RFID-Based Logistics Monitoring with Semantics-Driven Event Processing

Mikko Rinne\textsuperscript{1)}, Monika Solanki\textsuperscript{2)} and Esko Nuutila\textsuperscript{1)}

23rd of June 2016
DEBS 2016

\textsuperscript{1)} Aalto University
\textsuperscript{2)} University of Oxford
Scenario: Pharmaceutical Manufacturing

- **Manufacturer process**
  1. **Commissioning**
     - An Electronic Product Code (EPC) is assigned to each item
  2. **Packing**
     - Items are packed in cases
     - An aggregation identifier (AGE) is assigned to each case
  3. **Shipping**
     - Cases are loaded onto pallets
     - Pallets are shipped

- **A code (EPC or AGE) is scanned at each step by a reader, e.g. RFID**

- **Monitored issues**
  - Lost after commissioning
    - A commissioned EPC is not found in packing
  - Counterfeit in packing
    - An uncommissioned EPC is found in packing
  - Lost after packing
    - A packed EPC is not found in shipping
Event Processing with Semantic Web Methods

- **INSTANS**\(^1\) platform based on semantic web technologies
  - Events encoded in RDF (TriG, an event object = an RDF graph)
  - Queries in SPARQL
- **Same task formulated for Esper**\(^2\)
  - Events encoded in XML (an event object = an XML document)
  - Queries in Esper proprietary *Event Processing Language* (EPL)
- Quantitative comparison of performance
- Qualitative observations of differences

---

\(^1\)http://instans.org
\(^2\)http://www.espertech.com
Requirements

• **R1 / Predictability and consistency:** The same expected result shall be consistently produced by all platforms on any number of repeated executions of the same experiment on any suitable execution hardware.

• **R2 / Stream time synchronisation:** Any time filtering operations shall be synchronised to a time reference in the event stream, not the internal clock of the host computer, so that the result is independent of the speed of execution. A consequence of **R1**.

• **R3 / Stability:** No memory residue shall be accumulated from processed events.

• **R4 / Memory efficiency:** Intermediate storage shall be done efficiently so that a maximum number of events can be buffered in a given computing environment.

• **R5 / Performance:** Meet or exceed an average rate of 2.83 EPC/s (based on interviews of people in the pharmaceutical industry quoting manufacturer production figures up to 102,000 EPCs during a 10-hour day).
## Differences between INSTANS and Esper

<table>
<thead>
<tr>
<th>Property</th>
<th>INSTANS</th>
<th>Esper</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmed using</td>
<td>Common Lisp</td>
<td>Java</td>
</tr>
<tr>
<td>Execution environment</td>
<td>Shell or Lisp</td>
<td>Java library (wrapped in our Scala application “XEvePro”)</td>
</tr>
<tr>
<td>Query language</td>
<td>SPARQL 1.1 Query &amp; Update</td>
<td>EPL</td>
</tr>
<tr>
<td>Operation principle</td>
<td>Rule network</td>
<td>Event-driven</td>
</tr>
<tr>
<td>Input formats</td>
<td>RDF as turtle, ntriples, TriG and nquads</td>
<td>POJO, Map, Object-array, XML DOM and Ext</td>
</tr>
<tr>
<td>Output formats</td>
<td>turtle, ntriples, TriG, nquads, SRX and CSV</td>
<td>EventBean objects, XML, JSON, CSV</td>
</tr>
</tbody>
</table>
Example Events (Commissioning)

context:event0 {
  eve:eve0 a eem:ObjectEvent;
  eem:hasEventID "eve0";
  eem:eventOccurredAt "2015-04-16T09:14:59.395+01:00"^^xsd:dateTime;
  rdr2:reader102 a eem:Reader;
  eem:logicalID "reader102".
  eve:eve0 eem:recordedByReader rdr2:reader102;
  eem:action eem:ADD;
  eem:hasBusinessStepType cbv:commissioning;
  eem:hasDisposition cbv:active;
  eem:eventRecordedAt "2015-04-16T09:14:59.395+01:00"^^xsd:dateTime;
  eem:associatedWithEPCList _:node19j0kn68ix1.
  _:node19j0kn68ix1 a eem:SetOfEPCs;
  co:element epc:030001.0012345.10000001001, ..., epc:030001.0012345.100000010020 .
}

