PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

MONITOREO DE DATOS MEDIANTE UN ADMINISTRADOR DE FLUJO DE DATOS.

FELIPE ANDRES LUCO PACHECO
BRIAN VALENZUELA RAMOS

INFORME FINAL DE PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO DE EJECUCIÓN EN INFORMÁTICA

ABRIL, 2013

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

MONITOREO DE DATOS MEDIANTE UN ADMINISTRADOR DE FLUJO DE DATOS.

FELIPE ANDRES LUCO PACHECO
BRIAN VALENZUELA RAMOS

Wenceslao Palma Muñoz
Profesor Guía

Rodolfo Villarroel Acevedo
Profesor Co-referente

ABRIL, 2013

Dedicatoria

A mi familia, profesores y amigos por creer en mi y apoyarme a lo largo de este proceso educativo

Felipe Andres Luco Pacheco.

Dedicatoria

A mi familia

Brian Valenzuela Ramos.

Agradecimientos

A mi Madre, por entregarme una gran educación y una excelente formación valórica y a mis hermanos Fernanda y Joaquin.

A mis profesores, por entregarme las herramientas necesarias para poder enfrentar este desafío, que a lo largo de este camino me brindaron apoyo y conocimiento.

Y finalmente a mis amigos, por creer en mi y darme su apoyo en todo momento.

Felipe Andres Luco Pacheco.

Agradecimientos

A Carlita por su constante apoyo e infinita paciencia.
A mi Padre por inculcarme el deseo de aprender.
A mi Madre por su inmensurable amor.
A mis Hermanos por su apoyo y consejos para superarme cada día.

Brian Valenzuela Ramos.

Resumen

Las aplicaciones actuales que trabajan con grandes flujos de información, tales como sensores de red, tráfico vehicular, monitoreo de transacciones financieras, entre otras, requieren procesar flujos continuos de información; función es realizada por los Sistemas administradores de flujos de datos. En el proyecto Monitoreo de flujos de datos, se analizan, identifican y describen los fundamentos teóricos de los flujos de datos, como también el procesamiento de las consultas sobre los flujos. El objetivo principal es implementar un sistema de consultas sobre un flujo de datos, para lo cual se utilizará Esper, un sistema administrador de flujos de datos open source.

Palabras Clave: Flujo de información, Procesamiento de Consultas, Esper.

Abstract

Current applications that work with large flows of information, such as sensor networks, vehicular traffic, monitoring financial transactions, among others, require continuous flows of information processing. This function is performed by the data streams systems administrators. The project "monitoring data streams", analyzes, identifies and describes the theoretical foundations of data flows, as well as the processing of queries about streams. The main objective is to implement a query system about a data stream, for which we will use Esper , an open source data stream administrator system.

Palabras Clave: Flows of information, Query system, Esper.

Índice

Lista de Figuras

Lista de Tablas

1. Introducción

En las últimas décadas, la utilización de la tecnología ha ido en ascenso, lo cual provoca aumento de confianza e interés por parte de las empresas y personas en disponer de la información privada y/o trivial, junto con las dependencias de conectividad a disposición de aplicaciones informáticas. Viajar en avión, ejecutar transacciones bancarias, entre otras, [16] son actividades realizadas por sistemas computaciones complejos, las cuales contienen riesgos permanentes[15] que son minimizadas con monitoreo y análisis constantes por dichos sistemas. Sin duda tanta confianza y dependencia crea una actividad constante y que crece en el tiempo. Este crecimiento trae consigo que las aplicaciones se vean obligadas a procesar grandes volúmenes de flujos de datos, que en general son conjuntos de datos que se van produciendo gradualmente en función del tiempo.

Las aplicaciones que procesan grandes volúmenes de datos, tales como: análisis de tráfico en Internet, servidores web, transacciones financieras y comerciales en línea, entre otras, [10] requieren necesariamente del análisis de los flujos de datos que circulan por el sistema. Lo mencionado anteriormente, ha ocasionado que los recientes trabajos en los campos de procesamiento de flujos de datos adquieran gran importancia con respecto al monitoreo de estos[9], en consecuencia se inició el estudio de los sistemas administradores de flujos de datos, con los cuales se realizan monitoreos o analisis de datos mediante consultas continuas en el tiempo, a la información que se está transmitiendo constantemente. En contraposición con los tradicionales DBMS que trabajan con datos estáticos[14], los cuales pueden ser consultados, modificados y borrados.

Este proyecto tiene por objetivo estudiar la infraestructura de software necesaria para la implementación de sistemas de consultas sobre flujos de datos (DSMS). Los DSMS son aplicaciones que controlan el mantenimiento y consultas de datos en flujos de datos, caracterizados por la capacidad de realizar consultas continuas sobre un flujo de datos[17]. El uso de un DSMS para manejar un flujo de datos es aproximadamente análogo a la utilización de un sistema de gestión de base de datos (DBMS) para administrar una base de datos convencional. En las BD tradicionales, los datos son consultados una sola vez y se obtiene el resultado de la consulta, no así en los DSMS, los cuales mantienen consultas continuas que se van ejecutando a través del tiempo a todos los flujos entrantes, lo cual generan resultados que se van actualizando como nuevos datos obtenidos. El enfoque de los DSMS se basa principalmente en poder realizar un monitoreo de los flujos de datos que transitan en diversas aplicaciones[25].

La estructuración del proyecto consta de la formalización de los fundamentos teóricos de los DSMS, abarcando sus operadores, procesamiento de consultas y lenguaje de consulta. Además, se realizará una definición formal de los flujos de datos con sus principios, con el propósito de aprender a cabalidad la arquitectura de los sistemas administradores de flujos de datos, y de esta manera implementar un prototipo basado en los flujos generados en la bolsa de valores.

La estructura del informe está basada en las principales definiciones y fundamentos de los sistemas administradores de flujos de datos. Se encuentra organizado en 10 secciones, con capítulos en cada una de ellas:

En la sección 1 se introduce a la temática del proyecto, situando un contexto de aplicaciones y presentando las principales nociones de los DSMS. Luego la Sección 2 presenta el objetivo general y los específicos que se desean lograr con este trabajo. la sección 3 son descritos en profundidad los sistemas administradores de flujos de datos.En la sección 4, se describe el sistema de gestión de flujos IBM Infosphere Streams en 4 capítulos.

La sección 5, aborda el DSMS Esper en 14 capítulos en los cuales se define el motor que ocupa esper, su lenguaje, la configuracion del motor, entre otros. La Sección 6, describe un caso de estudios para ejemplificar el funcionamiento de la arquitectura de Esper. Y finalmente la Sección 7 donde se desarrolla la conclusión del trabajo realizado.

2. Definición de Objetivos

A continuación se presentan los objetivos tanto general como específicos.

2.1. Objetivo General

Implementar un sistemas de consultas sobre un flujos de datos.

2.2. Objetivos Específicos

- Analizar los fundamentos teóricos de los DSMS tales como: operadores, procesamiento de consultas y lenguaje de consulta.
- Identificar los principales DSMS disponibles.
- Probar los diferentes sistemas disponibles.
- Describir los fundamentos teóricos de Esper (Motor de Esper, procesamiento de consultas y lenguaje de consulta)
- Implementar consultas basadas en los flujos generados en la bolsa de valores.

3. Sistemas Administradores de flujos de Datos

El incremento de aplicaciones que necesitan analizar constantemente un gran flujo de datos, generan la necesidad de crear sistemas exclusivos para dicha función. Motivo por el cual, nacen los sistemas administradores de flujos de datos, cuyos sistemas permiten realizar consultas continuas sobre un flujo especifico de datos, mediante la aplicación de ventanas deslizantes en las consultas, rangos de tiempo, entre otros. A continuación se detallarán la Arquitectura de los DSMS's, las principales diferencias con los sistemas administradores de bases de datos tradicionales, el lenguaje que utilizan estos sistemas y los sistemas existentes actualmente.

3.1. Arquitectura

El modelo que se muestra en la figura 1 obtenido de [22], es el típico esquema de un Administrador de Flujo de Datos (DSMS), el cual se representa como un conjunto de consultas recurrentes y esta compuesto por uno o mas flujos de entrada de datos, cuatro salidas posibles y las consultas Q.

- **The Stream:** esta formada por todos los elementos de la respuesta que se producen una vez y nunca ha cambiado
- **The Store:** esta compuesto por las partes de la respuesta que pueden cambiar o ser removidas en el futuro. The Stream y the store hacen la respuesta completa de las consultas Q.
- **The Scratch:** representa la memoria de trabajo del sistema, es decir, un repositorio donde es posible almacenar los datos que no son parte de la respuesta, pero que pueden ser utiles para calcular la respuesta.
- **The Throw:** es una especia de papelera de reciclaje, que sirve para tirar las tuplas que no sean necesarias.

El modelo descrito en la figura 1 es el mas completo para definir el comportamiento de un DSMS. Muestra explícitamente como los DSMS pueden cubrir por completo las necesidades de los Procesamientos de los flujos de Información [22] (IFP), los IFP o flujos de información son el motor de una herramienta capaz de procesar un flujo de gran cantidad de información a medida que fluye desde un periférico hacia el sistema, siendo una extensión de un sistema de base de datos. Los DSMS se enfocan en la producción de respuestas a las consultas, que son continuamente actualizadas para adaptarse al constante cambio de los contenidos de los datos entrantes.

