1 How Often are Recommendations Followed

As learners engage in activities supported by a learning ecosystem, they will build up a history of learning experiences. When the digital resources of that learning ecosystem adhere to a framework dedicated to supporting and understanding the learner, such as the Total Learning Architecture (TLA), it becomes possible to retell their learning story through data and data visualization. One important aspect of that story is the recommendations provided to the learner and whether or not the learner follows those recommendations.

1.1 Ideal Statements

In order to accurately determine if a learner is following recommendations, there are a few requirements of the data produced by a LRP and the recommender itself. They are as follows:

- Every time the recommender makes a recommendation, a statement should be produced which uses the verb *https : //w3id.org/xapi/dod−isd/verbs/recommended*[1] and has the recommended piece of content as the object.
  - the content should be uniquely and consistently identified across all statements.
- When a learner launches recommended content, the resulting launched statement should use the verb *http : //adlnet.gov/expapi/verbs/launched*[2] and contain a refrence to the recommened content statement within $.*context.statement*
  - Launching of content should use the above IRI regardless of why the content was launched
  - If it not possible to refrence the exact recommended content statement, the launch statement should have some indication that it was the result of a recommendation.[3]

[1] See footnote 4

[2] See footnote 4

[3] It is possible to determine if recommendations are followed (with some level of error) without this explicit linking of launched to recommended but this severly complicates the algorithm. In this case, in order to optmize for accuracy, the algorithm would need to consider the actor and their general activity within a session, the object of both launched and recommended statements generated within the session, the time lapse between recommendations and launches with a predefined lapse value which determines if a launch was close enough in time to a recommendation to be considered a result of the recommendation. An additonal constraint on the above case is the recommendation statements should contain a reference to to the person recieving the recommendation, otherwise determining the 1:1 relationships between recommendations and launches requires aditional complexity and will still not be 100% accurate due to the reliance on the time lapse value.

1.2 Input Data Retrieval

How to query an LRS via a GET request to the Statements Resource via curl.[4][5][6]

```
R = ”verb=https://w3id.org/xapi/dod-isd/verbs/recommended”
L = ”verb=http://adlnet.gov/expapi/verbs/launched”

Since = ”since=2018-07-20T12:08:47Z”
Until = ”until=2018-07-21T12:08:47Z”
Base = ”https://example.endpoint/statements?”

endpoint1 = Base + R + ”&” + Since + ”&” + Until
endpoint2 = Base + L + ”&” + Since + ”&” + Until

Auth = Hash generated from basic auth

SR = curl -X GET -H ”Authorization: Auth”
          -H ”Content-Type: application/json”
          -H ”X-Experience-API-Version: 1.0.3”
          endpoint1
SL = curl -X GET -H ”Authorization: Auth”
          -H ”Content-Type: application/json”
          -H ”X-Experience-API-Version: 1.0.3”
          endpoint2

S = SR + SL
```

1.3 Statement Parameters to Utilize

The statement parameter locations here are written in JSONPath. This notation is also compatable with the xAPI Z notation due to the defined hierarchy of components. Within the Z specifications, a variable name will be used instead of the $

- $*.verb.id*
- $*.context.statement*

1.4 2018 Pilot TLA Statement Problems

At the time of writing this document, launched statements do not include a statement reference or any indication of a connection between recommendations and launches. The authors of this document do not have access to the LRS containing the recommended statements and thus can not draw any conclusions

[4] footnote 1 applies to both S1 and S2.

[5] See footnote 2.

[6] See footnote 3.

about any issues which may be present within those statements or any aspects of those statements which may correlate them to launch statements. The following algorithm is going to assume that the input set of statements follow the guidlines outlined in section 5.1 as the additional algorithmic considerations brought on by non ideal statements, as specified within footnote 16, result in an algoirthm which is not optimal for near real time visualizations.

1.5 Summary

1. Query an LRS via a GET request to the statements endpoint using the paramters verb, since and until to gather all statements with the verb $http://adlnet.gov/expapi/verbs/launched$.

