A Filter-Map-Reduce Framework for Extracting Features from Log Files

Russell G. Almond

Educational Psychology and Learning Systems, FSU

Work Products and Work Process

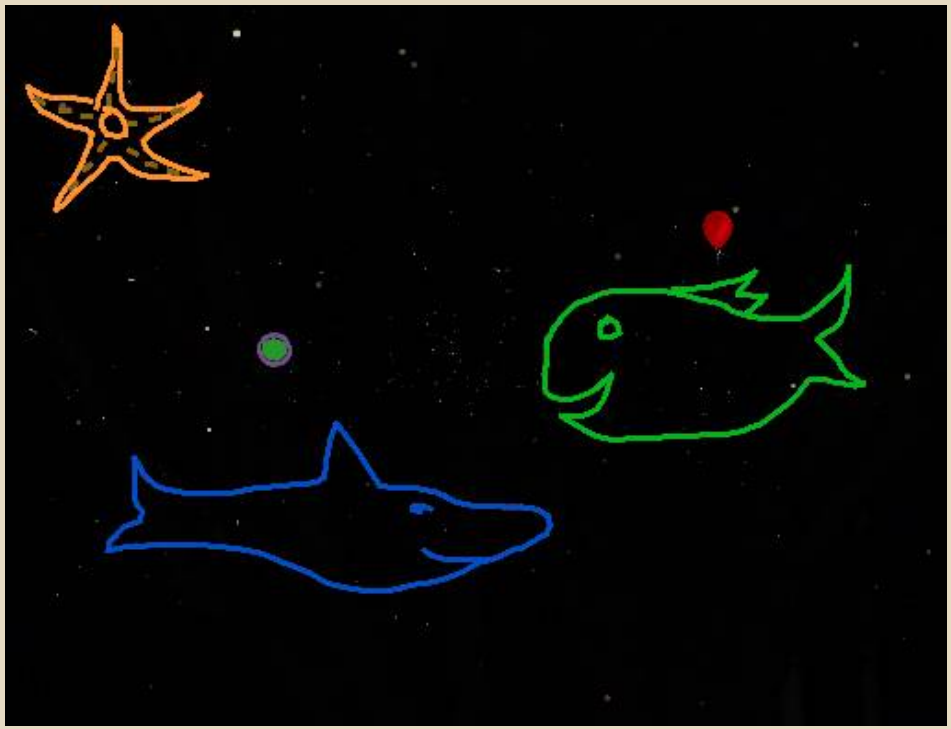
Work Product Final product that the learner produces. - Answer to question - Essay - Simulator state

Work Process Steps learner takes to get to (candidate) solution - Action sequence - Pauses - Event log

- Often have advice about problem solving strategy
  - Science and Engineering Practices
  - Writing and Revision

Case Study: Physics Playground

Use knowledge of physics to get ball to balloon.



Adjust parameters (mass of ball, gravity, air resistance).

Draw objects (e.g., lever) on screen.

Players earn gold, silver or no trophy.

Learning support videos.

Four-Processes Architecture

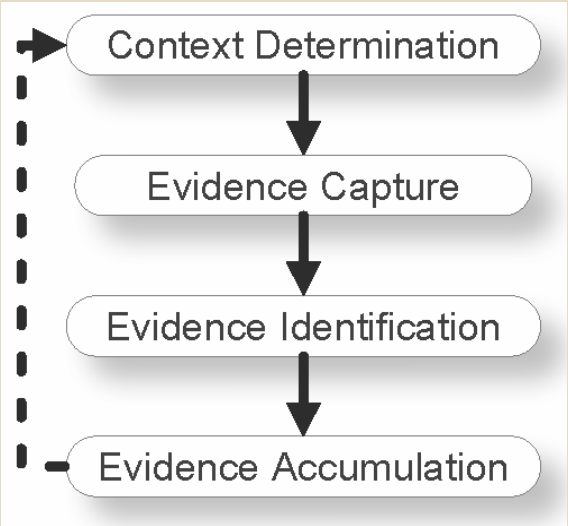


Figure 1: Four-Process Architecture for Assessments

- Context Determination (aka Activity Selection)
  - Rules for what activity (task, problem, game level, item) to do next
  - Rules for when to stop
- Evidence Capture (aka Presentation)
  - Converts physical events (e.g., mouse click) into logical events (e.g., move slider)
  - Logs logical events
  - Determines *scoring context*
- Evidence Identification
  - Summarizes events *within* a scoring context
  - Produces a vector of *observeable outcomes* for each scoring context
- Evidence Accumulation
  - Accumulates evidence *across* scoring contexts
  - Produces the *scores*

