

CEP/ESP: Procesamiento y correlación de gran cantidad de eventos en arquitecturas SOA

Víctor Ayllón (vayllon@novayre.es) – Director Gerente de Novayre
Juan M. Reina (jmreina@novayre.es) – Director de Tecnología de Novayre

Abstract. El “matrimonio” entre ESP/CEP y las arquitecturas orientadas a servicios son un tema de actualidad y cuya aplicación práctica en proyectos SOA verá la luz en los próximos meses. En este modelo, se deben combinar los mecanismos de publicación-suscripción basados en mensajes con la capacidad de procesar y correlacionar en tiempo real eventos de negocio de alto nivel. En este artículo se exponen los principales conceptos asociados al modelo de procesamiento y correlación de grandes cantidades de eventos complejos, conocido con sus siglas inglesas por ESP (Event Stream Processing) y CEP (Complex Event Processing) y cómo integrar esta infraestructura en una arquitectura SOA. Asimismo se expone un ejemplo de aplicación de ESP/CEP haciendo uso de la solución Open-Source Esper.

Keywords: BAM, CEP, ESP, SOA, Esper

1 Introducción

Los sistemas de publicación y suscripción basados en eventos siempre han formado parte de las arquitecturas relacionadas con la integración de sistemas o EAI a través de plataformas MOM (Message Oriented Middleware). Más recientemente se han integrado en las arquitecturas SOA dentro de una infraestructura ESB (Enterprise Service Bus).

Asimismo, desde hace ya varios años, los sistemas BAM [1] (Business Activity Monitoring) permiten el procesamiento de eventos de negocio desde distintas fuentes así como su notificación para que los responsables de la monitorización tomen una decisión en función del contexto del evento.

Por lo tanto, el reto actual, no es la generación, captura o notificación de dichos eventos, sino el poder procesar gran cantidad de sucesos producidos en tiempo real, la correlación de dichos eventos generando respuestas automáticas basadas en la semántica del evento y por último, su integración en una arquitectura SOA.

El objetivo de este artículo es exponer los principios y conceptos básicos del modelo ESP/CEP, cómo se puede integrar dicho modelo en una arquitectura SOA, la relevancia de un nuevo enfoque y lenguaje para procesar eventos (EPL) y por último un pequeño ejemplo de aplicación (incluido fragmentos de código fuente y trazas) que hace uso de la solución Open-Source ESP/CEP de referencia: el proyecto Esper [2]

2 Definición de ESP/CEP

En primer lugar es necesario distinguir entre ESP y CEP, aunque es importante señalar que aún no hay una definición formal y consensuada por toda la industria[3].

Podríamos afirmar que el objetivo de ESP es la captura y el procesamiento de una gran cantidad de eventos en una ventana temporal concreta. Es lo que se conoce como “corriente” de eventos (*‘event stream’* en inglés). Esta corriente se podría definir como una secuencia de eventos, del mismo tipo, ordenados en el tiempo. Por su parte, CEP tiene por meta la captura y procesamiento de eventos de diferente tipo de una forma desordenada en la llamada “nube” de eventos (*‘event cloud’* en inglés). Digamos pues que una nube de eventos puede contener muchas corrientes de eventos y una corriente es un caso “especial” de nube (los eventos son del mismo tipo).

Una arquitectura ESP/CEP debe ser capaz de procesar las “nubes y corrientes” de eventos generados por uno o varios productores, almacenar y clasificar los eventos en un repositorio teniendo en cuenta la dimensión temporal (cuándo se producen los eventos), tener mecanismos para detectar patrones complejos y generar respuestas automáticas a los consumidores de los patrones detectados.