2. Query an LRS via a GET request to the statements endpoint using the paramters verb, since and until to gather all statements with the verb $https://w3id.org/xapi/dod-isd/verbs/recommended$.[7]

3. Group all collections of statements by a $TIMEUNIT$

4. seperate the collection of grouped launched statements into a collection of those which were the result of a recommendation and those which were not.

5. Take the count of all groups of statements
   - Recommended statements per $TIMEUNIT$
   - Launches due to recommendations per $TIMEUNIT$
   - Launches not due to recommendations per $TIMEUNIT$

6. Calculate summary statistics for the overall time range and per $TIMEUNIT$
   - Divide launches due to recommendations by the total number of launches to determine the percentage of launches due to recommendations
   - Divide launches due to recommendations by the total number of recommendations to determine the percentage of recommendations which are followed.

[7] If since and until are specified, they should be the same in both requests.

1.6 Formal Specification

1.6.1 System State

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{FollowedRecommendations}} \\
\textit{Statements} \\
\textit{CountPerGroup} \\
S_{recommended}, S_{launched} : \mathbb{F}_1 \\
ordered_L, ordered_R, grouped_{launched}, grouped_{recommended}, \\
onlyRecommended, cPerGroup_{launched}, cPerGroup_{recommended}, \\
cPerGroup_{followed}, combined : \mathrm{seq} \\
t_{start}, N_{launched}, N_{recommended}, N_{followed}, P_{followed}, P_{dueto} : \mathbb{N} \\
tr_{start}, tr_{end} : \mathbb{F} \\
unit? : TIMEUNIT \\
\hline
S_{recommended} = statements \\
S_{launched} = statements \\
combined = \langle (tr_{start}, tr_{end}, N_{launched}, N_{recommended}, N_{followed}, P_{followed}, P_{dueto}) \rangle \\
count(grouped_{launched}) = count(grouped_{recommended}) \\
count(onlyRecommended) = count(grouped_{launched}) \Rightarrow \\
count(onlyRecommended) = count(grouped_{recommended}) \\
count(cPerGroup_{launched}) = count(cPerGroup_{followed}) = count(cPerGroup_{recommended}) \\
\hline
\end{array}
$$

- $S_{recommended}, S_{launched}$ are both non-empty, finite sets.
  - $S_{recommended}$ and $S_{launched}$ contain the results of querying an LRS for recommended and launched statements respectively.
- $ordered_L, ordered_R, grouped_{launched}, grouped_{recommended}, onlyRecommended$, $cPerGroup_{launched}, cPerGroup_{recommended}, cPerGroup_{followed}$ and $combined$ are all finite sequences.
  - $ordered_L$ and $ordered_R$ are the sequences of statements within $S_{launched}$ and $S_{recommended}$ respectively and sorted by timestamp.
  - $grouped_{launched}$ is the result of grouping the statements within $ordered_L$ by $unit?$.
  - $grouped_{recommended}$ is the result of grouping the statements within $ordered_R$ by $unit?$.
  - $onlyRecommended$ is the result of filtering the statements within the sequence $grouped_{launched}$ to only include statements where $statement.context.statement$ is present
  - $cPerGroup_{launched}, cPerGroup_{recommended}, cPerGroup_{followed}$ are all sequences of numbers which represent the count within each subsequence of $grouped_{launched}$, $grouped_{recommended}$ and $onlyRecommended$ respectively.

  - *combined* is a sequence of ordered pairs where each pair consists of tr_{start}, tr_{end}, $N_{launched}$, $N_{recommended}$, $N_{followed}$, $P_{followed}$ and P_{dueto}

- $t_{start}, N_{launched}, N_{recommended}, N_{followed}, P_{followed}, P_{dueto}$ are all natural numbers

- tr_{start}, tr_{end} are both timestamps which represent the the start and end of the time range for each a group of statements.

- *unit*? is an input representing a time interval, ie day vs month vs hour.

- all sequences are the same length so that each subsequence represents the same time grouping. In other words, indexes are comparable across sequences.

