Procesamiento de Eventos Complejos con Esper

Itinerario Formativo de Doctorado 7009

Juan Boubeta Puig

Grupo UCASE de Ingeniería del Software
Departamento de Ingeniería Informática

30 de mayo de 2013

¿Qué es Esper? (I)

- Motor de procesamiento de eventos complejos o *Complex Event Processing* (CEP) de la compañía EsperTech Inc.
- Código abierto (licencia GPL) y disponible en Java y .NET.
- Fundamental para arquitecturas dirigidas por eventos o *Event-Driven Architecture* (EDA) que necesitan procesar infomación en tiempo real.
- Permite lanzar acciones personalizadas cuando se cumplan ciertas condiciones sobre los flujos de eventos.
- Capacidad para correlacionar miles de eventos por segundo.
- Se almacenan las “consultas” (patrones) en lugar de los “datos”.
- Las consultas/patrones se escriben en EPL (*Event Processing Language*).

¿Qué es Esper? (II)

Escenario de detección de un caso sospechoso de gripe aviar

Patrón de evento complejo

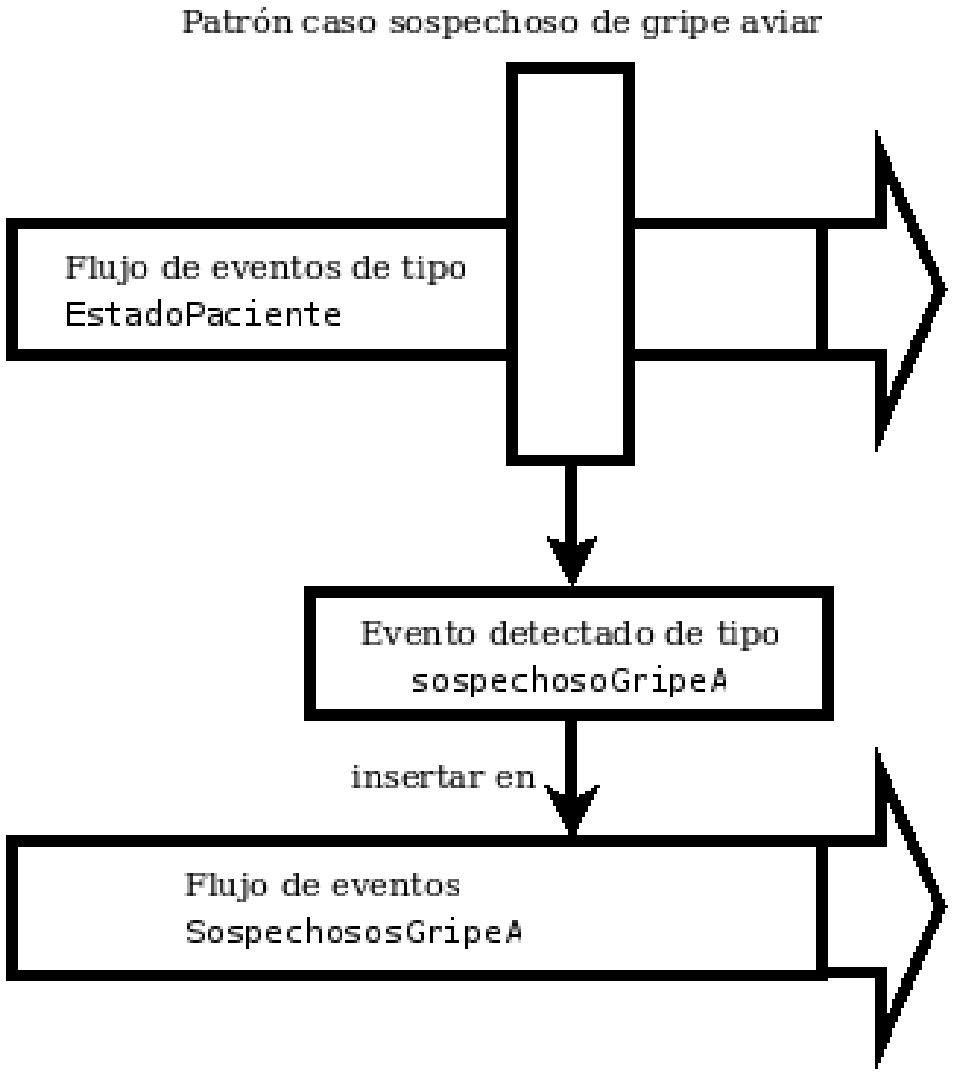
detección

Sospechoso de gripe aviar

Evento complejo

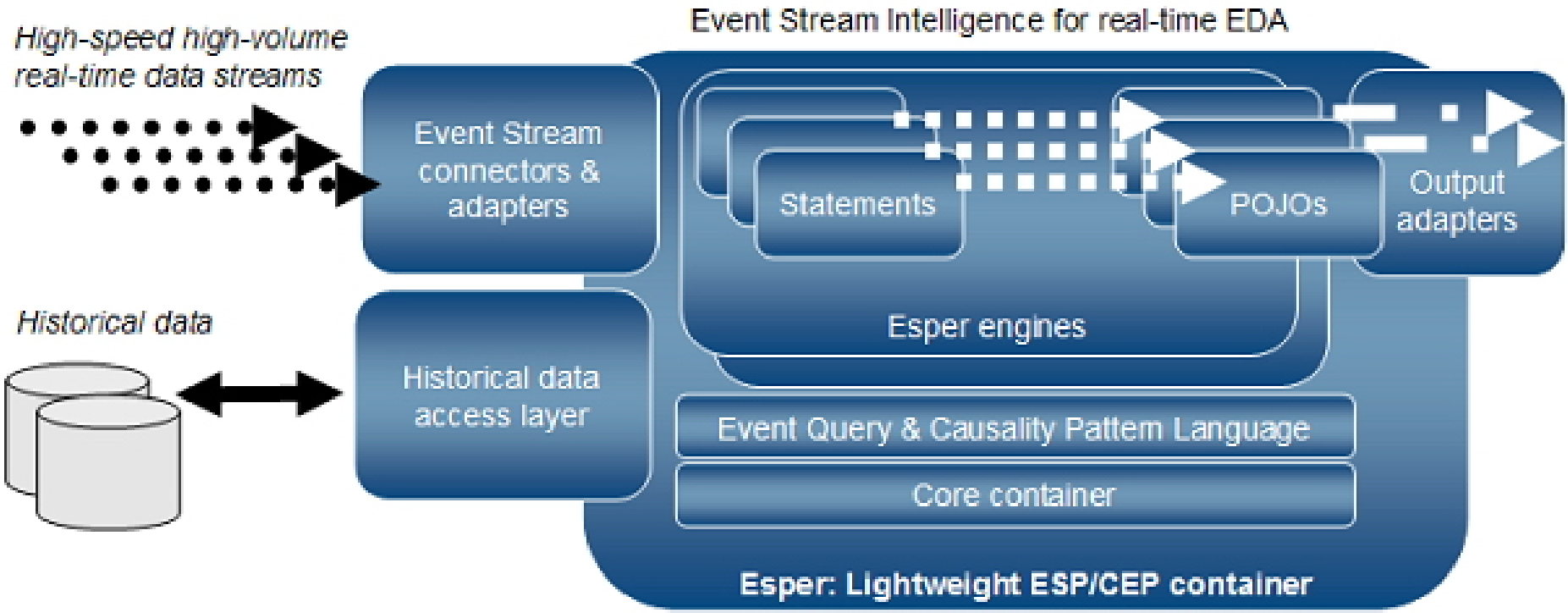
¿Qué es Esper? (III)

Escenario de detección de un caso sospechoso de gripe aviar



Arquitectura de Esper

Eventos, patrones de eventos y *listeners* (interfaces para notificación de eventos)



Modelo de datos

- Flujos de eventos:
  - Los atributos de los eventos se definen con tipos y valores: string, boolean, integer, long, float, byte, etc.
  - Varios formatos de eventos: POJO (*Plain Old Java Object*), documentos XML y *map* de Java.
- Tipos de flujos:
  - Flujos de ”entrada” (`istream`): por defecto, los nuevos eventos que llegan se incluyen en una ventana de datos.
  - Flujo de ”eliminación” (`rstream`): los eventos antiguos dejan una ventana de datos.
  - Ambos tipos (`irstream`).

