2023
CEP - Complex Event processing

Knowledge based systems

SADRŽAJ

01

Complex Event Processing

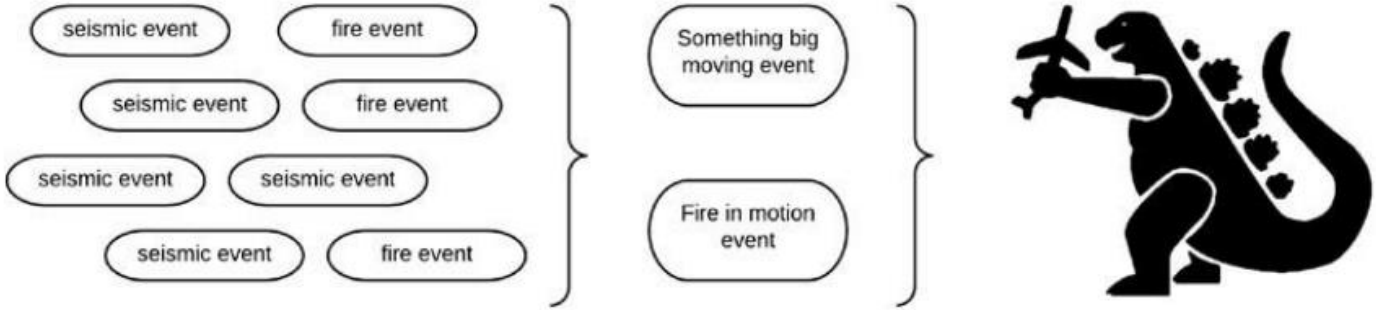
01 Complex Event Processing

- Događaj (event) se definiše kao zapis (record) neke značajne promene stanja domena aplikacije u datom vremenskom trenutku.
- Complex Event Processing ili CEP je koncept za obradu događaja čiji je zadatak procesiranje više događaja sa ciljem identifikovanja značajnih događaja u sklopu „oblaka“ događaja
- CEP koristi tehnike za detekciju složenih obrazaca,apstrakciju i korelaciju događaja, tehnike za hijerarhiju događaja, kao i tehnike za detekciju veza između događaja kao što su uzročnost, članstvo, vreme...
- CEP nastoji da pronađe interesantne događaje u moru događaja, da pronađe veze između njih i stvori nove podatke.

01 Complex Event Processing

- Complex Event Processing (CEP) postaje vodeća tehnologija za kreiranje i upravljanje informacionim sistemima kao što su:
  - Business ActivityMonitoring
  - Business Process Management
  - Enterprise Application Integration
  - Event–Driven Architectures
  - Network and business levelSecurity
  - Real time conformance to regulations and policies

01 Complex Event Processing



Mastering JBoss Drools 6

02

Events

02 Events

- Dva osnovna tipa događaja:
  - Punctual events: Događaji koji su se desili u određenom vremenskom trenutku.
  - Interval events: Događaji koji imaju dva značajna vremenska trenutka: trenutak nastajanja i trenutak završetka.
- Obično su imutabilni
- Imaju upravljiv životni ciklus
- Imaju jaka vremenska ograničenja – događaji su vremenski povezani

03

Declaring Events

03 Declaring Events

- Deklarisanje događaja u Drools-u se vrši dodavanjem atributa činjenicama:
  - Role of the type: Jedini obavezni metapodatak pomoću kojeg definišemo neki tip kao događaj. Postoje dva tipa: fact i event.
  - Timestamp: Opcioni property. Označava atribut modela koji definiše trenutak kada se događaj dogodio. Ukoliko nije navedeno, timestamp svake instance događaja biće trenutak kada je taj događaj ubačen u radnu memoriju.
  - Duration: Opcioni property. Označava atribut modela koji definiše trajanje događaja. Ukoliko nije navedeno, događaj se tretira kao punctual event. Ukoliko želimo da događaj tretiramo kao interval event, ovo polje postaje obavezno.
  - Expires: Opcioni property. String koji označava koliko dugo će događaj biti prisutan u radnoj memoriju, pre automatskog brisanja.

