# Esper as DAQ expert system backbone - evaluation

DAQ Topical

Tomasz Bawej, supervisor: Hannes Sakulin

20.02.2014

# Agenda

1. Introduction to Complex Event Processing with Esper
   - Complex Event Processing
   - Overview of Esper
2. Esper Event Processing Language by example
   - Streams and windows
   - Contexts
   - Patterns
   - Match recognize: detect state changes of DAQ
3. Expert system prototype
   - Prototype overview
   - Consistency with DaqDoctor
   - Performance
4. Evaluation of Esper
5. Discussion

# Complex Event Processing

## Goals

Derive useful information from streams of data and respond to it as quickly as possible.

# Complex Event Processing

## Goals

Derive useful information from streams of data and respond to it as quickly as possible.

## Means

- Event-pattern detection
- Event abstraction, filtering, aggregation, transformation and correlation
- Modeling event hierarchies and abstracting event-driven processes

# Esper (http://esper.codehaus.org/) by Espertech

## What is Esper?

- Event series analysis and event correlation engine.
- Lightweight kernel written in Java (plus a version for .NET).
- Available both as open source and an Enterprise Edition.

# Esper (http://esper.codehaus.org/) by Espertech

## What is Esper?

- Event series analysis and event correlation engine.
- Lightweight kernel written in Java (plus a version for .NET).
- Available both as open source and an Enterprise Edition.

## Notable customers:

- PayPal
- Huaweii
- Oracle

# Esper: building blocks

- *Events* pass through the Esper engine in *streams*.

## Event definitions

Statement registration must be preceded by defining all event types the statement refers to.
The event can be represented by an instance of:

- `java.util.Map`
- `java.lang.Object[]`
- POJO
- XML document

# Esper: building blocks

- *Events* pass through the Esper engine in *streams*.
- *Windows* utilise *views* to retain events.

## Event definitions

Statement registration must be preceded by defining all event types the statement refers to.
The event can be represented by an instance of:

- `java.util.Map`
- `java.lang.Object[]`
- POJO
- XML document

# Esper: building blocks

- *Events* pass through the Esper engine in *streams*.
- *Windows* utilise *views* to retain events.
- *Views* can be of four types:

### Event definitions

Statement registration must be preceded by defining all event types the statement refers to.

The event can be represented by an instance of:

- `java.util.Map`
- `java.lang.Object[]`
- POJO
- XML document

# Esper: building blocks

- *Events* pass through the Esper engine in *streams*.
- *Windows* utilise *views* to retain events.
- *Views* can be of four types:
  1. data `win:` views - *n* (seconds of) events, expiry expression, batch, accumulating...
  2. `std:` views - unique, grouped sub-views, size, first, last
  3. `stat:` views - statistics for specified properties
  4. `ext:` views - sorting

## Event definitions

Statement registration must be preceded by defining all event types the statement refers to.
The event can be represented by an instance of:

- `java.util.Map`
- `java.lang.Object[]`
- POJO
- XML document

# Esper: building blocks

- *Events* pass through the Esper engine in *streams*.
- *Windows* utilise *views* to retain events.
- *Views* can be of four types:
  1. data `win:` views - *n* (seconds of) events, expiry expression, batch, accumulating...
  2. `std:` views - unique, grouped sub-views, size, first, last
  3. `stat:` views - statistics for specified properties
  4. `ext:` views - sorting
- *Statements* are instructions for the engine, executed whenever applicable.
  - Statements can refer to events entering the window, leaving it or both. (`irstream`)

## Event definitions

Statement registration must be preceded by defining all event types the statement refers to.
The event can be represented by an instance of:

- `java.util.Map`
- `java.lang.Object[]`
- POJO
- XML document

# Esper: building blocks

- *Events* pass through the Esper engine in *streams*.
- *Windows* utilise *views* to retain events.
- *Views* can be of four types:
  1. data `win:` views - $n$ (seconds of) events, expiry expression, batch, accumulating...
  2. `std:` views - unique, grouped sub-views, size, first, last
  3. `stat:` views - statistics for specified properties
  4. `ext:` views - sorting
- *Statements* are instructions for the engine, executed whenever applicable.
  - Statements can refer to events entering the window, leaving it or both. (`irstream`)
- *Listeners* (*subscribers*) can be registered with *statements* to receive their updates.

## Event definitions

Statement registration must be preceded by defining all event types the statement refers to.
The event can be represented by an instance of:

- `java.util.Map`
- `java.lang.Object[]`
- POJO
- XML document

# Esper Event Processing Langage

Statements can be constructed using an SQL-like Event Processing Language (EPL).

## SQL features present in EPL

- tables, rows and columns translate to windows, events and properties
- clauses (e.g. `CREATE`, `SELECT`, `INSERT`, `DELETE`, `UPDATE`, `MERGE`, `FROM`, `WHERE`, `GROUP BY`, `HAVING` and `ORDER BY`)
- aggregation functions (e.g. `sum()`, `avg()`, `count()`, `max()`)
- triggers resembled by event driven queries (`ON...`)

# Esper Event Processing Langage

Statements can be constructed using an SQL-like Event Processing Language (EPL).

## SQL features present in EPL

- tables, rows and columns translate to windows, events and properties
- clauses (e.g. `CREATE`, `SELECT`, `INSERT`, `DELETE`, `UPDATE`, `MERGE`, `FROM`, `WHERE`, `GROUP BY`, `HAVING` and `ORDER BY`)
- aggregation functions (e.g. `sum()`, `avg()`, `count()`, `max()`)
- triggers resembled by event driven queries (`ON...`)

Simple SQL-like esper query.

```
select systime, srctime, fromState, toState from
    DaqStateChangeStream;
```

# Streams and windows: prompt on average stream A rate > 500Hz

```
create objectarray schema StreamRates as (name String, rate double,
     srctime long);
```

# Streams and windows: prompt on average stream A rate > 500Hz

Stream definition, populated by summing data over Storage Managers.

```
create objectarray schema StreamRates as (name String, rate double,
    srctime long);
```

# Streams and windows: prompt on average stream A rate > 500Hz

Stream definition, populated by summing data over Storage Managers.

```
create objectarray schema StreamRates as (name String, rate double,
     srctime long);
```

```
create window RatesWin.win:time(30 sec) as select * from StreamRates;
```

# Streams and windows: prompt on average stream A rate > 500Hz

Stream definition, populated by summing data over Storage Managers.

```
create objectarray schema StreamRates as (name String, rate double,
    srctime long);
```

Window retaining events for 30 seconds. Copies structure from `RatesStream`.

```
create window RatesWin.win:time(30 sec) as select * from StreamRates;
```