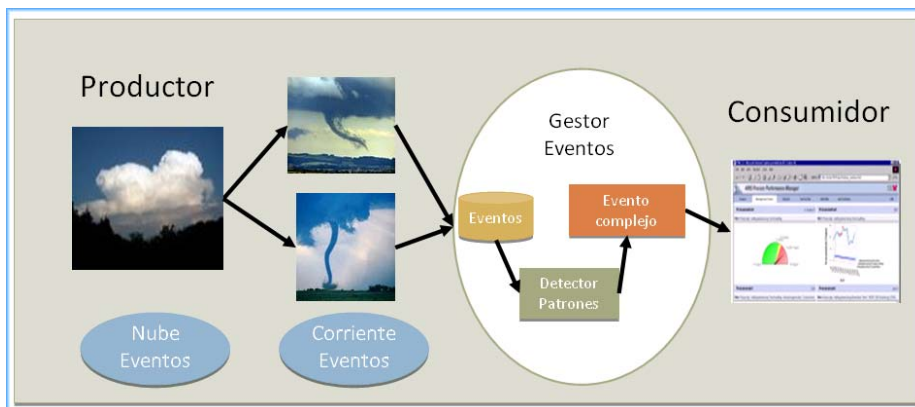


Fig. 1. Componentes de ESP/CEP

Pongamos un sencillo ejemplo de ESP/CEP: Supongamos un sistema financiero que tiene por objetivo el procesamiento en tiempo real de las acciones y de las noticias asociadas a las compañías que cotizan en la bolsa de Madrid.

- Mediante ESP, podríamos obtener la cotización media que ha tenido una acción en una ‘corriente’ de eventos concreta, es decir, en un intervalo de tiempo determinado: por ejemplo, el valor medio de una compañía como media de los valores recibidos en los últimos 5 minutos de sesión.
- Mediante CEP, podríamos ir más allá y definir patrones complejos que correlacionen la publicación de una noticia con la cotización de una compañía y que tome decisiones al respecto: por ejemplo, una regla podría ser si aparece una noticia de una compañía y en un intervalo temporal determinado la cotización de la acción sufre una brusca bajada entonces vender automáticamente las acciones de dicha empresa.

En el último apartado expondremos cómo se podría implementar este sistema utilizando la solución Open Source Esper.

3 Arquitectura solución ESP/CEP

Una típica arquitectura SOA a alto nivel esta formada por los siguientes elementos:

- **Sistemas Legacy** que a través de ‘Binding Component’ se conecte a un ESB (Enterprise Service Bus)
- **Servicios Web** que proporcionen funcionalidad de negocio al ESB
- Un **motor BPEL** para realizar la coreografía de Servicios Web
- Un **workflow BPM** para el modelado de negocio de aquellos procesos que requieran de intervención humana
- **Aplicaciones** de usuario final y **herramientas de monitorización** (BAM)

El componente a añadir sería un Gestor de Eventos Complejos el cual tendría que procesar y almacenar todos los eventos producidos en el ESB y en base a patrones ser capaz de inferir eventos complejos y generar respuestas automáticas.

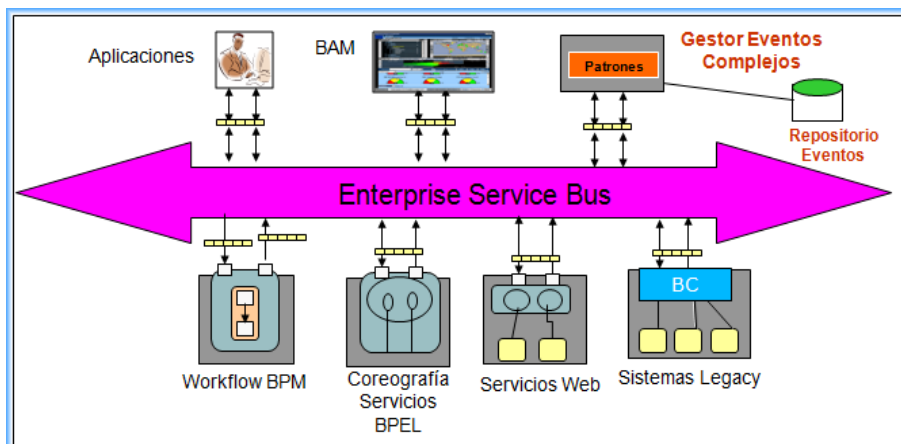


Fig. 2. Arquitectura SOA con gestor de eventos complejos

El Gestor de Eventos Complejos deberá ser capaz de registrar y procesar en tiempo real los eventos generados en todo el ESB. Esto implica que las implementaciones de este gestor deben optimizar su rendimiento. Mientras en otros elementos de una arquitectura SOA es admisible, aunque no deseable, cierta latencia, el Gestor de Eventos Complejos debe tener como primer requerimiento el rendimiento y como segundo la capacidad de gestionar la “ventana temporal” de los eventos.

