Waldek Kot

Meet Complex Event Processing

Let's move the Java world!
Motivation

• Extend your developer’s toolbox
• Build more sophisticated business logic in a simpler way
„more sophisticated business logic”?

• „sense-and-response” kind of applications
• process („observe”) streams of events
  – usually large number of events
  – multiple streams
• looking for complex relationships between events („the situations”)
  – patterns, trends, correlations, missing events, ...
  – create synthetic events
• (usually) in time-sensitive regime (real-time)
„simpler”

- declarative
  - think: „like using SQL”
- higher level constructs
  - event-processing network (EPN)
  - event processors
  - streaming queries (rules)
„simpler” = reusing your skills

• you = Java developer
• familiar concepts:
  – Java – the language, the JVM, the APIs
  – CQL = Continuous Query Language
    • SQL with extensions
  – notion of application container
    • enriching the components you create
    • well known deployment model
    • enterprise features (-ilities)
      – manageability, security, scalability, high availability, ...
      – high performance, low latency
Examples of applications

• real-time monitoring
  – finance: algorithmic trading, arbitrage
  – energy: smart grids, smart metering
  – fraud detection – patterns detection
    • intrusion detection (IT, networks, security systems)
    • online-gaming, betting
  – asset tracking (RFID, electronic tagging)
    • logistics: vehicle tracking, toll-collection, proximity tracking
    • security (e.g. police): offenders, criminals tracking
    • surveillance
Examples of applications (cont.)

• real-time calculations
  – pricing engines, payment processing, billing
  – finance: trading desktops

• real-time recommendation engines
  – cross-selling / up-selling, discounting
  – geo-fencing
    • smart malls, smart supermarkets

• real-time visualisations
  – decision making support
  – business processes
Typical challenges & requirements

- High volume of incoming events delivered in bursts
- Complex processing logic (patterns, streaming, ...)
- Many independent, yet interacting components
- Low latency, high throughput events processing
- Quickly distribute processing results to many other systems
- Ability to modify processing parameters on-the-fly
- Ability to mix & match dynamic (events) with static data (databases, buffers/caches, geo-data sources)
Back to basics

• Events
• Event Producers (emitters)
• Event Consumers (responders, processors)
• Channels and Streams
• Event-driven Architecture
Complex Event Processing

• special kind of event processing
  – in addition to: event passing, brokering, persistence, ...

• focus on:
  – pattern detection, aggregations, streaming queries, causality, timing
  – time and other dimensions (e.g. spatial)
  – ability to synthetize complex events
    • from simpler ones
Anatomy of CEP applications

• event sources
  – sensors, GPS, RFID, applications, market streams, networks (web, IT, telco, energy, ...)

• event consumers (responders)
  – other applications, business intelligence (BAM), services (SOA), business processes (BPM), middleware, code, ...

• EPNs – Event Processing Network(s)

• queries / rules
  – CQL – Continuous Query Language

• other forms fo business logic
  – Java components, rules engines, services, processes, ...

• supporting components
  – databases, in-memory data/object grids, reporting engines, integration platforms, adapters, ...
Event Processing Networks (EPN)
Real-life EPN
powered by the Oracle CEP Server

- Event Streams -

EPN (Event Processing Network) Elements

- Adapter
- Processor
- Cache
- POJO
- Channel

JSON

HTTP Pub/Sub
Real-life CEP app example

Oracle CEP Application (EPN)

Resource Locations

Matches and Alerts

SQL

AQ

ZONE Manager

Zone Definitions
Complex event processing functions

- Real-time data streams
  - High Velocity Events
  - RFID events
  - Order data
  - News feeds

- Aggregation
- Complex Continuous Queries
- Correlation
- Sliding Time Windows
- Causality
# Types of Event Flows

<table>
<thead>
<tr>
<th>Type</th>
<th>Characteristics</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>Characteristics: Irregular flow, Manageable volume</td>
<td>Examples: Customer arrival, Order ships</td>
</tr>
<tr>
<td>Streaming</td>
<td>Characteristics: Continuous flow, Need to “keep up”</td>
<td>Examples: Temperature sensors, Stock ticker</td>
</tr>
<tr>
<td>High-Volume</td>
<td>Characteristics: Possibly bursty flow, Need to handle worst case</td>
<td>Examples: Battlefield inputs, Shipping RFID sensors</td>
</tr>
</tbody>
</table>
Types of Event Processing

Filtering
Action: Only pass through events that meet criteria for “interesting”
Examples: Sensor data—only significant change
Equipment monitor—only values above certain threshold

Aggregation
Action: Combine events into composite or summary complex events
Examples: Average stock price every 30min
Temperature+wind+precipitation

Correlation
Action: Look for complex patterns across multiple event sources
Examples: Intrusion detection
Impending machine failure
CQL

• what you know from SQL
  – SELECT..FROM..WHERE..ORDER BY..GROUP BY..HAVING...
  – INNER... LEFT/RIGHT/FULL OUTER JOIN
  – difference from SQL: event attributes instead of column names after SELECT
  – difference from SQL: event stream name instead of table name after FROM
  – Example: SELECT symbol, price FROM inputStream WHERE volume > 1000