# Streams and windows: prompt on average stream A rate > 500Hz

Stream definition, populated by summing data over Storage Managers.

```
create objectarray schema StreamRates as (name String, rate double,
    srctime long);
```

Window retaining events for 30 seconds. Copies structure from `RatesStream`.

```
create window RatesWin.win:time(30 sec) as select * from StreamRates;
```

```
insert into RatesWin select * from StreamRates(name="A");
```

# Streams and windows: prompt on average stream A rate > 500Hz

Stream definition, populated by summing data over Storage Managers.

```
create objectarray schema StreamRates as (name String, rate double,
    srctime long);
```

Window retaining events for 30 seconds. Copies structure from RatesStream.

```
create window RatesWin.win:time(30 sec) as select * from StreamRates;
```

Forwards events from RatesStream into the window.

```
insert into RatesWin select * from StreamRates(name="A");
```

# Streams and windows: prompt on average stream A rate > 500Hz

Stream definition, populated by summing data over Storage Managers.

```
create objectarray schema StreamRates as (name String, rate double,
    srctime long);
```

Window retaining events for 30 seconds. Copies structure from RatesStream.

```
create window RatesWin.win:time(30 sec) as select * from StreamRates;
```

Forwards events from RatesStream into the window.

```
insert into RatesWin select * from StreamRates(name="A");
```

```
@Verbose(label="out", fields={"avgRate"}, extraNfo="A rate high")
select avg(rate) as avgRate from RatesWin
  having avg(rate)>500 output first every 10 seconds;
```

# Streams and windows: prompt on average stream A rate > 500Hz

Stream definition, populated by summing data over Storage Managers.

```
create objectarray schema StreamRates as (name String, rate double,
    srctime long);
```

Window retaining events for 30 seconds. Copies structure from `RatesStream`.

```
create window RatesWin.win:time(30 sec) as select * from StreamRates;
```

Forwards events from `RatesStream` into the window.

```
insert into RatesWin select * from StreamRates(name="A");
```

Java-style annotation.

```
@Verbose(label="out", fields={"avgRate"}, extraNfo="A rate high")
select avg(rate) as avgRate from RatesWin
  having avg(rate)>500 output first every 10 seconds;
```

## Annotation - extra information about a statement

- Esper provides a set of built-in annotations.
- Custom annotations can also be provided and used.
  - `@Verbose` marks the statement as producing relevant output.
  - `@Watched` marks statements as producing coninuous output.

From now on the `@Verbose` annotation will be omitted for the sake of clarity.

# Streams and windows: prompt on average stream A rate > 500Hz

Stream definition, populated by summing data over Storage Managers.

```
create objectarray schema StreamRates as (name String, rate double,
    srctime long);
```

Window retaining events for 30 seconds. Copies structure from RatesStream.

```
create window RatesWin.win:time(30 sec) as select * from StreamRates;
```

Forwards events from RatesStream into the window.

```
insert into RatesWin select * from StreamRates(name="A");
```

Java-style annotation.

```
@Verbose(label="out", fields={"avgRate"}, extraNfo="A rate high")
select avg(rate) as avgRate from RatesWin
  having avg(rate)>500 output first every 10 seconds;
```

Clause limiting the output rate.

## Annotation - extra information about a statement

- Esper provides a set of built-in annotations.
- Custom annotations can also be provided and used.
    - `@Verbose` marks the statement as producing relevant output.
    - `@Watched` marks statements as producing coninuous output.

From now on the `@Verbose` annotation will be omitted for the sake of clarity.

# Streams and windows: prompt on stream A rate > 500Hz

```
select average
  from StreamRates.win:time(30 seconds).stat:uni(rate)
  where average>500 output first every 10 seconds;
```

# Streams and windows: prompt on stream A rate > 500Hz

average is a field of the `stat:uni (univariate)` view.

```
select average
  from StreamRates.win:time(30 seconds).stat:uni(rate)
  where average>500 output first every 10 seconds;
```

# Streams and windows: prompt on stream A rate > 500Hz

average is a field of the stat:uni (univariate) view.

Chained views: the second is applied to the results of applying the first.

```
select average
  from StreamRates.win:time(30 seconds).stat:uni(rate)
  where average>500 output first every 10 seconds;
```

# Streams and windows: extend frlcontrollerLink with fedId

```
create objectarray schema FrlCtlLnk(fedSrcId int) copyfrom
    frlcontrollerLink;
```

# Streams and windows: extend frlcontrollerLink with fedId

Extending a copied schema definition.

```
create objectarray schema FrlCtlLnk(fedSrcId int) copyfrom
     frlcontrollerLink;
```

# Streams and windows: extend `frlcontrollerLink` with fedId

Extending a copied schema definition.

```
create objectarray schema FrlCtlLnk(fedSrcId int) copyfrom
     frlcontrollerLink;
```

```
insert into FrlCtlLnk select *,
  fedSrcId(x.context,x.slotNumber,x.linkNumber,CmsHw.FRL) as fedSrcId
  from frlcontrollerLink as x where
  fedSrcId(x.context,x.slotNumber,x.linkNumber,CmsHw.FRL)
  in (select fedSrcId from FedMask(slinkEnabled=true));
```

# Streams and windows: extend `frlcontrollerLink` with fedId

Extending a copied schema definition.

```
create objectarray schema FrlCtlLnk(fedSrcId int) copyfrom
     frlcontrollerLink;
```

Copying all the frlcontrollerLink fields.

```
insert into FrlCtlLnk select *,
  fedSrcId(x.context,x.slotNumber,x.linkNumber,CmsHw.FRL) as fedSrcId
  from frlcontrollerLink as x where
  fedSrcId(x.context,x.slotNumber,x.linkNumber,CmsHw.FRL)
  in (select fedSrcId from FedMask(slinkEnabled=true));
```

# Streams and windows: extend `frlcontrollerLink` with fedId

Extending a copied schema definition.

```
create objectarray schema FrlCtlLnk(fedSrcId int) copyfrom
     frlcontrollerLink;
```

Inserting `fedSrcId` returned from a Java helper method.

Copying all the frlcontrollerLink fields.

```
insert into FrlCtlLnk select *,
  fedSrcId(x.context,x.slotNumber,x.linkNumber,CmsHw.FRL) as fedSrcId
  from frlcontrollerLink as x where
  fedSrcId(x.context,x.slotNumber,x.linkNumber,CmsHw.FRL)
  in (select fedSrcId from FedMask(slinkEnabled=true));
```

# Streams and windows: extend `frlcontrollerLink` with fedId

Extending a copied schema definition.

```
create objectarray schema FrlCtlLnk(fedSrcId int) copyfrom
     frlcontrollerLink;
```

Inserting `fedSrcId` returned from a Java helper method.

