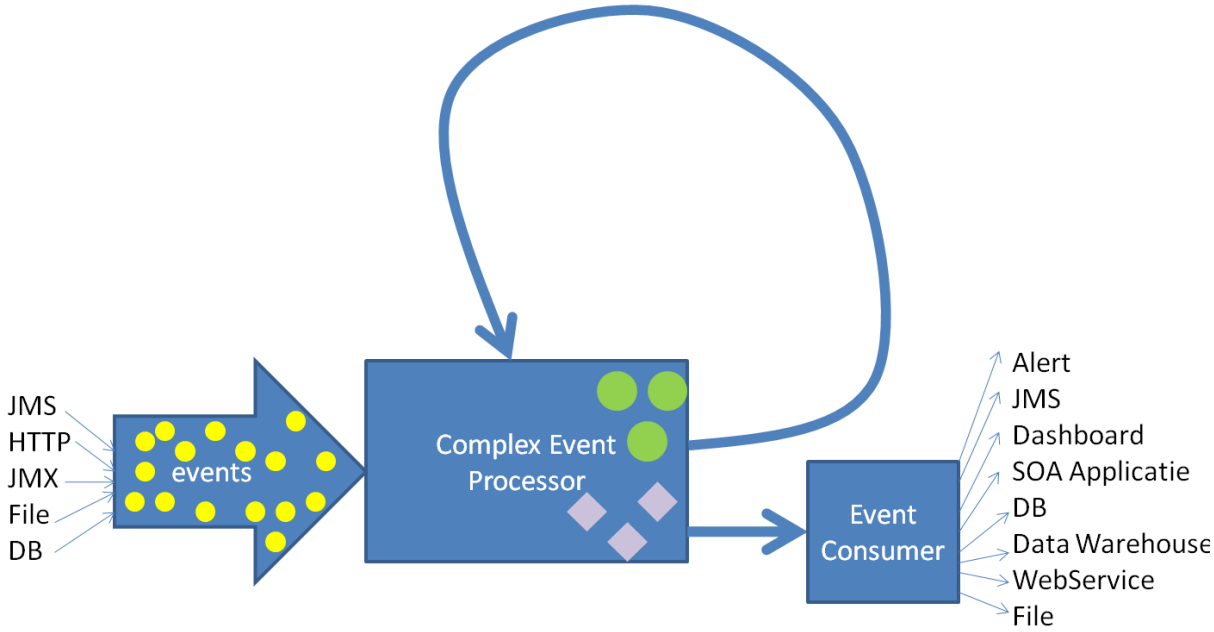
Complex Event Processing - het vervolg

Integratie tussen Complex Event Processing en Java applicaties

In het vorige nummer van Java Magazine stond deel een van deze tweedelige serie over Complex Event Processing (kortweg CEP vanaf nu). In dat artikel hebben we kennisgemaakt met het concept van continue queries –filters die voortdurend de events evalueren die op verschillende kanalen kunnen binnenkomen, op zoek naar uitzonderingen, aggregaties en speciale patronen – waaronder het non-event (het uitblijven van een verwacht event, hetgeen op zichzelf weer betekenisvol is). De queries die in CEP worden uitgevoerd worden geprogrammeerd in speciale talen, veelal leverancier-specifiek. Een soort standaard lijkt de kop op te steken in de vorm van CQL (Continuous Query Language), nauw verwant aan en zelfs deels overlappend met SQL.

Toepassingen van CEP zijn er legio, grotendeels in situaties waar voortdurend grote aantallen events met meestal een beperkte payload binnenkomen en er in real-time evaluaties van die events nodig zijn – om bijvoorbeeld direct fraude te constateren, of om een koffer die in het vliegveld-bagage-afhandelingssysteem dreigt kwijt te raken te lokaliseren of om op een dashboard iedere 15 seconden de actuele stand van zaken aan te geven met bijvoorbeeld het aantal bezoekers van de website en hun click gedrag.