4 EPL: Event Pattern Language

No obstante, además de un Gestor de Eventos Complejos, la disciplina ESP/CEP necesita de un nuevo enfoque por el cual, a través de un lenguaje de programación, se permita no solo la consulta y el procesamiento de eventos sino la posibilidad de añadir una “dimensión temporal” y lo que es más importante inferir nuevos eventos basados en patrones semánticos.

Este lenguaje se conoce como EPL (Event Pattern Language). EPL amplía y extiende a SQL en la definición de reglas temporales y en la aplicación de reglas no lineales para el procesamiento de eventos.

Algunos ejemplos de EPL's son:

- RAPIDE-EPL [4]
- StreamSQL [5]
- Coral8 CCL [6]

La aproximación de la plataforma Open Source Esper, ha sido la creación de un lenguaje de consulta similar a SQL, llamado EQL (Event Query Language), que permite filtrar eventos en función de una ventana temporal, crear nuevas corrientes de eventos a partir de una o varias corrientes existentes y definir patrones que faciliten aplicar reglas de negocio a los eventos capturados.

5 Ejemplo ESP/CEP utilizando Esper

a. Proyecto Esper

El proyecto Esper es una implementación Java Open Source del concepto CEP/ESP, y ha sido desarrollado por la compañía EsperTech Inc.

La siguiente figura muestra la arquitectura a alto nivel de Esper:

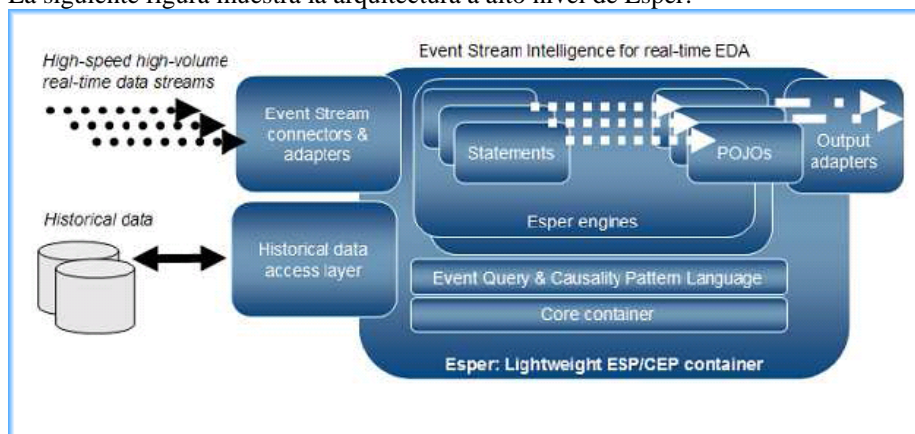


Fig. 3. Arquitectura a alto nivel de Esper

Como se aprecia en la figura anterior (figura 3), la arquitectura de Esper se basa en un contenedor ligero formado por diversos componentes, entre los que se incluye:

- Diversos motores: gestión de eventos, multithread, etc
- POJOs: asociados a los propios eventos
- EPL: lenguaje similar a SQL para la detección de patrones complejos
- Listener: interfaces Java para la notificación de eventos

Como implementación de ESP/CEP, Esper permite el procesamiento a gran escala y con excelente rendimiento de flujos de eventos utilizando ventanas temporales y permitiendo la ejecución de consultas complejas de forma continua que correlacionan distintas corrientes de eventos para detectar el patrón buscado.

