduction for the algorithm
    (b) the structure of the ideal input data
    (c) how to retrieve input data from an LRS
    (d) the statement parameters which the algorithm will utilize
    (e) any issues with the data collected during the 2018 pilot test of the TLA
    (f) a summary of the algorithm
    (g) the formal specification of the algorithm
    (h) pseudocode representation of the algorithm
    (i) JSONSchema for the output of the algorithm
    (j) a description of the associated visualization
    (k) a prototype of the visualization
    (l) a collection of suggestions describing how the algorithm could be adapted to improve the quality of the visualization prototype

1 xAPI Formal Specification

The current formal specification only defines xAPI statements abstractly within the context of Z. A concrete definition for xAPI statements it outside the scope of this document.

1.1 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 type for the results of querying an LRS

$[AGENT, GROUP]$

- Basic types for Agents and collections of Agents

1.2 Id Schema

```
┌─ Id ──────────────────────
│ id : 𝔽₁ #1
└───────────────────────────
```

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

1.3 Schemas for Agents, Groups and Actors

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

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

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Member}} \\ \hline
Agent \\
member : \mathbb{F}_1 \\ \hline
member = \{a : AGENT \mid \forall a : a_0..a_n \bullet a = agent\} \\ \hline
\end{array}$$

- 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*

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

- 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*

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Actor}} \\ \hline
Agent \\
Group \\
actor : AGENT \vee GROUP \\ \hline
actor = agent \vee group \\ \hline
\end{array}$$

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

1.4 Verb Schema

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

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

1.5 Object Schema

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

- 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

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

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

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

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

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

- 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

1.6 Result Schema

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

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

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

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

1.7 Context Schema

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

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

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

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

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

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

1.8 Timestamp and Stored Schema

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

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

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

1.9 Attachements Schema

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

- The schema *Attachements* introduces the componenet *attachements* which is a non-empty finite set of the component *attachement*
- The component *attachment* is a non-empty finite set of the componenets *usageType*, *display*, *contentType*, *length*, *sha*2 with optionally *description* and/or *fileUrl*

1.10 Statement and Statements Schema

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

- 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

$$
\begin{array}{|l}
\hline
\mathit{Statements} \\
Statement \\
statements : \mathbb{F}_1 \\
\hline
statements = \{s : statement\} \\
\hline
\end{array}
$$

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

2 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.

2.1 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 statements
- 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 statements
- 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

2.2 Input Data Retrieval

How to query an LRS via a GET request to the Statements Resource [1] [2] [3]

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

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

[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] If you want to query across the entire history of a LRS, omit Since and Until

```
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
```

2.3 Statement Parameters to Utilize

The statement parameter locations here are written in JSONPath. This notation is also compatable with the xAPI Z notation

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

2.4 2018 Pilot TLA Statement Problems

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

2.5 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. process the filtered data

- extract the timestamp
- extract the score values from the statement and convert them to a scale of 0..100
- create a pair of [timestamp, scaled-score]

2.6 Formal Specification

2.6.1 Basic Types

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

$SUCCESS ::== \{true\}$

2.6.2 System State

$TimelineLearnerSuccess$

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

- 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

2.6.3 Initial System State

$InitTimelineLearnerSuccess$

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

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

2.6.4 Filter for Completion

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

- 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

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

- 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*

2.6.5 Filter for Success

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

- 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 $\$.\textit{result}.\textit{success}$ is of the type *SUCCESS* otherwise $s!$ is an empty set

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

- the set *successes* is the set $S_{completion}$
- The set $\textit{successes}'$ contains elements s of type *STATEMENT* where $\textit{success}(s)$ is not an empty set
- The updated set $S'_{success}$ is the union of the previous state of $S_{success}$ and *successes*

2.6.6 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

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

- 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 aruguments 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 $processed'$

2.6.7 Sequence of Operations

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

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

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

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

2.6.8 Return

```
Return
  $\Xi TimelineLearnerSuccess$
  $ProcessedStatements$
  $S_{processed}! : \mathbb{F}$
  ----------
  $S_{processed}! = S_{processed}$
```

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

2.7 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.

