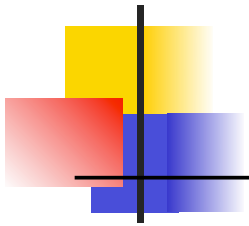
Complex Event Processing

Developing event driven applications with Esper

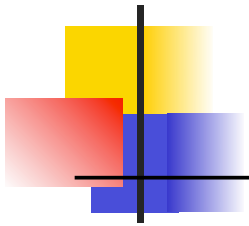
Dan Pritchett

Rearden Commerce



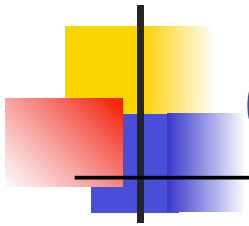
Why Event Driven Architectures?

- Events are a natural occurrence in all systems:
  - ATM transactions
  - Online purchases
  - Application/Resource failures
- Event Driven Architectures (EDA) are an obvious fit for modeling naturally occurring events.



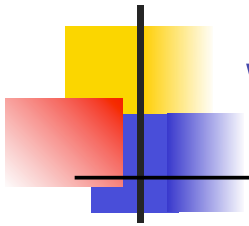
Key Benefits of EDA

- Loose coupling
  - Event producers and consumers only share event semantics
  - Decoupling deployments and bindings improves flexibility.
- Scalability
  - Event operations tend to be more concise
  - Each component can be scaled independently.



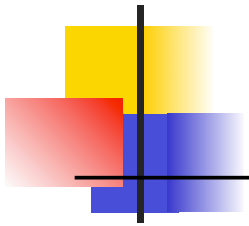
Challenges of EDA

- Unavoidable latency
  - Passing events between decoupled components takes time.
  - Can range from milliseconds to seconds.
- Different Paradigm
  - Engineers find request/response more natural
    - From SOA down to method calls, request/response is course-de-riguer.



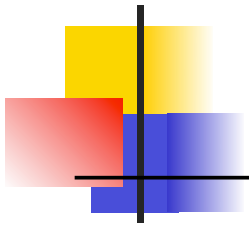
What is an event?

- Events identify an action that has occurred in the system.
  - State transitions
  - Alerts
- Events should be verbs, not nouns
  - Good: Change Profile Address - indicates the state transition of the user's address
  - Poor: User Profile - sends the current state of the user's profile.



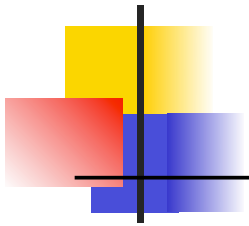
Why are noun events poor?

- The definition of event implies action:
  - Websters: something that happens or is regarded as happening; an occurrence, esp. one of some importance.
- What triggers a noun?
  - These are always an attempt to work around a modeling issue in the system.
  - Typically if you need the data, retrieve it.
    - Nouns are database accesses, not events.



What is stream processing?

- Event Stream Processing (ESP)
- Applies relational calculus to streams of events.
  - Filters
  - Projections
  - Joins
- Generalizes event processing in the way SQL generalized database access.

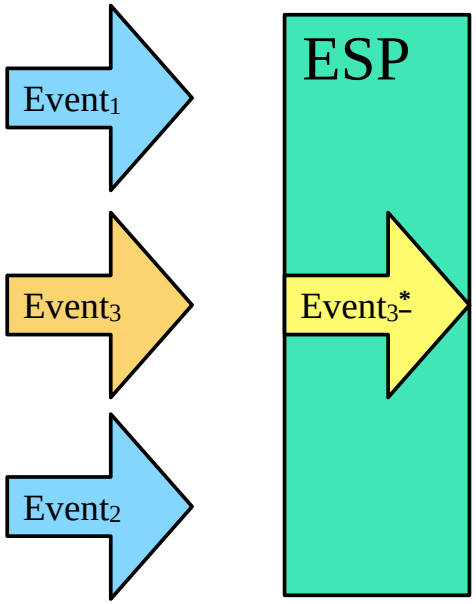


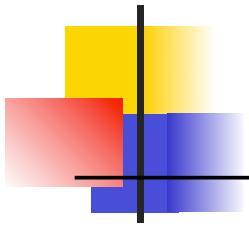
ESP *vs.* Messaging

- How does this relate to messaging?
  - Messaging is a transport.
  - Defines format, topology, and SLA
- Stream processing is application logic
  - Provides algorithms and semantics to message stream.