Copying all the frlcontrollerLink fields.

```
insert into FrlCtlLnk select *,
  fedSrcId(x.context,x.slotNumber,x.linkNumber,CmsHw.FRL) as fedSrcId
  from frlcontrollerLink as x where
  fedSrcId(x.context,x.slotNumber,x.linkNumber,CmsHw.FRL)
  in (select fedSrcId from FedMask(slinkEnabled=true));
```

Subquery against a named window.

# Streams and windows: extend `frlcontrollerLink` with fedId

Extending a copied schema definition.

```
create objectarray schema FrlCtlLnk(fedSrcId int) copyfrom
     frlcontrollerLink;
```

Inserting `fedSrcId` returned from a Java helper method.

Copying all the frlcontrollerLink fields.

```
insert into FrlCtlLnk select *,
  fedSrcId(x.context,x.slotNumber,x.linkNumber,CmsHw.FRL) as fedSrcId
  from frlcontrollerLink as x where
  fedSrcId(x.context,x.slotNumber,x.linkNumber,CmsHw.FRL)
  in (select fedSrcId from FedMask(slinkEnabled=true));
```

Subquery against a named window.

An imported enum.

# Contexts: computing FED backpressure

```
create objectarray schema FrlBackpressureStream as
  (fedSrcId Integer, bpFraction double, timestamp long);
```

# Contexts: computing FED backpressure

```
create objectarray schema FrlBackpressureStream as
  (fedSrcId Integer, bpFraction double, timestamp long);
```

```
create context BpPerFedId
  context RunOngoingNested start RunStart end RunStop,
  context BpPerFedIdNested partition by fedSrcId from FrlCtlLnk;
```

# Contexts: computing FED backpressure

```
create objectarray schema FrlBackpressureStream as
  (fedSrcId Integer, bpFraction double, timestamp long);
```

A compound context - product of all sub-contexts.

```
create context BpPerFedId
  context RunOngoingNested start RunStart end RunStop,
  context BpPerFedIdNested partition by fedSrcId from FrlCtlLnk;
```

# Contexts: computing FED backpressure

```
create objectarray schema FrlBackpressureStream as
  (fedSrcId Integer, bpFraction double, timestamp long);
```

A compound context - product of all sub-contexts.

A non-overlapping sub-context.

```
create context BpPerFedId
  context RunOngoingNested start RunStart end RunStop,
  context BpPerFedIdNested partition by fedSrcId from FrlCtlLnk;
```

# Contexts: computing FED backpressure

```
create objectarray schema FrlBackpressureStream as
  (fedSrcId Integer, bpFraction double, timestamp long);
```

A compound context - product of all sub-contexts.

A non-overlapping sub-context.

```
create context BpPerFedId
  context RunOngoingNested start RunStart end RunStop,
  context BpPerFedIdNested partition by fedSrcId from FrlCtlLnk;
```

A sub-context partitioned by `fedSrcId`.

# Contexts: computing FED backpressure

```
create objectarray schema FrlBackpressureStream as
  (fedSrcId Integer, bpFraction double, timestamp long);
```

A compound context - product of all sub-contexts.

A non-overlapping sub-context.

```
create context BpPerFedId
  context RunOngoingNested start RunStart end RunStop,
  context BpPerFedIdNested partition by fedSrcId from FrlCtlLnk;
```

A sub-context partitioned by `fedSrcId`.

```
context BpPerFedId
  insert into
    FrlBackpressureStream
  select
    fedSrcId,
    timestamp.getTime() as timestamp,
    (fifoAlmostFullCnt-prior(1,fifoAlmostFullCnt))
      /(clockCount-prior(1,clockCount)) as bpFraction
  from
    FrlCtlLnk
  where
    clockCount>prior(1,clockCount);
```

# Contexts: computing FED backpressure

```
create objectarray schema FrlBackpressureStream as
  (fedSrcId Integer, bpFraction double, timestamp long);
```

A compound context - product of all sub-contexts.

A non-overlapping sub-context.

```
create context BpPerFedId
  context RunOngoingNested start RunStart end RunStop,
  context BpPerFedIdNested partition by fedSrcId from FrlCtlLnk;
```

A sub-context partitioned by fedSrcId.

Compute backpressure value.

```
context BpPerFedId
  insert into
    FrlBackpressureStream
  select
    fedSrcId,
    timestamp.getTime() as timestamp,
    (fifoAlmostFullCnt-prior(1,fifoAlmostFullCnt))
      /(clockCount-prior(1,clockCount)) as bpFraction
  from
    FrlCtlLnk
  where
    clockCount>prior(1,clockCount);
```

# Contexts: computing FED backpressure

```
create objectarray schema FrlBackpressureStream as
  (fedSrcId Integer, bpFraction double, timestamp long);
```

A compound context - product of all sub-contexts.

A non-overlapping sub-context.

```
create context BpPerFedId
  context RunOngoingNested start RunStart end RunStop,
  context BpPerFedIdNested partition by fedSrcId from FrlCtlLnk;
```

A sub-context partitioned by fedSrcId.

```
context BpPerFedId
  insert into
    FrlBackpressureStream
  select
    fedSrcId,
    timestamp.getTime() as timestamp,
    (fifoAlmostFullCnt-prior(1,fifoAlmostFullCnt))
      /(clockCount-prior(1,clockCount)) as bpFraction
  from
    FrlCtlLnk
  where
    clockCount>prior(1,clockCount);
```

Compute backpressure value.

prior single row function,
returns i-th previous event in order of arrival

# Patterns: notify if jobcontrol flashlist is not updated within a minute

```
create variable int timeout = 50;
```

# Patterns: notify if jobcontrol flashlist is not updated within a minute

Variable holding the timeout value minus update interval.

```
create variable int timeout = 50;
```

# Patterns: notify if jobcontrol flashlist is not updated within a minute

Variable holding the timeout value minus update interval.

```
create variable int timeout = 50;



create context jobctlHostOutdated
  context ByHost partition by hostname from jobcontrol,
  context Outdated start pattern[a=jobcontrol->
      (timer:interval(timeout) and not jobcontrol(hostname=a.hostname))]
    end pattern [jobcontrol(hostname=a.hostname)];
```

# Patterns: notify if jobcontrol flashlist is not updated within a minute

Variable holding the timeout value minus update interval.

```
create variable int timeout = 50;
```

Creates a separate context partition per each hostname.

```
create context jobctlHostOutdated
  context ByHost partition by hostname from jobcontrol,
  context Outdated start pattern[a=jobcontrol->
      (timer:interval(timeout) and not jobcontrol(hostname=a.hostname))]
    end pattern [jobcontrol(hostname=a.hostname)];
```

# Patterns: notify if jobcontrol flashlist is not updated within a minute

Variable holding the timeout value minus update interval.

```
create variable int timeout = 50;
```

Creates a separate context partition per each hostname.

```
create context jobctlHostOutdated
 context ByHost partition by hostname from jobcontrol,
 context Outdated start pattern[a=jobcontrol->
   (timer:interval(timeout) and not jobcontrol(hostname=a.hostname))]
  end pattern [jobcontrol(hostname=a.hostname)];
```

Starts when the pattern detects an absence of flashlist.