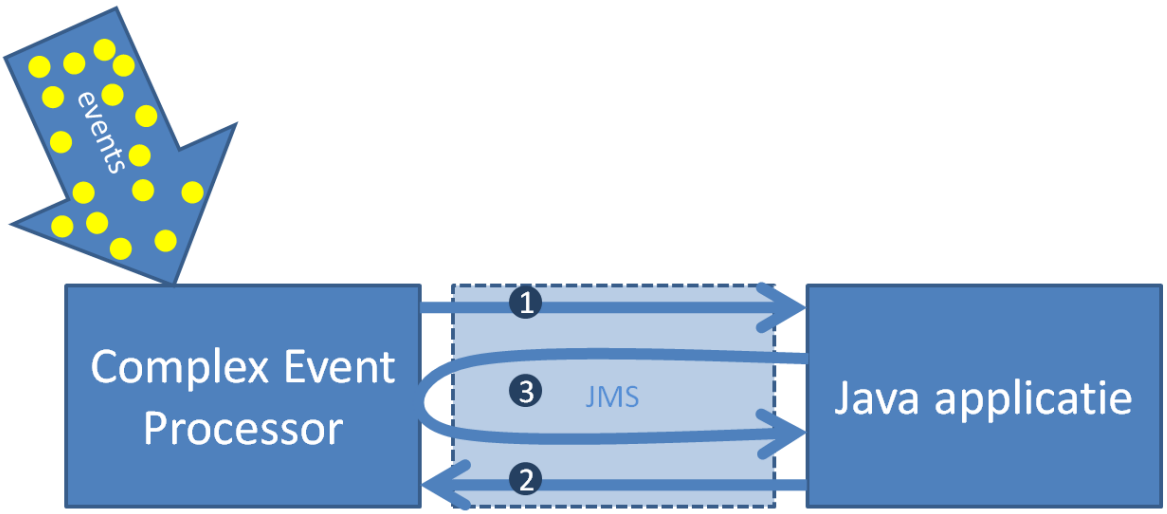
In deze tweede aflevering bekijken we wat de relevantie is van CEP voor Java applicaties en hoe interactie en integratie tussen CEP applicaties en willekeurige Java applicaties eruit zou kunnen zien. Om het verhaal concreet te maken, gaan we aan de slag met een van de vele CEP producten, namelijk het vroegere BEA WebLogic Event Server, tegenwoordig bekend onder de naam Oracle CEP. Dit product ondersteunt zowel een eigen specifieke event processing language (EPL) als ook de opkomende standaard CQL. In dit artikel zullen we alleen van CQL gebruikmaken. De manier van werken met Oracle CEP is sterk vergelijkbaar met de manier waarop je ook met andere producten aan de slag zou kunnen gaan. Dit artikel zou je dus een goede indruk moeten geven hoe je met CEP aan de slag kan gaan, ook als je een andere CEP product inzet.



Figuur 1: *de Complex Event Processor verwerkt events uit allerlei bronnen; de CEP publiceert zijn resultaten ook in de vorm van events – die veelal rijker zijn qua data inhoud en bedrijfsbetekenis en veel kleiner in aantal*

CEP en Java applicaties

Java applicaties kunnen grofweg op drie manieren samenwerken met en gebruikmaken van CEP applicaties.



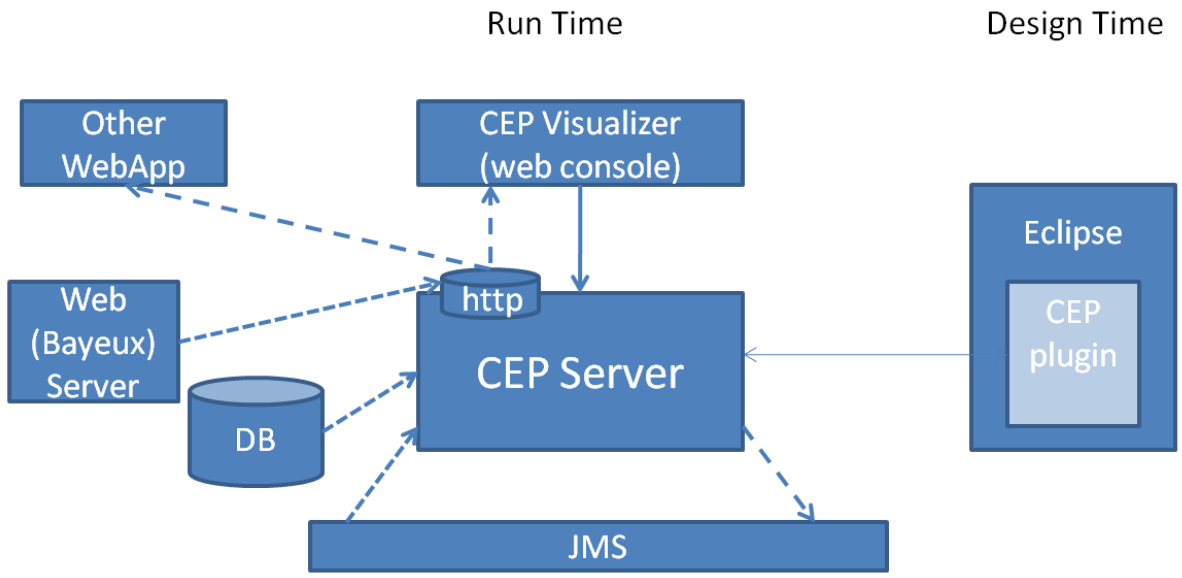
- De Java applicatie laat CEP als een soort pre-processor of filter grote aantallen events verwerken en ontvangt zelf alleen de uitkomsten in de vorm van events: de gevonden uitzonderingen, de resultaten van aggregatie of de constateringen van het voorkomen van patronen

- De Java applicatie publiceert events – bijvoorbeeld trace informatie over de wijze waarop het programma wordt uitgevoerd of de manier waarop de gebruiker door de web-applicatie klikt en navigeert – die via bijvoorbeeld JMS in de CEP belanden
- De Java applicatie publiceert events naar CEP en consumeert de uitkomsten van CEP – bijvoorbeeld alle zoekacties in de website worden aan CEP gemeld die daaruit de populairste zoektermen top 10 destilleert (en continue blijft destilleren)

In dit artikel zullen we van alle drie deze patronen voorbeelden uitwerken.