Declaración de eventos (I)

Schema

```
create schema schema_name[as] (property_name property_type
   [,property_nameproperty_type[,...]) [inherits
   inherited_event_type[, inherited_event_type] [,...]]
```

XML

Como instancias de *org.w3c.dom.Node*.

Map de Java

java.util.Map

Declaración de eventos (II)

Clase Java

```
package org.myapp.event;

public class OrderEvent {
  private String itemName;
  private double price;

  public OrderEvent(String itemName, double price) {
    this.itemName= itemName;
    this.price= price;
  }

  public String getItemName() { return itemName; }

  public double getPrice() { return price; }
}
```

¿Cómo crear un evento?

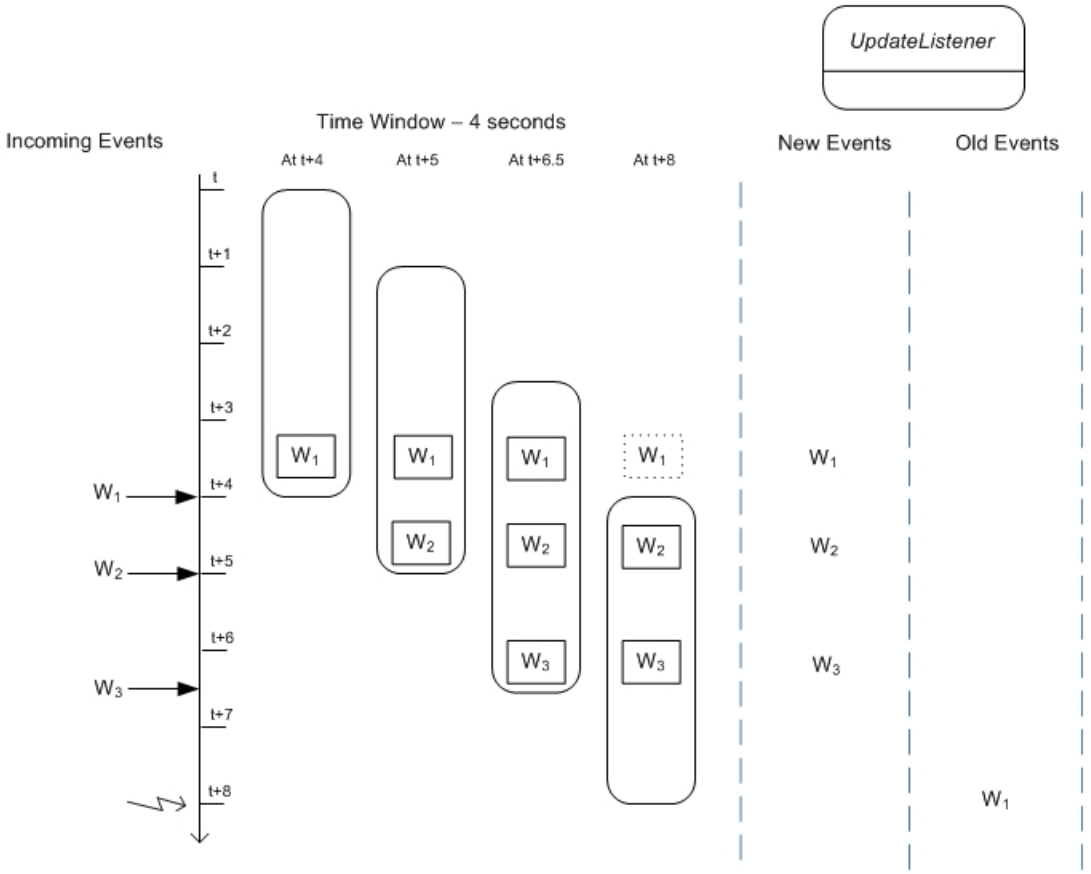
```
// Crear un nuevo objeto de evento
OrderEvent event = new OrderEvent(''shirt'', 74.50);

// Enviar el objeto a la instancia del motor
epService.getEPRuntime().sendEvent(event);
```

Ventanas (I)

