Event Driven Drools

CEP (Complex Event Processing) Explained

Luca Molteni

Before we start:

Drools community

- **Zulip** https://kie.zulipchat.com/ (**#drools** and **#kogito** channels)
- **Stack Overflow** https://stackoverflow.com/questions/tagged/drools)
- **Drools Usage** https://groups.google.com/g/drools-usage

Agenda

- Real time computing
- Event Processing for orchestration
- Drools CEP Setup
- Writing CEP Rules in Drools

Real Time computing

The reponse time of the system is a key component of the result

Example of real time computation vs non real time

Not Real Time: A Company Report

Example of real time computation vs non real time

Real Time: An Airplane Monitoring System

What is an event?

What is event processing?

Drools is not a real time system

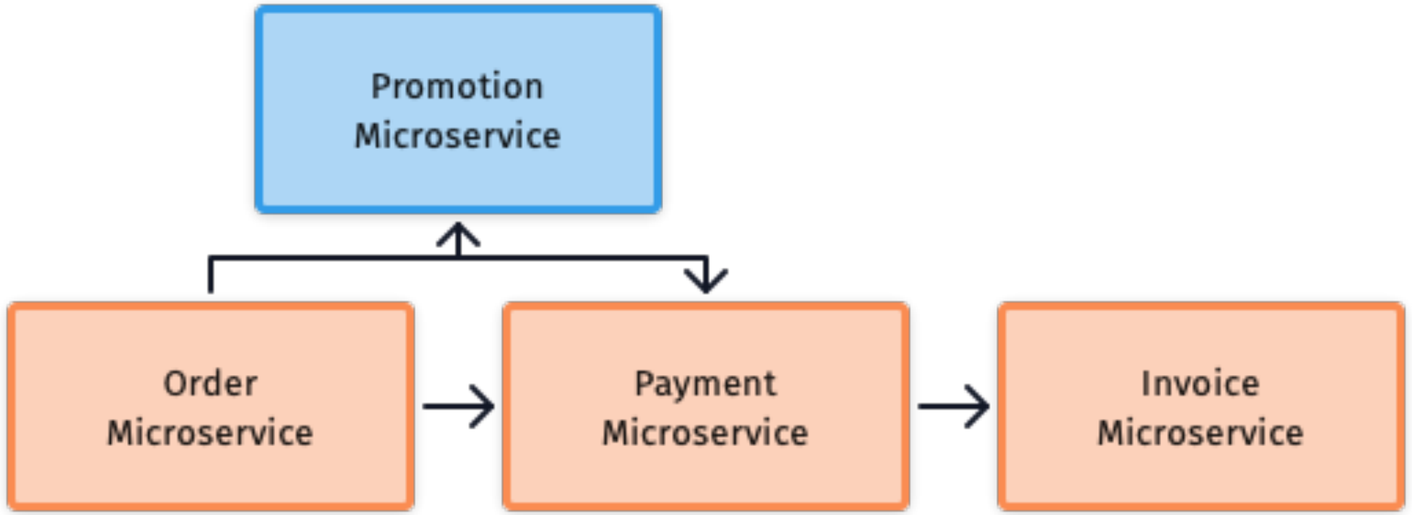
... but it does provide (Complex) Event Processing functionalities

What is CEP

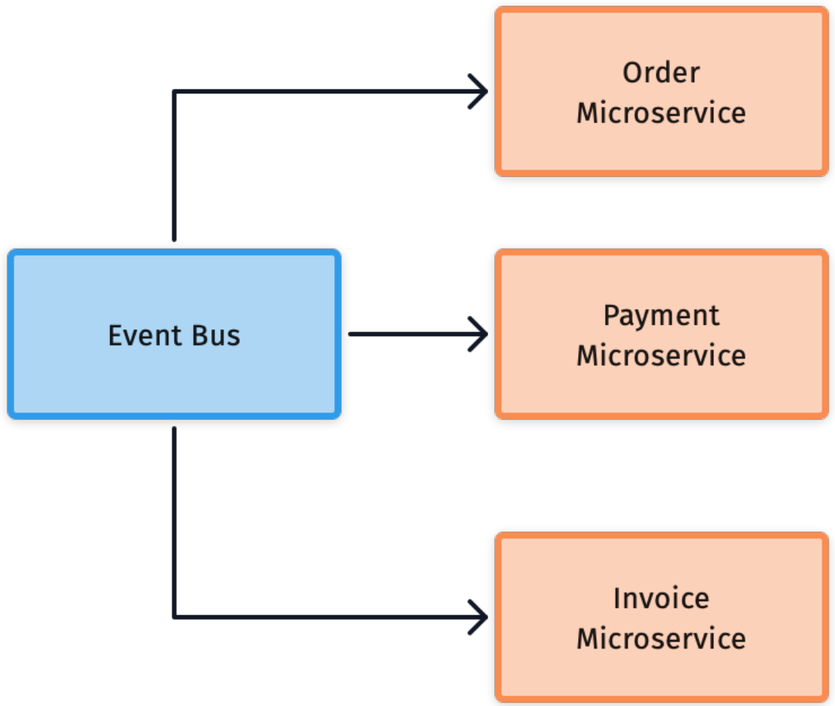
Complex Event Processing

Tightly coupled microservice

Tightly coupled microservice, a new service in between



Decoupling the architecture



Event Driven architecture

- Modern reactive Java: Vert.X
- https://vertx.io/docs/vertx-camel-bridge/java/

Use case scenarios

What is CEP useful for?

