Applying Complex Event Processing

Edson Tirelli
etirelli@redhat.com
Lead CEP Designer
JBoss, a Division of Red Hat
Agenda

- Brief introduction on CEP and Terminology
- Drools Vision
- Drools Fusion: Complex Event Processing extensions
  - Event Declaration and Semantics
  - Event Cloud, Streams and the Session Clock
  - Temporal Reasoning
  - Sliding Window Support
  - Streams Support
  - Memory Management
- Questions & Answers
"An event is an observable occurrence."
"An event in the Unified Modeling Language is a notable occurrence at a particular point in time."

"Anything that happens, or is contemplated as happening."
"An object that represents, encodes or records an event, generally for the purpose of computer processing"
For the scope of this presentation:

“An event is a **significant change of state** at a particular **point in time**”
Complex Event, is an abstraction of other events called its members.”

Examples:

- The 1929 stock market crash – an abstraction denoting many thousands of member events, including individual stock trades
- The 2004 Indonesian Tsunami – an abstraction of many natural events
- A completed stock purchase - an abstraction of the events in a transaction to purchase the stock
- A successful on-line shopping cart checkout – an abstraction of shopping cart events on an on-line website

Source: http://complexevents.com
“Complex Event Processing, or CEP, is primarily an event processing concept that deals with the task of processing multiple events with the goal of identifying the meaningful events within the event cloud.

CEP employs techniques such as detection of complex patterns of many events, event correlation and abstraction, event hierarchies, and relationships between events such as causality, membership, and timing, and event-driven processes.”

-- wikipedia
Examples:

- The Drools Bootcamp impact:
  - The Eyjafjallajokull glacier volcano eruption in Iceland
  - Followed by the ash cloud drifting southeast over Europe
  - Causing air traffic disruption in over 20 European and Asian countries
  - Affecting plans of the Drools Bootcamp in San Diego, CA

- Paul’s pickpocket event on Rome’s subway
- Credit card fraud detection
- Logistics Real-Time Awareness solution
- Neonatal ICU: infant vital signs monitoring
Complex Event Processing, or CEP, and Event Stream Processing, or ESP, are two technologies that were born separate, but converged.

An oversimplification: In their origins...

- Event Stream Processing focused on the ability to process high volume **streams** of events.
- Complex Event Processing focused on defining, detecting and processing the **relationships** among events.
For the scope of this presentation:

“CEP is used as a common term meaning both CEP and ESP.”
**Terminology: EDA**

“**Event Driven Architecture (EDA)** is a software architecture pattern promoting the production, detection, consumption of, and reaction to events. An event can be defined as "a significant change in state"[1]. For example, when a consumer purchases a car, the car's state changes from "for sale" to "sold". A car dealer's system architecture may treat this state change as an event to be produced, published, detected and consumed by various applications within the architecture.”
EDA vs CEP

CEP is a **component** of the EDA

EDA vs SOA

- EDA is **not** SOA 2.0
- Complementary architectures
- Metaphor
  - In our **body**:
    - SOA is used to build our muscles and organs
    - EDA is used to build our sensory system
EDA vs SOA

A few characteristics of common CEP scenarios:

- Huge volume of events, but only a few of real interest
- Usually events are immutable
- Usually queries/rules have to run in reactive mode
- Strong temporal relationships between events
- Individual events are usually not important
- The composition and aggregation of events is important
“A common platform to model and govern the business logic of the enterprise.”
Motivation

- Business Rules, Event Processing and Business Processes are all modelled declaratively.
- A business solution usually involves the interaction between these technologies.
- In short:
  - Technology overlap
  - Business overlap
- Several (good) products on the market:
  - Better either at CEP/ESP or Rules Processing or Business Processes
- The approach: attribute the same importance to the three complementary business modeling techniques
Drools Fusion: Enables...

- **Event Detection:**
  - From an event cloud or set of streams, select all the meaningful events, and only them.