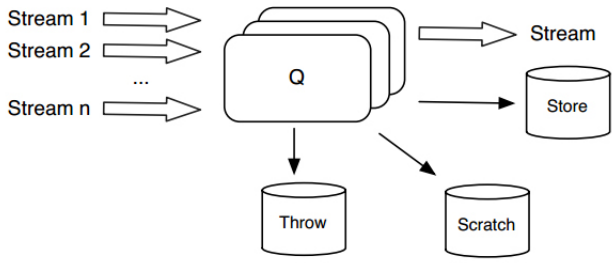


Figura 1: Arquitectura típica de un DSMS.
[1]

[1] Imagen obtenida de [22]

3.2. Principales diferencias respecto de los DBMS's

- Una DBMS tradicional asume un modelo pasivo, donde la mayoría de los resultados de los datos procesados, son a partir de transacciones o consultas realizadas por el sistema.

- La gestión de flujo de datos requiere un mayor enfoque activo, ya que los datos de seguimiento se alimentan de impredecibles fuentes externas (por ejemplo, sensores) y alertan cuando una actividad anormal en los datos es detectada.

- Las DBMS tradicionales gestionan los datos que existan en sus tablas. El manejo de flujo de datos a menudo requiere un procesamiento de datos que son delimitados por algún grupo finito de valores.

- Un DBMS tradicional ofrece respuestas exactas a consultas exactas, y es ciego en tiempo real a los plazos. La gestión de flujo de datos responde a los plazos en tiempo real (por ejemplo, aplicaciones militares monitoreando la posición de plataformas enemigas) y por lo tanto deben proporcionar una razonable aproximación del valor de los datos generados a las consultas con los datos reales.

- Un procesador de consultas tradicionales optimiza todas las consultas de la misma manera (típicamente centrado en el tiempo de respuesta). Un Administrador de flujo de datos beneficia de los criterios específicos de optimización de una aplicación (QoS).

3.2.1. Diferencias en la Arquitectura

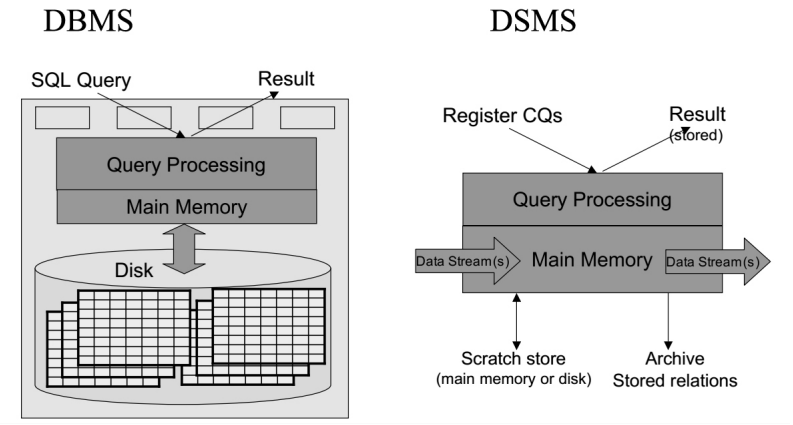


Figura 2: Diferencias entre la Arquitectura de un DBMS y DSMS.
[2]

[2]Imagen obtenida de [21]

En la figura 2 se pueden apreciar las diferencias existentes entre las arquitecturas. En primer lugar, mencionar que en los DBMS los datos son almacenados en discos, luego pasan a la memoria principal, donde los datos son procesados con la o las diferentes consultas que se desean generar sobre los datos, arrojando un resultado. Por el contrario, los DSMS administran flujo de datos que se almacena en la memoria principal, donde son procesadas con todas las consultas continuas(CQL) para luego generar una respuesta y almacenarla. Además, se guardan datos que no pertenecen a la respuesta, pero que son necesarios para poder generarla.

3.2.2. Comparación DBMS con DSMS's

Los resultados se muestran en la tabla 1.

	DBMS	**DSMS**
Datos	Relaciones persistentes	Flujos, ventanas de tiempo
Acceso a los datos	Aleatorio	Secuencial
Actualización	Arbitrarias	Solo para agregar
Velocidad Actualización	Relativamente lenta	Altas
Modelo de Proceso	Basado en las consultas	Basado en los datos
Consultas	Un tiempo	Continuas
Plan de consultas	fijo	adaptativo
Optimización de Consultas	Una consulta	Consultas múltiples
Respuesta de Consultas	Exacta	Exacta o aproximada
Latencia	Relativamente alta	Baja

Tabla 1: Comparación

3.3. CQL: Lenguaje de Consultas Continuas

Para el análisis de los flujos de datos es necesario utilizar un lenguaje de consultas recurrentes, en el cual las consultas que están almacenadas en el sistema, se vayan actualizando según el comportamiento del flujo. El lenguaje utilizado es CQL (Continuous Query Language), el cual es una extensión del conocido lenguaje de consultas SQL, en donde se deben reemplazar las referencias a las relaciones por referencias a los flujos de datos.

3.4. Procesamiento de las Consultas

El procesamiento comienza cuando una consulta continua se registra en un DSMS, lo cual ejecuta un plan de consulta que es compilado por el propio administrador. Los planes de consultas se componen por los operadores, las colas y la sinopsis[4], los cuales se describirán a continuación:

Operadores

Los operadores son los encargados de realizar el procesamiento real de la consulta, donde existen dos tipos de datos fundamentales; los flujos y las relaciones, términos que serán definidos en detalle en el capítulo 4.7. Estos dos tipos son unificados en la aplicación como secuencias de registros de tiempo en tuplas, donde cada tupla está marcada, como una inserción (+) o una eliminación (-)[4].

Cabe mencionar que los flujos sólo incluyen elementos de (+), mientras que las relaciones pueden incluir tanto elementos de (+) y (-) para capturar el estado de cambios en las relaciones a través del tiempo. Los operadores serán definidos y descritos con mayor precisión en el capitulo 4.7

3.4.1. Colas

Las colas en un plan de consulta contienen colecciones de elementos que representan una porción de un flujo o una relación. Estos elementos son producidos o consumidos por un productor y consumidor, respectivamente. A la vez el plan de consulta se conecta a su operador y consumidor, donde el productor inserta los elementos que produce al buffer de la cola hasta que sean procesadas por el consumidor[4].

Los operadores del sistema requieren que los elementos en las colas de entrada se lean en orden no decreciente, según su marca de tiempo. Por ejemplo, consideremos un operador de ventana Ov en un flujo de F. Por un lado, si recibe un elemento Ov (f, t,+) y en su cola de entrada se garantiza que no llegue en orden no decreciente, el operador Ov sabrá que ha recibido

todos los elementos con marca de tiempo t' <t, y se puede construir el estado de la ventana en el momento t -1. Si por otro lado, Ov no tiene esta garantía, nunca podrá estar seguro de tener la suficiente información para construir cualquier ventana correctamente. Por lo tanto, para cumplir este criterio con todas las colas se requiere de un mecanismo de tuplas por buffering que generen marcas de tiempo para asegurarse de no ser no decreciente, sin sacrificar la exactitud e integridad[7].

3.4.2. Sinopsis

En general la sinopsis pertenece a un operador de plan específico, donde se guardara el estado del operador, puesto que puede ser necesario en una evaluación futura del operador. Su principal utilización es para materializar el estado actual de una relación, tales como el contenido de una ventana deslizante o la relación producida por una sub-consulta. La sinopsis también se puede utilizar para almacenar un resumen de las tuplas en un flujo o relación de respuestas de consultas aproximadas. Cabe destacar que una sinopsis al igual que las colas, se deben mantener en la memoria, para cumplir con los requisitos de un buen desempeño[4].

3.4.3. Ejemplo de plan de consulta

Un plan se construye cuando se registran consultas CQL, en el sistema administrador de flujos de dato, el cual contiene sus respectivos operadores, unidos por colas, con sinopsis adjuntas a los operadores, según sea necesario[4]. Como ejemplo, se considera un plan para la siguiente consulta:

Select * from S1 [Rows 1000], S2 [Range 2 Minutes] Where S1.A = S2.A and S1.a >10

En el plan graficado, en la figura 3, se encuentran cuatro operadores: un select, join binario y una instancia de seq-windows para cada entrada del flujo. De la misma forma, se establecen las colas q1 y q2, las cuales mantienen los elementos de flujo de entrada que podrían, por ejemplo, ser recibidas por la red y colocadas en las colas por un operador del sistema (no representada). La cola q3, que es la cola de salida del operador seq-window, contiene elementos que representan la relación «S1 [Rows 100].» La cola q4 tiene los elementos para «S2 [Range 2 minutes]». La cola q5 tiene los elementos producto de la operación join de la relación «S1 [rows 1000] con S2 [Range 2 Minutes]», y a partir de estos elementos, la cola q6 contiene los elementos que pasan el operador de selección. Q6 puede dar lugar a un

operador de salida el envío de elementos a la aplicación, o para otro operador de plan de consulta dentro del sistema.

El plan contiene cuatro sinopsis, donde cada operador seq-window contiene una sinopsis, de modo que pueda generar elementos cuando expiran las tuplas de la ventana deslizante. El operador binary-join mantiene una sinopsis, materializando cada una de sus entradas relacionadas para unirlas con las tuplas de la entrada opuesta , como se describió anteriormente. Finalmente el operador Select no necesita mantener un estado, por lo cual no tiene una sinopsis.