# Patterns: notify if jobcontrol flashlist is not updated within a minute

Variable holding the timeout value minus update interval.

```
create variable int timeout = 50;
```

Creates a separate context partition per each hostname.

```
create context jobctlHostOutdated
  context ByHost partition by hostname from jobcontrol,
  context Outdated start pattern[a=jobcontrol->
      (timer:interval(timeout) and not jobcontrol(hostname=a.hostname))]
    end pattern [jobcontrol(hostname=a.hostname)];
```

Ends when the flashlist
with the same hostname arrives.

Starts when the pattern detects
an absence of flashlist.

# Patterns: notify if jobcontrol flashlist is not updated within a minute

Variable holding the timeout value minus update interval.

```
create variable int timeout = 50;
```

Creates a separate context partition per each hostname.

```
create context jobctlHostOutdated
  context ByHost partition by hostname from jobcontrol,
  context Outdated start pattern[a=jobcontrol->
      (timer:interval(timeout) and not jobcontrol(hostname=a.hostname))]
    end pattern [jobcontrol(hostname=a.hostname)];
```

Ends when the flashlist
with the same hostname arrives.

Starts when the pattern detects
an absence of flashlist.

Accessing context property.

```
context jobctlHostOutdated
  select context.ByHost.key1 as hostname,
    date(current_timestamp()) as systime,
    (current_timestamp()-context.Outdated.a.fetchstamp)/1000.0
      as secondsSinceUpdate
  from pattern [every timer:interval(10 seconds)];
```

# Patterns: notify if jobcontrol flashlist is not updated within a minute

Variable holding the timeout value minus update interval.

```
create variable int timeout = 50;
```

Creates a separate context partition per each hostname.

```
create context jobctlHostOutdated
  context ByHost partition by hostname from jobcontrol,
  context Outdated start pattern[a=jobcontrol->
      (timer:interval(timeout) and not jobcontrol(hostname=a.hostname))]
    end pattern [jobcontrol(hostname=a.hostname)];
```

Ends when the flashlist
with the same hostname arrives.

Starts when the pattern detects
an absence of flashlist.

Accessing context property.

```
context jobctlHostOutdated
  select context.ByHost.key1 as hostname,
    date(current_timestamp()) as systime,
    (current_timestamp()-context.Outdated.a.fetchstamp)/1000.0
      as secondsSinceUpdate
  from pattern [every timer:interval(10 seconds)];
```

Esper built-in method.

# Patterns: notify if jobcontrol flashlist is not updated within a minute

Variable holding the timeout value minus update interval.

```
create variable int timeout = 50;
```

Creates a separate context partition per each hostname.

```
create context jobctlHostOutdated
  context ByHost partition by hostname from jobcontrol,
  context Outdated start pattern[a=jobcontrol->
      (timer:interval(timeout) and not jobcontrol(hostname=a.hostname))]
    end pattern [jobcontrol(hostname=a.hostname)];
```

Ends when the flashlist
with the same hostname arrives.

Starts when the pattern detects
an absence of flashlist.

Accessing context property.

Accessing context starting event.

```
context jobctlHostOutdated
  select context.ByHost.key1 as hostname,
    date(current_timestamp()) as systime,
    (current_timestamp()-context.Outdated.a.fetchstamp)/1000.0
      as secondsSinceUpdate
  from pattern [every timer:interval(10 seconds)];
```

# Match recognize

```
select srctime, systime, fromState, toState
  from levelZeroFM_subsys match_recognize(
    partition by SUBSYS
    measures A.STATE as fromState, B.STATE as toState,
      B.timestamp as srctime, date(B.fetchstamp) as
          systime
    after match skip to current row
    pattern (A B)
    define A as A.SUBSYS='DAQ',
      B as B.STATE != A.STATE
  );
```

# Match recognize

```
select srctime, systime, fromState, toState
  from levelZeroFM_subsys match_recognize(
    partition by SUBSYS
    measures A.STATE as fromState, B.STATE as toState,
      B.timestamp as srctime, date(B.fetchstamp) as
           systime
    after match skip to current row
    pattern (A B)
    define A as A.SUBSYS='DAQ',
      B as B.STATE != A.STATE
  );
```

## Match recognize vs patterns

- Match recognize works with one event type at a time.
- Match recognize is claimed to be faster and less memory consuming.
- Match recognize has been proposed for incorporation into SQL standard.

# Match recognize

```
select srctime, systime, fromState, toState
  from levelZeroFM_subsys match_recognize(
    partition by SUBSYS
    measures A.STATE as fromState, B.STATE as toState,
      B.timestamp as srctime, date(B.fetchstamp) as
          systime
    after match skip to current row
    pattern (A B)
    define A as A.SUBSYS='DAQ',
      B as B.STATE != A.STATE
  );
```

Defines returned values.

## Match recognize vs patterns

- Match recognize works with one event type at a time.
- Match recognize is claimed to be faster and less memory consuming.
- Match recognize has been proposed for incorporation into SQL standard.

# Match recognize

```
select srctime, systime, fromState, toState
  from levelZeroFM_subsys match_recognize(
    partition by SUBSYS
    measures A.STATE as fromState, B.STATE as toState,
      B.timestamp as srctime, date(B.fetchstamp) as
           systime
    after match skip to current row
    pattern (A B)
    define A as A.SUBSYS='DAQ',
      B as B.STATE != A.STATE
  );
```

Defines returned values.

Allows a closing event of one pattern to be the opening event of another.

## Match recognize vs patterns

- Match recognize works with one event type at a time.
- Match recognize is claimed to be faster and less memory consuming.
- Match recognize has been proposed for incorporation into SQL standard.

# Match recognize

```
select srctime, systime, fromState, toState
  from levelZeroFM_subsys match_recognize(
    partition by SUBSYS
    measures A.STATE as fromState, B.STATE as toState,
      B.timestamp as srctime, date(B.fetchstamp) as
           systime
    after match skip to current row
    pattern (A B)
    define A as A.SUBSYS='DAQ',
      B as B.STATE != A.STATE
  );
```

Defines returned values.