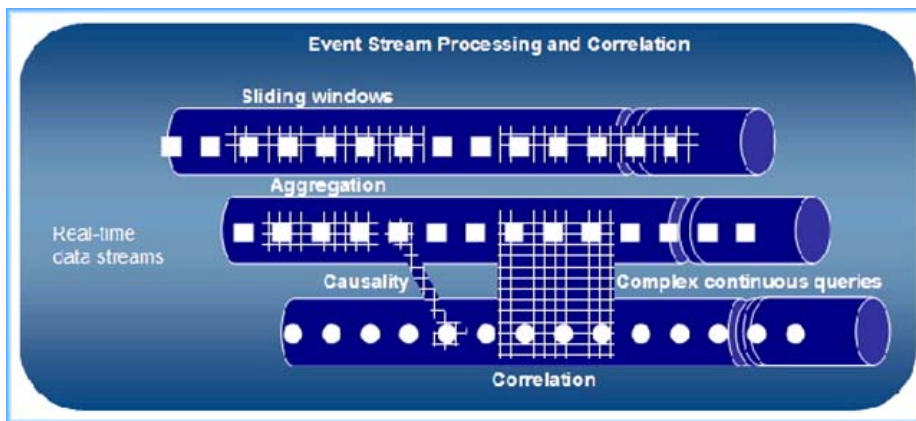


Fig. 4. Modelo de Procesamiento y Correlación de Esper

Como todos sabemos, en el mundo real, existen casos en los que un retraso o latencia de pocos segundos puede resultar en un grave perjuicio económico, organizativo o incluso vital. En el siguiente ejemplo vamos a plantear, desarrollar y resolver un problema que requiere de respuesta en tiempo real. Afortunadamente el perjuicio de no resolver adecuadamente la situación que se plantea a continuación tan sólo es económico.

b. Descripción del problema

El problema a resolver sería el siguiente: Poseemos una pequeña empresa de gestión de carteras de valores, denominada GestCarteVa. Nuestra clientela ha aumentado considerablemente en un período de tiempo muy corto gracias a una correcta gestión de la información financiera. Nos hemos especializado en reaccionar con mucha celeridad ante bruscas caídas o subidas de los valores de nuestros clientes justo tras la publicación de alguna noticia relacionada con estos valores. Esto nos ha llevado a ganar prestigio y más clientes pero nos ha creado un problema, ya que no somos capaces de responder con la celeridad del principio. Lo que antes hacíamos a mano porque teníamos pocos clientes, noticias y pocos valores a analizar, debido al repentino aumento de nuestra clientela ahora ya no es posible, debemos automatizar al máximo esta tarea.

Para ello hemos contactado con una empresa que proporciona información de la cotización bursátil y de noticias a través de Internet utilizando los siguientes servicios web:

- El servicio web de cotización bursátil nos enviará una notificación en tiempo real con la cotización de cada valor en una sesión por lo que el volumen de información y su frecuencia es alto.
- El servicio web de noticias financieras nos enviará una notificación cuando se produce una noticia relativa a alguna compañía que cotice en bolsa, estas noticias no tienen una periodicidad determinada.

Vamos a utilizar un enfoque CEP/ESP y la plataforma Open Source Esper para el desarrollo de una sistema que permita solucionar el problema.

c. Enfoque de la solución

Actualmente, GestCarteVa utiliza un enfoque estático para ver si una noticia que se publique referente a una compañía puede hacer que su cotización baje o suba. Digamos que sobre la “nube” de eventos de cotizaciones y noticias que se producen en cada momento, los empleados de la compañía deben detectar las relaciones entre las cotizaciones y las noticias publicadas.

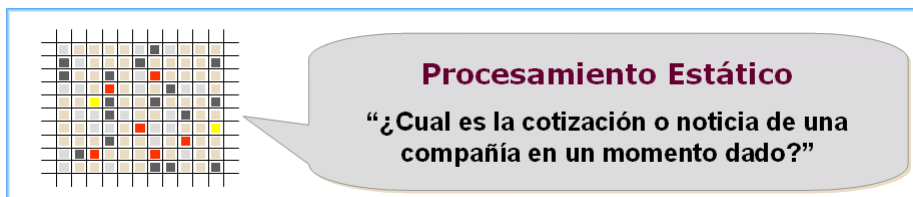


Fig. 5. Procesamiento Estático: nube de eventos en un instante determinado

El enfoque de la una solución CEP/ESP debe permitir procesar de forma independiente las distintas “corrientes” de eventos relacionados con cotizaciones y noticias y correlacionarlos mediante un patrón semántico que podría ser el siguiente: “una bajada en una cotización de una compañía que supere un determinado umbral y

que se haya producido en un intervalo temporal concreto tras el anuncio de una noticia de la compañía debe generar una orden de venta”