Sliding window o ventana deslizante

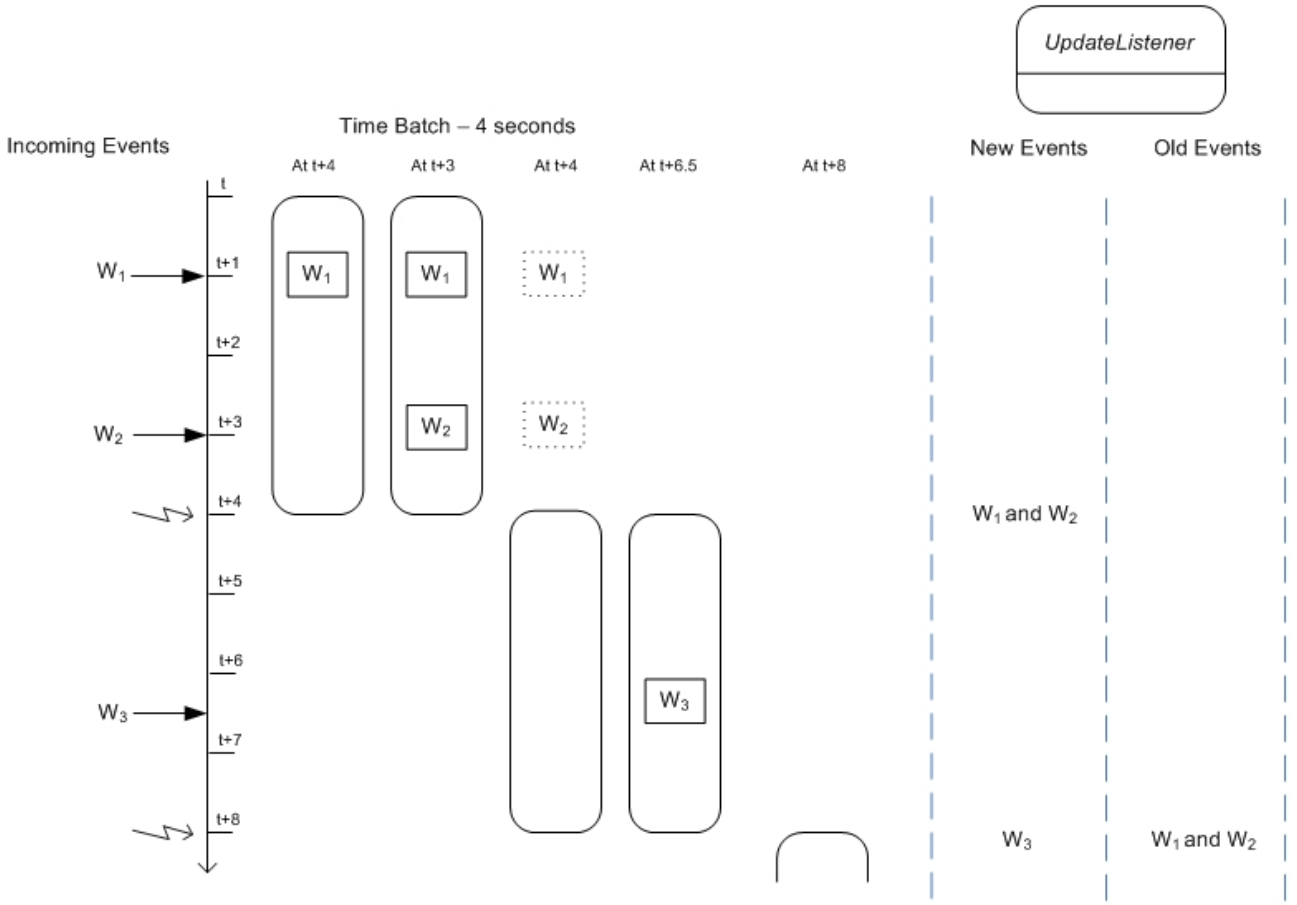
- Se desplazan paso a paso en pequeños incrementos.
- Un evento en concreto puede formar parte de una misma ventana durante varias iteraciones.



Ventanas (II)

Tumbling window o ventana abatible

- Se desplazan en incrementos correspondientes a su longitud.
- Cada evento forma parte de una ventana solamente una vez.