<Turtle / TriG>

<eem:ObjectEvent xmlns:eem="http://purl.org/FIspace/eem#"
                  xmlns:cbv="http://purl.org/FIspace/cbv#">
  <hasEventID>0</hasEventID>
  <eventOccurredAt>2015-04-16T09:14:59.395+01:00</eventOccurredAt>
  <recordedByReader>
    <logicalID>102</logicalID>
  </recordedByReader>
  <action resource="eem:ADD"/>
  <hasBusinessStepType resource="cbv:commissioning"/>
  <hasDisposition resource="cbv:active"/>
  <eventRecordedAt>2015-04-16T09:14:59.395+01:00</eventRecordedAt>
  <associatedWithEPCList xmlns:epc="http://fispace.aston.ac.uk/pharmaCo1/data/epc/id/sgtin/">
    <element resource="epc:030001.0012345.10000001001"/>
    <element resource="epc:030001.0012345.100000010020"/>
  </associatedWithEPCList>
</eem:ObjectEvent>

<XML>

• Automatic conversion from TriG to RDF/XML breaks the event to three entities
• Manually generated XML schema
• Our TriG files are 33% smaller (depends on choices in label lengths)
Solution algorithms

INSTANS

EPC \text{c, eve}_{\text{nc}}, t_{\text{c}} \rightarrow \text{EPC}\_c \rightarrow t_{\text{c}}

EPC \text{p, eve}_{\text{np}}, t_{\text{p}} \rightarrow \text{EPC}\_p \rightarrow t_{\text{p}}

\text{report lost}

\text{report counterfeit}

\text{true}

\text{true}

\text{t}_{\text{c}} + t_{\text{winc}} \leq t_{\text{e}}

\text{t}_{\text{p}} + t_{\text{wint}} \leq t_{\text{e}}

Esper

\text{EPC}\_c = \text{EPC}\_p

\text{EPC}\_c \text{ not in win}_m

\text{EPC}\_p \text{ not in win}_m

\text{report lost}

\text{report counterfeit}

\text{true}

\text{true}

\text{remove stream}

\text{remove stream}
Example Queries to Extract EPCs from Commissioning Events

DELETE { GRAPH ?g { ?elist co:element ?epc } }
INSERT { ?epc :commissionedEPC ?e ;
   :commissionedInSeconds ?current_sec } WHERE { GRAPH ?g {
   ?e a eem:ObjectEvent ;
   eem:hasBusinessStepType cbv:commissioning ;
   eem:action eem:ADD ;
   eem:hasDisposition cbv:active ;
   eem:associatedWithEPCList ?elist .
   ?elist co:element ?epc } ;
latest :eventInSeconds ?current_sec } ;

insert into CommissioningEPCs
select hasEventID, resource as EPCNumber
from ObjectEvent(
   hasBusinessStepType.resource="cbv:commissioning",
   action.resource="eem:ADD",
   hasDisposition.resource="cbv:active")
[select hasEventID, resource
from associatedWithEPCList.element];

SPARQL Update

EPL
Stream Time Synchronisation (R2)

- **Clocks synchronised to the recorded event stream**
  - Use value of `eventOccurredAt`
  - Time has to progress after end-of-file to trigger detection
    - Otherwise last anomalies will not be detected

- **Esper**
  - Internal timer disabled
  - `CurrentTimeEvents` sent based on incoming stream
  - End-of-file detected by file reader, time moved forward

- **INSTANS**
  - A custom current time triple updated with rules in the main graph
  - All other time-dependent queries match said triple
  - Another input file with an end-marker sent after recorded stream
Why INSTANS and Esper?