1.6.2 Initial System State

$InitFollowedRecommendations$

$FollowedRecommendations$

$$
\begin{aligned}
&S_{recommended} \neq \emptyset\\
&S_{launched} \neq \emptyset\\
&unit? = \{day\}\\
&ordered_L = \langle\rangle\\
&ordered_R = \langle\rangle\\
&grouped_{launched} = \langle\rangle\\
&grouped_{recommended} = \langle\rangle\\
&onlyRecommended = \langle\rangle\\
&cPerGroup_{launched} = \langle\rangle\\
&cPerGropu_{recommended} = \langle\rangle\\
&cPerGroup_{followed} = \langle\rangle\\
&combined = \langle\rangle\\
&t_{start} = 0\\
&N_{launched} = 0\\
&N_{recommended} = 0\\
&N_{followed} = 0\\
&P_{followed} = 0\\
&P_{dueto} = 0
\end{aligned}
$$

- $S_{recommended}$ and $S_{launched}$ are initially not empty sets

- all sequences are initially empty

- all numbers are initially zero

- the default $TIMEUNIT$ is set to day

1.6.3 Group by Timestamp

$$
\begin{array}{l}
\textit{SortByTimestamp} \\
\hline
\textit{Statement} \\
\textit{IsoToUnix} \\
\textit{orderByTimestamp} : \mathbb{F}_1 \rightarrow \textit{seq}_1 \\
o? : \mathbb{F}_1 \\
o! : \textit{seq}_1 \textit{statement} \\
\hline
o? = \{o : \textit{statement}\} \\
o! = \textit{orderByTimestamp}(o?) \\
o! = \langle o_i..o_j \rangle \bullet \forall o_n : o_i..o_j \bullet o_n : \textit{STATEMENT} \land i \leq n \leq j \bullet \\
\qquad\qquad \textit{convert}(o_i.\textit{timestamp}) \leq \textit{convert}(o_n.\textit{timestamp}) \leq \textit{convert}(o_j.\textit{timestamp})
\end{array}
$$

- The schema *SortByTimestamp* introduces the function *orderByTimestamp* which takes in a non-empty, finite set and returns a non-empty, finite sequence.
- *orderByTimestamp* is a sequence of statements ordered from earliest to latest.

$$
\begin{array}{l}
\textit{WithinRange} \\
\hline
\textit{withinRange} : (\mathbb{N}, \mathbb{N}, \mathbb{N}, \textit{TIMEUNIT}) \rightarrow \mathbb{F}_1 \#1 \\
\textit{in}?, \textit{start}?, \textit{state}? : \mathbb{N} \\
\textit{unit}? : \textit{TIMEUNIT} \\
\textit{out}! : \{\textit{TRUE}\} \lor \{\textit{FALSE}\} \\
\hline
\textit{unit}? = \{\textit{second}\} \Rightarrow 1 \lor \{\textit{minute}\} \Rightarrow 60 \lor \{\textit{hour}\} \Rightarrow 3600 \lor \\
\qquad \{\textit{day}\} \Rightarrow 86400 \lor \{\textit{week}\} \Rightarrow 604800 \lor \\
\qquad \{\textit{month}\} \Rightarrow 2629743 \lor \{\textit{year}\} \Rightarrow 31556926 \\
\textit{out}! = \textit{withinRange}(\textit{in}?, \textit{start}?, \textit{state}?, \textit{unit}?) \\
\textit{withinRange}(\textit{in}?, \textit{start}?, \textit{state}?, \textit{unit}?) = \\
\qquad\qquad \textbf{if } \textit{in}? \leq \textit{start}? + ((\textit{state}? + 1) * \textit{unit}?) \\
\qquad\qquad\qquad \textbf{then } \textit{out}! = \{\textit{TRUE}\} \\
\qquad\qquad\qquad \textbf{else } \textit{out}! = \{\textit{FALSE}\}
\end{array}
$$