Oracle CEP – architectuur en installatie

Oracle CEP draait in of beter gezegd als standalone server – die vaag overeenkomsten heeft met een soort uitgeklede WebLogic omgeving. Deze server maakt bij voorkeur gebruik van een JVM met speciale voorzieningen voor real time verwerking, zoals Sun Hotspot of JRockit RealTime. The CEP Server publiceert een web applicatie – de CEP Visualizer – die dienstdoet als beheertool.



De CEP Server consumeert events uit JMS, uit relationele tabellen die zich via een adapter voordoen als continue event producent en op basis van het Bayeux protocol via web server push. Daarnaast kunnen custom adapters worden gecreëerd die als event source dienst doen – en bijvoorbeeld files of RSS feeds lezen om inspiratie voor events te verzamelen.

De resultaten van de verwerking in CEP worden naar JMS gepubliceerd, via de Bayeux-methode naar web clients gepusht of via custom event sinks op een andere manier verwerkt.

De ontwikkelomgeving voor Oracle CEP is Eclipse met een specifieke plugin. De plugin maakt onder andere het visueel ontwikkelen van het EPN (Event Processing Network) mogelijk en biedt ook directe interactie met de CEP Server, handig voor voor starten en stoppen van de server en het deployen en ook debuggen van CEP applicaties.

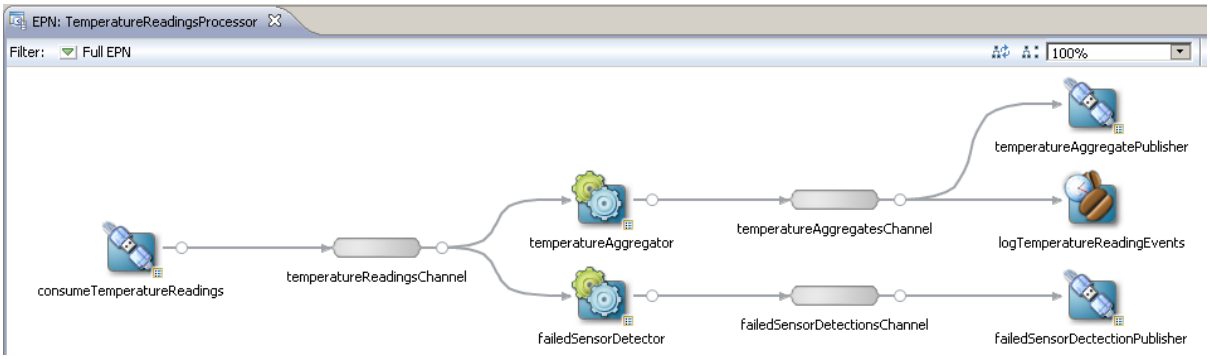
Aan het eind van het artikel staat een verwijzing naar een weblog artikel met de referenties naar de software en de installatie-instructies.

Temperatuur-bewaking

In het eerste voorbeeld kijken we naar een ziekenhuis waar op verschillende plekken specifieke eisen worden gesteld aan de temperatuur: de kamer met de couveuses moet heel warm worden gehouden –24 C, terwijl de koelruimte met transfusiebloed rond de 5 C moet blijven. De kamers waar patiënten verblijven zitten rond de 21C, en de diepvriezers op -12C.

In het ziekenhuis zijn honderden sensoren geplaatst, die elk iedere seconde hun de temperatuur meten en hun signaal op een JMS queue plaatsen. De sensoren zijn steeds in clusters van drie in een bepaalde locatie geplaatst– om individuele schommelingen uit te middelen en om bestand te zijn tegen de uitval van enkele temperatuursensoren zonder direct geen informatie meer te hebben over de temperatuur in een bepaalde ruimte.

Al deze sensoren produceren gezamenlijk vele duizenden signalen per minuut. En de meeste data is nauwelijks interessant voor de achterliggende Java applicatie die de temperatuur onder controle moet houden. In deze situatie wordt CEP ingezet om die duizenden signalen per minuut terug te brengen tot hooguit tientallen per minuut. In plaats van de temperatuur te horen van iedere individuele sensor wordt de Java applicatie alleen geïnformeerd over de gemiddeldes per cluster. En in plaats van iedere seconde een aangepaste waarde te krijgen wordt eens per halve minuut de gemiddelde temperatuur over de afgelopen 45 seconden per cluster uitgerekend en doorgegeven.