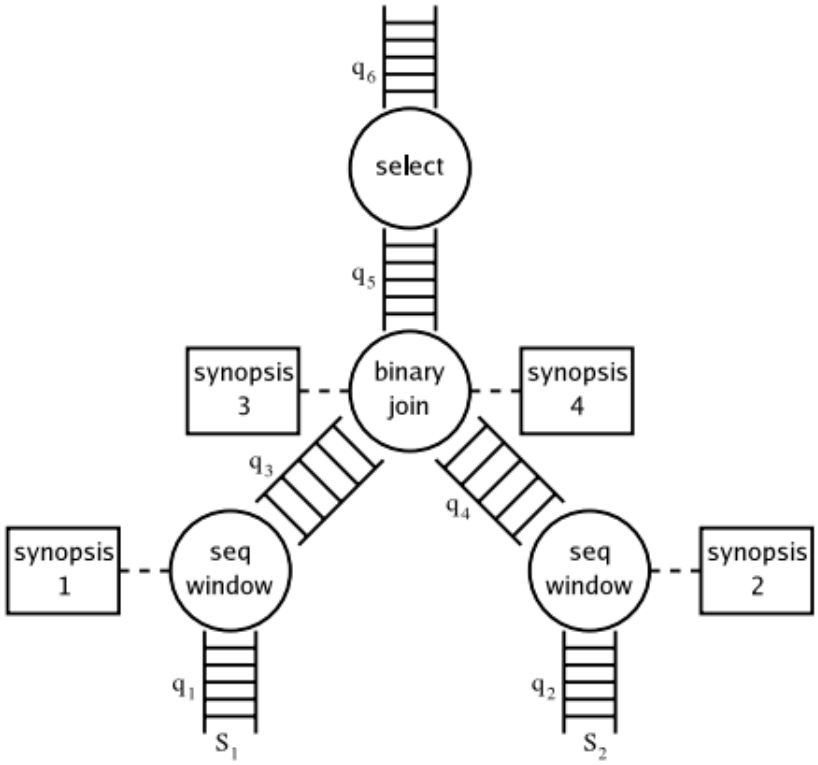


Figura 3: Ejemplo plan de consultas.
[3]

[3]Imagen obtenida de [4]

3.5. Operadores

Existen dos tipos de datos fundamentales en el lenguaje CQL, las cuales son los flujos de datos (streams) y las relaciones.

- **Streams:** Un flujo S es una bolsa de elementos (s,t), donde s es un tupla que pertenece al esquema de S y t es una marca de tiempo del elemento.
- **Relaciones:** Una relación R es un mapeo de cada instante de tiempo en T a una bolsa finita de tuplas que pertenecen al esquema de R.

A partir de estos dos tipos de datos, se definen tres clases de operadores, como se muestran en la figura 4, los cuales son Relación - Relación, Stream - Relación y Relación - Stream.

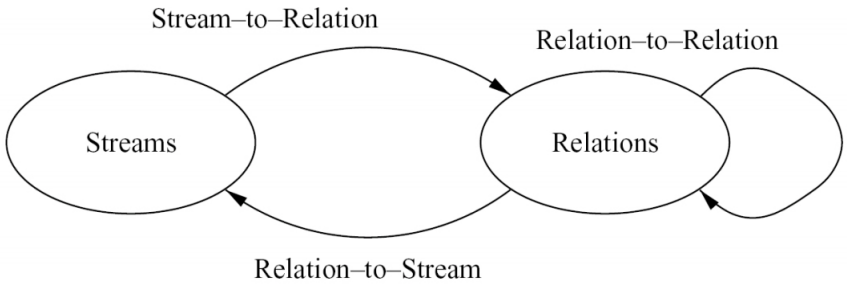


Figura 4: Tipos de datos y Operadores .
[4]

[4]Imágen obtenida de [4]

3.5.1. Operador Relación a Relación

El operador relación a relación en CQL es un derivado de las tradicionales consultas relacionales en SQL que definen consultas sobre tablas de base de datos.

3.5.2. Operador Stream a Relación

El operador stream a relación está basado en el concepto de ventana deslizante sobre un flujo y son expresadas utilizando una especificación de la ventana, a continución se explicaran algunos tipos de ventana:

- **Constant value:** Es un rango de tiempo constante C sobre un identificador (ID).

S[Range C on ID].

- **Time-based:** La ventana deslizante en un flujo S toma un intervalo de tiempo w como un parámetro y se produce una relación R. En un tiempo t, R(t) contiene toda las tuplas de S con una marca de tiempo entre [t-w, t]. Como caso especial [Now] denota a la ventana como w=0, también existe el caso en donde la marca de tiempo es ilimitada (unbounded).

S[Range t].

S[Range t1 Slide t2].

S[Now].

S[Range Unbounded].

- **Tuple-based:** La ventana deslizante en un flujo S toma un entero N>0 como un parámetro y produce la relación R. En un tiempo t, donde R(t) contiene las N tuplas de S con una marca de tiempo $\leq t$.

S[Rows N].

S[Rows N1 Slide N2].

- **Partitioned:** La ventana deslizante en un flujo S toma un entero N y un conjunto de atributos {A1,.,.,.,Ak} de S como parámetros, y se especifica siguiendo S con [Partition By A1,.,.,., Ak Rows N] lógicamente particiones de S en diferentes sub-flujos basados en la igualdad de los atributos A1,.,.,.,Ak , calcula un tupla basada en ventana deslizante de tamaño N independiente en cada sub-flujo, luego toma la unión de estas ventanas y produce la relación de salida.

S[Partition By A1 ... Ak Rows N].

S[Partition By A1 ... Ak Rows N Range T].

S[Partition By A1 ... Ak Rows N Range T1 Slide T2].

3.5.3. Operador Relación a Stream

Cql posee 3 operadores de relación - stream:

- **IStream:** Se utiliza para insertar un flujo. Este operador es usado comúnmente con una ventana infinita (Unbounded) para expresar una condición de filtrado. Aplicado a una relación R que contiene <s,t>cuando un tupla s esta entre R(t) - R(t - 1). Por ejemplo cuando s es insertado en R en un tiempo t. La siguiente consulta continua es un filtro a un flujo S:

  Select Istream(*) From S [Rows Unbounded] Where S.A >10

  El flujo S es convertido en una relación aplicando una ventana infinita. El filtro relación a relación «S.A >10» actúa sobre esta relación y las inserciones a está son transmitidas como el resultado de la consulta. Gracias a las sintaxis que incluye CQL la consulta también puede ser reescrita de la siguiente forma:

  Select * From S Where S.A >10

- **DStream:** Para borrar un flujo.Este operador es usado comúnmente con una ventana Now para expresar una condición de filtrado. Aplicado a una relación R que contiene <s,t>cuando un tupla s esta entre R(t-1) - R(t). Por ejemplo cuando s es eliminado de R en un tiempo t. La siguiente consulta continua es un join de ventada de dos flujos S1 y S2:

  Select * From S1 [Rows 1000], S2 [Range 2 Minutes] Where S1.A = S2.A And S1.A >10

  La respuesta a esta consulta es una relación. En cualquier tiempo dado, la relación de respuesta contiene el join (sobre un atributo A con A>10) de las ultimas 1000 tuplas del flujo S1 con las tuplas de S2 que han llegado en los 2 minutos previos. Si se prefiere producir un flujo que contenga los nuevos valores de A y estos aparezcan en el join, se puede escribir IStream (S1.A)en lugar del «*» en la cláusula del Select, ya que el operador IStream se utiliza para insertar un flujo nuevo.

- **RStream:** Para relación de flujos, en una relación R que contiene <s,t>donde s esta en R en un tiempo t. Por ejemplo cuando cada tupla actual de R se transmite a cada instante de tiempo t. La siguiente consulta continua es una tabla almacenada R basada en cada tupla del flujo S:

  Select Rstream(S.A, R.B) From S [Now], R Where S.A = R.A

3.5.4. Operadores utilizados en STREAM

En la tabla 2, se pueden apreciar los diferentes operadores que se utilizan en el DSMS's STREAM creado en la universidad de Stanford:

Nombre	Tipo de Operador	Descripción
select	Relación a relación	Filtra elementos basados en el predicado
project	Relación a relación	Duplica-preserva una proyección
binary-join	Relación a relación	Join de dos relaciones de entrada
mjoin	Relación a relación	Joinn multipunto
union	Relación a relación	Bolsa de uniones
except	Relación a relación	Bolsa de diferencias
intersect	Relación a relación	Bolsa de intersecciones
antisemijoin	Relación a relación	antisemijoin de dos relaciones de entrada
aggregate	Relación a relación	Realiza agrupación y agregación
duplicate-eliminate	Relación a relación	Realiza la eliminación de duplicados
seq-window	Stream a relación	Implementa un tipo de ventana
I-Stream	Relación a Stream	Implementa la semánticas I-Stream
D-Stream	Relación a Stream	Implementa la semánticas D-Stream
R-Stream	Relación a Stream	Implementa la semánticas R-Stream

Tabla 2: Operadores usados en STREAM

3.6. Esquemas de consultas de STREAM

Como se ha mencionado anteriormente, el procesamientos de flujos de datos es una actividad habitual en aplicaciones, tales como el monitoreo de red, gestión de los datos de telecomunicaciones, redes de sensores, entre otros. Para los distintos campos y aplicaciones se requiere capturar los datos y especificar los esquemas, las consultas y las expresiones de las consultas, en diferentes lenguajes de consultas. Por lo cual, se presentarán las características de los esquemas y las consultas necesarias para los siguientes campos y/o aplicaciones.