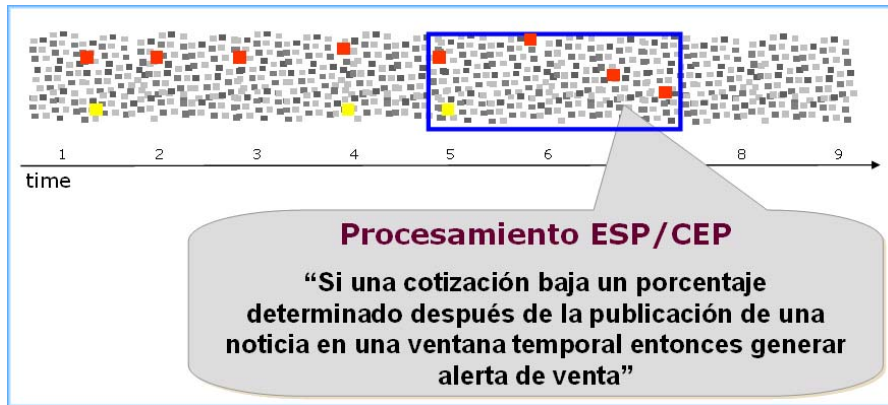


Fig. 6. Procesamiento ESP/CEP: nube de eventos y correlación en ventana temporal

d. Desarrollo técnico

En este apartado vamos a detallar como se implementaría una solución al problema con Esper, mostrando incluso fragmentos del código fuente.

Técnicamente, lo primero que debemos hacer es inicializar Esper y obtener un proveedor de servicios que nos pueda ayudar en nuestra tarea.

Esto lo podemos hacer en una clase de utilidad como la que se muestra en la figura siguiente:

```

package com.novayre.esper.jsweb;

import com.espertech.esper.client.Configuration;

public class EsperUtil {

    private static EPServiceProvider epService;

    public static EPServiceProvider getService() {

        synchronized(EsperUtil.class) {

            if(epService == null) {

                Configuration config = new Configuration();
                config.addEventTypeAutoAlias("com.novayre.esper.jsweb.dto");

                epService = EPServiceProviderManager.getDefaultProvider(config);
            }

        }

        return epService;
    }

    public static EPStatement getStatement(String expression) {
        return getService().getEPAdministrator().createEPL(expression);
    }

    public static EPStatement getPattern(String expression) {
        return getService().getEPAdministrator().createPattern(expression);
    }
}

```

Fig. 7. Clase de utilidad para inicializar Esper y generar Sentencias y Patrones EPL

Esta clase contiene tres métodos estáticos, el primero `getService()` inicializa Esper y nos devuelve una instancia del proveedor de servicios. En este caso le estamos diciendo a Esper que nuestros eventos serán instancias de clases del paquete `com.novayre.esper.jsweb.dto`. Esto nos evitará en el futuro tener que incluir el nombre completo de la clase en cada sentencia Esper. Los otros dos métodos, utilizados para generar sentencias y patrones EPL, los comentamos más adelante.

Tras la obtención de un proveedor de servicios necesitamos varios generadores de eventos, es decir, clases que nos permitan introducir en los flujos de eventos de Esper corrientes las notificaciones recibidas de los servicios web.

Para el caso de las cotizaciones bursátiles la clase transformará cada mensaje recibido en una instancia de la clase `MarketValueEvent` y a continuación insertará las instancias en el flujo.

En la siguiente figura se muestra el fragmento de la clase que inserta una instancia de `MarketValueEvent` en Esper:

```
MarketValueEvent mve = new MarketValueEvent(cv.getCompany(), cv.getValue(),
    System.currentTimeMillis());
EsperUtil.getService().getEPRuntime().sendEvent(mve);
```

Fig. 8. Fragmento que inserta un evento en Esper

Para los mensajes recibidos del servicio web de noticias financieras haremos lo mismo generando eventos de la clase `CompanyNewsEvent`.

Una vez realizados los pasos anteriores, tendremos los eventos en flujos manejados por Esper. El siguiente reto es poder tratar estos eventos según nuestras necesidades.

Queremos que Esper detecte cuándo la desviación estándar de alguno de los valores supere un cierto límite, en este caso el umbral es un euro.

Para ello vamos a definir una sentencia Esper, utilizando el lenguaje EQL, con la ayuda del método estático `getStatement(String)` de la clase `EsperUtil` que definimos anteriormente:

```
EPStatement stmt = EsperUtil.getStatement(
    "insert into FilteredMarketValueEvent " +
    "select company, stddev(value) " +
    "from MarketValueEvent.win:time_batch(5 sec) " +
    "group by company " +
    "having stddev(value) > 1");

stmt.addListener(new FilteredMarketValueListener());
```

Fig. 9. Fragmento que define una sentencia EQL y añade un listener para su tratamiento

El método `getStatement(String)` nos devuelve una sentencia Esper validada por el gestor de eventos y puesta en ejecución de forma inmediata, es decir, Esper evalúa y valida el texto enviado y si es válido lo ejecuta. Una sentencia Esper permanece en ejecución hasta que es detenida por la aplicación.