Defines temporal pattern: A directly followed by B.

Allows a closing event of one pattern to be the opening event of another.

## Match recognize vs patterns

- Match recognize works with one event type at a time.
- Match recognize is claimed to be faster and less memory consuming.
- Match recognize has been proposed for incorporation into SQL standard.

# Match recognize

```
select srctime, systime, fromState, toState
  from levelZeroFM_subsys match_recognize(
    partition by SUBSYS
    measures A.STATE as fromState, B.STATE as toState,
      B.timestamp as srctime, date(B.fetchstamp) as
           systime
    after match skip to current row
    pattern (A B)
    define A as A.SUBSYS='DAQ',
      B as B.STATE != A.STATE
  );
```

Defines returned values.

Defines temporal pattern: A directly followed by B.

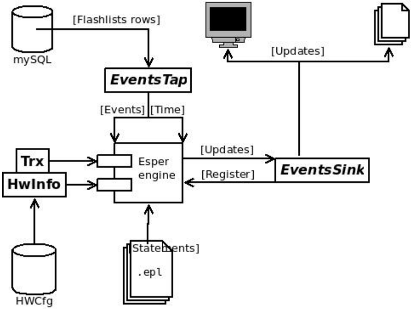
Imposes constraints on events A and B.

Allows a closing event of one pattern to be the opening event of another.

## Match recognize vs patterns

- Match recognize works with one event type at a time.
- Match recognize is claimed to be faster and less memory consuming.
- Match recognize has been proposed for incorporation into SQL standard.

# Prototype expert system

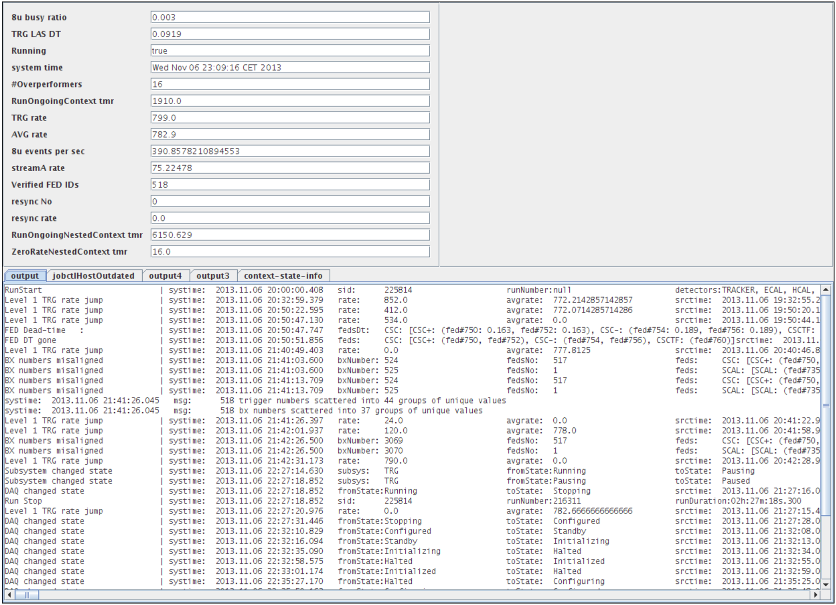


- Events
  - unique flashlist rows from November global run
  - stored in a mysql database, indexed by reception timestamps (*fetchstamps*)
  - flashlists in use:
    - EventProcessorStatus
    - EVM
    - FMMInput
    - FMMStatus
    - frlcontrollerCard
    - frlcontrollerLink
    - gt_cell_lumiseg
    - hostInfo
    - jobcontrol
    - levelZeroFM_static
    - levelZeroFM_subsys
    - StorageManagerPerformance;
- EPL statements stored in files
- `EventSinks` as statement listeners
- Limitations:
  - Non-interactive
  - Output repeated for persisting problems.

# Implemented checks

- In-run tests:
  - Message on Run Start giving: run NR, SID, detectors in, FEDs in per detector.
  - Message on Run Stop giving: run Nr, SID, detectors in, FEDs in per detector, avg L1 rate, avg stream A rate, avg dead time (from trigger LAS), duration.
  - Message on L1 trigger rate jump $> 10\%$.
  - Message on subsys going to (ERROR, RUNNING_*, PAUSING, PAUSED, RESUMING)
  - Message on any state change of DAQ.
  - Message on jobcontrol flashlist not being updated for 1 minute.
  - FEDs with dead-time $> 1\%$.
  - FEDs with backpressure $> 1\%$.
  - Message on stream A $> 500$ Hz.
  - Detection of FEDs in illegal state.
- In-run test with rate stuck at 0:
  - Check BX alignment and print message if not aligned. Repeat after 10 seconds.
  - Check triggers(events) alignment + print message if not aligned. Repeat after 10 seconds.
  - FEDs stuck in ERROR/OOS/WARNING/BUSY (anything not READY).
  - List FEDs with backpressure or deadtime.
  - Check if number of resyncs and the last resync event number is the same for all FEDs.
- Continuous display:
  - Stream A rate
  - L1 rate
  - Resync rate and number of resyncs
  - Event processing rate per group (determined by number of processors on machine).
  - CPU utilization per group (as above)
  - Deadtime (from trigger LAS)