ESP Operations

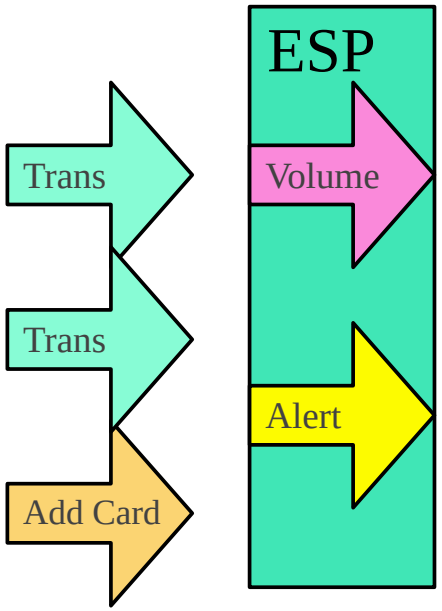
- Filter
  - Eliminate events
- Projection
  - Reduce content of event.

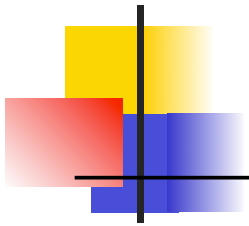




More ESP Operations

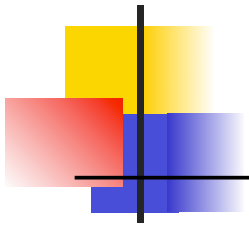
- Aggregation
  - Sequence of homogenous events.
- Join
  - Sequence of heterogenous events.





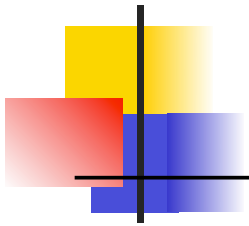
Esper, Java ESP

- Esper provides:
  - Event Processing Language (EPL) - Full set of stream operations.
  - Flexible event representations
    - POJO
    - java.util.Map
    - org.w3c.dom.Node
  - High performance - O(10^4) events/second on commodity hardware.



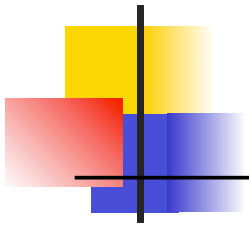
Esper IO

- I/O Adapter interface for delivering events
  - Delivered with two adapters
    - CSV - Events from a CSV stream
    - Spring JMS - Events from a JMS stream
  - Can be used to create additional adapters.
- Not strictly required.
  - Esper engine is decoupled from I/O.



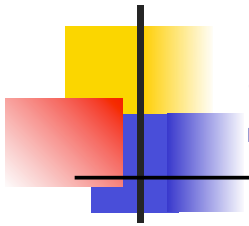
Esper Events

- Events are defined by a set of properties
  - Simple, *e.g.* - address
  - Indexed, *e.g.* - address[0]
  - Map, *e.g.* - address('Home')
  - Nested, *e.g.* - address.street
- Properties map into fields of event container (POJO, Map, or DOM).



Event Processing Language

- Esper's Event Processing Language (EPL) is how all stream processing logic is expressed.
  - SQL like syntax
    - Extensions for event windowing
    - Extensions for event property references.



SQL *vs.* EPL

SQL

A	B	C	D	E
0	k	j	3	8
1	b	t	4	7
2	v	e	2	5
3	y	l	2	7

```
select A, B, D
from t where
D>2
```