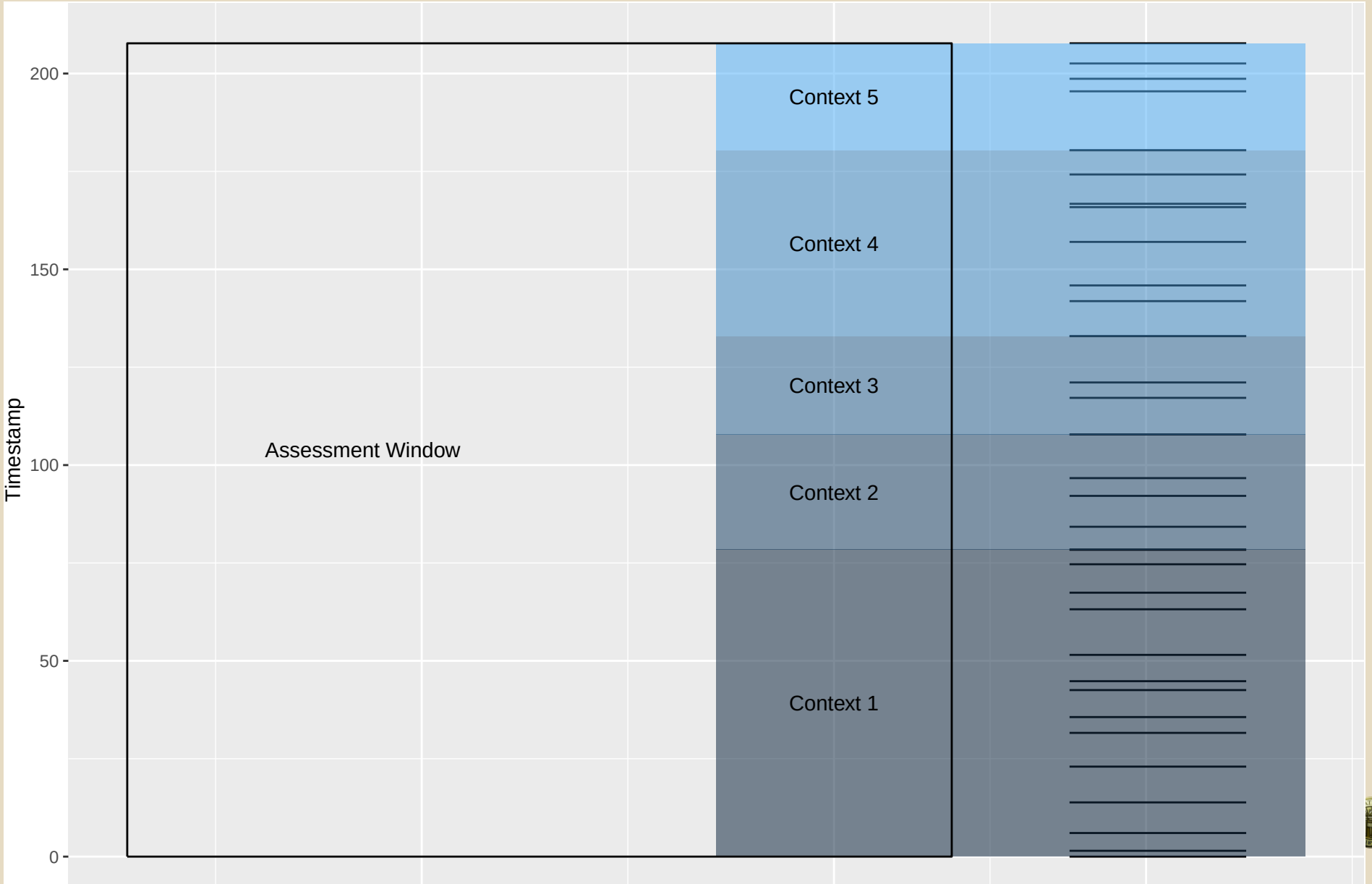
Message Format is like Email message

To:
From:
Topic:
Date:
Context:

Body (Contains topic dependent data)

Signature & Watermarks

Scoring Contexts and Windows



Observable Outcomes

- Output of EI is a row in a data table, with key (`Learner ID`, `Task ID`)
- Columns are *observable outcomes*

Goal is to have educators design these without needing programmers to implement them.