3.6.1. Sistemas de subastas en línea

En los sistemas de subastas en línea, como eBay y Mercado Libre, constantemente se están efectuando distintos flujos de operaciones con diferentes datos cada uno. Por lo cual, en estos sistemas, cientos de subastas, para los elementos individuales, están abiertas en un momento dado. Nuevas personas son continuamente registradas en el sistema, los nuevos artículos están siendo sometidas a subastas constantemente, y las ofertan están permanentemente llegando a distintos artículos.

3.6.1.1. Esquema

El esquema consta de tres relaciones: categoría, artículo y persona. A la vez, contiene tres flujos: OpenAuction, closeAuction y bid.

Categoría: La información que contiene sobre varias categorías(Ej: Libros, juguetes, etc) de los artículos subastados.

Categoria(id, nombre, descripción);

Artículo: La información que contiene sobre cada artículo que se subasta.

Articulo(id, nombre, descripción, categoriaId, registroTiempo);

Persona: La información que contiene la relación de los usuarios registrados que tinen permiso para iniciar y hacer oferta a las subastas.

Persona(id, nombre, email, ciudad, comuna, registroTiempo);

OpenAuction: flujo de las aberturas de las subastas

OpenAuction(idArticulo, vendedor, precioInicio, fechaYhora);

ClosedAuction: Flujo de los cierres de la subasta.

ClosedAuction(idArticulo, comprador, fechaYhora);

Oferta: Flujos de las ofertas para los artículos. Un idArticulo, por flujo, sólo aparece después de la tupla Openauction y antes de la closedAuction.

Oferta(idArticulo, precio, Postor, fechaYhora);

3.6.1.2. Consultas

Algunas de las consultas que se pueden emplear, en este caso de estudio, en CQL son:

Consulta de selección: Selecciona todas las ofertas, en un conjunto específico de 5 artículos.

Select idArticulo, precio
From Oferta
Where idArticulo = 1007 or idArticulo = 1020 or
idArticulo 2001 or idArticulo 2019 or idArticulo =1087

Consulta la subasta más alta: Cada 10 minutos, entregar la oferta más alta realizadas en dicho periodo de tiempo.

Select Rstream(idArticulo, precio)
From Oferta [Range 10 minute
Slide 10 Minute]
Where precio = (Select Max(precio)
From Oferta[Range 10 Minute
Slide 10 minute]

3.6.2. Red de gestión de tráfico

Las gestión de tráfico, consiste en el monitoreo de los paquetes de la red, especificamente en la información que entregan la cabecera de éstos y las mediciones de rendimiento de la red, a través de un conjunto de elementos, tales como routers y switches. A continuación, se proporciona el esquema y las consultas para este caso.

3.6.2.1. Esquema

El esquema consta de un solo flujo de paquetes de red.
Paquetes: flujo de trazas de paquetes de red.

Paquetes(srcIP, srcPort, destIP, destProt, len, flags, id, colID, timestamp);

srcIP = Host de origen dirección IP.
srcPort = Número de puerto en el host de origen.
destIP = Host de destino dirección IP.
destProt = Número de puerto en el host de destino.
len = La longitud del paquete.
flags = Bandera de mapa de bits que utiliza el protocolo TCP.
id = (Casi) Identificador único. Por ejemplo, el hash de los campos que permanecen estables durante la ruta del paquete.
colID = Identificador único del colector de seguimiento de paquetes.
timestamp = Tiempo de cuando el paquete se recogió.

3.6.2.2. Consultas

Algunas de las consultas que se pueden emplear en este caso de estudio en CQL son:

Consulta Top-k de tráfico de consultas: Controlar los pares origen-destino en el top 5, percentil en términos de tráfico total, en los últimos 20 minutos, en un enlace troncal B.

Load: Select srcIP, destIP, Sum(len) as traffic
From Packets [Range 20 minute]
Where colID = 'B' Group By srcIP, destIP
Q: Select srcIP, destIP, traffic
From Load as L1
Where (Select Count(*)
From Load as L2
Where L2.traffic <L1.traffic) >
(Select 0.95 * count(*)
From Load) Order by traffic

Consulta Protocolo de análisis: Para cada dirección IP de origen y cada intervalo de 5 minutos, cuente el número de bytes y el número de paquetes que resultan de las peticiones HTTP.

Select Rstream(srcIP, Sum(len), count(*))
From Packets [Range 5 Minute
Slide 5 Minute]
Where desPort = '80'
Group by srcIP

Esta consulta asume que todas las solicitudes HTTP son del puerto 80.

3.7. DSMS's existentes

Uno de los primeros administradores de este tipo de consultas continuas fue telegraphCQ[11], el cual fue diseñado el año 2000 por la Universidad de California, con el propósito de desarrollar una función adaptativa del flujo de datos, de tal manera que se pudiera soportar una gran cantidad de datos para aplicaciones en red, y así tratar con flujos ilimitados. Este sistema (telegraphCQ) utiliza el lenguaje basado en SQL, llamado Streaquel, el cual contiene a todos los operadores de relación de SQL, incluyendo los aggregates(por ejemplo la función count de sql). En suma, la principal contribución de telegraphCQ fue crear los operadores para el procesamiento de consultas adaptativas.

En el año 2002, nace Aurora[1] por las Universidades de Brown y Brandeis, que es un sistema administrador de flujo de datos desarrollado en Java, el cual tiene como propósito demostrar como la programación de un algoritmo influye en la latencia de las tuplas y el largo de las colas. En este mismo año, también es diseñado Gigascope[12], con el objetivo de incluir análisis de tráfico, detección de intrusos y monitoreo de rendimiento para aplicaciones de red. Este sistema define un lenguaje similar a SQL compuesto solo de filtros, joins, group by y aggregates llamado GSQL.

En año 2004, es desarrollado STREAM[4] por la universidad de Stanford, el cual tenía como finalidad investigar el manejo de los datos y el procesamiento de las consultas para ilimitados flujos continuos de datos. Además se creó el lenguaje de consultas (CQL), basado en SQL, el cual agrega tres operadores del tipo relación a flujo; IStream, DStream y RStream que permiten trabajar con las tuplas. El principal aporte de este proyecto es el nuevo lenguaje CQL[8].

En el año 2009, se desarrollan tres software comerciales capaces de procesar flujos de información. En primer lugar, StreamBase[27], el cual es una plataforma que incluye: un sistema para procesar flujos de datos, un conjunto de adaptadores para reunir información desde fuentes diferentes y una herramienta basada en Eclipse. Cabe destacar, que Streambase posee su propio lenguaje de consultas, llamado StreamSQL. En segundo lugar, Oracle CEP (complex event processing), el cual es un sistema que permite procesar flujos de información, en tiempo real. Este DSMS utiliza el lenguaje de consultas continuas CQL[24]. Tercero y último, mencionar que en este año es desarrollado Esper [13], el cual es considerado el CEP líder de código libre (open-source). Esper define un lenguaje declarativo de reglas llamado EPL, Lenguaje de Procesamiento de Eventos, el cual posee todos los operadores de SQL incluyendo constructores para la definición de ventanas deslizantes.

En el año 2010, es anunciado IBM InfoSphere Streams, desarrollado por IBM, el cual pretende ser una plataforma que posibilita el desarrollo y ejecución de aplicaciones que procesan la información de flujos de datos. Streams permite el análisis continuo de grandes volúmenes de flujos de datos, utilizando SPL, Lenguaje de procesamiento de flujos. En el próximo capítulo se dan a conocer las características de este sistema.

Los sistemas expuestos, TelegraphCQ, Aurora, STREAM, StremBase, Oracle CEP, Esper e IBM InfoSphere Streams, han sido la base para la recopilación de información necesaria para el desarrollo de este proyecto.

4. IBM InfoSphere Streams

Streams es un producto diseñado por IBM, liberado bajo licencia privada, específicamente para ayudar a las empresas a analizar volúmenes masivos de transmisión de datos a velocidades extremas, para mejorar la visión empresarial y la toma de decisiones.

InfoSphere Streams, trabaja en un entorno de desarrollo integrado basado en la aplicación de código abierto Eclipse proyect. De esta forma, los desarrolladores pueden especificar un operador para ser utilizado en el procesamiento del flujo, tal como filtros, funciones de agregación, matemáticas complejas, entre otras. Por lo tanto, InfoSphere es un producto de software, integrado con el sistema operativo para un alto rendimiento y es compatible con J2EE, servidores LDAP y servidores de bases de datos.

4.1. Conceptos y Términos

Una instancia de Streams es un completo y autónomo tiempo de ejecución de flujos. Se compone de un conjunto de servicios que interactúan a través de uno o varios Host. A continuación se definirán los términos anteriormente mencionados.

Flujos de Instancia: Sirven como un contenedor para las entidades físicas o host. Las instancias se inician y luego se detienen, cuyas entidades dentro de una instancia de Streams se modifican, compilan, presentan, desarrollan, y así sucesivamente. Desde un punto de vista administrativo, una instancia de flujo se crea y luego se puede eliminar. Cabe destacar, que la creación de una instancia es un procedimiento de costo relativamente bajo.