# Swing view screenshot



| | |
|---|---|
| 8u busy ratio | 0.003 |
| TRG LAS DT | 0.0919 |
| Running | true |
| system time | Wed Nov 06 23:09:16 CET 2013 |
| #Overperformers | 16 |
| RunOngoingContext tmr | 1910.0 |
| TRG rate | 799.0 |
| AVG rate | 782.9 |
| 8u events per sec | 390.8578210894553 |
| streamA rate | 75.22478 |
| Verified FED IDs | 518 |
| resync No | 0 |
| resync rate | 0.0 |
| RunOngoingNestedContext tmr | 6150.629 |
| ZeroRateNestedContext tmr | 16.0 |

```
RunStart                         | systime:  2013.11.06 20:00:00.408   sid:      225814                    runNumber:null                      detectors:TRACKER, ECAL, HCAL,
Level 1 TRG rate jump            | systime:  2013.11.06 20:32:59.379   rate:     852.0                     avgrate:  772.2142857142857         srctime:  2013.11.06 19:32:55.2
Level 1 TRG rate jump            | systime:  2013.11.06 20:50:22.595   rate:     412.0                     avgrate:  772.0714285714286         srctime:  2013.11.06 19:50:20.1
Level 1 TRG rate jump            | systime:  2013.11.06 20:50:47.130   rate:     534.0                     avgrate:  0.0                       srctime:  2013.11.06 19:50:44.1
FED Dead-time   :                | systime:  2013.11.06 20:50:47.747   fedsDt:   CSC: [CSC+: (fed#750: 0.163, fed#752: 0.163), CSC-: (fed#754: 0.189, fed#756: 0.189), CSCTF:
FED DT gone                      | systime:  2013.11.06 20:50:51.856   feds:     CSC: [CSC+: (fed#750, fed#752), CSC-: (fed#754, fed#756), CSCTF: (fed#760)]srctime:  2013.11.
Level 1 TRG rate jump            | systime:  2013.11.06 21:40:49.403   rate:     0.0                       avgrate:  777.8125                  srctime:  2013.11.06 20:40:46.8
BX numbers misaligned            | systime:  2013.11.06 21:41:03.600   bxNumber: 524                       fedsNo:   517                       feds:     CSC: [CSC+: (fed#750,
BX numbers misaligned            | systime:  2013.11.06 21:41:03.600   bxNumber: 525                       fedsNo:   1                         feds:     SCAL: [SCAL: (fed#735
BX numbers misaligned            | systime:  2013.11.06 21:41:13.709   bxNumber: 524                       fedsNo:   517                       feds:     CSC: [CSC+: (fed#750,
BX numbers misaligned            | systime:  2013.11.06 21:41:13.709   bxNumber: 525                       fedsNo:   1                         feds:     SCAL: [SCAL: (fed#735
systime:  2013.11.06 21:41:26.045   msg:      518 trigger numbers scattered into 44 groups of unique values
systime:  2013.11.06 21:41:26.045   msg:      518 bx numbers scattered into 37 groups of unique values
Level 1 TRG rate jump            | systime:  2013.11.06 21:41:26.397   rate:     24.0                      avgrate:  0.0                       srctime:  2013.11.06 20:41:22.9
Level 1 TRG rate jump            | systime:  2013.11.06 21:42:01.937   rate:     120.0                     avgrate:  778.0                     srctime:  2013.11.06 20:41:58.9
BX numbers misaligned            | systime:  2013.11.06 21:42:26.500   bxNumber: 3069                      fedsNo:   517                       feds:     CSC: [CSC+: (fed#750,
BX numbers misaligned            | systime:  2013.11.06 21:42:26.500   bxNumber: 3070                      fedsNo:   1                         feds:     SCAL: [SCAL: (fed#735
Level 1 TRG rate jump            | systime:  2013.11.06 21:42:31.173   rate:     790.0                     avgrate:  0.0                       srctime:  2013.11.06 20:42:28.9
Subsystem changed state          | systime:  2013.11.06 22:27:14.630   subsys:   TRG                       fromState:Running                   toState:  Pausing
Subsystem changed state          | systime:  2013.11.06 22:27:18.852   subsys:   TRG                       fromState:Pausing                   toState:  Paused
DAQ changed state                | systime:  2013.11.06 22:27:18.852   fromState:Running                   toState:  Stopping                  srctime:  2013.11.06 21:27:16.0
Run Stop                         | systime:  2013.11.06 22:27:18.852   sid:      225814                    runNumber:216311                    runDuration:02h:27m:18s.300
Level 1 TRG rate jump            | systime:  2013.11.06 22:27:20.976   rate:     0.0                       avgrate:  782.6666666666666         srctime:  2013.11.06 21:27:15.4
DAQ changed state                | systime:  2013.11.06 22:27:31.446   fromState:Stopping                  toState:  Configured                srctime:  2013.11.06 21:27:28.0
DAQ changed state                | systime:  2013.11.06 22:32:10.829   fromState:Configured                toState:  Standby                   srctime:  2013.11.06 21:32:08.0
DAQ changed state                | systime:  2013.11.06 22:32:16.094   fromState:Standby                   toState:  Initializing              srctime:  2013.11.06 21:32:13.0
DAQ changed state                | systime:  2013.11.06 22:32:35.090   fromState:Initializing              toState:  Halted                    srctime:  2013.11.06 21:32:34.0
DAQ changed state                | systime:  2013.11.06 22:32:58.575   fromState:Halted                    toState:  Initialized               srctime:  2013.11.06 21:32:55.0
DAQ changed state                | systime:  2013.11.06 22:33:01.174   fromState:Initialized               toState:  Halted                    srctime:  2013.11.06 21:32:59.0
DAQ changed state                | systime:  2013.11.06 22:35:27.170   fromState:Halted                    toState:  Configuring               srctime:  2013.11.06 21:35:25.0
```

# Consistency with DaqDoctor

DaqDoctor's output

| | | |
|---|---|---|
| 2013-11-08 07:10:35 | NO BEAM | FED BX numbers are not aligned. 1 different bc numbers encountered |
| 2013-11-08 07:10:35 | NO BEAM | Most (475) had the bc set to 2454 in the last event seen. 28 frls saw the last bc at 2442: HBHEb (711), HBHEb (706), HBHEb (709), HBHEb (708), HBHEb (707), HBHEb (710), HBHEc (712), HBHEc (715), HBHEc (716), HBHEc (714), HBHEc (717), HBHEc (713), HO (727), HO (728), HO (730), HO (724), HO (731), HO (729), HO (725), HO (726), GT (812), GT (813), HBHEa (701), HBHEa (703), HBHEa (705), HBHEa (702), HBHEa (704), HBHEa (700) 1 frls saw the last bc at 2446: DTTF (780) 1 frls saw the last bc at 2443: SCAL (735) |

Prototype's output: BX numbers misaligned

| systime | bxn | fedsNo | feds |
|---|---|---|---|
| 2013.11.08 07:10:39.646 | 2454 | 475 | [REMOVED] |
| 2013.11.08 07:10:39.646 | 3490 | 28 | HCAL: [HBHEa: (700, 701, 702, 703, 704, 705), HBHEb: (706, 707, 708, 709, 710, 711), HBHEc: (712, 713, 714, 715, 716, 717), HO: (724, 725, 726, 727, 728, 729, 730, 731)], TRG: [GT: (812, 813)] |
| 2013.11.08 07:10:39.646 | 3491 | 1 | SCAL: [SCAL: (735)] |
| 2013.11.08 07:10:39.646 | 3494 | 1 | DT: [DTTF: (780)] |

# Performance

Tested on a set of 111M events recorded between 2013-11-06 20:00:00 and 2013-11-08 20:00:00.

- Engine clock, controlled by incoming data, progresses 42 times as fast as system clock (Intel i7 CPU 4x3.40GHz).
- This metric reaches 44 with no statements or no data sent to the engine.
- It reaches 34000 if, additionally, nothing but timestamps is being pulled from DB.