Ventanas (III)

Por lotes (*batch windows*) o normales (*non-batch windows*)

Por lotes No lanzan ningún evento hasta que no se llenan por completo.

Normales Producen un evento de salida cada vez que la entrada cambia.

Ventanas (IV)

A una ventana (*window*) se le denomina vista (*view*) en Esper

Vista	Sintaxis	Descripción
Length	win:length(size)	Ventana deslizante según el número de eventos.
LengthBatch	win:length_batch(size)	Ventana abatible que agrupa eventos y los lanza cuando un número de eventos ha sido recogido.
Time	win:time(time_period)	Ventana deslizante según período de tiempo especificado.
TimeAccumulating	win:time_accum(time_period)	Ventana deslizante que acumula eventos hasta que no llegan más eventos durante un período de tiempo.
TimeBatch	win:time_batch(time_period ...)	Ventana abatible que agrupa eventos y los lanza por cada período de tiempo especificado.
TimeLengthBatch	win:time_length_batch(time_period, size)	Ventana abatible según el período de tiempo y la longitud.

Registros y períodos de tiempo

- Tiempo implícito: Esper utiliza el reloj del sistema.
- Tiempo explícito: Se pueden utilizar los registros de tiempo de los datos de los flujos de eventos.
- Los períodos de tiempo se definen en las consultas:

```
time-period : [day-part] [hour-part] [minute-part]
  [seconds-part] [milliseconds-part]
day-part : (number|variable_name) ("days" | "day")
hour-part : (number|variable_name) ("hours" | "hour")
minute-part : (number|variable_name) ("minutes" | "minute" | "min")
seconds-part : (number|variable_name) ("seconds" | "second" | "sec")
milliseconds-part : (number|variable_name)
  ("milliseconds" | "millisecond" | "msec")
```

Sintaxis de EPL

- Una sentencia en EPL es una consulta continua que se evalúa constantemente sobre el flujo de eventos.
- La sintaxis de EPL es:

```
[annotations]
[expression_declarations]
[context context_name]
[insert into insert_into_def]
select select_list
from stream_def [as name] [, stream_def [as name]] [,...]
[where search_conditions]
[group by grouping_expression_list]
[having grouping_search_conditions]
[output output_specification]
[order by order_by_expression_list]
[limit num_rows]
```

Análisis de flujos de eventos (I)

```
// Ejemplo 1
select avg(precio) as mediaPrecio
from EventoMarcaStock.win:time(20 sec)

// Ejemplo 2
select símbolo, avg(precio) as mediaPrecio
from EventoMarcaStock.win:length(50)
group by símbolo
```

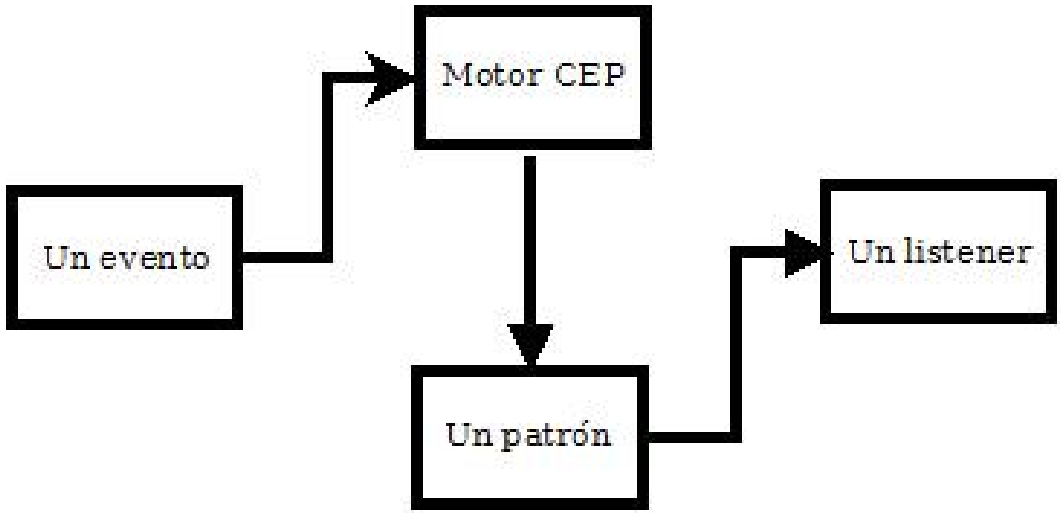
Análisis de flujos de eventos (II)