- Stock Market
- IOT
- Fraud Detection
- Monitoring
  - https://github.com/lucamolteni/CEPExplained

Questions?

- ~~Real time computing~~
- ~~Event Processing for orchestration~~
- Drools CEP Setup
- Writing CEP Rules in Drools

CEP In Drools - setup

- Cloud vs Stream mode
- Stateful Session
- How to define an Event?
- Punctual vs Interval Event
- Immutability

Cloud mode vs Stream mode

- Cloud Mode (default):
  - Facts have no particolar orders
- Stream Mode: CEP enabled
  - Events are ordered and have timestamp
  - Temporal operators supported
  - Support sliding windows

Cloud mode vs Stream mode

```
<kbase name="CEPExplained" eventProcessingMode="stream" packages="kie.live">
  <ksession name="default" type="stateful" />
</kbase>
```

or

```
import org.kie.api.conf.EventProcessingOption;
import org.kie.api.KieBaseConfiguration;
import org.kie.api.KieServices.Factory;

KieBaseConfiguration config = KieServices.Factory.get().newKieBaseConfiguration();

config.setOption(EventProcessingOption.STREAM);
```

Why we need a stateful session?

- Stateless computation: only user input is needed to get the results
- Stateful computation: need both user input and the state of the system
- We need the memory of the events to correlate events
- Example: a rule that fires if 5 minutes are passed from the last

What is an event in Drools?

- A fact with a timestamp (or twos)
- If a timestamp field is not provided, the insertion time will be used
- `@org.kie.api.definition.type.Role(EVENT)` on the POJO class
- Annotate the declared type

Point-in-time (Punctual) vs Interval Event

- Point-in-time (Punctual) events have only a single timestamp
  - Example: event from central heating system
- Interval events have starting timestamp and a duration
  - Example: phone call

Immutability

- Events are immutable
- We cannot enforce immutability since it's Java
- Events can be enriched with new data, but please don't change the existing

CEP In Drools - setup summary

- Cloud vs Stream mode
- Stateful Session
- How to define an Event?
- Punctual vs Interval Event
- Immutability

CEP In Drools - writing CEP rules

- DEMO
- Testing
- Temporal Operators
- Sliding Windows
- Memory Management
- Continuous Versus Discrete Rule Firing

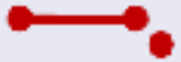
DEMO

Session Clock

Pseudo Clock vs Real Time Clock

- A Virtual clock that lets user manage time programmatically
- Other uses: synchronizing time between different instances

All the Temporal Operators

	Point-Point	Point-Interval	Interval-Interval
A before B			
A meets B			
A overlaps B			
A finishes B			
A includes B			
A starts B			
A coincides B			

Temporal Operators definition in the documentation

before

This operator specifies if the current event occurs before the correlated event. This operator can also define an amount of time before which the current event can precede the correlated event, or a delimiting time range during which the current event can precede the correlated event.

For example, the following pattern matches if `$eventA` finishes between 3 minutes and 30 seconds and 4 minutes before `$eventB` starts. If `$eventA` finishes earlier than 3 minutes and 30 seconds before `$eventB` starts, or later than 4 minutes before `$eventB` starts, then the pattern is not matched.

```
$eventA : EventA(this before[3m30s, 4m] $eventB)
```

You can also express this operator in the following way:

```
3m30s <= $eventB.startTimestamp - $eventA.endTimeStamp <= 4m
```

Sliding Windows

- Powerful feature to filter facts from working memory or entry point
  - Length Based
  - Time Based

Sliding Windows

Time based Example: Average Temperature

```
rule "Sound the alarm if temperature rises above threshold"
when
  TemperatureThreshold($max : max)
  Number(doubleValue > $max) from accumulate(
    SensorReading($temp : temperature) over window:time(10m),
    average($temp))
then
  // Sound the alarm.
end
```

Sliding Windows

Length based Example: Avarage Temperature

```
rule "Sound the alarm if temperature rises above threshold"
when
  TemperatureThreshold($max : max)
  Number(doubleValue > $max) from accumulate(
    SensorReading($temp : temperature) over window:length(100),
    average($temp))
then
  // Sound the alarm.
end
```

Declared Sliding Windows

```
declare window
  @doc("last 100 temperature events")
  SensorReading($temp : temperature) over window:length(100)
end
```

Memory Management

- Memory is not infinite, memory allocation needs have to be addressed when thinking about our domain model
- Two types of memory management
  - Explicit Expiration
  - Inferred Expiration

Memory Management Example: Explicit Expiration

```
declare StockPoint
  @expires( 30m )
end
```

Memory Management Example: Inferred expiration

```
rule "Correlate orders"
when
  $bo : BuyOrder($id : id)
  $ae : AckOrder(id == $id, this after[0,10s] $bo)
then
  // Perform an action.
end
```

Discrete vs Continuous Rule Firing

Two ways to evaluate rules

* In Discrete rule firing, the user will decide when to run the rules, by calling `fireAllRules` on the session

* In Continuous rule firing, Drools will automatically fire the rules when an event matches

 * This is enabled using the `fireUntilHalt` method on the session and it will spawn a different thread

Kogito Distributed CEP (HA)

Kogito

- Same features as Drools but cloud native
- High Availability
- Support CloudEvents messages to trigger rules

The end

Questions?