• ESP/CEP extensions
  – [] after FROM – defines a window in the stream
    • time-based: [RANGE 3 SECONDS], [NOW]
    • length-based: [ROWS 3]
    • sliding, batched: [ROWS 3 SLIDE 3]
  – sub-partitions in the window: [PARTITION BY attribute ROWS 3]
  – windows changes: additions (ISTREAM), falling out (RSTREAM), deletions (DSTREAM)
  – MATCH_RECOGNIZE – patterns
    • mix and match with the window concept

• named subqueries, parametrized queries,
  – VIEW name(attributes) AS SELECT * FROM inputChannel WHERE salary > 1000

• joins
  – between streams
  – between streams and static data (RDBMS, cache, own)
Example – event filtering

<table>
<thead>
<tr>
<th>Time</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{“AAA”, 10.0, 10.5}</td>
<td>{“AAA”, 10.0, 10.5}</td>
</tr>
<tr>
<td>2</td>
<td>{“AAA”, 10.0, 10.5}</td>
<td>{“AAA”, 10.0, 10.5}</td>
</tr>
<tr>
<td>3</td>
<td>{“BBB”, 11.0, 12.5}</td>
<td></td>
</tr>
</tbody>
</table>
Example – event filtering - CQL

- Solution:

  ```sql
  SELECT *
  FROM stockstream [NOW]
  WHERE symbol = 'AAA'
  ```

  - Specify STREAM source
  - Specify a WINDOW operator
  - Define predicate for filtering
Example – new event detection

<table>
<thead>
<tr>
<th>Time</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{“AAA”, 10.0, 10.5}</td>
<td>{“AAA”, 10.0, 10.5}</td>
</tr>
<tr>
<td>2</td>
<td>{“AAA”, 10.0, 10.5}</td>
<td>crossed out</td>
</tr>
<tr>
<td>3</td>
<td>{“BBB”, 11.0, 12.5}</td>
<td>{“BBB”, 11.0, 12.5}</td>
</tr>
</tbody>
</table>
Example – new event detection - CQL

```
ISTREAM(
    SELECT *
    FROM stockstream [ROWS 1]
)
```
Example – event partitioning

So, there is a small problem with the previous solution...

<table>
<thead>
<tr>
<th>Time</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{“AAA”, 10.0, 10.5}</td>
<td>{“AAA”, 10.0, 10.5}</td>
</tr>
<tr>
<td>2</td>
<td>{“AAA”, 10.0, 10.5}</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>{“BBB”, 11.0, 12.5}</td>
<td>{“BBB”, 11.0, 12.5}</td>
</tr>
<tr>
<td>4</td>
<td>{“AAA”, 10.0, 10.5}</td>
<td>{“AAA”, 10.0, 10.5}</td>
</tr>
</tbody>
</table>
Example – event partitioning - CQL

ISTREAM (  
    SELECT *  
    FROM stockstream  
    [PARTITION BY symbol ROWS 1]  
)
## Example – event partitioning - CQL

<table>
<thead>
<tr>
<th>Time</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{&quot;AAA&quot;, 10.0, 10.5}</td>
<td>{&quot;AAA&quot;, 10.0, 10.5}</td>
</tr>
<tr>
<td>2</td>
<td>{&quot;AAA&quot;, 10.0, 10.5}</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>{&quot;BBB&quot;, 11.0, 12.5}</td>
<td>{&quot;BBB&quot;, 11.0, 12.5}</td>
</tr>
<tr>
<td>4</td>
<td>{&quot;AAA&quot;, 10.0, 10.5}</td>
<td></td>
</tr>
</tbody>
</table>

Much better now...
Example – event correlation

<table>
<thead>
<tr>
<th>Time</th>
<th>Input (BIDS)</th>
<th>Input (ASKS)</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1s</td>
<td>{“AAA”, 12.0, cust1}</td>
<td>{“BBB”, 9.0, cust2}</td>
<td></td>
</tr>
<tr>
<td>5s</td>
<td>{“AAA”, 10.0, cust3}</td>
<td>{“AAA”, 10.0, cust2}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{“AAA”, 10.0, cust3, 10.0, cust2}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{“AAA”, 12.0, cust1, 10.0, cust2}</td>
</tr>
<tr>
<td>15s</td>
<td>{“BBB”, 10.0, cust4}</td>
<td>{“CCC”, 11.0, cust5}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{“AAA”, 10.0, cust3, 10.0, cust2}</td>
</tr>
<tr>
<td>20s</td>
<td></td>
<td>{“BBB”, 10.0, cust6}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{“BBB”, 10.0, cust4, 10.0, cust6}</td>
</tr>
</tbody>
</table>
Example – event correlation - CQL

- Solution:
  ```sql
  SELECT bid.symbol, bidPrice, bid.cust, askPrice, ask.cust,
  FROM bidstream [RANGE 10 seconds] AS bid,
  askstream [RANGE 10 seconds] AS ask
  WHERE bid.symbol = ask.symbol
  ```

- You can perform regular join operations between streams, including outer joins.
Example – pattern detection