1. Events with nested structures
   • Available RDF-based implementations at time of testing worked on
timestamped triples
     – RDF stream processing community group working on a common specification for timestamped graphs
   • Distributed stream computing platforms typically run on tuples

2. Query-based processing

3. Free license permitting result publication
   • Most query-based non-RDF complex event processing platforms are
commercial offerings
     – Esper is available with a community license, which permits publication of results

4. Event patterns
   • Most RDF stream processing platforms are based on the data stream
management system (DSMS) paradigm with stream-level windowing
Stream Windows vs. Event Patterns

- Non-overlapping (tumbling) windows would produce random results
- Overlapping windows catch both correct results and false positives
- Cancelling false positives would require
  - Post-processing with inter-window memory
    - Auxiliary streams
    - Persistent states specified in TEF-SPARQL\(^1\) (no implementation available)
  - Restrictions for the time position of a match within the window

\(^1\) Gao, S., Scharrenbach, T., & Bernstein, A. Running out of Bindings? Integrating Facts and Events in Linked Data Stream Processing. CEUR Vol-1488.
Experimental Setup

• TriG and XML input files generated separately (no run-time conversion)
• Identical output confirmed by diff on sorted CSV output files
• Execution time measured as the median time of three last runs in a batch of four
  • Output to /dev/null to minimize I/O impact
• Memory consumption monitored with top using 1 second sample intervals
• Complete datasets, queries, results, execution scripts and instructions for repeating the experiment available¹)

¹)https://bitbucket.org/aaltodsg/manufacturer-logistics-monitoring
Event Timing Diagram

- **Commissioning**
  - Parameter: $c(1) \rightarrow \Delta t_c \rightarrow c(2) \rightarrow c(cp)$
  - Parameter Values:
    - $c(1)$: $EPC_1 \rightarrow EPC_{ec}$
    - $c(2)$: $\Delta t_c$
    - $c(cp)$: $\Delta t_{cp}$

- **Packing**
  - Parameter: $p(1) \rightarrow \Delta t_p \rightarrow p(2) \rightarrow p(ps)$
  - Parameter Values:
    - $p(1)$
    - $p(2)$
    - $p(ps)$

- **Shipping**
  - Parameter: $s(1) \rightarrow \Delta t_{ps} \rightarrow s(2)$
  - Parameter Values:
    - $s(1)$
    - $s(2)$

**Parameter Values**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>ec [nr]</td>
<td>20</td>
</tr>
<tr>
<td>cp [nr]</td>
<td>5</td>
</tr>
<tr>
<td>ps [nr]</td>
<td>20</td>
</tr>
<tr>
<td>$\Delta t_c$</td>
<td>0.5 s</td>
</tr>
<tr>
<td>$\Delta t_{cp}$</td>
<td>3.5 – 25,000 s</td>
</tr>
<tr>
<td>$\Delta t_p$</td>
<td>2.5, 3 s</td>
</tr>
<tr>
<td>$\Delta t_{ps}$</td>
<td>110 s</td>
</tr>
<tr>
<td>$\Delta t_s$</td>
<td>50 s</td>
</tr>
</tbody>
</table>
E1: Speed and Memory Stability

- Commissioning to packing only (E1-4)
- Buffering kept low
- Esper up to ~15x faster
- INSTANS memory consumption shows a slight upward trend

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta t_{cp}$</td>
<td>3.5 s</td>
</tr>
<tr>
<td>$t_{winc}$</td>
<td>5 s</td>
</tr>
<tr>
<td>Counterfeit EPC</td>
<td>8%</td>
</tr>
</tbody>
</table>

**EPC-codes per second**

<table>
<thead>
<tr>
<th>Nr. Of EPC</th>
<th>100</th>
<th>1,000</th>
<th>10,000</th>
<th>100,000</th>
<th>1,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Esper</td>
<td>734</td>
<td>2,867</td>
<td>8,146</td>
<td>21,790</td>
<td>39,243</td>
</tr>
<tr>
<td>INSTANS</td>
<td>964</td>
<td>1,952</td>
<td>2,566</td>
<td>2,686</td>
<td>2,514</td>
</tr>
</tbody>
</table>

**Memory consumption during 1M EPC**

- Commissioning to packing only (E1-4)
- Buffering kept low
- Esper up to ~15x faster
- INSTANS memory consumption shows a slight upward trend
E2: Performance Impact of Counterfeit Detection