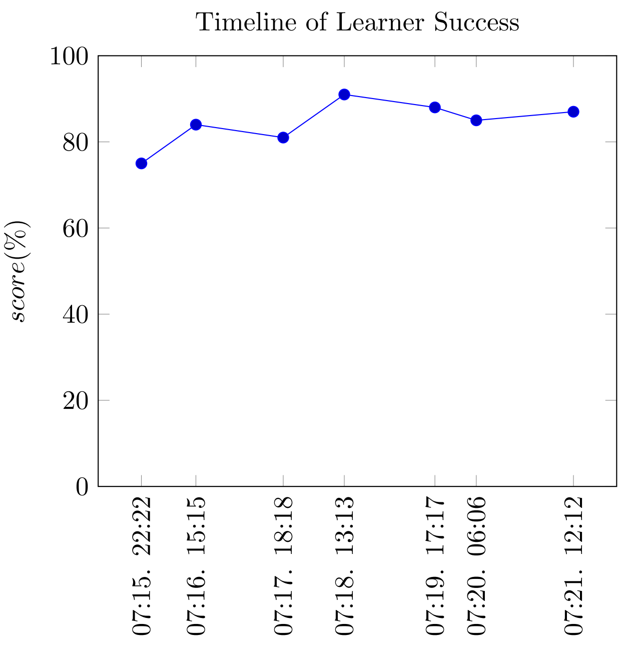
2.8 JSON Schema

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

2.9 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.

2.10 Visualization prototype



2.11 Prototype Improvement Suggestions

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. Examples of fields which could be used to populate the visualization include but are not limited to:

- 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 Which Assessment Questions are the Most Difficult

As learners engage in a blended eLearning ecosystem, they will experiecne teaching content as well as assessment content. Assessments are designed to measure the effectiveness of learning content and help measure learning. It is possible that certain assessment questions do not accurately represent the concepts contained within teaching material and this may be indicated by a majority of learners getting the question wrong. It is also possible that the question accurately represents the learning content but is just hard. The following algorithm will identify these questions but will not be able to deduce the why learners answer them incorrectly.

3.1 Ideal Statements

In order to accurately determine which assessment questions are the most dificult, there are a few requirements of the data produced by a LRP. They are as follows:

- statements describing a learner answering a question must report if the learner got the question correct or incorrect via $\$.result.success$
- if it is possible to get partial credit on a question, the amount of credit should be reported within the statement
  - the credit earned by the learner should be reported within $\$.result.score.raw$
  - the minimum and maximum possible credit amount should be reported within $\$.result.score.min$ and $\$.result.score.max$ respectively
- If it is possible to get partial credit on a question, it must still be reported if the learner reached the threshold of success via $\$.result.success$

- Statements describing a learner answering a question should contain activities of the type *cmi.interaction*
- activities must be uniquely and consistently identified across all statements
- Statements describing a learner answering a question should[4] use the verb *http : //adlnet.gov/expapi/verbs/answered*
- 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

3.2 Input Data Retrieval

How to query an LRS via a GET request to the Statements Resource [5] [6] [7]

```
Verb = "verb=http://adlnet.gov/expapi/verbs/answered"

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

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

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

endpoint = Base + Verb + "&" + 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
```

[4] it is possible to use another verb but if another is used, that will need to be accounted for in data retrieval
[5] See footnote 1.
[6] See footnote 2.
[7] See footnote 3.

3.3 Statement Parameters to Utilize

3.4 2018 Pilot TLA Statement Problems

3.5 Summary

3.6 Formal Specification

3.6.1 Basic Types

3.6.2 System State

3.6.3 Initial System State

3.6.4 Filter for Failure

3.6.5 Processes Results

3.6.6 Sequence of Operations

3.6.7 Return

3.7 Pseudocode

3.8 JSON Schema

3.9 Visualization Description

3.10 Visualization prototype

3.11 Prototype Improvement Suggestions