03 Declaring Events

```java
@Role(Role.Type.EVENT)
@Timestamp("executionTime")
@Expires("2h30m")
public class TransactionEvent implements Serializable {

    private static final long serialVersionUID = 1L;
    private Date executionTime;
    private Long customerId;
    private Double totalAmount;
```

```
declare PhoneCallEvent
    @role(event)
    @timestamp(whenDidWeReceiveTheCall)
    @duration(howLongWasTheCall)
    @expires(2h30m)
    whenDidWeReceiveTheCall: Date
    howLongWasTheCall:Long
    callInfo: String
end
declare SuspiciousCustomerEvent
    @role(event)
    customerId: Long
    reason: String
end
```

04

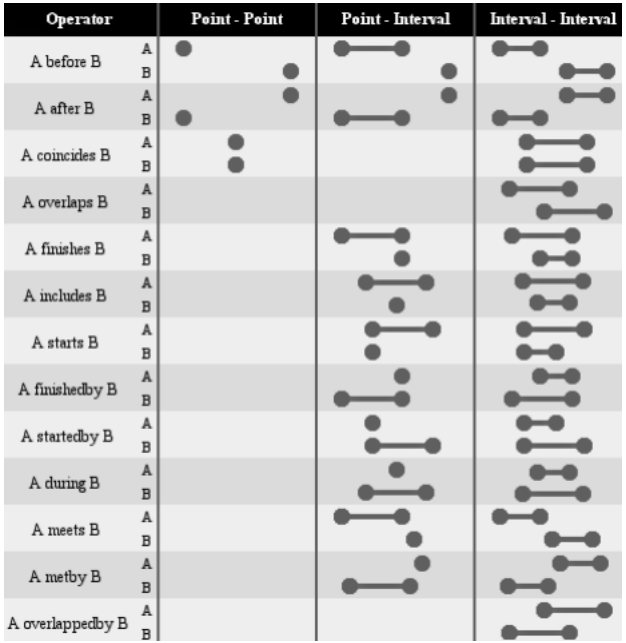
Temporal operators

04 Temporal Operators

```
declare MyEvent
    @role(event)
    @timestamp(executonTime)

end

rule "my first time operators example"
    when
        $e1: MyEvent()
        $e2: MyEvent(this after[5m] $e1)
    then
        System.out.println("We have two events 5 minutes apart ");
end
```



05

Event-driven architecture

05 Event-driven architecture

- Ideja event-driven arhitekture (EDA) jeste da se komponente klasifikuju u četiri kategorije:
  - Event Producer: Njihova uloga u EDA je da budu kreatori događaja. Sve što može da stvori događaj smatra se Event Producer–om, bez obrzira na to da li se radi o hardverskom senzoru, poslovnom procesu ili bilo komdrugom obliku aplikacije koja može proizvoditi nove događaje.
  - Event Consumer: Njihova uloga jeste da sluša događaje koje proizvode druge komponente. Ove komponente mogu da variraju od jednostavnih listener–a, do kompleksnih kontrolnih tabli. Obično predstavljaju završne rezultate cele arhitekture i tačku gde se ti rezultati predaju spoljnom svetu