- Everything else stable, alter the share of counterfeits
- INSTANS slowed down by more than 40% from 0 to 100% error rate due to an increased amount of output to process
- Esper speed increased over 30%
  - Difference in algorithms: as there are fewer entries going to the $w_{in_m}$ matching window in the higher counterfeit cases (nothing from commissioning matches packing in the 100% case), there are also fewer comparisons with $w_{in_m}$ contents.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta t_{cp}$</td>
<td>3.5 s</td>
</tr>
<tr>
<td>$t_{winc}$</td>
<td>5 s</td>
</tr>
<tr>
<td>Counterfeit EPC</td>
<td>0%, 50%, 100%</td>
</tr>
</tbody>
</table>
E3: Intermediate Buffering Impact

- All test batches equally long (100k EPC)
- Time between commissioning and packing varies, long commissioning-window
- Esper
  - No clear performance impact
  - 20% memory consumption increase
- INSTANS
  - Addition of new EPC codes to the graph slows down when the number of buffered codes increases due to the increased number of bindings to be verified for each addition (130x slowdown)
  - 121% memory consumption increase

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta t_{cp}$</td>
<td>10; 100; 1,000; 2,500 s</td>
</tr>
<tr>
<td>$t_{winc}$</td>
<td>2,600 s</td>
</tr>
<tr>
<td>Counterfeit EPC</td>
<td>8%</td>
</tr>
</tbody>
</table>
E4: Buffer Everything

- Variable test batch length
- Everything buffered between commissioning and packing
- Esper
  - Performance only slows down with 1M EPCs
- INSTANS
  - 100k EPCs at 20 EPC/s, 1M EPC did not complete in six days (vs. Esper 91 secs)
    - Preliminary result with latest optimisations (after paper submission)
    - 86x improvements achieved for 100k, 1736 EPC/s and for 1M EPC 1117 EPC/s

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta t_{cp}$</td>
<td>25; 250; 2,500; 25,000 s</td>
</tr>
<tr>
<td>$t_{winc}$</td>
<td>30,000 s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nr. of EPC</th>
<th>1,000</th>
<th>10,000</th>
<th>100,000</th>
<th>1,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Esper</td>
<td>3,008</td>
<td>7,941</td>
<td>20,360</td>
<td>11,028</td>
</tr>
<tr>
<td>INSTANS</td>
<td>1,083</td>
<td>228</td>
<td>20</td>
<td>N/A</td>
</tr>
</tbody>
</table>

EPC-codes per second

<table>
<thead>
<tr>
<th>Nr of EPC-codes</th>
<th>1,000</th>
<th>10,000</th>
<th>100,000</th>
<th>1,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Esper</td>
<td>452</td>
<td>336</td>
<td>516</td>
<td>604</td>
</tr>
<tr>
<td>INSTANS</td>
<td>1,280</td>
<td>1,195</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
E5: Complete Processing Chain

- As experiment 1, but include shipping
- Esper
  - 8.5 times slower than itself in exp1 at 1M EPCs
  - 1,600 times requirement $R_5$ (1M EPC)
- INSTANS
  - 1M EPCs did not complete due to a token collision error
  - 950 times requirement $R_5$ (100k EPC)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_s$ [nr]</td>
<td>20</td>
</tr>
<tr>
<td>$\Delta t_c$</td>
<td>0.5 s</td>
</tr>
<tr>
<td>$\Delta t_{cp}$</td>
<td>3.5 s</td>
</tr>
<tr>
<td>$\Delta t_p$</td>
<td>2.5 s</td>
</tr>
<tr>
<td>$\Delta t_{ps}$</td>
<td>110 s</td>
</tr>
<tr>
<td>$\Delta t_s$</td>
<td>50 s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nr. Of EPC</th>
<th>100</th>
<th>1,000</th>
<th>10,000</th>
<th>100,000</th>
<th>1,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Esper</td>
<td>724</td>
<td>3,012</td>
<td>3,161</td>
<td>4,526</td>
<td>4,639</td>
</tr>
<tr>
<td>INSTANS</td>
<td>888</td>
<td>1,811</td>
<td>2,573</td>
<td>2,714</td>
<td>N/A</td>
</tr>
</tbody>
</table>