Host: Es un término físico, y casi por completo se compara con una serie única de sistema operativo. En ciertos contextos, un host también se llama nodo. Con el fin de que exista una instancia de flujos, se componen de uno o varios hosts.

Tipos de Host: Un host puede ser exclusivamente de los siguientes tipos: host administrativo, host de aplicación y host de uso mixto.

Host Administrativo: Ejecuta los servicios para los residentes que son el entorno de ejecución de secuencias adecuadas, como la autorización de flujos y servicios de autenticación, que entre otras responsabilidades, verifica la identidad del usuario y el nivel de permisos.

Host de aplicación: Se dedica a la ejecución de los flujos reales, es decir realizan la consulta sobre el conjunto de tuplas a analizar.

Host mixto: Se ejecuta tanto en los servicios de residentes, como a los flujos de aplicaciones, y son comunes en los entornos de desarrollo.

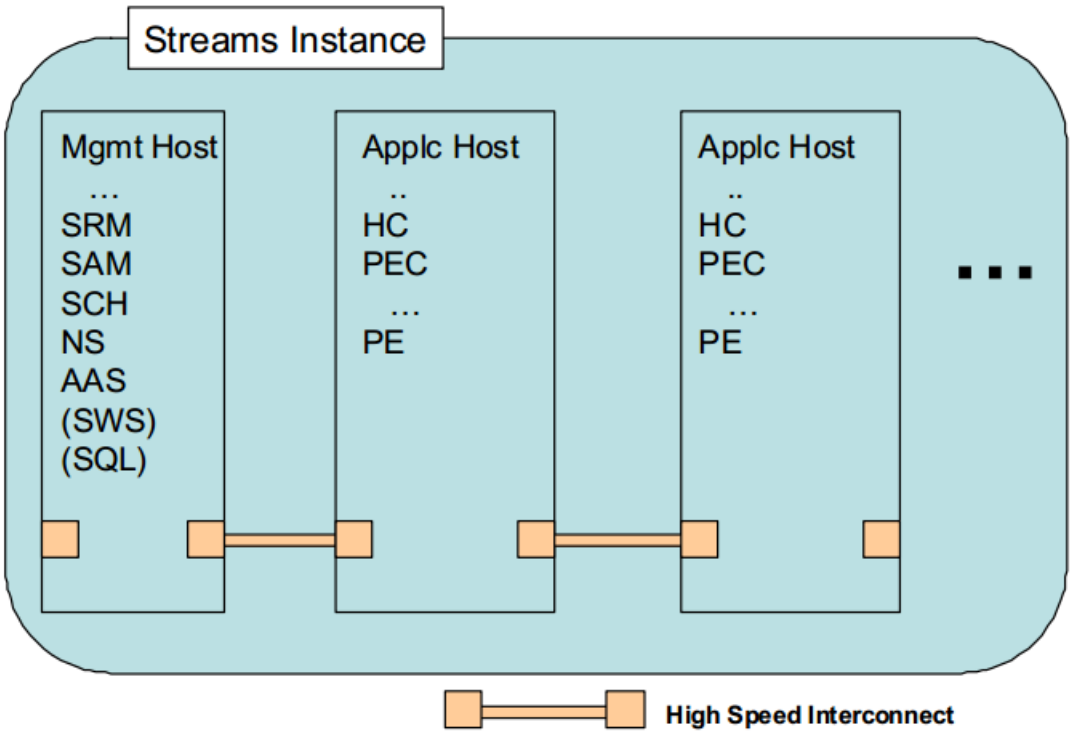


Figura 5: Flujos de instancias y host's.
[5]

[5]Imagen obtenida de [29]

4.2. Tiempo de ejecución de Secuencias

El tiempo de ejecución de secuencias se compone de una serie de procesos de servicios, que interactúan para gestionar y ejecutar las aplicaciones de Streams. Un conjunto completo de estos servicios actúan juntos y forman una instancia de Streams. Las instancias se pueden crear y configurar de forma independiente el uno del otro, a pesar de que pueden compartir algunas del mismo hardware físico. La figura 6 muestra una instancia de flujo, con sus servicios a los constituyentes.

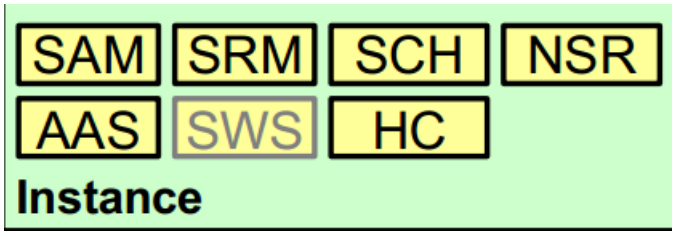


Figura 6: Servicios asociados con instancias de Streams.
[6]

[6]Imagen obtenida de [29]

Los servicios que componen una instancia son los siguientes:

Stream Application Manager (SAM): Es un servicio de gestión que administra las aplicaciones en el tiempo de ejecución de flujos. En concreto, el Sam se encarga de las tareas de gestión, tales como el envío de trabajos y la cancelación. También está relacionada con el planificador para calcular la ubicación de los elementos de proceso que comprenden una aplicación. También interactúa con el controlador de host para desplegar y cancelar la ejecución de estos elementos de procesos.

Streams Resource manager (SRM): Este es un servicio de gestión que inicializa la instancia de Streams, y controla todos sus servicios. El SRM acumula y agrupa las métricas de todo el sistema, incluidos los estados de los host que forman parte de una instancia y el estado de los componentes de flujos, así como las métricas de rendimiento pertinentes necesarias para la programación y la administración del sistema al interactuar con el controlador de host.

Scheduler (SCH): El programador es un servicio de gestión que calcula las decisiones de ubicación para las aplicaciones que se desplegarán en el sistema de ejecución de Streams. También está relacionada, principalmente, con el servicio de SAM, para atender las peticiones de trabajo de implementación. A la vez, interactúa con el componente SRM para obtener mediciones de tiempo de ejecución, necesarias para informáticos de procesamientos de las decisiones de colocación de elementos.

Name Service (NSR): Este es un servicio de gestión que almacena referencias de servicios para todos los componentes de la instancia. Es utilizado por todos los elementos que componen la instancia, para localizar su distribución en la misma para efectos de la comunicación entre componentes.

Authorization and Authentication Service (AAS): Es un servidor de administración que se ha autenticado y autorizado las operaciones para la instancias

Streams Web Service (SWS): Este servicio de gestión basada en web ofrece acceso a los servicios de la instancia. Este servicio es opcional y se puede agregar o quitar de la instancia si es necesario.

Host Controller (HC): El servicio de aplicación de HC se ejecuta en cada host de la aplicación en una instancia. Este servicio lleva a cabo todas las solicitudes de administración de trabajos realizados por el servicio SAM, que incluye iniciar, parar y seguir los elementos del procesamiento.

Processing Element Container (PEC): Cada elemento de proceso en marcha (PE) está alojado en un proceso de PEC, que es iniciado y está controlado por el host donde el PE se está ejecutando. Los servicios PEC se ejecutan cuando las aplicaciones se inician en la instancia de Stream y no se muestran en los diagramas de tiempos de ejecución.

4.3. Elementos del Lenguaje

En esta sección se describe los elementos del lenguaje que utiliza IBM Sphere Streams. Para introducir lo componentes se utilizará un ejemplo de una empresa de telefonía móvil.

Ejemplo: Una empresa de telefonía móvil operativo está preocupado por el desempeño de su infraestructura inalámbrica. Por lo cual, la compañía requiere la utilización para controlar los flujos de llamadas.

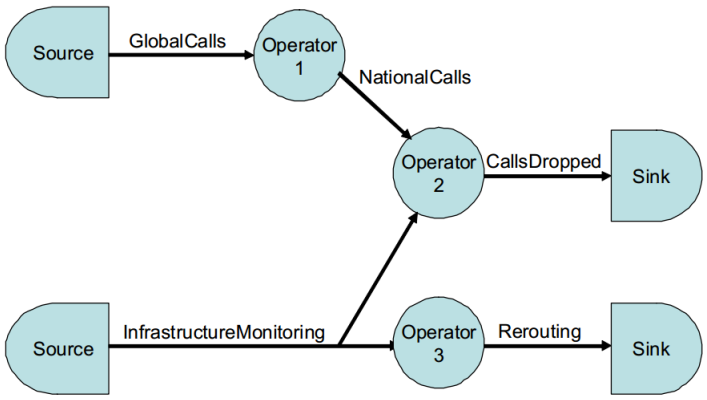


Figura 7: Ejemplo de Llamada Telefónica.
[7]

[7] Imagen obtenida de [29]

Los flujos de entrada a la solicitud son:

Los datos sobre las llamadas que realizan los clientes de la empresa en tiempo real (GlobalCalls). Datos en tiempo real sobre la salud de las antenas de transmisión (InfrastructureMonitoring)

Para evitar el impacto en el procesamiento de todas las tuplas en el flujo GlobalCalls más adelante, la aplicación filtra el flujo GlobalCalls, eliminando los datos sobre las llamadas realizadas en la infraestructura de otros países. Este filtrado se realiza en operador 1.