Het ontwikkelen van de benodigde CEP applicatie gaat in een paar eenvoudige stappen:

- Creëer een nieuw Oracle CEP project in Eclipse – op basis van het HelloWorld template bijvoorbeeld
- Ontwerp het Event Processing Network (EPN) in de visuele editor
- Configureer de JMS adapters (inbound en outbound)
- Programmeer de CQL processor die de binnenkomende events verwerkt en de uitgaande events produceert

NB: onderaan dit artikel staat een link naar een weblog artikel dat alle code bevat die hier op hoofdlijnen wordt beschreven.

In een EPN maken we gebruik van met name adapters (om events het EPN binnen te krijgen of ze juist naar buiten te publiceren), channels – om events aan een of meerdere geïnteresseerde afnemers aan te leveren en processors om de events die uit een of meerdere kanalen aankomen te verwerken.

In dit geval lezen we TemperatureReadings van een JMS queue met een JMS adapter. De betreffende events komen aan als JMS MapMessages met drie velden: sensor id, cluster id en temperatuurwaarde. Een timestamp wordt door de CEP zelf toegekend. We definieren een eenvoudige Java Bean met properties voor deze drie velden. Deze event bean belichaamt het event en zal door de channels van het EPN stromen.

De configuratie file onder het EPN ziet er als volgt uit:

```
<wlevs:adapter id="consumeTemperatureReadings" provider="jms-inbound">
  <wlevs:listener ref="temperatureReadingsChannel" />
</wlevs:adapter>

<wlevs:channel id="temperatureReadingsChannel"
               event-type="TemperatureReading">
  <wlevs:listener ref="temperatureAggregator" />
</wlevs:channel>

<wlevs:processor id="temperatureAggregator">
  <wlevs:listener ref="temperatureAggregatesChannel" />
</wlevs:processor>

<wlevs:channel id="temperatureAggregatesChannel"
               event-type="TemperatureFinding">
  <wlevs:listener ref="temperatureAggregatePublisher" />
</wlevs:channel>

<wlevs:adapter id="temperatureAggregatePublisher" provider="jms-outbound"
/>
```

De configuratie van de beide JMS adapters staat in de META-INF/wlevs/config.xml en specificeert de toegangsdetails voor de JMS Queues.

De temperatureAggregator processor staat in een aparte file beschreven: temperatureAggregator.xml. De kern van deze file is de CQL query, tegelijkertijd het hart van de CEP applicatie:

```
select avg(temperatureReadingsChannel.temperature)  as temperature
,      temperatureReadingsChannel.clusterId as clusterId
from   temperatureReadingsChannel [range 45 slide 30]
group by temperatureReadingsChannel.clusterId
```

En lijkt dat op SQL of niet… Deze continue query leest voortdurend de events uit het temperatureReadingsChannel en berekent de gemiddelde temperatuur per cluster. Het gemiddelde wordt steeds berekend over een range van 45 seconden – alle waardes die in de afgelopen 45 seconded zijn gelezen uit het channel. En deze berekening vindt iedere 30 seconden (slide 30) opnieuw plaats. Hoeveel events er ook per cluster in het kanaal belanden, er komt er per cluster maar een per 30 seconden uit – die via het channel op de JMS outbound adapter belandt en door de Java applicatie uit de queue wordt gelezen.

De applicatie die je kunt downloaden bij dit artikel bevat een sensor simulator – een eenvoudige Java class die als custom adapter fungeert in een tweede CEP applicatie met als taak een dozijn sensoren te simuleren door meetwaarden te gaan publiceren op de JMS Queue.

Defect-detectie

Zolang alle sensoren goed functioneren zal de bovenstaande applicatie elke 30 seconden de gemiddelde temperaturen rapporteren van alle clusters. Maar er kan natuurlijk wel eens een sensor kapot gaan. En hoe kom je daar nou achter? De gemiddelde waardes blijven wel komen, ook al zijn ze over twee in plaats van drie sensoren berekend.

Het constateren van non-events – een aanwijzing in dit geval voor een kapotte sensor – is een kolfje naar de hand van CEP applicaties. Door op zoek te gaan naar het patroon dat bestaat uit een event in combinatie met het uitblijven van een tweede verwacht event kunnen we een sterke aanwijzing vinden over een kaduke sensor.

In het EPN van de temperatuur processor is een kleine aanpassing voldoende om op zoek te gaan naar deze aanwijzingen.