EPC-codes per second
## Results vs. Requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Esper + XEvePro</th>
<th>INSTANS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predictability and consistency (R1)</td>
<td>Ok&lt;sup&gt;1)&lt;/sup&gt;</td>
<td>Ok&lt;sup&gt;1)&lt;/sup&gt;</td>
</tr>
<tr>
<td>Stream time synchronisation (R2)</td>
<td>Ok&lt;sup&gt;1)&lt;/sup&gt;</td>
<td>Ok&lt;sup&gt;1)&lt;/sup&gt;</td>
</tr>
<tr>
<td>Stability (R3)</td>
<td>Ok</td>
<td>Slow memory residue buildup observed. Exp5 1M did not complete due to a runtime error.</td>
</tr>
<tr>
<td>Memory efficiency (R4)&lt;sup&gt;2)&lt;/sup&gt;</td>
<td>166 EPC/MiB at 100k EPC, 837 EPC/MiB at 1M EPC (Exp4).</td>
<td>78.1 EPC/MiB at 100k EPC, 1M EPC did not complete (Exp4).</td>
</tr>
<tr>
<td>Performance (R5)</td>
<td>Stable performance, target exceeded in all tests. Exp5 1M EPC exceeds target 1,600 times.</td>
<td>Performance impacted by output (Exp2) and buffering (Exp3, Exp4). Exp5 100k EPC exceeds target 950 times.</td>
</tr>
</tbody>
</table>

<sup>1)</sup> identical results verified  
<sup>2)</sup> sampled with *top*
# Qualitative Comparison (1/2)

<table>
<thead>
<tr>
<th></th>
<th>Esper + XEvePro</th>
<th>INSTANS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Configuration parameters</strong></td>
<td>EPL and command line</td>
<td>RDF (Turtle) and command line</td>
</tr>
<tr>
<td><strong>Programming</strong></td>
<td>EPL and Scala</td>
<td>SPARQL</td>
</tr>
<tr>
<td><strong>Synchronisation to timestamps in events</strong></td>
<td>EPL and Scala providing an external sync to the Esper system timer</td>
<td>SPARQL</td>
</tr>
<tr>
<td><strong>Buffering of EPC codes</strong></td>
<td>Built-in window mechanism</td>
<td>In-memory graph storage</td>
</tr>
<tr>
<td><strong>Output formatting</strong></td>
<td>Scala listeners attached to EPL queries</td>
<td>SPARQL SELECT queries</td>
</tr>
<tr>
<td><strong>Use of globally accessible ontologies</strong></td>
<td>XML namespaces but not RDF/OWL ontologies (ontology names preserved in the XML format)</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Semantic web tool compatibility</strong></td>
<td>No</td>
<td>Through input data (TriG), output (TriG, turtle, ntriples, nquads available) and federated query support</td>
</tr>
<tr>
<td><strong>Reasoning and entailments</strong></td>
<td>No</td>
<td>INSTANS supports entailment rules, RDF is compatible with other reasoning tools</td>
</tr>
</tbody>
</table>
Qualitative Comparison (2/2)

- **Considering the next step in the logistics chain**
  - The manufacturer ships the pallets and a shipping manifest (a “pedigree”) to a warehouse

- **Semantic Web / INSTANS**
  - The pedigree can be a graph or dataset and exposed to the warehouse as an IRI
  - The warehouse can send federated queries to compare the contents of a shipment against the pedigree.
  - INSTANS could
    1. construct a graph or dataset of the successfully shipped items in the sending site
    2. perform the federated queries against the observed item codes in the receiving site.

- **Non-semantic web / Esper**
  - SQL database connectivity supported
  - A similar outcome could be achieved by saving the pedigree to a database and exposing the database to the warehouse
  - Extra coding effort required
Conclusions

• The example industrial logistics monitoring task was successfully implemented on both RDF/SPARQL and XML/EPL frameworks
  • Identical results confirmed
• Esper demonstrated clearly better performance and higher maturity of the platform
  • Platform of choice for best performance
• INSTANS also met or exceeded real-world derived requirements, when excessive buffering was not required
  • Work in progress, but with benefits in integration to the semantic web framework