1. "Which trophy (if any) was earned?"
2. Did the player try to draw a lever?
3. How many times did the player adjust the gravity slider?
4. How much time did the player spend viewing support videos?

Context Sets

Most observables are meaningful in more than one scoring context.

Many observables are only meaningful in some scoring contexts.

Associate observables with sets of contexts:

- All Game Levels
  1. Trophy
  2. Time spent on learning supports
- Sketching Level
  2. Lever drawn
- Manipulation Level
  3. Gravity slider manipulation count

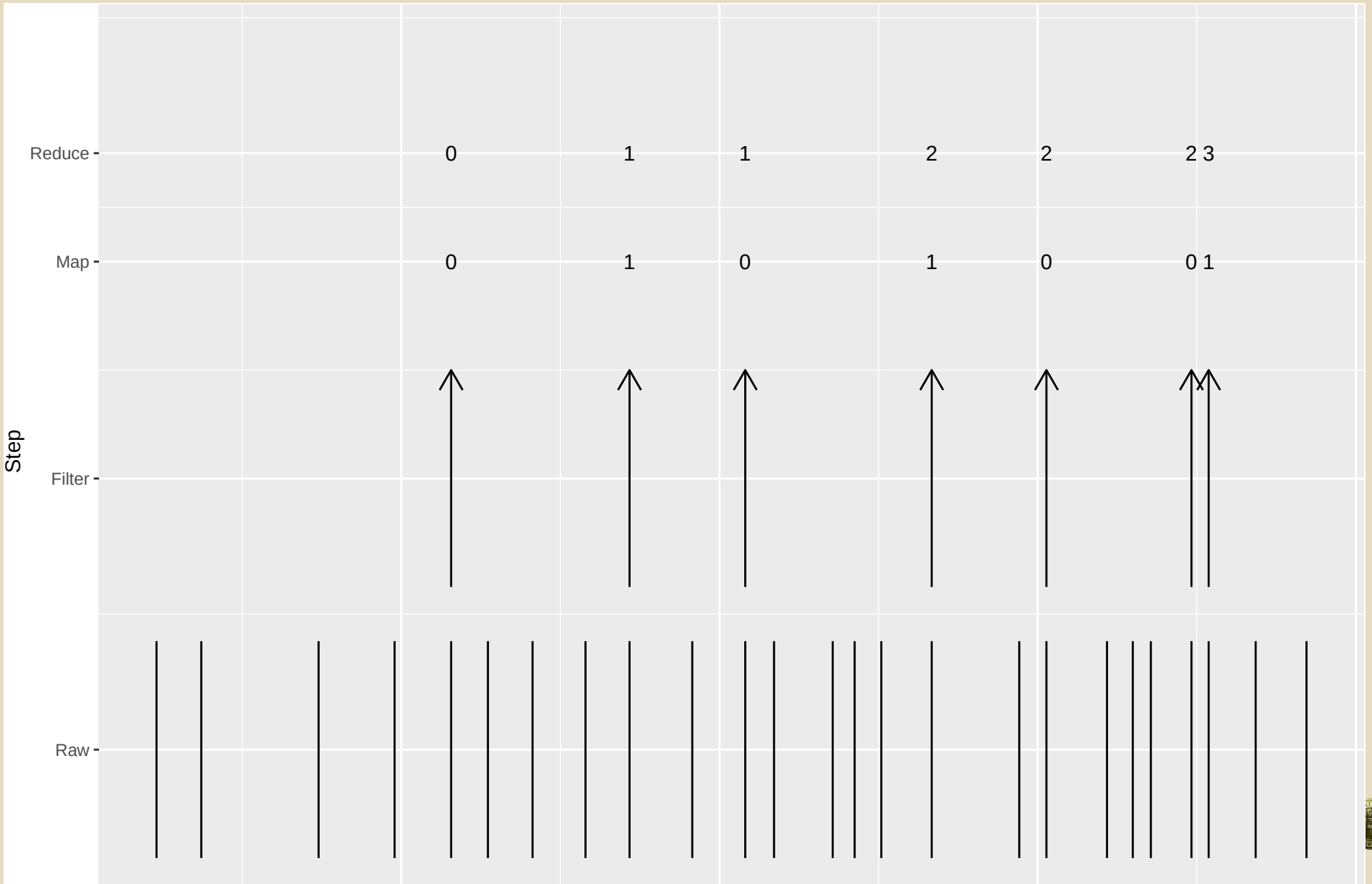
Filter-Map-Reduce

Algorithm for most observables can be partitioned into three phases:

