Oracle CEP

Minor Review
19th April 2011

Oracle CEP

- What is “Complex Event Processing”?
- What is Oracle CEP?
- Applications at CERN

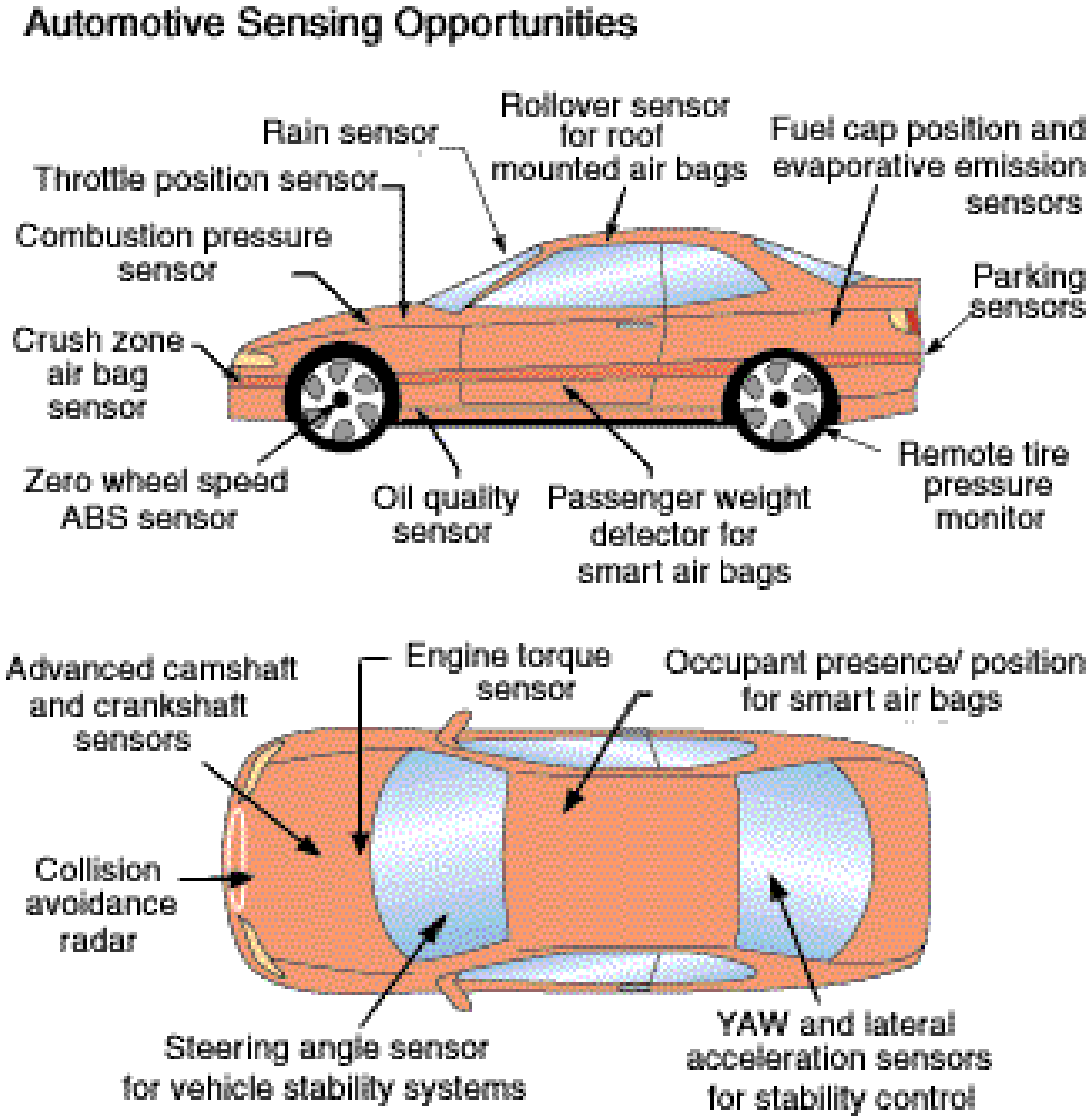
What is “Complex Event Processing”?

- **Definition (wikipedia):** Complex event processing (CEP) consists of processing **many events** happening across all the layers of an organization, **identifying the most meaningful** events within the event cloud, analyzing their impact, and **taking subsequent action in real time**.