- **[Temporal] Event Correlation:**
  - Ability to correlate events and facts declaring both temporal and non-temporal constraints between them.
  - Ability to reason over event aggregation

- **Event Abstraction:**
  - Ability to compose complex events from atomic events AND reason over them
Drools Fusion

**Features:**

- Event Semantics as First Class Citizens
- Allow Detection, Correlation and Composition
- Temporal Constraints
- Session Clock
- Stream Processing
- Sliding Windows
- CEP volumes (scalability)
- (Re)Active Rules
- Data Loaders for Input
Event Declaration and Semantics

- **Event semantics:**
  - **Point-in-time and Interval**
  - An **event** is a fact with a few special characteristics:
    - Usually **immutable**, but not enforced
    - Strong **temporal** relationships
    - **Lifecycle** may be managed
    - Allow use of **sliding windows**

- “All **events are facts**, but not all facts are events.”
Temporal Reasoning

- **Semantics for:**
  - **time:** discrete
  - **events:** point-in-time and interval
- **Ability to express temporal relationships:**
  - Allen’s 13 temporal operators

- **James F. Allen** defined the 13 possible temporal relations between two events.
- **Eiko Yoneki** and **Jean Bacon** defined a unified semantics for event correlation over time and space.
rule “Shipment not picked up in time”
when
    Shipment( $pickupTime : scheduledPickupTime )
    not ShipmentPickup( this before $pickupTime )
then
    // shipment not picked up... action required.
end
rule "Shipment not picked up in time"
when
  Shipment( $pickupTime : scheduledPickupTime )
  not ShipmentPickup( this before $pickupTime )
then
  // shipment not picked up... Action required.
end
### Allen’s 13 Temporal Operators

<table>
<thead>
<tr>
<th></th>
<th>Point-Point</th>
<th>Point-Interval</th>
<th>Interval-Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>A before B</td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
</tr>
<tr>
<td>A meets B</td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
</tr>
<tr>
<td>A overlaps B</td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
</tr>
<tr>
<td>A finishes B</td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
</tr>
<tr>
<td>A includes B</td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
</tr>
<tr>
<td>A starts B</td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
</tr>
<tr>
<td>A coincides B</td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
<td><img src="#" alt="Diagram" /></td>
</tr>
</tbody>
</table>
Allen’s 13 Temporal Operators

<table>
<thead>
<tr>
<th></th>
<th>Point-Point</th>
<th>Point-Interval</th>
<th>Interval-Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>A after B</td>
<td>A B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A metBy B</td>
<td>A B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A overlapedBy B</td>
<td>A B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A finishedBy B</td>
<td>A B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A during B</td>
<td>A B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A finishes B</td>
<td>A B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Some references

- **Allen, J. F.** *Maintaining knowledge about temporal intervals.* 1983
- **Yoneki, Eiko and Bacon, Jean.** *Unified Semantics for Event Correlation Over Time and Space in Hybrid Network Environments.* 2005.
- **Bennett, Brandon and Galton, Antony P.** *A Unifying Semantics for Time and Events.* 2000.
Simple Example Scenario
Stream Support (entry-points)

- **A scoping abstraction for stream support**
  - Rule compiler gather all entry-point declarations and expose them through the session API
  - Engine manages all the scoping and synchronization behind the scenes.

```java
package com.sample

rule "Stock Trade Correlation"
  when
    $c: Customer( type == "VIP" )
    BuyOrderEvent( customer == $c, $id : id ) from entry-point "Home Broker Stream"
    BuyAckEvent( relatedEvent == $id ) from entry-point "Stock Trader Stream"
  then
    // take some action

StatefulSession session = ...;
EntryPoint ep = session.getEntryPoint("Home Broker Stream");
...
ep.insert(...);
ep.insert(...);
...```
Cloud Mode, Stream Mode, Session Clock

**CLOUD**
- No notion of “flow of time”: the engine sees all facts without regard to time
- No attached Session Clock
- No requirements on event ordering
- No automatic event lifecycle management
- No sliding window support

**STREAM**
- Notion of “flow of time”: concept of “now”
- Session Clock has an active role synchronizing the reasoning
- Event Streams must be ordered
- Automatic event lifecycle management
- Sliding window support
- Automatic rule delaying on absence of facts
Reference Clock

- Reference clock defines the flow of time
- Named **Session Clock**
  - is assigned to each session created
- Synchronizes time sensitive operations
  - duration rules
  - event streams
  - process timers
  - sliding windows
Session Clock

- Uses the strategy pattern and multiple implementations:
  - Real-time operation
  - Tests
  - Simulations
  - etc