- The schema *WithinRange* introduces the function *withinRange* which takes in three numbers and a *TIMEUNIT* and returns either $\{TRUE\}$ or $\{FALSE\}$
- *withinRange* checks to see if *in*? is less than or equal to a start time *start*? plus the result of multiplying the numeric conversion for *unit*? by the *state*?.
- *state*? represents the current group, ie. day 1 vs day 2 vs day 3. The +1 is to account for array indexes starting at 0.

$$
\begin{array}{l}
\textit{GroupByTimeUnit} \\
\hline
\textit{Statement} \\
\textit{IsoToUnix} \\
\textit{WithinRange} \\
\textit{groupByTimeUnit} : (\mathrm{seq}_1, \mathbb{N}, \textit{TIMEUNIT}) \rightarrow \mathrm{seq}_1 \\
g? \, , g! : \mathrm{seq}_1 \\
t_{start}? : \mathbb{N} \\
\hline
g? = \langle g?_i \mathinner{..} g?_j \rangle \bullet \forall g?_n : g?_i \mathinner{..} g?_j \bullet i \leq n \leq j \bullet g?_n = \textit{statement} \wedge \\
\quad\quad \textit{convert}(g?_i \, .\textit{timestamp}) \leq \textit{convert}(g?_n \, .\textit{timestamp}) \leq \textit{convert}(g?_j \, .\textit{timestamp}) \\
g! = \textit{groupByTimeUnit}(g? \, , t_{start}? \, \textit{state}? \, , \textit{unit}? \,) \\
g! = \langle g : \textit{seq} \mid \forall g?_n : g?_i \mathinner{..} g?_j \bullet \exists_1 \langle g_r \rangle : \langle g_q \rangle \mathinner{..} \langle g_s \rangle \bullet q \leq r \leq s \wedge r = \textit{state}? \bullet \\
\quad\quad\quad\quad\quad\quad \textbf{if } \textit{withinRange}(\textit{convert}(g?_n \, .\textit{timestamp}), t_{start}, r, \textit{unit}? \,) = \{\textit{TRUE}\} \\
\quad\quad\quad\quad\quad\quad\quad\quad \textbf{then } g? \upharpoonright g?_n \wedge g?_n \; \mathsf{in} \; \langle g_r \rangle \Rightarrow \langle g_r \rangle = \langle g_{ri} \mathinner{..} g_{rn} \mathinner{..} g_{rj} \rangle \bullet ri \leq rn \leq rj \bullet g_{rn} = g?_n \\
\quad\quad\quad\quad\quad\quad\quad\quad \textbf{else if } \forall g_n? : g_i? \mathinner{..} g_j? \bullet \textit{withinRange}(g?_n \, , t_{start}, r, \textit{unit}? \,) = \{\textit{FALSE}\} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \textbf{then } \textit{groupByTimeUnit}((g? \upharpoonright \langle g_r \rangle), t_{start}? \, (r+1), \textit{unit}? \,) \bullet \langle g_r \rangle = \langle\rangle \vee \neq \langle\rangle\rangle \\
\hline
\end{array}
$$

- The schema *GroupByTimeUnit* intorudces the function *groupByTimeUnit* which takes as arguments a non-empty, finite sequence, a natural number and a *TIMEUNIT* and outputs a non-empty, finite sequence of sequences.

- For every statement within the input sequence, *groupByTimeUnit* checks to see if the timestamp of that statement is within the range of t_{start} and *unit*?. If it is, that statement is removed from the input sequence $g?$ and added to the current subsequence $\langle g_r \rangle$. If none of the remaining statements within the input sequence are within the range of t_{start} and *unit*?, then the variable *state*? is incremented, the current subsequence $\langle g_r \rangle$ is either a collection of matched statements or is an empty sequence and the search for remaining subsequences $\langle g_{r+state?} \rangle$ continues.

- because the input sequence $g?$ is orderd chronologically, this implies that once a statement does not fit into a range, the rest of the statements remaining in the input sequence will not fit into that range and *state*? must be incremented to generate a new subsequence $\langle g_{r+state?} \rangle$ so that the remaining statements can be grouped.