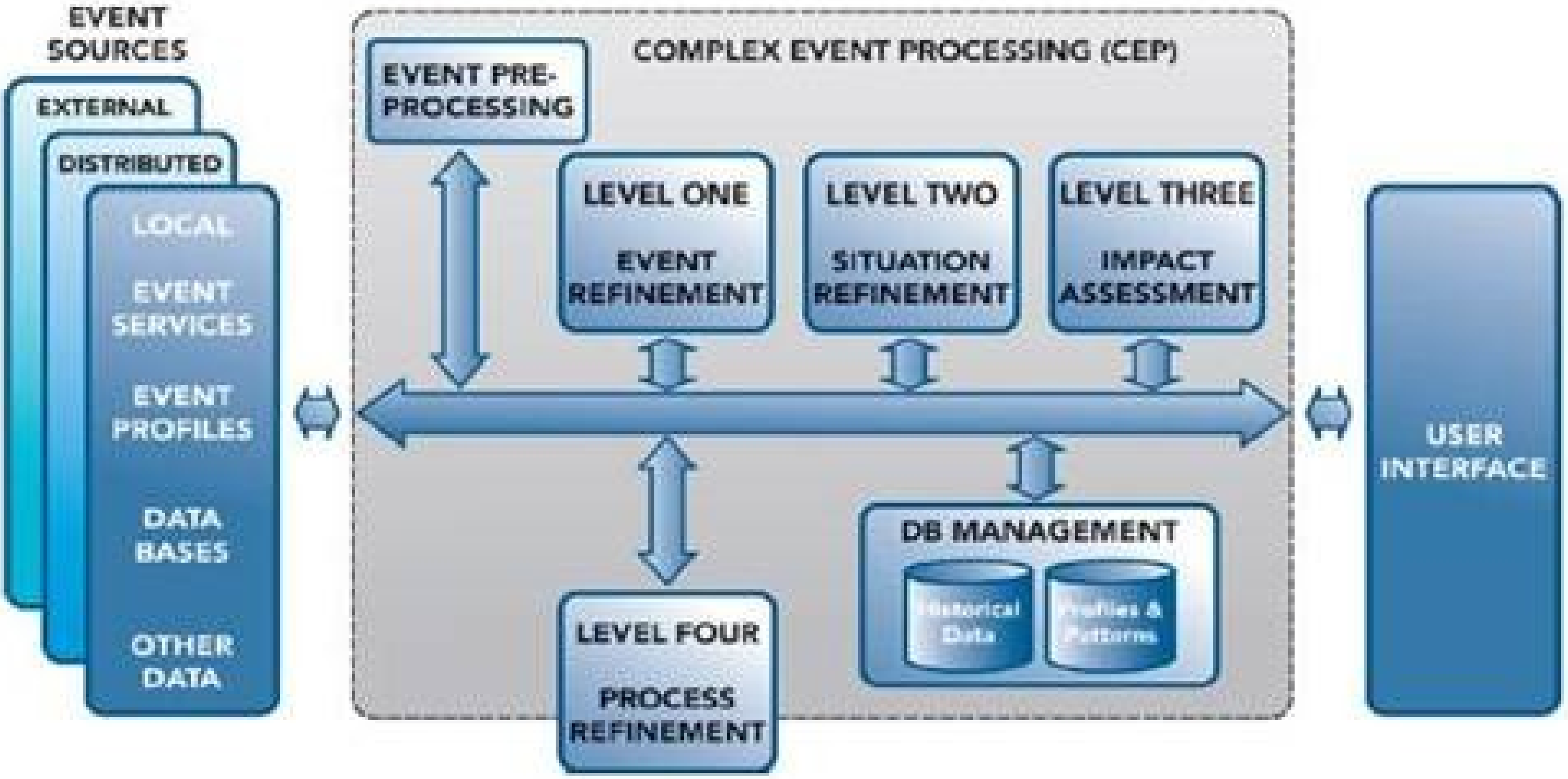
Basic concept

- Instead of running queries against the data, the **continuous stream** of data is **evaluated** against the **static query**

Real life example?