```
SessionClock

 RealTimeClock
PseudoClock
HeartBeatClock
(custom clocks)
```
Selecting the session clock:

API:

```java
KnowledgeSessionConfiguration conf = ...;
conf.setOption( ClockTypeOption.get( "realtime" ) );
```

System Property or Configuration File:

```java
drools.clockType = pseudo
```
Sliding Window Support

- Allows reasoning over a moving window of “interest”
  - Time
  - Length

```plaintext
rule “Average Order Value over 12 hours”
when
  $c : Customer()$a : Number() from accumulate (BuyOrder( customer == $c, $p : price ) over window:time( 12h ),
  average( $p ) )
then
  // do something
end
```
Negative patterns may require rule firings to be delayed.

```java
rule "Order timeout"
when
    $bse : BuyShares ( $id : id )
    not BuySharesAck( id == $id, this after[0s,30s] $bse )
then
    // Buy order was not acknowledged. Cancel operation
    // by timeout.
end
```
Negative patterns may require rule firings to be delayed.

```plaintext
rule "Order timeout"
when
    $bse : BuyShares ( $id : id )
    not BuySharesAck( id == $id, this after[0s,30s] $bse )
then
    // Buy order was not acknowledged. Cancel operation
    // by timeout.
end
```

Forces the rule to wait for 30 seconds before firing, because the acknowledgement may arrive at any time!
Some references

✓ Ghanem, Hammad, Mokbel, Aref and Elmagarmid. *Incremental Evaluation of Sliding-Window Queries over Data Streams.*
Temporal Dimension

- Requires the support to the temporal dimension
  - A rule/query might match in a given point in time, and not match in the subsequent point in time

- That is the single most difficult requirement to support in a way that the engine:
  - stays deterministic
  - stays a high-performance engine

- Achieved mostly by compile time optimizations that enable:
  - constraint tightening
  - match space narrowing
  - memory management
Temporal Dimension Support

- CEP scenarios are **stateful** by nature.
- Events usually are only **interesting during a short period of time**.
- Hard for applications to know when events are not necessary anymore
  - Temporal constraints and sliding windows describe such “**window of interest**”
rule “Bag was not lost”
when
  $c : BagEvent( ) from entry-point “check-in”
  $l : BagEvent( this == $c.bagId, this after[0,5m] $c )
      from entry-point “pre-load”
then
  // bag was not lost
end
rule “reasoning on events over time”
when
   $a : A( )
   $b : B( this after[-2,2] $a )
   $c : C( this after[-3,4] $a )
   $d : D( this after[1,2] $b, this after[2,3] $c )
   not E( this after[1,10] $d )
then
   // do something
end
Temporal Distance Algorithm

1. Gather all temporal relationships between events
2. Create the temporal dependency graph as a dependency matrix
3. Calculate the reflexive and transitive closures
   - Floyd-Warshall algorithm: $O(n^3)$
4. Check for unbound intervals
   - Infinite time-windows
5. Calculate the maximum expiration time for each of the event types