1.6.4 Processes Results

$$
\begin{array}{|l}
\hline
\mathit{OrderStatements} \\
\Delta \mathit{FollowedRecommendations} \\
\mathit{SortByTimestamp} \\
\hline
\mathit{ordered}'_L = \mathit{orderByTimestamp}(S_{launched}) \\
\mathit{ordered}'_R = \mathit{orderByTimestamp}(S_{recommended}) \\
t'_{start} = \mathit{convert}((\mathit{head}\,\mathit{ordered}'_L).\mathit{timestamp}) \\
\hline
\end{array}
$$

- The schema *OrderStatements* updates the system state defined by the schema *FollowedRecommendations*.
- $\mathit{ordered}'_L$ is the result of ordering the statements contained within the set $S_{launched}$ chronologically.
- $\mathit{ordered}'_R$ is the result of ordering the statements contained within the set $S_{recommended}$ chronologically.
- t'_{start} is the timestamp from the first statement within $\mathit{ordered}'_L$ converted to unix time.

$$
\begin{array}{|l}
\hline
\mathit{GroupByTime} \\
\Delta \mathit{FollowedRecommendations} \\
\mathit{GroupByTimeUnit} \\
\hline
\mathit{grouped}'_{launched} = \mathit{groupByTimeUnit}(\mathit{ordered}'_L, t'_{start}, 0, \mathit{unit}?) \\
\mathit{grouped}'_{recommended} = \mathit{groupByTimeUnit}(\mathit{ordered}'_R, t'_{start}, 0, \mathit{unit}?) \\
\hline
\end{array}
$$

- The schema *GroupByTime* updates the state defined by the schema *FollowedRecommendations*.
- $\mathit{grouped}'_{launched}$ is the result of passing $\mathit{ordered}'_L$, t'_{start}, 0 and $\mathit{unit}?$ to the function *groupByTimeUnit*.
- $\mathit{grouped}'_{recommended}$ is the result of passing $\mathit{ordered}'_R$, t'_{start}, 0 and $\mathit{unit}?$ to the function *groupByTimeUnit*.

$$
\begin{array}{|ll}
\hline
\mathit{OnlyRecommendedLaunches} & \\
\Delta \mathit{FollowedRecommendations} & \\
\hline
\mathit{onlyRecommended}' = & \langle o : \mathit{seq} \mid \mathbf{let}\ \mathit{grouped}'_{launched} == \mathit{gl} \Rightarrow \\
& \langle \langle \mathit{gl}_i \rangle .. \langle \mathit{gl}_j \rangle \rangle \Rightarrow \langle \langle \mathit{gl}_{ii} .. \mathit{gl}_{ij} \rangle .. \langle \mathit{gl}_{ji} .. \mathit{gl}_{jj} \rangle \rangle \bullet \\
& \forall \langle \mathit{gl}_n \rangle : \langle \mathit{gl}_i \rangle .. \langle \mathit{gl}_j \rangle \bullet \exists_1 \langle o_n \rangle : \langle o_i \rangle .. \langle o_j \rangle \Rightarrow \langle \langle o_{ii} .. o_{ij} \rangle .. \langle o_{ji} .. o_{jj} \rangle \rangle \bullet \\
& ((\forall o_{in} : o_{ii} .. o_{ij} \bullet o_{in}.\mathit{context}.\mathit{statement} \neq \emptyset \land o_{in}\ \mathsf{in}\ \mathit{gl}_i) \land \\
& (\forall o_{jn} : o_{ji} .. o_{jj} \bullet o_{jn}.\mathit{context}.\mathit{statement} \neq \emptyset \land o_{jn}\ \mathsf{in}\ \mathit{gl}_j)) \lor \\
& \langle o_n \rangle = \langle \rangle \rangle \\
\hline
\end{array}
$$

- The schema *OnlyRecommendedLaunches* updates the state defined by the schema $FollowedRecommendations$.

- $onlyRecommended'$ is the sequence of objects o where o is a sequence consisting of statements (or no statements) from the corresponding sequences within $grouped'_{launched}$ where *statement.context.statement* exists.