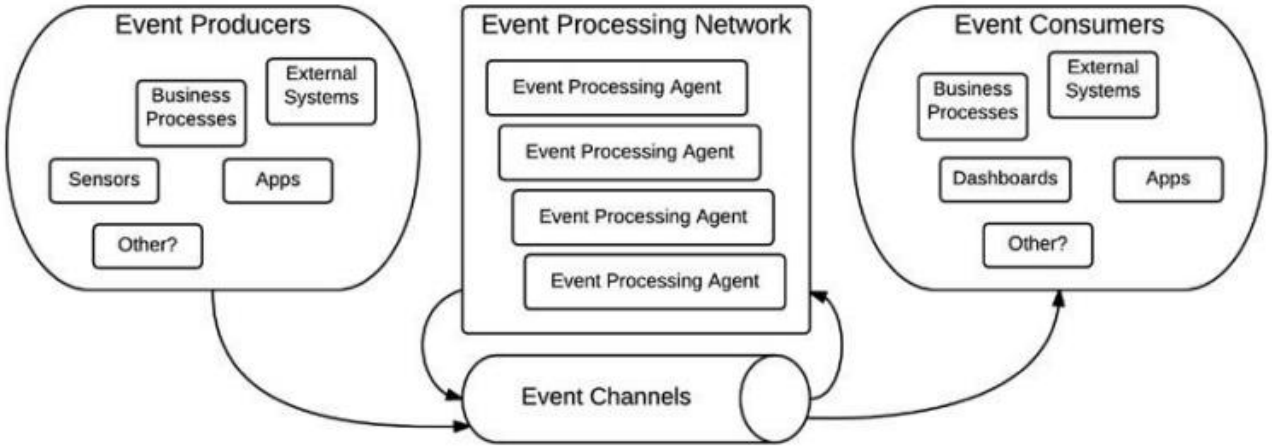
05 Event-driven architecture

- Event Channels – predstavljaju komunikacione protokole između drugih komponenti. Ove komponente enkapsuliraju druge komponente koje se koriste za predošenje događaja iz jedne komponente u drugu.Java Message Service (JMS)

05 Event-driven architecture

- Event Processing Agents: ključne komponente koje grupišu događaje kako bi otkrile i procesirale složene događaje. U Drools-u, svako pravilo koje radi sa CEP-om posmatra se kao event processing agent. Grupisanje agenata sa ciljem detektovanja i reagovanja u složenijim situacijama naziva se event processing network.

05 Event-driven architecture



06

Entry Points

06 Event-driven architecture

- Entry points u Drools mogu se posmatrati kao način podele radne memorije.
- Svaka sesija može imati više ulaznih tačaka koje se mogu koristiti kao način za pronalaženje izvora dolazećih podataka.
- U kompleksnijim procesiranjima događaja, ulazne tačke predstavlaju odličan način definisanja različitih izvora podataka.
- U Drools-u se ulazne tačke definišu implicitno, tako što se koriste u pravilima.

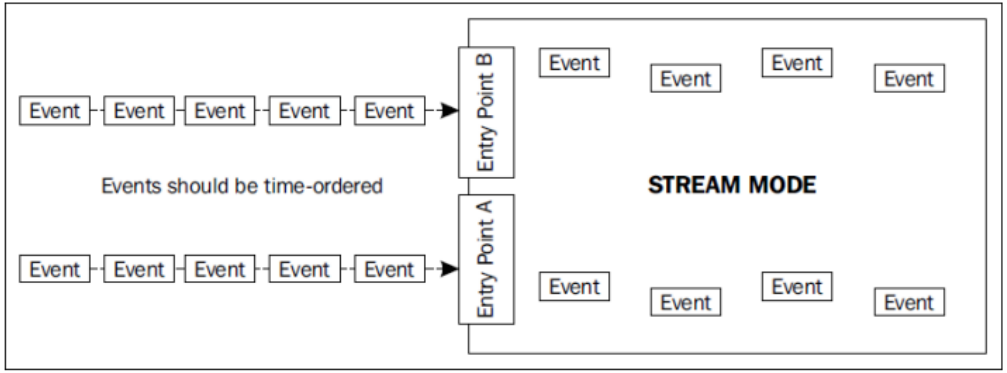
06 Event-driven architecture - Adding ecents to entry point

```
// create your rulebase and your session as usual
KieSession session = ...

// get a reference to the entry point
EntryPoint atmStream = session.getEntryPoint( "ATM Stream" );

// and start inserting your facts into the entry point
atmStream.insert( aWithdrawRequest );
```