No siempre es posible determinar una razón del término de una llamada; el cliente puede haber terminado la llamada normal o puede haber sido terminada, debido a que está interrumpida por un fallo. Operador 2 correlaciona los datos de las llamadas, que entran a través de los flujos de llamada nacional, con la información de vigilancia del estado del monitoreo de la información, en el flujo InfrastructureMonitoring. Con esto se puede

determinar que las llamadas se terminaron especificamentes debido a un problema de insfraestructura.

Operador 3 contiene un intento de enrutamiento, sobre la base de los fallos. Envía el re-encaminamiento al disipador de salida inferior.

stream GlobalCalls(...) := Source()[...]...
stream NationalCalls(...) := operator1(GlobalCalls)[...]...
stream InfrastructureMonitoring(...) := Source ()[...]...
operator2(NationalCalls,InfrastructureMonitoring)[...]...
stream Rerouting(... := operator 3(InfrastructureMonitoring)[...]...
Null := Sink(CallsDropped)[...]...
Null := Sink(Rerouting)[...]...

Note que cada flujo se define por primera vez en el lado izquierdo de una asignación (representada por :=). Inmediatamente después de la asignación := es el nombre del operador responsable de la producción de ese flujo.

4.4. Estructura del archivo de procesamiento de flujos del lenguaje

El procesamiento del archivo de lenguaje, es en archivos de texto ASCII que alarmante tienen una extensión .dps. Cada archivo .dps expresa un flujo desde el código fuente (s) al Skin (s) a través de uno o más operadores conectados por flujos. Cada archivo de código fuente se compone de un máximo de seis secciones de las cuales dos son obligatorias. Las seis secciones, son:

1. Aplication
2. Typedefs
3. Libdefs
4. Program
5. Nodepools
6. FunctionDebug

El Aplication y Program son obligatorias y se describen juntas. Las otras secciones son opcionales.

4.4.1. Ejemplo de un archivo de lenguaje

Call Re-routing appllication
[Application]
Routing info
[Program]
stream GlobalCalls(...) := Source()[...]...
stream NationalCalls(...) := operator1(GlobalCalls)[...]...
stream InfrastructureMonitoring(...) := Source ()[...]...
stream CallsDropped(... :=
operator2(NationalCalls,InfrastructureMonitoring)[...]...
stream Rerouting(... := operator 3(InfrastructureMonitoring)[...]...
Null := Sink(CallsDropped)[...]...
Null := Sink(Rerouting)[...]...

- Aplication es el nombre de la aplicación (en este ejemplo).
- Program, es el cuerpo del programa.
- typedefs, con esta sección se pueden crear alias para incorporar en los tipos de datos que se propone utilizar en el programa.

  Ej:

  [Typedefs]

  typedef ListofMasts IntegerList
- Libdefs, puede incluir referencias a archivos de cabecera que describen las interfaces de bibliotecas externas y los del sistema de archivos con las rutas de acceso.
- Nodepools, los grupos de host UNIx, pueden ser definidos.
- FunctionDebug, se pueden escribir expresiones para la prueba exclusivamente.

5. Esper

Las bases de datos relacionales, o sistemas basados en mensajes, hacen bastante difícil el trabajo con datos temporales y consultas en tiempo real. En efecto, las bases de datos requieren consultas explicitas para retornar datos significativos y no están adaptadas para enviar datos, a medida que van cambiando.

Esper es un procesador de flujos de eventos y un motor de correlación de eventos, dirigido a las arquitecturas de eventos en tiempo real (EDA). Esper es capaz de disparar acciones personalizadas escritas como Plain Old Java Objects (POJO), cuando las condiciones de eventos ocurren entre flujos de eventos. Está diseñado para la correlación de eventos de grandes volúmenes donde millones de eventos llegan y es imposible almacenarlos todos para luego consultarlos usando la arquitectura clásica de bases de datos. Un lenguaje adaptado de consultas de eventos (EQL) que permite expresar condiciones a los eventos, correlaciones, ventanas, esto minimiza el esfuerzo de desarrollo al mínimo para configurar un sistema que pueda reaccionar a situaciones complejas.

Esper es un kernel ligero escrito en Java, que es totalmente integrable en cualquier proceso de Java, JEE aplicación de servidor o basada en Java Enterprise Service Bus. Esper permite el desarrollo rápido de aplicaciones que procesan grandes volúmenes de mensajes entrantes o eventos.

5.1. Arquitectura Esper

En esta sección se describiran los diferentes objetos que pertenecen a la Arquitectura de Esper que se pueden apreciar en la siguiente figura 8.

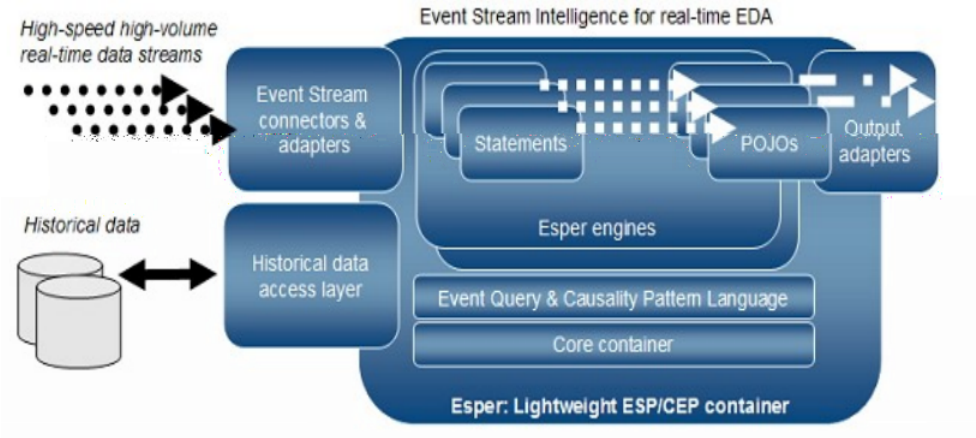


Figura 8: Arquitectura de Esper
[9]

[9] Imagen Obtenida de [13]

Esper se basa en un contenedor ligero formado por diversos componentes, entre los que se incluye:

- **Diversos motores:** gestión de eventos, multi hilos, etc.
- **POJOS:** Asociado 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 distintos flujos de eventos para detectar el patrón buscado.

5.2. Motor de Esper

EL motor de Esper funciona de manera similar a las bases de datos, pero al revés. A diferencia de las bases de datos que almacenan datos y ejecutan consultas sobre éstos, el motor de Esper permite que las aplicaciones almacenen las consultas y las ejecuta sobre un flujo de datos. La respuesta del motor de Esper es en tiempo real, coinciden las condiciones con las consultas y su modelo de ejecución es continuo, no solo cuando se realiza una consulta.

Esper proporciona 2 métodos o mecanismos principales para procesar los eventos, los eventos patrones y consultas de flujo de eventos. Esper provee un lenguaje de patrones de eventos para especificar eventos basados en expresiones y coincidencia de patrones. Detrás del motor de coincidencia de patrones hay una implementación de una máquina de estados. Este método de procesamiento de eventos coincidentes espera la presencia o ausencia de eventos o combinación de eventos. Esto incluye eventos correlacionados basados en el tiempo.

También ofrece consultas a flujos de eventos que satisfagan las necesidades de las aplicaciones de procesamiento complejo de eventos. Las consultas de flujos de eventos provee ventanas, uniones (join) y funciones de análisis para el uso de los flujos. Estas consultas siguen la sintaxis EPL. EPL fue diseñado de manera similar a SQL, con la diferencia que EPL trabaja con vistas en lugar de tablas. Las vistas representan las diferentes operaciones necesarias para estructurar datos y derivar datos de un flujo de eventos.

5.3. Lenguaje de Procesamiento de Eventos

El Lenguaje de Procesamiento de eventos (EPL) es un lenguaje similar a SQL con las clausulas select, from, where, group by, having y order by. Los flujos reemplazan a las tablas como fuente de datos y los eventos reemplazan las filas como unidades básicas de datos. Dado que los eventos se componen de datos, los conceptos de correlación de SQL a través de uniones, filtrado y agregación, mediante agrupación pueden ser efectivamente aprovechados.

La cláusula INSERT INTO se redefine como un medio de transmisión de eventos a otras corrientes para su posterior procesamiento. Los datos externos accesibles a través de JDBC pueden ser consultados y unidos con el flujo de datos. Las cláusulas adicionales, tales como PATTERN y OUTPUT están disponibles para proveer los constructores faltantes en SQL para el procesamiento de eventos.

El propósito de la cláusula UPDATE es actualizar las propiedades de los eventos. Las actualizaciones se realizan antes que los eventos se apliquen a una selección de statements o a statements patrones.

Los statements de EPL se utilizan para obtener y agregar información de uno o más flujos de eventos, y para unir o combinar flujos de eventos. Esta sección describe la sintaxis de EPL y se se exponen los puntos de vista incorporados, que son los bloques de construcción para la derivación y la agregación de información de flujos de eventos.

En EPL los statements contienen definiciones de una o más vistas. Similares a las tablas en una sentencia SQL, las vistas definen los datos disponibles para consultar y filtrar. Algunas vistas representan las ventanas a través de un flujo de eventos y otras se derivan de las estadísticas de las propiedades de eventos, grupos de eventos o manejar valores de las propiedades únicas del evento. Las vistas pueden ser escalonadas una sobre otra para crear una cadena de vistas. El motor de Esper se asegura de que las vistas se vuelven a utilizar entre los statementsde EPL para ser más eficiente.Los tipos de vista son:

Ventanas: win:length, win:length_batch, win:time, win:time_batch, win:time_length_batch, win:time_accum, win:ext_timed, ext:sort, ext:rank, ext:time_order, std:unique, std:groupwin, std:lastevent, std:firstevent, std:firstunique, win:firstlength, win:firsttime.