# Esper: for and against

## Pros:

- An out-of-the-box event processing framework.
- Extensibility and ease of embedding.
- Separation of event-processing logic and Java code.
- Satisfactory documentation.

## Cons:

- EPL
  - sometimes too rigid and conter-intuitive:
    - Variables can only be set using triggering event in `ON-SET` statements,
    - pattern-triggered queries cannot select from unnamed windows (`unidirectional join` as a workaround)
  - incomplete: e.g. forbids using `rstream` in patterns
  - evolving:
    - bugs fixed with each version (prototype crashes on 4.9)
    - `group by` in subqueries since 4.11
    - subqueries within `having` since 4.10
  - steep learning curve
  - cumbersome data transformations
- Hard to debug.
- Lack of community, books, thorough tutorials.

# Discussion

Questions, suggestions?

# Discussion

Thank you!

# Discussion

Thank you!

Thank you!

# Variables

```
create objectarray schema ResyncStream as (resyncRate double,
    resyncNo long);
create variable String desiredContext = "";
create variable Integer desiredSlotNumber = -1;

on FrlCtlLnk(fedSrcId=812) as ctl set desiredContext = ctl.context,
    desiredSlotNumber= ctl.slotNumber;


insert into ResyncStream
select resyncRate, resyncNo from frlcontrollerCard match_recognize(
  partition by context, slotNumber
  measures 1000*(B.myrinetResync - A.myrinetResync)/(B.timestamp.
      getTime()-A.timestamp.getTime()) as resyncRate, B.
      myrinetResync as resyncNo
  after match skip to current row
  pattern (A B)
  define A as A.context = desiredContext and A.slotNumber=
      desiredSlotNumber
);
```

# Expressions

```
create objectarray schema RunAvgRate as (rate double);
```

# Expressions

```
create objectarray schema RunAvgRate as (rate double);

  select avg(rate) as rate from L1RatesStream;
```

# Expressions

```
create objectarray schema RunAvgRate as (rate double);

  select avg(rate) as rate from L1RatesStream;
```

Using contexts to reinit results.

Expression with a subquery.

```
create expression runAvgRate{
  (select rate from RunAvgRate.win:length(1))};
create expression runNumber{
  (select RUN_NUMBER from levelZeroFM_static.std:unique(SID) where
      SID=sid)};
```

# Expressions

```
create objectarray schema RunAvgRate as (rate double);

  select avg(rate) as rate from L1RatesStream;
```

Using contexts to reinit results.

Expression with a subquery.

```
create expression runAvgRate{
  (select rate from RunAvgRate.win:length(1))};
create expression runNumber{
  (select RUN_NUMBER from levelZeroFM_static.std:unique(SID) where
      SID=sid)};
```

Stream with a combination of views.

```
        runAvgRate() as avgRunRate, formatMs(runDuration()) as
            runDuration ...
        from pattern [every p=RunStop];
```

# Expressions

```
create objectarray schema RunAvgRate as (rate double);

  select avg(rate) as rate from L1RatesStream;
```

Using contexts to reinit results.

Expression with a subquery.

```
create expression runAvgRate{
  (select rate from RunAvgRate.win:length(1))};
create expression runNumber{
  (select RUN_NUMBER from levelZeroFM_static.std:unique(SID) where
      SID=sid)};
```

Stream with a combination of views.

```
        runAvgRate() as avgRunRate, formatMs(runDuration()) as
            runDuration ...
        from pattern [every p=RunStop];
```

# Expressions

```
create objectarray schema RunAvgRate as (rate double);

  select avg(rate) as rate from L1RatesStream;
```

Using contexts to reinit results.

Expression with a subquery.

```
create expression runAvgRate{
  (select rate from RunAvgRate.win:length(1))};
create expression runNumber{
  (select RUN_NUMBER from levelZeroFM_static.std:unique(SID) where
      SID=sid)};
```

Stream with a combination of views.

```
        runAvgRate() as avgRunRate, formatMs(runDuration()) as
            runDuration ...
        from pattern [every p=RunStop];
```

# Expressions

```
create objectarray schema RunAvgRate as (rate double);

  select avg(rate) as rate from L1RatesStream;
```

Using contexts to reinit results.

Expression with a subquery.

```
create expression runAvgRate{
  (select rate from RunAvgRate.win:length(1))};
create expression runNumber{
  (select RUN_NUMBER from levelZeroFM_static.std:unique(SID) where
      SID=sid)};
```

Stream with a combination of views.

```
        runAvgRate() as avgRunRate, formatMs(runDuration()) as
            runDuration ...
        from pattern [every p=RunStop];
```

# Expressions

```
create objectarray schema RunAvgRate as (rate double);

  select avg(rate) as rate from L1RatesStream;

create expression runAvgRate{
  (select rate from RunAvgRate.win:length(1))};
create expression runNumber{
  (select RUN_NUMBER from levelZeroFM_static.std:unique(SID) where
      SID=sid)};

       runAvgRate() as avgRunRate, formatMs(runDuration()) as
            runDuration ...
       from pattern [every p=RunStop];
```

Using contexts to reinit results.

Expression with a subquery.

Expression invokations.

Stream with a combination of views.

# Expressions

```
create objectarray schema RunAvgRate as (rate double);

  select avg(rate) as rate from L1RatesStream;
```

Using contexts to reinit results.

Expression with a subquery.

```
create expression runAvgRate{
  (select rate from RunAvgRate.win:length(1))};
create expression runNumber{
  (select RUN_NUMBER from levelZeroFM_static.std:unique(SID) where
      SID=sid)};
```

Expression invokations.

Stream with a combination of views.

```
          runAvgRate() as avgRunRate, formatMs(runDuration()) as
              runDuration ...
          from pattern [every p=RunStop];
```

Using a registered single-row function.

# Expressions

```
create objectarray schema RunAvgRate as (rate double);

  select avg(rate) as rate from L1RatesStream;
```

Using contexts to reinit results.

Expression with a subquery.

```
create expression runAvgRate{
  (select rate from RunAvgRate.win:length(1))};
create expression runNumber{
  (select RUN_NUMBER from levelZeroFM_static.std:unique(SID) where
      SID=sid)};
```

Expression invokations.

Stream with a combination of views.

```
        runAvgRate() as avgRunRate, formatMs(runDuration()) as
            runDuration ...
        from pattern [every p=RunStop];
```

Using a registered single-row function.

## Extensibility

Esper allows users to provide custom aggregation functions, single row functions, annotations, views and others.