- $onlyRecommended'$ maintains the same number and ordering of time groups (subsequences) as $grouped'_{launched}$ and $grouped'_{recommended}$.

$$
\begin{array}{|l}
\textit{GetCounts} \\
\hline
\Delta FollowedRecommendations \\
CountPerGroup \\
\hline
cPerGroup'_{launched} = \langle c : \mathbb{N} \mid \mathbf{let}\ grouped'_{launched} == gl \Rightarrow \langle \langle gl_i \rangle .. \langle gl_j \rangle \rangle \bullet \\
\qquad \forall \langle gl_n \rangle : \langle gl_i \rangle .. \langle gl_j \rangle \bullet \exists_1 c_n : \mathbb{N} \bullet \\
\qquad \mathbf{if}\ gl_n = \langle \rangle \\
\qquad\quad \mathbf{then}\ c_n = 0 \\
\qquad\quad \mathbf{else}\ c_n = count(\langle gl_n \rangle) \rangle \\
cPerGroup'_{recommended} = \langle c : \mathbb{N} \mid \mathbf{let}\ grouped'_{recommended} == gr \Rightarrow \langle \langle gr_i \rangle .. \langle gr_j \rangle \rangle \bullet \\
\qquad \forall \langle gr_n \rangle : \langle gr_i \rangle .. \langle gr_j \rangle \bullet \exists_1 c_n : \mathbb{N} \bullet \\
\qquad \mathbf{if}\ gr_n = \langle \rangle \\
\qquad\quad \mathbf{then}\ c_n = 0 \\
\qquad\quad \mathbf{else}\ c_n = count(\langle gr_n \rangle) \rangle \\
cPerGroup'_{followed} = \langle c : \mathbb{N} \mid \mathbf{let}\ onlyRecommended' == or \Rightarrow \langle \langle or_i \rangle .. \langle or_j \rangle \rangle \bullet \\
\qquad \forall \langle or_n \rangle : \langle or_i \rangle .. \langle or_j \rangle \bullet \exists_1 c_n : \mathbb{N} \bullet \\
\qquad \mathbf{if}\ or_n = \langle \rangle \\
\qquad\quad \mathbf{then}\ c_n = 0 \\
\qquad\quad \mathbf{else}\ c_n = count(\langle or_n \rangle) \rangle \\
\hline
\end{array}
$$

- The schema *GetCounts* updates the state defined by the schema $FollowedRecommednations$.

- $cPerGroup'_{launched}$ is a sequence of numbers c where each c is either 0 or the result of passing the current subsequence of $grouped'_{launched}$ (gl_n) to the function *count*.

- $cPerGroup'_{recommended}$ is a sequence of numbers c where each c is either 0 or the result of passing the current subsequence of $grouped'_{recommended}$ (gr_n) to the function *count*.

- $cPerGroup'_{followed}$ is a sequence of numbers c where each c is either 0 or the result of passing the current subsequence of $onlyRecommended'$ (or_n) to the function *count*.