Allereerst een listener in het channel temperatureReadingsChannel om de events met meetwaarden ook naar de nieuwe processor te brengen

```
<wlevs:listener ref="failedSensorDetector" />
```

Dan de failedSensorDetector processor die het failedSensorDetectionsChannel voedt dat uitmondt in de failedSensorDectectionPublisher adapter die events voor falende sensoren meldt op een JMS Queue.

```
<wlevs:processor id="failedSensorDetector">
  <wlevs:listener ref="failedSensorDetectionsChannel" />
</wlevs:processor>
<wlevs:channel id="failedSensorDetectionsChannel"
               event-type="FailedSensorDetection">
  <wlevs:listener ref="failedSensorDectectionPublisher" />
</wlevs:channel>
<wlevs:adapter id="failedSensorDectectionPublisher" provider="jms-
outbound"/>
```

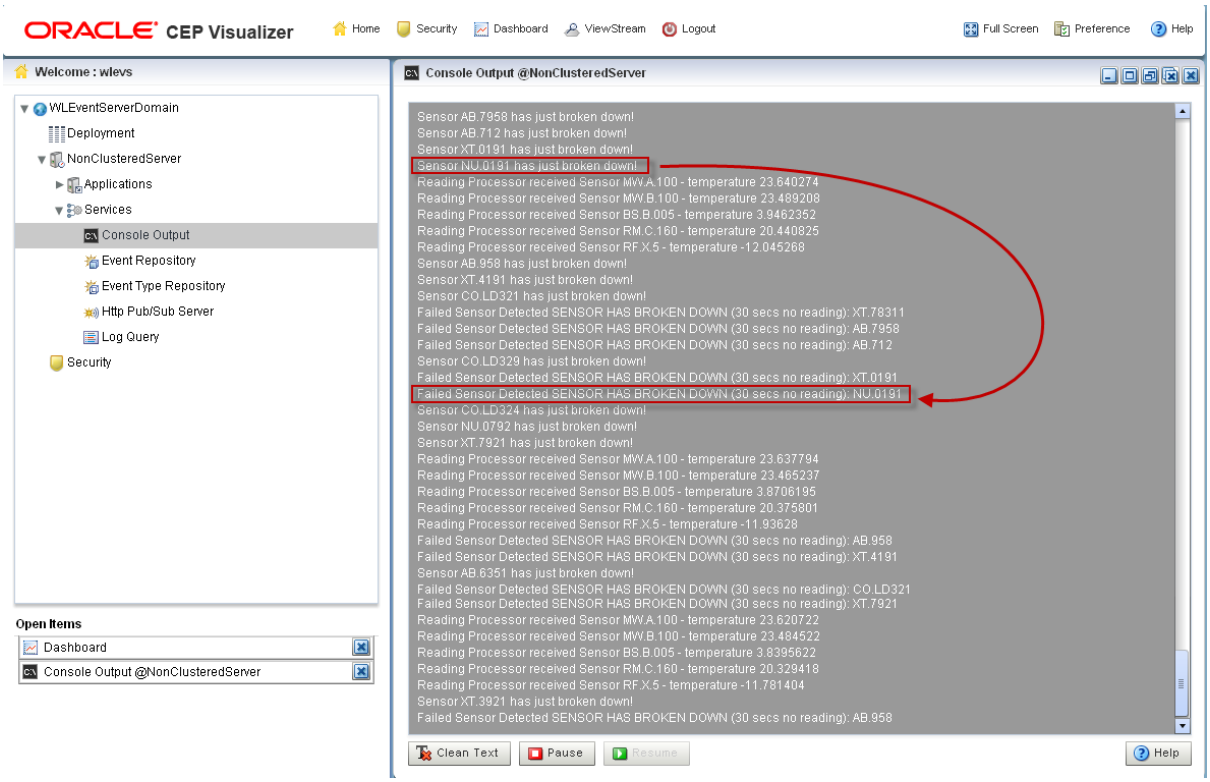
Ook hier geldt dat de CQL query de kern van de zaak vormt:

```
select 'SENSOR HAS BROKEN DOWN (30 secs no reading): '
      ||sensorReadings.sensorId  as sensorId
,      sensorReadings.clusterId as clusterId
from   temperatureReadingsChannel
MATCH_RECOGNIZE
( partition by sensorId
  MEASURES A.sensorId as sensorId
  ,        A.clusterId as clusterId
  all matches
  include timer events
  PATTERN(A B*)
```

```
  duration multiples of 30
  DEFINE A as A.temperature > -100, B as B.sensorId != A.sensorId
) as sensorReadings
```