Automotive Sensing Opportunities
Rain sensor
Rollover sensor for roof mounted air bags
Fuel cap position and evaporative emission sensors
Throttle position sensor
Combustion pressure sensor
Parking sensors
Crush zone air bag sensor
Zero wheel speed ABS sensor
Oil quality sensor
Passenger weight detector for smart air bags
Remote tire pressure monitor
Advanced camshaft and crankshaft sensors
Engine torque sensor
Occupant presence/ position for smart air bags
Collision avoidance radar
Steering angle sensor for vehicle stability systems
YAW and lateral acceleration sensors for stability control

How events are processed



EVENT SOURCES

EXTERNAL

DISTRIBUTED

LOCAL

EVENT SERVICES

EVENT PROFILES

DATA BASES

OTHER DATA

COMPLEX EVENT PROCESSING (CEP)

EVENT PRE-PROCESSING

LEVEL ONE

EVENT REFINEMENT

LEVEL TWO

SITUATION REFINEMENT

LEVEL THREE

IMPACT ASSESSMENT

LEVEL FOUR

PROCESS REFINEMENT

DB MANAGEMENT

Historical Data

Profiles & Patterns

USER INTERFACE

How does Oracle CEP works?

- Oracle CEP is an application server designed specifically for real-time, event-driven applications
- Events are delivered to an Oracle CEP instance from an **external source**
- Oracle provides some **adaptors** to facilitate this input.
- There is always the possibility of use the **eclipse plug-in** to implement a customized solution

How does Oracle CEP works?

- Easy way to convert a java-analyzed input into a adapter.

- Oracle CQL (Oracle Continuous Query Language) based in SQL, **has added constructs that support streaming data**.

Oracle CQL syntax example

```
<processor>
    <name>proc</name>
    <rules>
        <query id="q1"><![CDATA[
            SELECT ExchangeStream.symbol,
     ExchangeStream.price, Stock.exchange
            FROM   ExchangeStream [Now], Stock
            WHERE  ExchangeStream.symbol = Stock.symbol
        ]]></query>
    </rules>
</procesor>
```

Foreign Exchange example

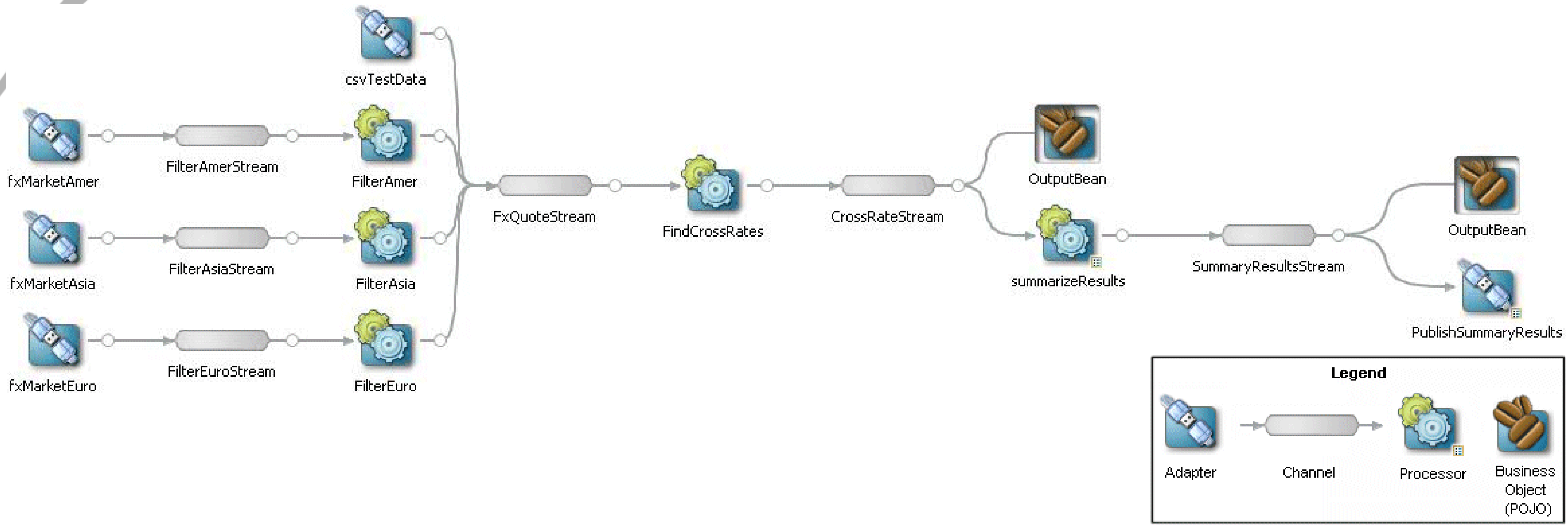
- **Three data feeds** send a constant pair of values from different parts of the world (ex. USD-EUR)
- The fxMarketAmer, fxMarketAsia, and fxMarketEuro **adapters** receive the data from the feeds, convert them into events, and pass them to the corresponding FilterAmer, FilterAsia, and FilterEuro **processors**.
- The **processor** also only selects a specific currency pair from a particular channel

Foreign Exchange example

- A different **processor**, called FindCrossRate, joins all events across all providers, calculates the mid-point between the maximum and minimum rate, and then applies a trader-specified spread.
- Finally, the processor forwards the rate to the **POJO** that contains the **business code**; in this example, the POJO simply publishes the rate to clients.