# Enumeration methods & output clause

```
context L1ZeroRate
  select
    date(max(b.timestamp)) as srctime,
    date(current_timestamp()) as systime,
    fedsInfoString(window(b.*).toMap(
      k=>k.fedSrcId, v=>format(v.bpFraction))) as feds
  from
    FrlBackpressure(timestamp is not null) as b,
    pattern[every FrlBackpressure->(timer:interval(1 msec)
      and not FrlBackpressure)] unidirectional
  output first every 5 seconds;
```

# Enumeration methods & output clause

Statement active only when the rate is 0.

```
context L1ZeroRate
  select
    date(max(b.timestamp)) as srctime,
    date(current_timestamp()) as systime,
    fedsInfoString(window(b.*).toMap(
      k=>k.fedSrcId, v=>format(v.bpFraction))) as feds
  from
    FrlBackpressure(timestamp is not null) as b,
    pattern[every FrlBackpressure->(timer:interval(1 msec)
      and not FrlBackpressure)] unidirectional
  output first every 5 seconds;
```

# Enumeration methods & output clause

Using a helper method fedsInfoString(Map<Long,String>).

```
context L1ZeroRate
  select
    date(max(b.timestamp)) as srctime,
    date(current_timestamp()) as systime,
    fedsInfoString(window(b.*).toMap(
      k=>k.fedSrcId, v=>format(v.bpFraction))) as feds
  from
    FrlBackpressure(timestamp is not null) as b,
    pattern[every FrlBackpressure->(timer:interval(1 msec)
      and not FrlBackpressure)] unidirectional
  output first every 5 seconds;
```

# Enumeration methods & output clause

window() aggregation function.

```
context L1ZeroRate
  select
    date(max(b.timestamp)) as srctime,
    date(current_timestamp()) as systime,
    fedsInfoString(window(b.*).toMap(
      k=>k.fedSrcId, v=>format(v.bpFraction))) as feds
  from
    FrlBackpressure(timestamp is not null) as b,
    pattern[every FrlBackpressure->(timer:interval(1 msec)
      and not FrlBackpressure)] unidirectional
  output first every 5 seconds;
```

# Enumeration methods & output clause

window() aggregation function.

toMap() enumeration function.

```
context L1ZeroRate
  select
    date(max(b.timestamp)) as srctime,
    date(current_timestamp()) as systime,
    fedsInfoString(window(b.*).toMap(
      k=>k.fedSrcId, v=>format(v.bpFraction))) as feds
  from
    FrlBackpressure(timestamp is not null) as b,
    pattern[every FrlBackpressure->(timer:interval(1 msec)
      and not FrlBackpressure)] unidirectional
  output first every 5 seconds;
```

# Enumeration methods & output clause

window() aggregation function.

toMap() enumeration function.

```
context L1ZeroRate
  select
    date(max(b.timestamp)) as srctime,
    date(current_timestamp()) as systime,
    fedsInfoString(window(b.*).toMap(
      k=>k.fedSrcId, v=>format(v.bpFraction))) as feds
  from
    FrlBackpressure(timestamp is not null) as b,
    pattern[every FrlBackpressure->(timer:interval(1 msec)
      and not FrlBackpressure)] unidirectional
  output first every 5 seconds;
```

## Enumeration methods: performing common tasks on collections of events

Some, like the aggregate() method, accept lambda expressions:

```
window(b.*).aggregate("", (result, row)=>result||" "||row.fedSrcId)
```

# Enumeration methods & output clause

window() aggregation function.

toMap() enumeration function.

```
context L1ZeroRate
  select
    date(max(b.timestamp)) as srctime,
    date(current_timestamp()) as systime,
    fedsInfoString(window(b.*).toMap(
      k=>k.fedSrcId, v=>format(v.bpFraction))) as feds
  from
    FrlBackpressure(timestamp is not null) as b,
    pattern[every FrlBackpressure->(timer:interval(1 msec)
      and not FrlBackpressure)] unidirectional
  output first every 5 seconds;
```

Using a helper method format(double).

## Enumeration methods: performing common tasks on collections of events

Some, like the aggregate() method, accept lambda expressions:

```
window(b.*).aggregate("", (result, row)=>result||" "||row.fedSrcId)
```

# Enumeration methods & output clause

window() aggregation function.

toMap() enumeration function.

Pattern triggered selection.

```
context L1ZeroRate
  select
    date(max(b.timestamp)) as srctime,
    date(current_timestamp()) as systime,
    fedsInfoString(window(b.*).toMap(
      k=>k.fedSrcId, v=>format(v.bpFraction))) as feds
  from
    FrlBackpressure(timestamp is not null) as b,
    pattern[every FrlBackpressure->(timer:interval(1 msec)
      and not FrlBackpressure)] unidirectional
  output first every 5 seconds;
```

## Enumeration methods: performing common tasks on collections of events

Some, like the aggregate() method, accept lambda expressions:

```
window(b.*).aggregate("", (result, row)=>result||" "||row.fedSrcId)
```

# Enumeration methods & output clause

window() aggregation function.

toMap() enumeration function.

```
context L1ZeroRate
  select
    date(max(b.timestamp)) as srctime,
    date(current_timestamp()) as systime,
    fedsInfoString(window(b.*).toMap(
      k=>k.fedSrcId, v=>format(v.bpFraction))) as feds
  from
    FrlBackpressure(timestamp is not null) as b,
    pattern[every FrlBackpressure->(timer:interval(1 msec)
      and not FrlBackpressure)] unidirectional
  output first every 5 seconds;
```

Output stabilizing clause.

## Enumeration methods: performing common tasks on collections of events

Some, like the aggregate() method, accept lambda expressions:

```
window(b.*).aggregate("", (result, row)=>result||" "||row.fedSrcId)
```

# Enumeration methods & output clause

window() aggregation function.

toMap() enumeration function.

```
context L1ZeroRate
  select
    date(max(b.timestamp)) as srctime,
    date(current_timestamp()) as systime,
    fedsInfoString(window(b.*).toMap(
      k=>k.fedSrcId, v=>format(v.bpFraction))) as feds
  from
    FrlBackpressure(timestamp is not null) as b,
    pattern[every FrlBackpressure->(timer:interval(1 msec)
      and not FrlBackpressure)] unidirectional
  output first every 5 seconds;
```

Output stabilizing clause.

## Enumeration methods: performing common tasks on collections of events

Some, like the aggregate() method, accept lambda expressions:

```
window(b.*).aggregate("", (result, row)=>result||" "||row.fedSrcId)
```

## Output clause: suppressing output and controlling its rate

The suppressing logic allows reading and setting variables, using crontab notation and more.