Deze query is eventjes wat complexer dan de vorige – en ook wat minder standaard SQL achtig. Opnieuw worden events verwerkt die op het temperatureReadingsChannel verschijnen. De MATCH_RECOGNIZE sectie gaat op zoek naar patronen in de eventstroom. In dit geval zoeken we naar een combinatie van A gevolgd door een of meer Bs. A is iedere willekeurige temperatuurmeting en B is een meting door een andere sensor dan A. Een B event kan niet voorkomen zou je zeggen, omdat we ook een partition by uitvoeren en wel op sensorId – en er dus alleen maar events met hetzelfde sensorId met elkaar worden vergeleken. Behalve dan als er op een gegeven geen metingen meer binnenkomen van een bepaalde sensor. De laatste ontvangen meting is het A event in ons patroon en als er na 30 seconden geen ander event van de sensor is ontvangen genereert CEP een soort heartbeat event voor de partitie van de betreffende sensor – en treedt het B event toch op. Als er een heartbeat event nodig is na 30 seconden, dan wordt het AB* patroon geconstateerd voor de betreffende sensor en wordt er een 'kapotte sensor' event gepubliceerd. Zo wordt een non-event (geen teken van leven) omgezet in een actief signaal.

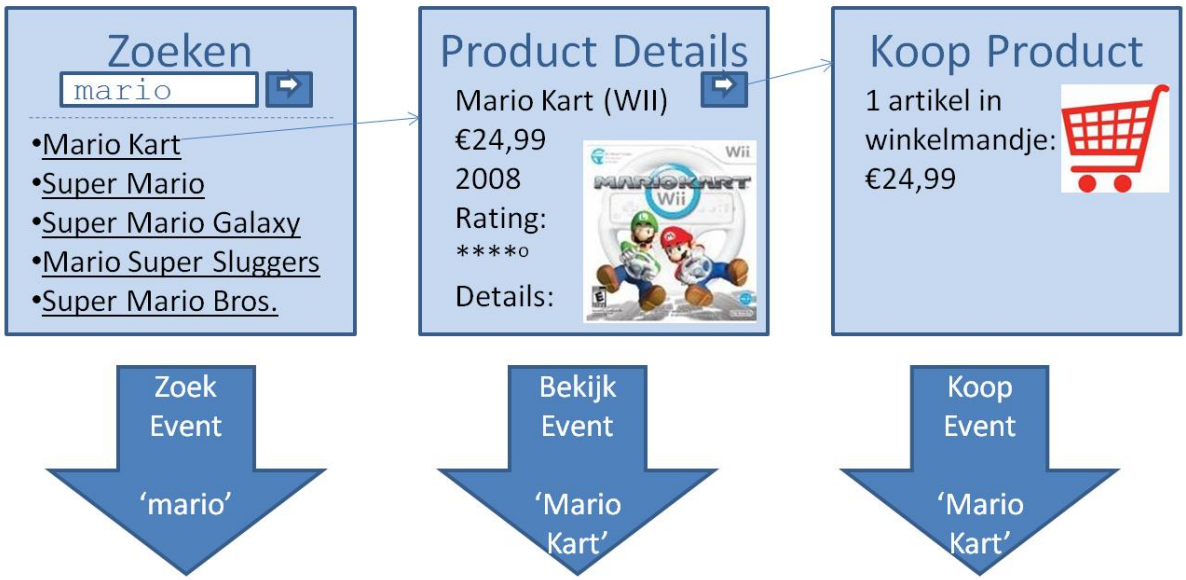
De figuur toont de CEP Visualizer, de administratie web applicatie voor Oracle CEP. De console output toont hoe de sensor simulator af en toe een sensor 'uitschakelt' – en hoe de CEP applicatie daar enige tijd later (pakweg 30 seconden) achter komt.



Analyse van klik- en koopgedrag in de Web Winkel

Dit tweede voorbeeld doet het in zekere zin andersom: in plaats dat CEP filtert namens de Java applicatie en alleen de bevindingen doorgeeft, is het hier de Java applicatie die het initiatief heeft in

de richting van CEP. We hebben te maken met een eenvoudige web winkel. Bezoekers kunnen op de eerste pagina een zoekopdracht laten uitvoeren. De resultaten voor de opgegeven zoekterm worden getoond. De bezoeker kan een zoekresultaat aanklikken en navigeert dan naar de detail pagina. Daar kan hij of zij besluiten het product te kopen.



We kunnen CEP inzetten om een paar vragen te beantwoorden. Daartoe moet de web applicatie een aantal events publiceren die door CEP kunnen worden verwerkt. JMS is het voordehandliggende medium tussen de web applicatie en de Complex Event Processor.