En esta sentencia lo que hacemos es agrupar por compañía la desviación estándar del valor de su acción en bolsa, si esta desviación supera el límite de un euro, Esper insertará un evento en el flujo `FilteredMarketValueEvent` y enviará una notificación al listener que hemos añadido en la segunda instrucción de la figura anterior.

En la sentencia EQL introducimos una ventana temporal, es decir, tan sólo tenemos en cuenta para nuestro cálculo los eventos producidos en los últimos 5 segundos.

Podemos ampliar, reducir o eliminar esta ventana temporal como deseemos o tengamos necesidad.

Tras esto, recibiremos notificaciones de Esper cada vez que un valor bursátil fluctúe más allá del límite que le hemos fijado.

La recepción de estas notificaciones se muestra en la siguiente figura:

```

package com.novayre.esper.jsweb.cep.listener;

import com.espertech.esper.client.UpdateListener;

public class FilteredMarketValueListener implements UpdateListener {

    @Override
    public void update(EventBean[] arg0, EventBean[] arg1) {

        EventBean event = arg0[0];

        String company = (String)event.get("company");
        Double stddev_value = (Double)event.get("stddev(value)");

        System.out.println("DANGER Company = " + company +
            ", stddev(value) = " + stddev_value);

    }

}

```

Fig. 10. Clase que recibe las notificaciones de los eventos asociados a sentencias EQL

Ahora lo que queremos recibir es una notificación de Esper por cada noticia financiera enviada por el servicio web de noticias. Podríamos hacerlo así:

```

stmt = EsperUtil.getStatement(
    "select company " +
    "from CompanyNewsEvent");

stmt.addListener(new CompanyNewsListener());

```

Fig. 11. Fragmento que define una sentencia EQL y añade un listener para su tratamiento

Y tratar las notificaciones de la siguiente forma:

```

package com.novayre.esper.jsweb.cep.listener;

import com.espertech.esper.client.UpdateListener;

public class CompanyNewsListener implements UpdateListener {

    @Override
    public void update(EventBean[] arg0, EventBean[] arg1) {

        EventBean event = arg0[0];

        String company = (String)event.get("company");

        System.out.println("NEWS!!!! Company = " + company);

    }

}

```

Fig. 12. Clase que recibe las notificaciones de los eventos asociados a sentencias EQL

Ahora podemos recibir las notificaciones de fluctuaciones acusadas tras alguna noticia financiera con el siguiente código:

```

stmt = EsperUtil.getPattern(
    "every cne=CompanyNewsEvent -> " +
    "FilteredMarketValueEvent (company=cne.company) " +
    "where timer:within(5 sec)");

stmt.addListener(new CrashListener());

```

Fig. 13. Fragmento para definir un patrón EQL

En la figura anterior utilizamos el método estático `getPattern(String)`.

Este método es similar al ya visto `getStatement(String)` con la diferencia que **genera un patrón semántico** en lugar de un flujo. El patrón es el siguiente: “Generar un nuevo evento cada vez que se genere una noticia y en los siguientes 5 segundos se genere una fluctuación”

Para procesar las notificaciones del evento asociado al patrón utilizamos un nuevo listener:

```

package com.novayre.esper.jsweb.cep.listener;

import com.espertech.esper.client.UpdateListener;

public class CrashListener implements UpdateListener {

    @Override
    public void update(EventBean[] arg0, EventBean[] arg1) {

        EventBean event = arg0[0];

        CompanyNewsEvent cne = (CompanyNewsEvent)event.get("cne");

        System.out.println("ESP/CEP NOTIFY about " + cne.getCompany() +
            " *****");

    }
}

```

Fig. 14. Clase que recibe las notificaciones de los eventos asociados al patrón EQL definido

En la aplicación de ejemplo hemos simulado la recepción de cotizaciones bursátiles con intervalos de un segundo.