Note: POJO=Plain Old Java Object

Foreign Exchange example



Foreign Exchange example

```
<?xml version="1.0" encoding="UTF-8"?>
<n1:config xmlns:n1="http://www.bea.com/ns/wlevs/config/application">
  <processor>
    <name>FilterAmer</name>
    <rules>
      <view id="UsdToEur" schema="lastPrice symbol"><![CDATA[
                          select lastPrice, symbol from FilterAmerStream [range 1 second slide 500 milliseconds] where
      symbol="USDEUR"
      ]]></view>
      <view id="UsdToEurPre" schema="price fromRate toRate"><![CDATA[
                  select avg(lastPrice) as price, "USD"  as fromRate, "EUR" as toRate
                  from UsdToEur
                  where lastPrice < 3.0 and lastPrice > 0.25
      ]]></view>
      <query id="AmerFilterCrossRates"><![CDATA[
        istream (select * from UsdToEurPre)
      ]]></query>
    </rules>
  </processor>
  <processor>
    <name>FilterAsia</name>
    <rules>
      <view id="EurToJpy" schema="lastPrice symbol"><![CDATA[
                          select lastPrice, symbol from FilterAsiaStream [range 1 second slide 500 milliseconds] where
      symbol="EURJPY"
      ]]></view>
      <view id="EurToJpyPre" schema="price fromRate toRate"><![CDATA[
                  select avg(lastPrice) as price, "EUR" as fromRate, "JPY" as toRate
                  from EurToJpy
                  where lastPrice < 200.0 and lastPrice > 100.0
      ]]></view>
      <query id="AsiaFilterCrossRates"><![CDATA[
        istream (select * from EurToJpyPre)
      ]]></query>
    </rules>
  </processor>
```