De events die de web applicatie zou moeten publiceren – met allemaal in elk geval de sessie identificatie in de payload:

- Zoek-event met als payload de zoekterm die de gebruiker heeft ingevoerd
- Bekijk-event met als payload het product dat de gebruiker bekeken heeft
- Koop-event met als payload het product dat de gebruiker koopt

Op basis van deze events kan CEP aan de slag op zoek naar antwoorden op ondermeer de volgende vragen:

1. Wat zijn de populairste zoektermen van dit moment (bijvoorbeeld de top 10 van de afgelopen 24 uur) – dit is een vrij eenvoudige aggregatie – gegroepeerd op zoekterm en met om de zoveel uur een rapportage in de vorm van tien events voor de top zoek-termen
2. Wat zijn de populairste waardeloze zoektermen? Dat zjin zoektermen die de gebruikers kennelijk niet de door hen gewenste resultaten opleveren omdat na het zoek-event met die term niet een bekijk-event volgt, maar een nieuw zoek-event of helemaal geen event meer.
3. Wat zijn kennelijk de meest succevolle zoektermen? Hier zoeken we naar het patroon van zoek-event gevolgd door een of meerdere bekijk–events en leidend tot een koop-event zonder dat er nog een ander zoek-event tussen zit
4. Wat zijn items die wel vaak bekeken maar niet vaak gekocht worden? Dat zijn producten waarvoor het percentage kijk+koop van het totaal aantal kijk-events laag is. Ook deze lijst kan bijvoorbeeld iedere 12 of 24 uur worden opgehoest door CEP.

Ontwikkeling van de CEP applicatie

Opnieuw maken we een nieuw Oracle CEP project in Eclipse. In dit project maken we een EPN met een inbound adapter die events leest uit een JMS Queue (en voor test doeleinden ook gebruik zou kunnen maken van een adapter die data uit een CVS file leest). De adapter is via een channel verbonden met een CQL processor, die de volgende queries bevat op de webWinkelEvents. Het webWinkelEvent bevat vier velden: eventType, sessieIdentificatie, zoekterm en productNaam.

```
<view id="V1">
<![CDATA[
  select count(webWinkelEventsChannel.sessieId) as aantal
  ,      webWinkelEventsChannel.zoekterm as zoekterm
  from   webWinkelEventsChannel [range 35 slide 15]
  where  webWinkelEventsChannel.eventType ='zoek'
  group by webWinkelEventsChannel.zoekterm
  ]]>
</view>
<view id="V2">
<![CDATA[
  select V1.aantal as aantal
  ,      V1.zoekterm as zoekterm
  from V1
  order by aantal desc rows 3
]]>
</view>
<query id="Top3Zoektermen">
<![CDATA[
  RSTREAM(
    SELECT aantal
    ,      zoekterm
    FROM   V2
    )
]]>
</query>
```

De view V1 telt de zoektermen die in de zoek events binnenkomen, telkens over de afgelopen 35 seconden en met iedere 15 second een nieuw resultaat. View V2 brengt het resultaat van V1 terug tot alleen de top 3 van zoekresultaten en de query Top3Zoektermen tenslotte maakt van de relatie (de uitkomst van V2) weer een event stream (die we aan JMS kunnen doorgeven). Uiteindelijk wordt allleen het resultaat van de query in het uitvoerkanaal gestopt:

Populairste waardeloze zoektermen

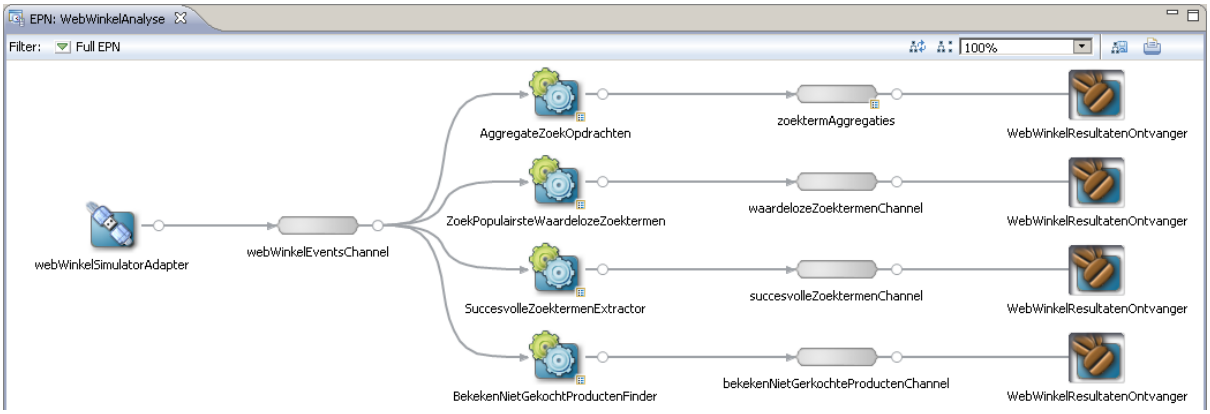
De zoektermen die wel veel gebruikt worden maar geen bevredigend resultaat oplevereden voor de gebruiker (omdat deze direct daarna een andere zoekterm ging proberen). We zoeken deze met een patroon match die zoekt naar de combinatie van zoek-event onmiddellijk gevolgd door een ander zoek-event binnen dezelfde sessie.

```
<view id="V1">
<![CDATA[
  select waardelozeZoektermen.zoekterm as zoekterm
  from   webWinkelEventsChannel
  MATCH_RECOGNIZE
  ( MEASURES A.zoekterm as zoekterm
    all matches
    PATTERN(A B+)
    DEFINE A as A.eventType='zoek'
```

```
        , B as B.eventType='zoek' and B.sessieId = A.sessieId
  ) as waardelozeZoektermen
]]>
</view>
<query id="WaardelozeZoektermen">
<![CDATA[
  SELECT count(*) as aantal
  ,      zoekterm
  FROM   V1 [range 30 slide 10]
  group by zoekterm
]]>
</query>
```