$CombineSequences$

$\Delta FollowedRecommendations$

$$\begin{array}{l}
combined' = \langle c : (tr'_{start}, tr'_{end}, N'_{launched}, N'_{recommended}, N'_{followed}, P'_{followed}, P'_{dueto}) \mid \\
\qquad\qquad\qquad \mathbf{let}\ grouped'_{launched} == gl \Rightarrow \langle \langle gl_i \rangle .. \langle gl_n \rangle .. \langle gl_j \rangle \rangle \\
\qquad\qquad\qquad\quad cPerGroup'_{launched} == cl \Rightarrow \langle cl_i .. cl_n .. cl_j \rangle \\
\qquad\qquad\qquad\quad cPerGroup'_{recommended} == cr \Rightarrow \langle cr_i .. cr_n .. cr_j \rangle \\
\qquad\qquad\qquad\quad cPerGroup'_{followed} == cf \Rightarrow \langle cf_i .. cf_n .. cf_j \rangle \\
\qquad\qquad\qquad \bullet \forall \langle gl_n \rangle : \langle gl_i \rangle .. \langle gl_j \rangle \bullet i \leq n \leq j \bullet \\
\qquad\qquad\qquad \exists_1 c_n : (tr_{startn}, tr_{endn}, N_{launchedn}, N_{recommendedn}, N_{followedn}, P_{followedn}, P_{dueton}) \bullet \\
\qquad\qquad\qquad tr_{startn} = (head\, gl_n).timestamp \\
\qquad\qquad\qquad tr_{endn} = (last\, gl_n).timestamp \\
\qquad\qquad\qquad N_{launchedn} = cl_n \\
\qquad\qquad\qquad N_{recommendedn} = cr_n \\
\qquad\qquad\qquad N_{followedn} = cf_n \\
\qquad\qquad\qquad P_{followedn} = cf_n \div cr_n \\
\qquad\qquad\qquad P_{dueton} = cf_n \div cl_n \rangle
\end{array}$$

- The schema $CombineSequences$ changes the state defined by the schema $FollowedRecommendations$.
- $combined'$ is a sequence of objects c where each c is an ordered pair of $tr'_{start}, tr'_{end}, N'_{launched}, N'_{recommended}, N'_{followed}, P'_{followed}, P'_{dueto}$.
- for each c_n:
  - $tr'_{start} \leadsto tr_{startn}$ which is equal to the timestamp for the first statement within gl_n
  - $tr'_{end} \leadsto tr_{endn}$ which is equal to the timestamp for the last statement within gl_n.
  - $N'_{launched} \leadsto N_{launchedn}$ which is equal to the current count of launched statements within the nth time grouping aka cl_n.
  - $N'_{recommended} \leadsto N_{recommendedn}$ which is equal to the current count of recommended statements within the nth time grouping aka cr_n.
  - $N'_{followed} \leadsto N_{followedn}$ which is equal to the current count of recommended statements within the nth time grouping aka cf_n.
  - $P'_{followed} \leadsto P_{followedn}$ which is equal to the result of dividing cf_n by cr_n.
  - $P'_{dueto} \leadsto P_{dueton}$ which is equal to the result of dividing cf_n by cl_n.

1.6.5 Sequence of Operations

$ProcessFollowedRecommendations \mathrel{\widehat=}$
$OrderStatements \fatsemi GroupByTime \fatsemi OnlyRecommendedLaunches \fatsemi$
$GetCounts \fatsemi CombineSequences$

- The schema *ProcessFollowedRecommendations* defines the order of operations for the steps within the *FollowedRecommendations* algorithm.

1.6.6 Return

$$
\begin{array}{|l}
\hline
\mathit{ReturnFollowedRecommendations} \\
\Xi \mathit{FollowedRecommendations} \\
\mathit{ProcessFollowedRecommendations} \\
\mathit{combined}! : \mathrm{seq} \\
\hline
\mathit{combined}! = \mathit{combined}' \\
\hline
\end{array}
$$

- The schema *ReturnFollowedRecommendations* describes the return value of the system defined by the schema *FollowedRecommendations*

- The return value *combined*! is the variable *combined*′ defined wtihin the schema *CombineSequences*