06 Event-driven architecture



07

Sliding Windows

07 Sliding Windows

- Sliding windows - način za obeležavanja događaja od interesa definisanjem prozora koji se konstrantno kreće
- Dva tipa:
  - Length-based sliding windows over window:length(10)
    - Najjednostavniji tip
    - U obzir uzima poslednjih događaja koji su ubačeni u radnu memoriju
    - Svaki put kada se novi događaj doda, poslednji element u prozoru biva zamenjen novim
  - Time-based sliding windows over window:time(2m)
    - Omogućuje korisniku da piše pravila koja će uzimat samo one događaje koji su se desili u poslednjih N jedinica vremena

07 Sliding Windows Length-based

```
rule "last 6 transactions are more than 100 dollars"
    when
        Number(doubleValue >100.00) from accumulate(
            TransactionEvent($amount:totalAmount)
            over window: length(6), sum($amount)
        )
    then
        // todo
end
```

07 Sliding Windows Time-based

```
rule "obtain last five hours of operation"
    when
        Number() from accumulate(
            TransactionEvent($amount:totalAmount)
            over window: time(5h), sum($amount)
        )
    then
        // todo
end
```

07 Sliding Windows Declared

- Sliding window se tipično definiše u okviru pravila koje ga koristi.
- Problem - pravila koja filtriraju elemente kroz isti prozor moraju da ga redefinišu u svakom pravilu.
- Rešenje - window declaration..
- Window declaration - definiše prozor kao prethodno utvrđena komponenta koja se po imenu poziva u bilo kom pravilu.

07 Sliding Windows declared

```
declare window Beats
    @doc("last 10 seconds heat beats")
    HeartBeat() over window:time(10s)
    from entry-point "heart beat monitor"
end

rule "beats in the window"
    when
        accumulate(
            HeartBeat() from window Beats,
            $cnt:count(1)
        )
    then
        // todo
end
```

07 Stream processing configuration

```
<kbase name="cepKbase" eventProcessingMode="stream" packages="chapter06.cep">
    <ksession name="cepKsession"/>
</kbase>
```

```
KieServices ks = KieServices.Factory.get();
KieContainer kc = ks.getKieClasspathContainer();
KieBaseConfiguration kbconf = ks.
newKieBaseConfiguration();
kbconf.setOption(EventProcessingOption.STREAM);
KieBase kbase = kc.newKieBase(kbconf);
```

07 Continuous versus Discrete rule firing

- ***Discrete rule firing*** će okinuti pravila u određenom trenutku u vremenu. Aplikacija će dodati događaje i činjenice u KieSession, i u određenom trenutku će iskoristiti *fireAllRules* metodu da okine pravila koja se podudaraju sa radnom memorijom u tom trenutku.

- ***Continuous rule*** firing će imati specifičnu nit koja je predoređena za okidanje pravila u svakom trenutku kada podaci odgovaraju pravilu. Koristiće *fireUntilHalt* metodu KieSession, dok će jedna ili više niti dodavati događaje i činjenice u KieSession.

07 Session clock

- Pri kreiranju Kie Session u svrhu pokretanja CEP baziranih scenarija moguće je konfigurisati njegov interni sat.
- Runtime clock
  - Sat mašine na kojoj je Drools pokrenut
- Pseudo clock
  - Sat kontrolisan od strane aplikacije
  - Napredovanje vremena se postiže pozivanjem *advanceTime* metode

07 Pseudo clock

```
SessionPseudoClock clock = ksession.
    getSessionClock();
clock.advanceTime(2, TimeUnit.HOURS);
clock.advanceTime(5, TimeUnit.MINUTES);
```

```
<kbase name="cepKbase" eventProcessingMode="stream"
    packages="chapter06.cep">
    <ksession name="cepKsession" clockType="pseudo"/>
</kbase>
```

```
KieServices ks = KieServices.Factory.get();
KieContainer kc = ks.getKieClasspathContainer();
KieSessionConfiguration ksconf = ks.
    newKieSessionConfiguration();
ksconf.setOption(ClockTypeOption.get(
    ClockType.PSEUDO_CLOCK.getId()));
KieSession ksession = kc.newKieSession(ksconf);
```