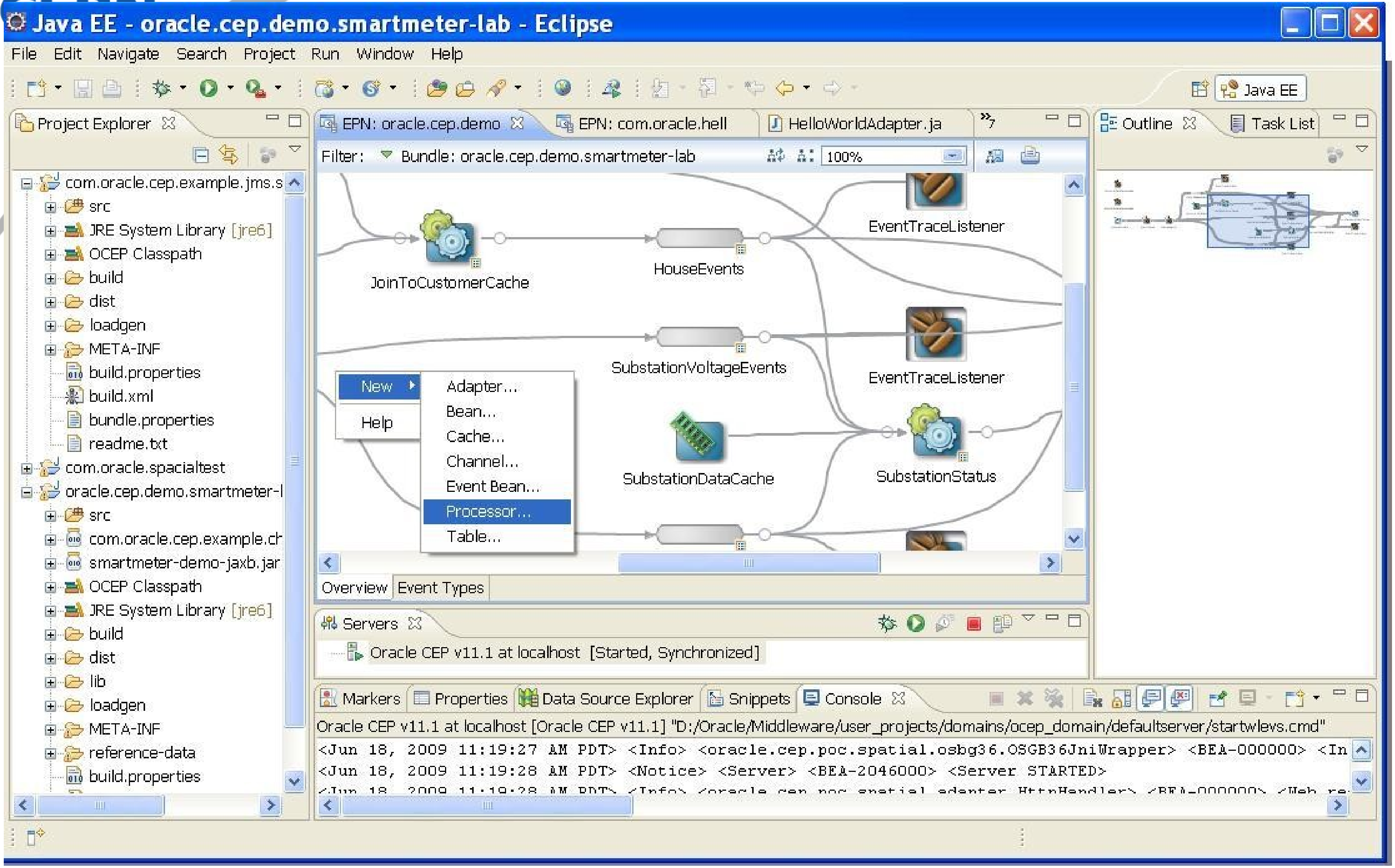
Foreign Exchange example

```
<processor>
    <name>FilterEuro</name>
    <rules>
      <view id="EurToGbp" schema="lastPrice symbol"><![CDATA[
                          select lastPrice, symbol from FilterEuroStream [range 1 second slide 500 milliseconds] where
      symbol="EURGBP"
      ]]></view>
      <view id="EurToGbpPre" schema="price fromRate toRate"><![CDATA[
        select avg(lastPrice) as price, "EUR" as fromRate, "GBP" as toRate
        from EurToGbp
        where lastPrice < 1.5 and lastPrice > 0.5
      ]]></view>
      <query id="EuroFilterCrossRates"><![CDATA[
        istream(select * from EurToGbpPre)
      ]]></query>
    </rules>
  </processor>
  <processor>
    <name>FindCrossRates</name>
    <rules>
      <query id="FindCrossRatesRule"><![CDATA[
       select ((a.price * b.price) + 0.05) as internalPrice,
          a.fromRate as crossRate1,
          b.toRate as crossRate2
                  from FxQuoteStream [range 1 second] as a, FxQuoteStream [range 1 second] as b
                  where
                    NOT (a.price IS NULL)
                  and
                    NOT (b.price IS NULL)
                  and
                    a.toRate = b.fromRate
      ]]></query>
    </rules>
  </processor>
```

Foreign Exchange example

```
 <processor>
    <name>summarizeResults</name>
    <rules>
      <query id="Rule"><![CDATA[
        select
         crossRate1 || crossRate2 as crossRatePair,
         count(*) as totalCount,
         avg(internalPrice) as averageInternalPrice
        from CrossRateStream
        group by crossRate1,crossRate2
        having count(*) > 0
      ]]></query>
    </rules>
  </processor>
</n1:config>
```

Eclipse plugin



Oracle CEP v11.1 at localhost [Started, Synchronized]

```
Oracle CEP v11.1 at localhost [Oracle CEP v11.1] "D:/Oracle/Middleware/user_projects/domains/ocep_domain/defaultserver/startwlevs.cmd"
<Jun 18, 2009 11:19:27 AM PDT> <Info> <oracle.cep.poc.spatial.osbg36.OSGB36JniWrapper> <BEA-000000> <In
<Jun 18, 2009 11:19:28 AM PDT> <Notice> <Server> <BEA-2046000> <Server STARTED>
<Jun 18, 2009 11:19:28 AM PDT> <Info> <oracle.cep.poc.spatial.adapter.HttpHandler> <BEA-000000> <Web re
```

Applications at CERN

- Network analysis: (first possible application)
  - Oracle doesn’t provide an adaptor for network analysis
  - jpcap
    - http://jpcap.sourceforge.net
    - A tool for real-time network traffic capture and analysis
    - An API for developing packet capture applications in Java

Questions?

- Sources:
  - Wikipedia.org
  - Thecepblog.com
  - Oracle.com
  - Designews.com