A	B	D
0	k	3
1	b	4

EPL

A=0,B=k, C=j,D=3, E=8

A=1,B=b, C=t,D=4, E=7

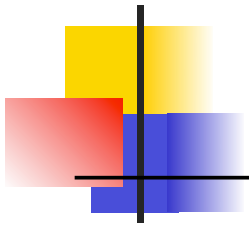
A=2,B=v, C=e,D=2, E=5

A=3,B=y, C=l,D=2, E=7

```
select A, B, D from
T(D>2).win:Length(4
)
```

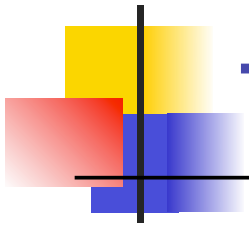
A=0,B=k , D=3

A=1,B=b , D=4



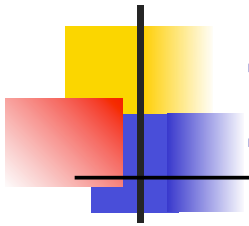
Example Esper Program

- Application Requirements
  - Monitor a stream of statistics from a web service.
  - Stream contains: service name, request size, response size, status, and execution time in ms.
  - Report the following:
    - Average sizes and execution time every minute.
    - Generate alerts for execution times above a provided threshold



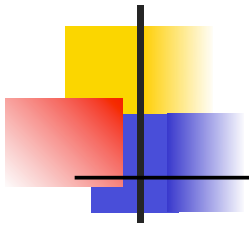
The Event Bean

```
public class WebEvent {
  private String service;
  private int requestSize;
  private int responseSize;
  private int status;
  private long execMS;

  // ... Bean methods left as exercise to
  // ... audience
}
```



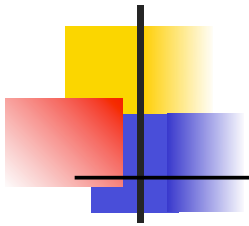
Initialize Processor

```
Configuration conf = new Configuration();
conf.addEventTypeAlias("WebEvent", WebEvent.class.getName());

EPServiceProvider service = EPServiceProviderManager
                .getProvider("Example", conf);
```



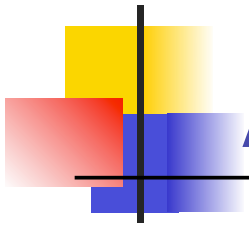
EPL Statements

```
String avgStmt = "select avg(requestSize) as avgRequest,
    avg(responseSize) as avgResponse,
    avg(exectMS) as avgExec
    from WebEvent.win:time_batch(60 sec)";

String alertStmt = "select * from WebEvent where execMS>30000";
```



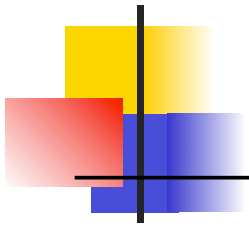
Register Statements

```
EPStatement avg = service.getEPAdministrator()
          .createEPL(avgStmt);
avg.addListener(new AvgListener());

EPStatement alert = service.getEPAdministrator()
          .createEPL(alertStmt);
alert.addListener(new AlertListener());
```



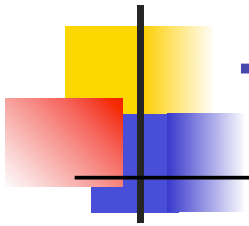
Average Listener

```
public class AvgListener implements UpdateListener {
  public void update(EventBean newEvents[],
                     EventBean oldEvents[]) {
    System.out.println("Average request size: " +
                     newEvents[0].get("avgRequest");
    System.out.println("Average response size: " +
                     newEvents[0].get("avgResponse");
    System.out.println("Average execution time: " +
                     newEvents[0].get("avgExec");
  }
};
```



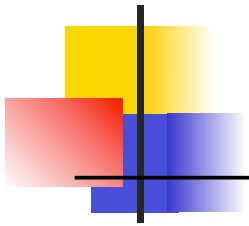
Process Events

```
WebEvent event = new WebEvent("testService", 100, 1024, 100);

service.getEPRuntime().sendEvent(event);
```



Time Windows

- select * from Stream.win:time(5 sec)
  - Sliding window over events.
    - t - 5 seconds delivered as new events.
    - t - 6 seconds delivered as old events.
- select * from Stream.win:time_batch(5 sec)
  - Events accumulate for 5 seconds.
    - This 5 seconds, new events.
    - Previous 5 seconds, old events.



More Information

- http://esper.codehaus.org/
- *The Power of Events* by David Luckham (ISBN 0-201-72789-7)