A continuación vemos el resultado de una ejecución de la aplicación:

```

NEWS!!!! Company = Novayre
MARKETVALUE: company=Novayre, value=77.11390971979608,
moment=10-07-2008 18:08:00
MARKETVALUE: company=Novayre, value=75.21365862759095,
moment=10-07-2008 18:08:01
MARKETVALUE: company=Novayre, value=78.25906608187235,
moment=10-07-2008 18:08:02
MARKETVALUE: company=Novayre, value=74.29336083407968,
moment=10-07-2008 18:08:03
MARKETVALUE: company=Novayre, value=77.94812518731693,
moment=10-07-2008 18:08:04
DANGER Company = Novayre, stddev(value) =
1.7371429929268212
MARKETVALUE: company=Novayre, value=78.06954121021784,
moment=10-07-2008 18:08:05
MARKETVALUE: company=Novayre, value=77.49799158888048,
moment=10-07-2008 18:08:06
MARKETVALUE: company=Novayre, value=77.15722030951656,
moment=10-07-2008 18:08:07
MARKETVALUE: company=Novayre, value=78.03564497449142,
moment=10-07-2008 18:08:08
NEWS!!!! Company = Novayre
MARKETVALUE: company=Novayre, value=80.42591608768689,
moment=10-07-2008 18:08:09
DANGER Company = Novayre, stddev(value) =
1.2817953422598312
ESP/CEP NOTIFY about Novayre
*****

```

Fig. 15. Traza en la ejecución de la solución

En la siguiente figura se muestra la secuencia de eventos en función de la corriente a la que pertenece:

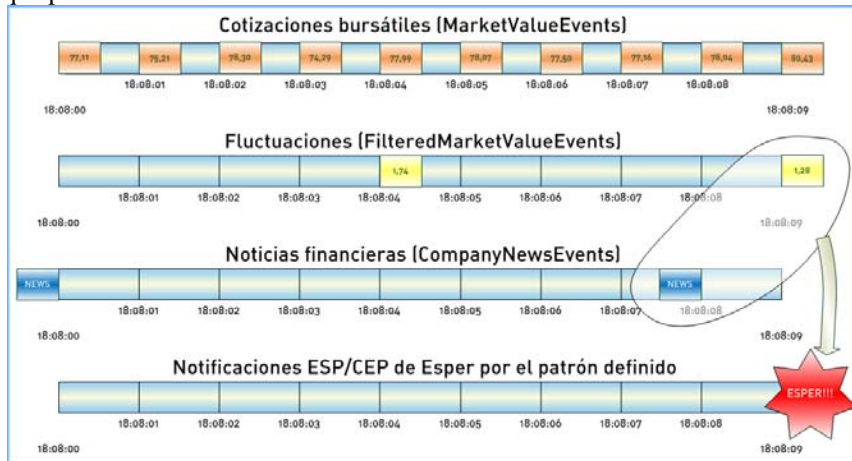


Fig. 16. Secuencia de eventos y patrones producidos en Esper

En la figura anterior vemos que la primera noticia no genera una notificación del patrón definido al producirse la fluctuación fuera de la ventana temporal. Sin embargo la siguiente noticia, al ir acompañada de una fluctuación en el segundo posterior, genera una notificación de que se ha cumplido el patrón.

6 Conclusiones

Hemos revisado los conceptos básicos de ESP/CEP y construido una pequeña solución que hace uso de la plataforma Esper. Como hemos, visto el patrón que permite integrar un motor ESP/CEP en una arquitectura SOA es el siguiente (figura 17).

- En primer lugar, el motor debe recibir los distintos tipos de eventos de negocio publicados por los servicios
- Dichos eventos deben ser procesados y correlacionados en el tiempo con el objetivo de detectar patrones complejos
- Una vez detectado el patrón, el motor debe generar alarmas que sea recibidas por distintos componentes ('listeners') para su procesamiento y presentación al usuario (por ejemplo en un 'cuadro de mando')