6. Calculate necessary delay for the rules with negative patterns
# Temporal Dependency Matrix

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>[0, 0]</td>
<td>[−2, 2]</td>
<td>[−3, 4]</td>
<td>[−∞, ∞]</td>
<td>[−∞, ∞]</td>
</tr>
<tr>
<td>B</td>
<td>[−2, 2]</td>
<td>[0, 0]</td>
<td>[−∞, ∞]</td>
<td>[1, 2]</td>
<td>[−∞, ∞]</td>
</tr>
<tr>
<td>C</td>
<td>[−4, 3]</td>
<td>[−∞, ∞]</td>
<td>[0, 0]</td>
<td>[2, 3]</td>
<td>[−∞, ∞]</td>
</tr>
<tr>
<td>D</td>
<td>[−∞, ∞]</td>
<td>[−2, −1]</td>
<td>[−3, −2]</td>
<td>[0, 0]</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>E</td>
<td>[−∞, ∞]</td>
<td>[−∞, ∞]</td>
<td>[−∞, ∞]</td>
<td>[−10, −1]</td>
<td>[0, 0]</td>
</tr>
</tbody>
</table>
### Temporal Dependency Matrix

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>[0, 0]</td>
<td>[-2, 2]</td>
<td>[-3, 4]</td>
<td>[-∞, ∞]</td>
<td>[-∞, ∞]</td>
</tr>
<tr>
<td>B</td>
<td>[-2, 2]</td>
<td>[0, 0]</td>
<td>[-∞, ∞]</td>
<td>[1, 2]</td>
<td>[-∞, ∞]</td>
</tr>
<tr>
<td>C</td>
<td>[-4, 3]</td>
<td>[-∞, ∞]</td>
<td>[0, 0]</td>
<td>[2, 3]</td>
<td>[-∞, ∞]</td>
</tr>
<tr>
<td>D</td>
<td>[-∞, ∞]</td>
<td>[-2, -1]</td>
<td>[-3, -2]</td>
<td>[0, 0]</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>E</td>
<td>[-∞, ∞]</td>
<td>[-∞, ∞]</td>
<td>[-∞, ∞]</td>
<td>[-10, -1]</td>
<td>[0, 0]</td>
</tr>
</tbody>
</table>

#### Transitive Closure

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>[0, 0]</td>
<td>[-2, 2]</td>
<td>[-3, 2]</td>
<td>[-1, 4]</td>
<td>[0, 14]</td>
</tr>
<tr>
<td>B</td>
<td>[-2, 2]</td>
<td>[0, 0]</td>
<td>[-2, 0]</td>
<td>[1, 2]</td>
<td>[2, 12]</td>
</tr>
<tr>
<td>C</td>
<td>[-2, 3]</td>
<td>[0, 2]</td>
<td>[0, 0]</td>
<td>[2, 3]</td>
<td>[3, 13]</td>
</tr>
<tr>
<td>D</td>
<td>[-4, 1]</td>
<td>[-2, -1]</td>
<td>[-3, -2]</td>
<td>[0, 0]</td>
<td>[1, 10]</td>
</tr>
<tr>
<td>E</td>
<td>[-14, 0]</td>
<td>[-12, -2]</td>
<td>[-13, -3]</td>
<td>[-10, -1]</td>
<td>[0, 0]</td>
</tr>
</tbody>
</table>
Temporal Dependency Matrix

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>[0,0]</td>
<td>[-2,2]</td>
<td>[-3,2]</td>
<td>[-1,4]</td>
<td>[0,14]</td>
</tr>
<tr>
<td>B</td>
<td>[-2,2]</td>
<td>[0,0]</td>
<td>[-2,0]</td>
<td>[1,2]</td>
<td>[2,12]</td>
</tr>
<tr>
<td>C</td>
<td>[-2,3]</td>
<td>[0,2]</td>
<td>[0,0]</td>
<td>[2,3]</td>
<td>[3,13]</td>
</tr>
<tr>
<td>D</td>
<td>[-4,1]</td>
<td>[-2,-1]</td>
<td>[-3,-2]</td>
<td>[0,0]</td>
<td>[1,10]</td>
</tr>
<tr>
<td>E</td>
<td>[-14,0]</td>
<td>[-12,-2]</td>
<td>[-13,-3]</td>
<td>[-10,-1]</td>
<td>[0,0]</td>
</tr>
</tbody>
</table>
Some references

- Teodosiu, Dan and Pollak, Günter. *Discarding Unused Temporal Information in a Production System.*
- **Drools project site:**
  - http://www.drools.org (http://www.jboss.org/drools/)

- **Documentation:**
  - http://www.jboss.org/drools/documentation.html

---

Edson Tirelli  
etirelli@redhat.com  
Lead CEP Designer  
JBoss, a Division of Red Hat