- example – missing event detection

<table>
<thead>
<tr>
<th>Time</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1s</td>
<td>{1, “ORDER”}</td>
<td></td>
</tr>
<tr>
<td>5s</td>
<td>{2, “ORDER”}</td>
<td></td>
</tr>
<tr>
<td>10s</td>
<td>{1, “SHIPMENT”}</td>
<td>10s</td>
</tr>
<tr>
<td>15s</td>
<td>{3, “ORDER”}</td>
<td></td>
</tr>
<tr>
<td>15+t</td>
<td></td>
<td>{“DELAYED”, 2}</td>
</tr>
<tr>
<td>20s</td>
<td>{3, “SHIPMENT”}</td>
<td></td>
</tr>
</tbody>
</table>
Example – event correlation - CQL

Solution:

```sql
SELECT "DELAYED" as alertType, orders.orderId,
FROM saletstream MATCH RECOGNIZE ( PARTITION BY orderId

MEASURES

    CustOrder.orderId AS orderId

PATTERN (CustOrder NotTheShipment*) DURATION 10 SECONDS

DEFINE

    CustOrder AS (type = 'ORDER'),
    NotTheShipment AS ((NOT (eventType = 'SHIPMENT'))) ) AS orders
```
Example – event correlation - CQL

- PATTERN (CustOrder NotTheShipment*) DURATION 10 SECONDS
  - Pattern does not match if:
    - Customer order is followed by a shipment within 10 seconds
  - Pattern does match if:
    - Customer order is followed by anything else order than a shipment
    - ...in 10 seconds
Example – pattern detection

Ract, when the stock price forms „W” shape.
You can call „W” as my favourite letter 😊
Example - „W” pattern detection - CQL

```sql
SELECT
    T.stockSymbol,
    T.lastPrice
FROM StockExchangeFeed
MATCH_RECOGNIZE (  
    MEASURES
        A.symbol as stockSymbol,
        last(Z) as lastPrice
    PATTERN
        (A X+ Y+ W+ Z+)
    DEFINE
        X as X.price < prev(X. price),
        Y as Y. price > prev(Y. price),
        W as W. price < prev(W. price),
        Z as Z. price > prev(Z. price)
) as T
```
CQL

• Very rich language
• Built-in functions
  – math, statistical, OLAP
  – spatial
  – XML processing
  – Java
• Extensibility
  – Java
Standardizing the CQL

• Initiative for a complete “continuous” query language

• **Start with SQL ’99 plus “continuous” query extensions**
  – Based on Stanford University research

• **Industry standards discussion**
  – Event Processing Technical Society (EPTS)
  – ANSI SQL
  – OMG

• **Adoption Today**
  – ANSI SQL Standards Proposal for CQL Pattern Matching
    • Oracle, IBM, Stanford University
  – OpenSource Adoption of CQL
  – Oracle Complex Event Processor (CEP)
Oracle CEP Server

- Light-weight Java container for event-driven, time sensitive apps
- Applications
  - have metadata (XML):
    - EPN(s), in/out adapters configuration
    - EPN’s components configuration
      - e.g. for CEP processor: set of CQL queries
  - have Java code
    - one of component types in the EPN
    - access to Java SE, part of Java EE, Spring Framework, Spring-DM and OSGi (Eclipse Equinox) APIs
    - business logic (action) executed in response to
    - also, used for all kinds of „glue” purposes (integration)
    - extensibility (e.g. custom event processors)
  - packaged as OSGi bundles
    - key Oracle CEP developers are founders and key commiters of Spring-DM (OSGi) project
    - probably the first industrial quality OSGi app server
Oracle CEP Tooling

• You can use notepad, but IDE helps...

• For development
  – plugins to the Eclipse IDE
    • plugins to JDeveloper coming soon
  – CEP-aware: wizards, editors, views, navigators, etc.
  – project, EPN and CQL construction (incl. visual)
    • navigation, validation, refactoring, right-click components’ configuration
  – Deployment to the CEP server and lifecycle control

• For administration / monitoring
  – web-based console, query plans, clustering, etc.
Oracle CEP development tooling in Eclipse
CQL Query Plan and Real Time monitoring
Oracle CEP Server

• enterprise features
  – clustering, security, manageability, legacy systems integration mechanisms

• performance
  – public benchmark (stock market signal generation):
    • > 1 mln events per second
    • 400 streaming queries
      – @1mln incoming event, about 40k match rate
    • average event processing latency: < 100 micro-seconds
    • low-end, 4 CPU x86 server

• Oracle ecosystem
Summary

• CEP is yet another tool in your dev toolbox!
  – for event-driven, time-sensitive applications

• best-fit scenario:
  – sensing large numbers of event streams, looking for complex relationships between events („situations”). When they happen, react with your own business logic.

• you already have most of the skills to build such advanced applications
  – CQL (≈SQL), Java
  – application container
More information

• Oracle Technology Network
  – otn.oracle.com
• software
• documentation
• tutorials and samples
• demos
• performance benchmarks
• articles
Thank you 😊!

Waldek Kot
waldemar.kot@oracle.com