Derivado de estadísticas: std:size, stat:uni, stat:linest, stat:correl, stat:weighted_avg.

EPL proporciona el concepto de ventana con nombre. Estas son ventanas de datos que pueden ser insertadas-en o borradas-desde uno o más statements, y que puede ser consultada por uno o más statements. Las ventanas con nombre tienen un carácter global, siendo visible y compartida a través de una instancia del motor más allá de una sola sentencia.

Clausulas:

- CREATE WINDOW: Para crear una ventana con nombre.
- ON MERGE: combinar atómicamente los eventos dentro de una ventana con nombre.
- INSERT INTO: Para insertar datos dentro de la ventana.
- ON DELETE: Remover eventos de una ventana.
- ON UPDATE: Actualizar eventos de una ventana.
- ON SELECT: para realizar una consulta provocada por un patrón o un evento que llega en una ventana.

El nombre de la ventana se produce en el statment FROM, utilizado para consultar una ventana con nombre o incluir una ventana en un join o sub-consulta.

Statements: *Es una consulta continua registrada en una instancia del motor de Esper que provee de resultados a los listener cuando llegan nuevos datos, en tiempo real o por demanda a través del iterador.*

5.4. Sintaxis del Lenguaje de procesamiento de eventos

Las consultas en EPL son creadas y almacenadas en el motor, y publica los resultados a los listener cuando los eventos son recibidos por el motor o el temporizador de eventos. A la vez produce que los criterios especificados en la consulta coincidan. Los eventos también se pueden obtener a través de un iterador.

La cláusula select en una consulta EPL especifica las propiedades del evento o eventos que desea recuperar. La cláusula from en una consulta EPL especifica las definiciones de flujo de eventos y los nombres de los flujos a utilizar. La cláusula where en una consulta EPL especifica las condiciones de búsqueda que especifica en que evento o combinación de eventos se buscara. Por ejemplo la siguiente consulta retorna el precio medio de los ticks en el stock de IBM en los últimos 30 segundos.

```
select avg(price) from Accion.win:time(30 sec) where nemo='IBM';
```

Las consultas en EPL siguen la siguiente sintaxis. Estas pueden ser consultas simples o consultas complejas. Una selección simple solo contiene una cláusula select y una definición simple del flujo. Las consultas complejas pueden ser construidas a partir de una lista de expresiones, uniendo multiples flujos, puede contener una clausula where con una condición de búsqueda entre otros. La sintaxis a continuación:

```
[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]
```

5.5. Especificación de periodos de tiempo

Las ventanas basadas en el tiempo como bien se observan los patrones se toma el periodo de tiempo como un parámetro. Los periodos de tiempo siguen la siguiente sintaxis:

time-period : [year-part] [month-part] [week-part] [day-part] [hour-part] [minute-part] [seconds-part] [milliseconds-part]
year-part : (number|variable_name) (ƨears"| ƨear")
month-part : (number|variable_name) ("months"| "month")
week-part : (number|variable_name) ("weeks"| "week")
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"

Ejemplos de periodos de tiempo:
10 seconds
10 minutes 30 seconds
20 sec 100 msec
1 day 2 hours 20 minutes 15 seconds 110 milliseconds
0.5 minutes
1 year
1 year 1 month

5.6. Modelo de Procesamiento

El modelo de procesamiento de Esper es continuo, actualiza los listener y/o suscribe un statement, recibe una actualización de datos tan pronto como el motor procesa los eventos del statement en uso, de acuerdo a la elección del statment, vistas, filtros y rangos de salida del flujo de eventos.

5.6.1. Insertar flujo

En esta sección observaremos la salida de un simple statment de EPL. El statment selecciona un flujo sin usar una ventana y sin aplicar filtros:

```
select * from Withdrawal
```

Este statement selecciona todo los eventos Withdrawal. Cada vez que el motor procesa un evento del tipo Withdrawal o cualquier sub-tipo withdrawal, este invoca todo los update listener, manejando los nuevos eventos de cada statement listener.

El termino insertar flujo denota la llegada de un nuevo evento, y la entrada de una ventana o agregado. El flujo de inserción en este ejemplo es la llegada de los eventos de tipo withdrawal y se los envía a los listener como nuevos eventos.

5.6.2. Insertar y remover flujos

Una longitud de ventana le indica al motor que solo debe mantener los últimos N eventos para un flujo. El siguiente statement aplica una longitud de ventana en un flujo de tipo Withdrawal. El statement sirve para ilustrar en el concepto de ventana, la entrada y salida de eventos.

```
select * from Withdrawal.win:length(5)
```

La longitud de la ventana del statement es de 5 eventos. El motor entra todo los eventos withdrawal dentro de la ventana. Cuando la ventana está llena, el evento más antiguo es expulsado de la ventana. El motor le indica a los listener que todos los eventos entrando a la ventana son nuevos eventos, y los eventos saliendo de la ventana son eventos antiguos.

El termino insertar flujo denota a los eventos que llegan, mientras que el termino remover un flujo se refiere a los eventos que están saliendo de la ventana, o cambiando los valores agregados. En este ejemplo los flujos removidos son los que están saliendo de la ventana de largo 5 y los cuales son puestos como eventos antiguos para los listener.

La figura 9 muestra como la ventana va cambiando a medida que ingresan los eventos y son puestos en el update listener

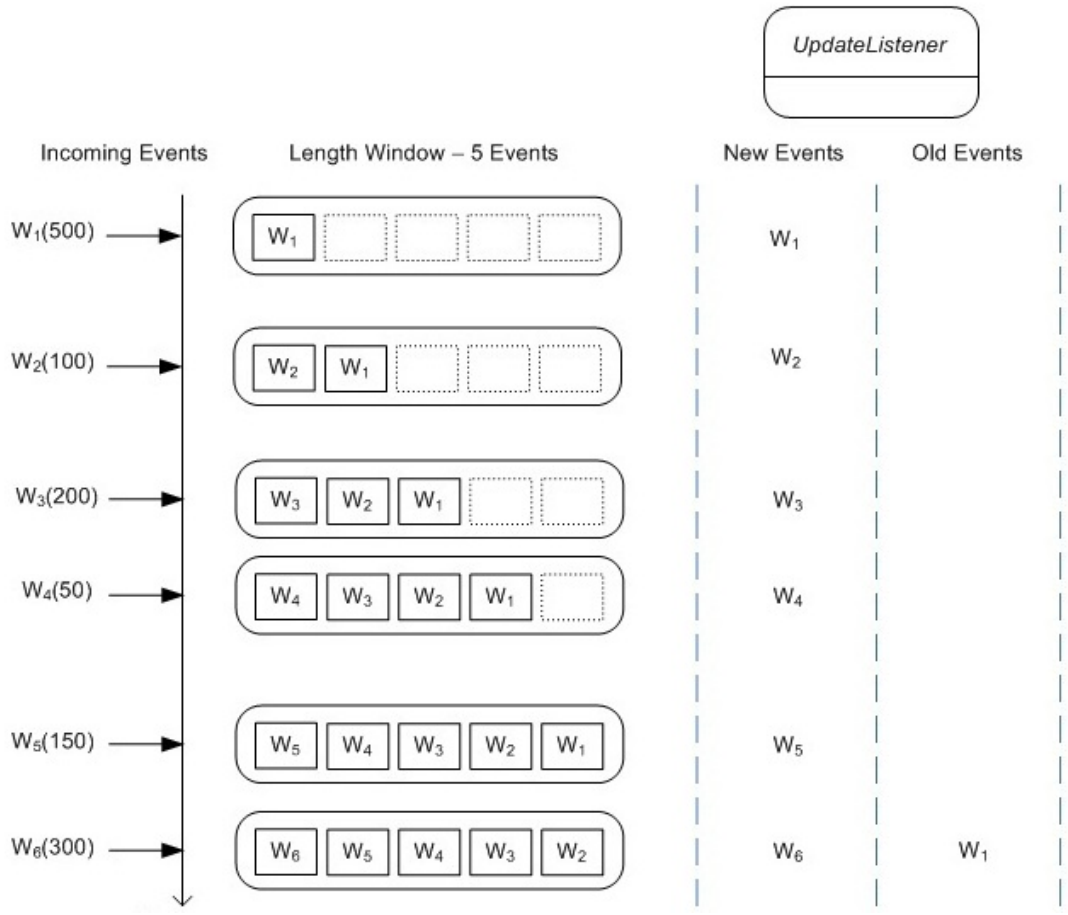


Figura 9: Ejemplo Insertar y remover flujo
[10]

[10]Imagen Obtenida de [13]

Al igual que antes, todos los eventos que llegan se publican como nuevos eventos a los listener. Además, cuando el evento W1 sale de la ventana de longitud a la llegada de evento W6 , se contabiliza como un evento antiguo para los listener. Similar a una ventana de longitud, una ventana de tiempo también mantiene los eventos más recientes hasta un período de tiempo

dado.Una ventana de tiempo 5 segundos por ejemplo, mantiene los últimos 5 segundos de eventos. Con el paso de los segundos la ventana de tiempo activamente va empujando los eventos más antiguos fuera de la ventana resultando en uno o más eventos puestos como eventos antiguos por los update listeners