Condiciones que pueden ser definidas para unir flujos de eventos o filtrar eventos

```
// Ejemplo 3
select fraude.númeroCuenta as numCuenta,
  fraude.aviso as aviso,
  sacarDinero.cantidad as cantidad,
  MAX(fraude.tiempo, sacarDinero.tiempo) as tiempo,
  'fraudeAlSacarDinero' as desc
from EventoAvisoFraude.win:time(30 min) as fraude,
  EventoSacarDinero.win:time(30 sec) as sacarDinero
where fraude.númeroCuenta = sacarDinero.númeroCuenta
```

Patrones de eventos (I)

- Los patrones en EPL se registran en el motor Esper.
- Si un patrón se activa cuando se recibe uno o varios eventos, el motor lo notificará al *listener* asociado al patrón.
- Cada patrón puede estar asociado con uno o varios *listeners*.
- Un *listener* realizará una tarea concreta: imprimir por pantalla la información del evento que recibe, envíar un correo electrónico, etc.



Patrones de eventos (II)

Átomos

Bloques básicos de construcción de patrones. Tipos:

- Expresiones de filtro que especifican un evento concreto a buscar: `MarcaStock(símbolo='ABC', precio > 100)`
- Observadores de eventos que especifican intervalos de tiempo o un tiempo específico:
  - **timer:interval** Espera el tiempo determinado antes de que el valor del observador sea cierto. `A -> timer:interval(10 seconds)`
  - **timer:at** La expresión se evalúa verdadera cuando se alcanza el tiempo especificado. `timer:at (minutes, hours, days of month, months, days of week [, seconds])`
- Observadores personalizados que observan eventos externos que no están bajo el control del motor.

Patrones de eventos (III)

Operadores (I)

Controlan el ciclo de vida de las expresiones y combinan lógica o temporalmente átomos de patrones. Tipos:

- Operadores temporales que controlan el orden de eventos: **->** (seguido-por). `A -> (B or C)`
- Operadores que controlan repeticiones de subexpresiones:
  - **[num]** El patrón se dispara cuando llegan los últimos *num* eventos. `[5] A`.
  - **until** El patrón se evalúa a verdadero hasta que se cumpla la condición especificada. `A until B`.
  - **every** El patrón se dispara cada vez que se encuentre con un evento del tipo especificado. `every A`.
    Si no se especifica este operador el patrón no seguirá buscando una vez se haya detectado la primera ocurrencia de ese tipo de evento.
  - **every-distinct** Elimina los resultados duplicados, a partir de las propiedades de eventos especificadas. `every-distinct(a.id) a=A`

Patrones de eventos (IV)

Operadores (II)

- Operadores lógicos: **and**, **or**, **not**.
- Guardas (condiciones *where* que controlan el ciclo de vida de las subexpresiones):
  - **timer:within** Si la expresión del patrón asociado no se evalúa a verdadera en el período de tiempo especificado terminará y será evaluada como falsa permanentemente.
    `A where timer:within(5 seconds)`
  - **timer:withinmax** La subexpresión termina cuando finaliza el período de tiempo o se alcanza el valor máximo del contador.
    `(every A) where timer:withinmax (5 seconds, 2)`
  - **while** La subexpresión del patrón termina cuando la expresión se evalúa a falsa. `(every a=A) while (a.size > 0)`

Patrones de eventos (V)

Operadores (III): ¡cuidado con el uso del operador *every*!