De view V1 verwerkt de zoek events en zoekt naar alle zoek-events die direct gevolgd worden door een ander zoek-event met hetzelfde sessieId – een duidelijk kenmerk van een mislukte zoekterm. De query telt vervolgens iedere 10 seconden het aantal voorkomens van de zoekterm – in de afgelopen 30 seconden.

Het blog-artikel waarnaar wordt verwezen bevat ook de code voor de andere twee zoekvragen: de meest succesvolle zoektermen (gevolgd door een of meer bekijk events en uiteindelijk een koop-event) en wat zijn producten die wel vaak bekeken maar niet vaak gekocht worden.



Terugvoeden naar de Java applicatie

De CEP applicatie kan zijn bevindingen op meerdere manieren beschikbaar stellen, aan verschillende afnemers. Een van de afnemers zou in dit geval ook prima de web applicatie kunnen zijn die ook de bron events publiceert. De populairste zoektermen zouden bijvoorbeeld in een tag-cloud kunnen worden gepresenteerd en de producten die wel bekeken maar niet gekocht worden kunnen meer of juist minder prominent worden vermeld of zelfs uit het assortiment worden verwijderd.

De web applicatie kan via JMS, maar ook via een http Pub/Sub kanaal de door CEP gepubliceerde events afvangen. Ook kan CEP deze in een database tabel inserten – vanwaar ze voor de web applicatie benaderbaar zijn. Tenslotte kan een CEP processor events doorsturen naar een custom adapter (een 'EventSink') waarin we zelf de Java code schrijven die iets met de ontvangen events gaat doen:

```
public class WebWinkelResultatenOntvanger implements StreamSink {
  EventTypeRepository  etr_;
```

```
  @Service
  public void setEventTypeRepository(EventTypeRepository etr) {
    etr_ = etr;
  }
  public void onInsertEvent(Object event) {
    EventType eventType = etr_.getEventType(event);
    if (eventType.getTypeName().equals("WebWinkelZoektermAggregatieEvent"))
{
      String zoekterm = (String)eventType.getPropertyValue(event,
"zoekterm");
      Integer aantal = (Integer)eventType.getPropertyValue(event,
"aantal");
      System.out.println("Top Zoekterm: " + zoekterm+" - aantal: "+aantal);
    }
  }
}
```

Conclusies

Als je aan een applicatie werkt die geacht wordt in real time te reageren op honderden of duizenden impulsen per seconde sta je voor een flinke uitdaging. Qua performance en continue verwerking van de aantallen signalen. Maar ook domweg vanwege de complexiteit van het zoeken naar patronen in die brei aan gegevens en het openhouden, doorschuiven van en aggregeren over tijd-vensters die langs diverse dimensies moeten worden gepartitioneerd.

In dit soort situaties zou je inzet kunnen overwegen van een Complex Event Processor – een component die speciaal is toegerust om deze voortdurende stortbui van events te absorberen en met de uitgekristalliseerde informatie te komen – in de vorm van rijkere bedrijfsevents. Deze events worden vaak via JMS aan de Java applicatie doorgeven die ze gaat interpreteren en omzetten in acties.

Java applicaties kunnen ook zelf de producent zijn van kleinschalige events – bijvoorbeeld in de vorm van trace gegevens of bij het publiceren van web applicatie klik-en navigatiegedrag. In zo'n geval stromen de events ook weer vaak via JMS naar de CEP die er zinnige informatie uithaalt. De Java applicatie die de events produceert zou ook heel goed de conclusies van CEP weer kunnen consumeren en toepassen – bijvoorbeeld door de publicatie van de top 10 van populaire zoektermen.

Er is een groot aantal producten voor Complex Event Processing, die op onderling sterk vergelijkbare manier worden gebruikt. Dit artikel gebruikt Oracle CEP – maar blijft grotendeels van toepassing als een ander product wordt ingezet.

Referenties

Op http://technology.amis.nl/blog/7193/complex-event-processing-java-magazine-sources-references vind je hyperlinks naar de Oracle CEP software (Server en Eclipse Plugin) en de installatie-instructies voor de ontwikkelomgeving. Ook kan je daar de source code downloaden van de applicaties die in dit artikel zijn beschreven.