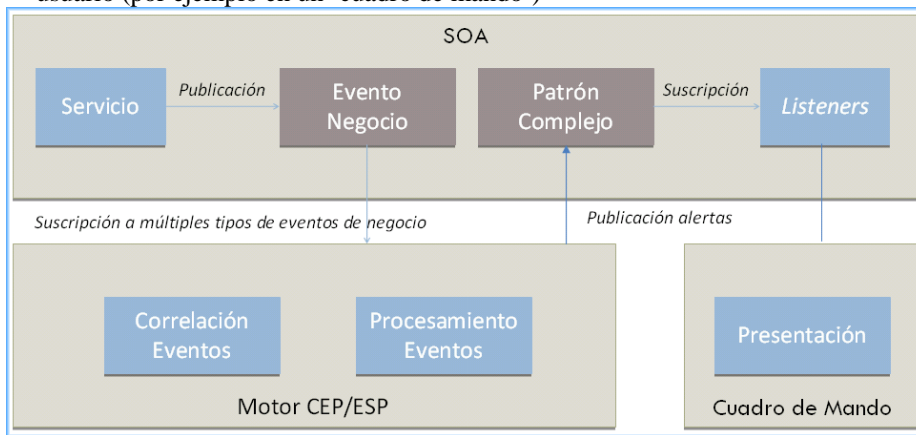


Fig. 17. Integración de Motor CEP/ESP en una arquitectura SOA

No obstante, la implantación de sistemas ESP/CEP deben formar parte del último escalón de los distintos niveles de madurez a la hora de adoptar arquitecturas SOA [7]

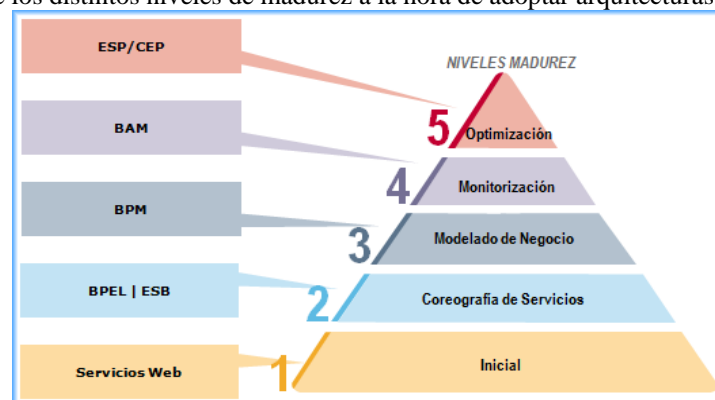


Fig. 18. Niveles de Madurez SOA

Podemos decir que tanto el nivel 4 (BAM) como el nivel 5 (ESP/CEP) se corresponden con sistemas que correlacionan eventos. La diferencia radica en que cuando el sistema encuentra dichas coincidencias, BAM lo notifica a un responsable de tomar decisiones para que actúe en consecuencia mientras que ESP/CEP puede orquestar una respuesta automática sin necesidad de intervención humana.

El integrar un modelo de procesamiento ESP/CEP en una arquitectura SOA, nos debe permitir decidir que eventos deben ser filtrados desde una perspectiva de monitorización (BAM) y cuales pueden ser correlacionados en el tiempo o agregados en eventos de negocio de alto nivel, es decir, de mayor contenido semántico.

Como hemos señalado en el comienzo del artículo, la capacidad de procesamiento y la latencia son dos requisitos importantes que hay que exigirle a una plataforma ESP/CEP. Las pruebas de rendimiento publicadas por los propios creadores de Esper [8], señalan la capacidad de la solución para procesar más de 500.000 eventos por segundo con una latencia menor a 3 microsegundos.

Entre las aplicaciones reales de ESP/CEP en una arquitectura SOA, la primera sería el dotar de una mayor capacidad de procesamiento e “inteligencia” humana a las herramientas de monitorización de actividad, generando respuestas automáticas en función del contexto. Sin duda, otra de las aplicaciones de CEP/ESP, que ya está viendo la luz, es la construcción de aplicaciones en tiempo real para el mercado financiero [9], la utilización para sistemas que detectan violaciones de seguridad [10] e incluso la implantación en soluciones RFID [11].

Referencias

- [1] BAM: <http://www.ebizq.net/topics/bam/features/4689.html>
- [2] Proyecto Esper: <http://esper.codehaus.org/>
- [3] Diferencias entre CEP y ESP: <http://complexevents.com/?p=103>
- [4] RAPIDE EPL : <http://www.informit.com/articles/article.aspx?p=29270&seqNum=2>
- [5] StreamSQL : <http://blogs.streamsql.org/>
- [6] Coral8: <http://www.coral8.com/>
- [7] Niveles de Madurez: <http://www.omg.org/>
- [8] Rendimiento Esper: <http://docs.codehaus.org/display/ESPER/Esper+performance>
- [9] Finanzas y CEP: <http://complexevents.com/?p=412>
- [10] Seguridad y CEP: <http://www.thecepblog.com/2007/07/10/extrusion-detection-is-ripe-for-cep/>
- [11] RFID y CEP: <http://www.rfidjournal.com/article/view/1196>