$A_1\ B_1\ C_1\ B_2\ A_2\ D_1\ A_3\ B_3\ E_1\ A_4\ F_1\ B_4$

- **every (A -> B)** Detecta un evento A seguido por un evento B. En el momento en el que ocurre B el patrón se dispara, entonces el patrón se reinicia y busca el próximo evento A: $\{A_1, B_1\}$, $\{A_2, B_3\}$, $\{A_4, B_4\}$.
- **every A -> B** Este patrón se dispara para cada evento A seguido por un evento B: $\{A_1, B_1\}$, $\{A_2, B_3\}$,$\{A_3, B_3\}$, $\{A_4, B_4\}$.
- **A -> every B** Se dispara para un A seguido por cada B: $\{A_1, B_1\}$, $\{A_1, B_2\}\{A_1, B_3\}\{A_1, B_4\}$.
- **every A -> every B** Este patrón se dispara para cada evento A seguido por cada evento B: $\{A_1, B_1\}$, $\{A_1, B_2\}$, $\{A_1, B_3\}$, $\{A_2, B_3\}$, $\{A_3, B_3\}$, $\{A_1, B_4\}$, $\{A_2, B_4\}$, $\{A_3, B_4\}\{A_4, B_4\}$.

Detección de patrones eventos + análisis de flujos de eventos

```
select a.id, count(*)
from pattern [
  every a=Estado -> (timer:interval(10 sec)
    and not Estado(id = a.id)]
group by id
```

Funciones de agregación

- Algunas de las funciones: **sum**, **avg**, **count**, **max**, **min**.
- Permiten el uso de **group-by** y **having**.

```
select sum(price)
from StockTickEvent.win:time(50 sec)
```

Ejemplo sencillo (I)

- Se registra la consulta que devuelve la media de los precios de los eventos *OrderEvent* (ver transparencias 11 y 12) que llegaron en los últimos 30 s:

```
EPServiceProvider epService =
   EPServiceProviderManager.getDefaultProvider();
String expression = "select avg(price)
   from org.myapp.event.OrderEvent.win:time(30 sec)";
EPStatementstatement =
   epService.getEPAdministrator().createEPL(expression);
```

Ejemplo sencillo (II)

- Se registra un *listener* en el motor que se encargará de “escuchar” la consulta:

```
public class MyListener implements UpdateListener{
  public void update(EventBean[] newEvents,
                                EventBean[] oldEvents) {
    EventBean event = newEvents[0];
    System.out.println("avg=" + event.get("avg(price)"));
  }
}
...
...
MyListener listener = new MyListener();
statement.addListener(listener);
```

Ejemplo sencillo (III)

- Se realiza la configuración del motor:

```
Configuration config= new Configuration();
config.addEventTypeAutoAlias("org.myapp.event");
EPServiceProvider epService =
  EPServiceProviderManager.getDefaultProvider(config);
```

Otros ejemplos

- http://coffeeonesugar.wordpress.com/2009/07/21/getting-started-with-esper-in-5-minutes/
- http://www.debugrelease.com/2012/06/14/complex-event-processing-with-esper-tutorial-for-beginner
- http://esper.codehaus.org/tutorials/tutorial/examples.html (StockTicker y MatchMaker para comenzar). Para ejecutarlo:
  1. Tener instalado Java 1.6 o superior y configurada la variable de entorno JAVA_HOME.
  2. Abrir una ventana de consola y cambiar al directorio: *esper-X.X.X/examples/etc*
  3. Ejecutar esper-X.X.X/examples/stockticker/etc/setenv.bat
  4. Ejecutar esper-X.X.X/examples/stockticker/etc/compile.bat
  5. Ejecutar esper-X.X.X/examples/stockticker/etc/run_stockticker.bat

Referencias bibliográficas I


EsperTech Inc.
Quick Start
http://esper.codehaus.org/tutorials/tutorial/quickstart.html, mayo 2013.


EsperTech Inc.
Tutorial
http://esper.codehaus.org/tutorials/tutorial/tutorial.html, mayo 2013.


EsperTech Inc.
Solution Patterns
http://esper.codehaus.org/tutorials/solution_patterns/solution_patterns.html, mayo 2013.

Referencias bibliográficas II


EsperTech Inc.
Esper EPL Online
http://esper-epl-tryout.appspot.com/epltryout/index.html, mayo 2013.


EsperTech Inc.
Documentation: Esper & EsperIO
http://esper.codehaus.org/esper/documentation/documentation.html, mayo 2013.


EsperTech Inc.
Download Esper
http://esper.codehaus.org/esper/download/download.html, mayo 2013.