Pre-filter	Only observables related to the context are computed.
Filter	Remove events which are not relevant (e.g., "Ball moved")
Map	Extract relevant datum from event message. (e.g., 1 if gravity slider, 0 if not)
Reduce	Reduce the mapped values to a single outcome (e.g., `sum()`, `count()`, `any()`)

Filter and mapping operations are inherently parallel and can exploit cluster computing. Reduction can be partitioned if the operator is commutative and associative.

A Graphical View of the algorithm



Filtering: Focusing on what is relevant

Three kinds of filtering:

- Certain observables only relevant in certain context sets (implicit filter).
- Only certain message topics are interesting
- Value of data field must be certain value or in a certain range.

Separating out the conditions allows optimizing computations.

Mapping: Extracting Key Values

Takes an event and returns *value* and *timestamp*.

Value could be:

- Constant
- Field from the data
- Transformed field from the data (common numerical operators)
- Conditional Rule:
  - If *condition 1* then *value expression 1*
  - If *condition 2* then *value expression 2*
  - Otherwise *default value expression*

Reduction: Summarizing Across Scoring Window

Takes vector of values and generate final observable values.

- Logical summaries: `any()` or `all()`
- Numerical summaries: `count()`, `sum()`, `mean()`
- Ordinal summaries: `min()`, `max()`
- Position order: `first()`, `last()`

Example 1: What trophy did the player earn?

Filter: `topic` in `{"Level Start", "Trophy Awarded"}`

Map:

- If `topic == "Level Start"` then `none`
- If `topic == "Trophy Awarded"` then `data.trophy`

Reduction: `max()`

Example 2: Did the player draw a lever?

Note that the EC process does the classification of drawings as engines because it requires access to the physics engine.

Filter: `topic` in `{"Agent Identified"}`

Map:

- If `data.agent == "Lever"` then `TRUE`
- Otherwise `FALSE`

Reduction: `any()`

Example 3: How many times did the player adjust the gravity slider

Filter:

- `topic` in `{"Adjusted Slider"}`
- `data.control == "Gravity Slider"`

Map: - Always `1`

Reduction: `count()`

Example 4: For how long did the player use learning supports

System logs start and stop times. Convert to number of seconds, and make start times negative. Then take sum.

Filter:

- `topic` in `{"Learning Support Start", "Learning Support Stop"}`
- `data.support` in `{<Physics Videos>}`

Map:

- If `topic %contains% "Start"` then `-((int) timestamp)`
- If `topic %contains% "Stop"` then `(int) timestamp)`

Reduction: `sum()`

How far we have come

- **Goal**: Get algorithm directly from the specification by content author.
- Algorithm can be naturally partitioned to take advantage of parallel computing
  - `Kafka` – Queuing algorithm
  - `Apache Spark` – Distributed Computing
  - `jq` and `MongoDB` pipelines
- Watermarks allow system to reason about completeness of data.
- Specification, not yet implementation.
- May need labeling preprocessing for some applications (e.g., text analysis).

Future Direction

- Big issue is maintaining consistencey between EC, EI and EA processes.
- Need to better capture possible data fields output from EC.
- Maybe possible to work with analyses of audio/video recordings (typically NVivo).

Thanks

https://ralmond.net/EvidenceStream

Message Format (JSON)

```
{
  messageType: "ESEvent",
  app: <Application GUID>,
  uid: <Unique learner ID within app>,
  window:  <Scoring Context (Task) ID>,
  timestamp: <ISO Time>,
  sender: <ProccessName>,
  topic: <Message Topic>| <verb>/<object>,

  data: { <JSON OBJECT>},

  processed: <logical>,
  pError: [<error message>],
  watermarks: {<mark>:<timestamp>,...}
  signatures: [<encrypted checksums>]
}
```