1.7 Pseudocode

Algorithm 1: Followed Recommendations

```
Input: $S_{recommended}$, $S_{launched}$ $timeUnit$
Result: $combined'$
$ordered'_L \leftarrow orderByTimestamp(S_{launched})$;
$ordered'_R \leftarrow orderByTimestamp(S_{recommended})$;
$t'_{start} \leftarrow convert((head\, ordered'_L).timestamp)$;
$grouped'_{launched} \leftarrow groupByTimeUnit(ordered'_L, t'_{start}, 0, timeUnit)$;
$grouped'_{recommended} \leftarrow$
  $groupByTimeUnit(ordered'_R, t'_{start}, 0, timeUnit)$;
$grouped_{followed} \leftarrow []$;
foreach $G$ in $grouped'_{launched}$ do
    $curGrouping \leftarrow []$;
    foreach $G_n$ in $G$ do
        if $G_n.context.statement \neq nil$ then
            do
            $curGrouping' \leftarrow curGrouping \frown G_n$;
            recur curGrouping'
        else
            recur curGrouping'
        end
    end
    $grouped'_{followed} \leftarrow grouped_{followed} \frown curGrouping'$;
    recur $grouped'_{followed}$
end
$C_{launched} \leftarrow \mathbf{map\, count()\, grouped'_{launched}}$;
$C_{recommended} \leftarrow \mathbf{map\, count()\, grouped'_{recommended}}$;
$C_{followed} \leftarrow \mathbf{map\, count()\, grouped'_{followed}}$;
$combined \leftarrow []$;
for $i \leftarrow 0$ to $count(c_{launched})$ by 1 do
    $tr_{starti} \leftarrow (first(nth(grouped'_{launched}, i))).timestamp$;
    $tr_{endi} \leftarrow (last(nth(grouped'_{launched}, i))).timestamp$;
    $N_{Li} \leftarrow nth(C_{launched}, i)$;
    $N_{Ri} \leftarrow nth(C_{recommended}, i)$;
    $N_{Fi} \leftarrow nth(C_{followed}, i)$;
    $P_{Fi} \leftarrow N_{fi} \div N_{Ri}$;
    $P_{duetoi} \leftarrow N_{fi} \div N_{Li}$;
    $subVec_i \leftarrow [tr_{starti}, tr_{endi}, N_{Li}, N_{Ri}, N_{Fi}, P_{Fi}, P_{duetoi}]$;
    $combined' \leftarrow combined \frown subVec_i$
end
return $combined'$
```

- **map count()** $grouped'_{...}$ means apply the function **count()** to every sequence within the sequence $grouped_{...}$ and put all results into a single array.

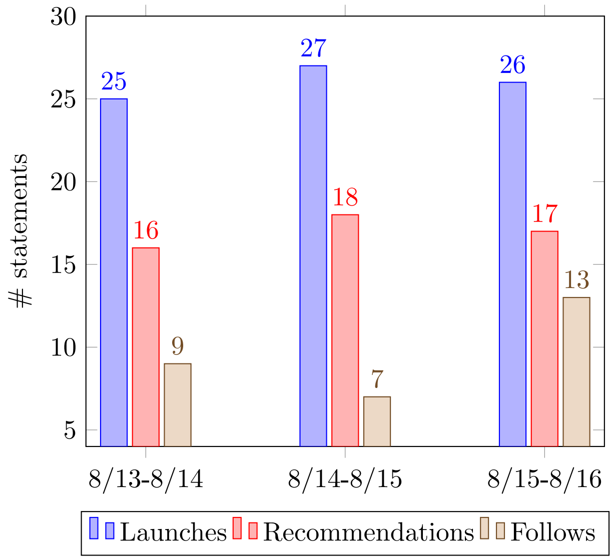
1.8 JSON Schema

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

1.9 Visualization Description

The **Followed Recommendations** visualization can be a bar chart where the domain is time ranges and the range is a number representing the total count of statements recorded. For each time range, there will be three groups: 1) the number of launched statements 2) the number of recommended statements 3) the number of launches which are due to recommendations. Above each grouping or on hover, summary statistics can be desplayed which describe the percentage of launches due to recommendations and the percentage of recommendations which were followed.

1.10 Visualization prototype



- The percentages described in section 5.9 are not displayed here.

1.11 Prototype Improvement Suggestions

Additional features may be implemented on top of this base specification but they would require adding aditional values to each subarray returned by the algorithm. These additional values can be retrieved via (1) performing metadata lookup within or independently of the algorithm (2) by utilizing additional xAPI statement paramters and/or (3) by performing additional computations. The following examples assume the metadata is contained within each statement available to the algorithm.

- populate a tooltip with the most popular launched, recommended and followed activity.
- populate a tooltip with the number of actors associated with the launches and follows.
- populate a tooltip with the actor who most often and the actor who lease often follows recommendations.