5.6.3. Filtrado y cláusula Where

Los filtros permiten a los flujos discriminar eventos antes que éstos entren a la ventana. El siguiente statement muestra cómo se filtra una selección a un evento de tipo withdrawal con un valor de 200 o mayor.

select * from Withdrawal(amount>=200).win:length(5)

Con el filtro los eventos que posean una cantidad menor a 200 no entraran en la ventana y no serán pasados por los update listeners

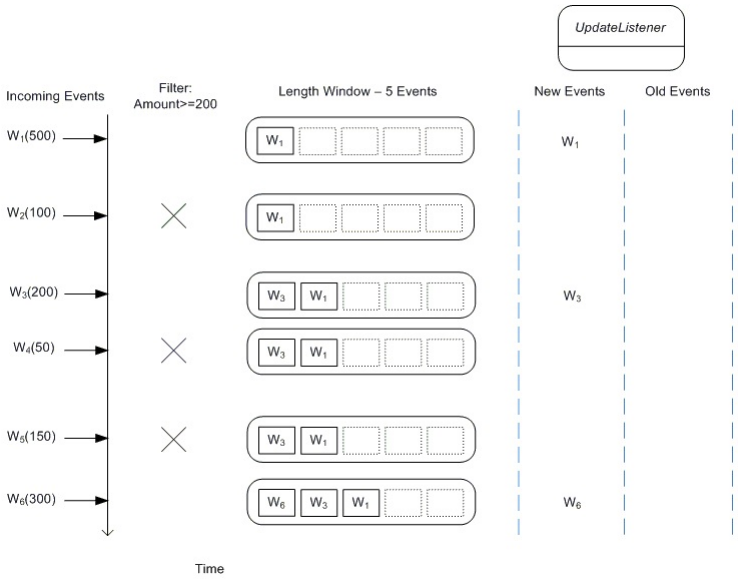


Figura 10: Ejemplo de filtrado
[11]

[11] Imagen Obtenida de [13]

En la figura 10 se observa que solo los eventos con cantidad mayor a 200 entran en la ventana.

El siguiente statement aplica una cláusula where a un evento withdrawal

select * from Withdrawal.win:length(5) where amount >= 200

La cláusula where aplica para ambos, tanto nuevos como antiguos eventos. Como la figura 11 muestra la entrada de eventos que van llegando a la ventana, sin embargo solo los eventos que pasen la cláusula where son pasados a los updates listeners. También los eventos que salen de la ventana, solo los eventos que cumplan con la condición son puestos como eventos antiguos para los listener.

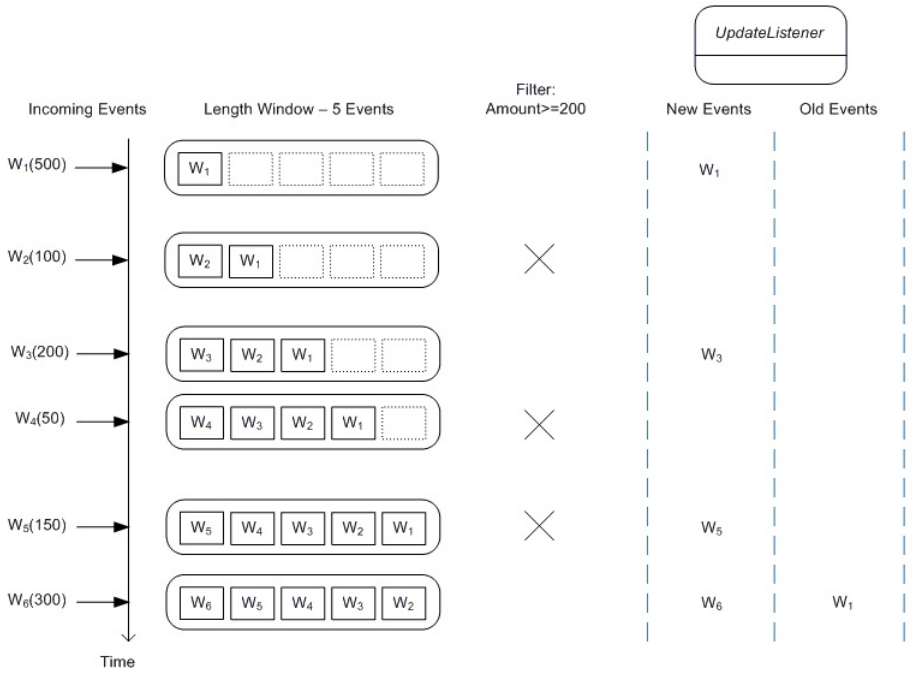


Figura 11: Ejemplo de cláusula where
[12]

[12]Imagen Obtenida de [13]

5.6.4. Ventana con tiempo

En esta sección se explicara el modelo de salida de un statement utilizando vista de ventana con tiempo y vista de tiempo por lotes.

5.6.4.1. Ventana con tiempo

Una ventana con tiempo es una ventana en movimiento que se extiende al intervalo de tiempo que posea. Las ventanas con tiempo nos permiten limitar el número de eventos considerados en una consulta, a lo largo de la ventana. Un ejemplo práctico es considerar la necesidad de determinar toda las cuentas donde el monto en los últimos 4 segundos es mayor a 1000. El statement que resuelve este problema es:

select account, avg(amount) from Withdrawal.win:time(4 sec) group by account having amount >1000

La figura 12 sirve para ilustrar el funcionamiento de una ventana con tiempo, para este caso se asume una consulta simple que selecciona los eventos, pero no los filtra ni agrupa. El diagrama comienza en un tiempo dado t y muestra los contenidos de la ventana con tiempo en t + 4 y t + 5 segundos y así sucesivamente

select * from Withdrawal.win:time(4 sec)

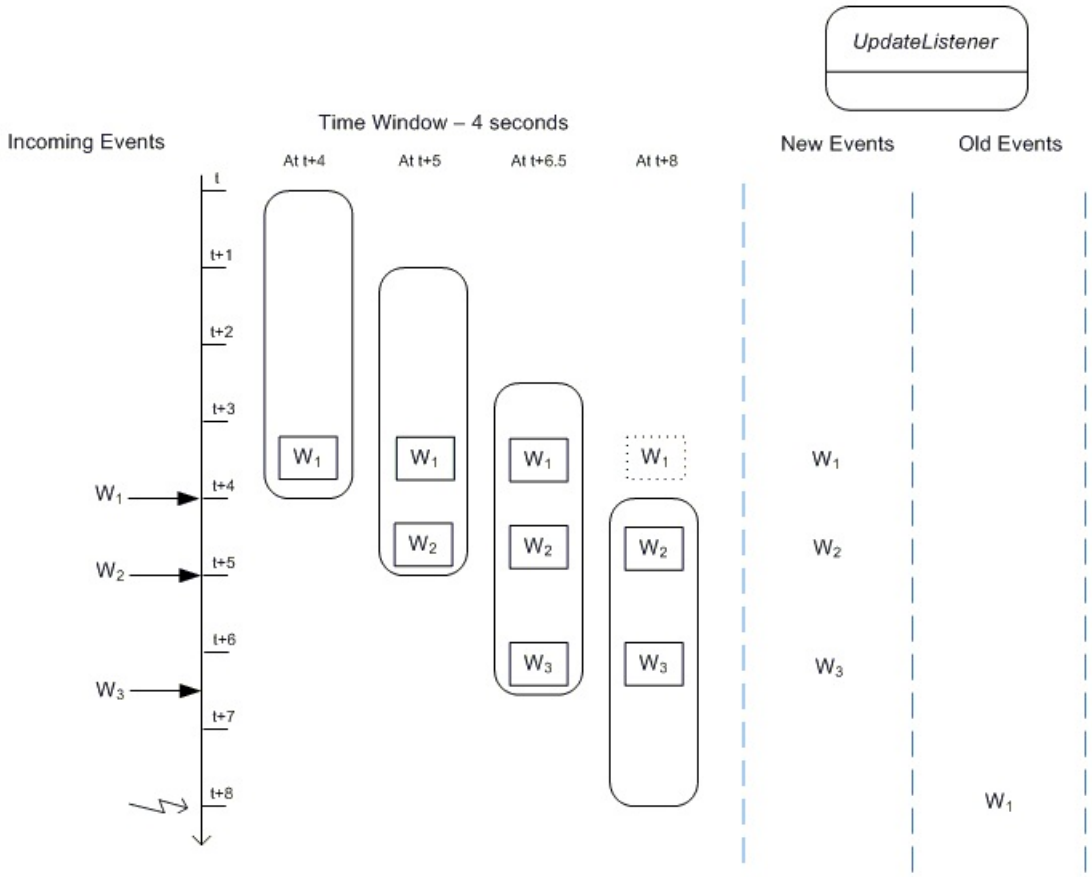


Figura 12: Ejemplo de ventana con tiempo
[13]

[13]Imagen Obtenida de [13]

5.6.4.2. Ventana por lotes

La vista de tiempo por lotes carga los eventos y los despacha cada un intervalo de tiempo específico. La ventana con tiempo controla la evaluación de los eventos, como el tamaño del lote. La figura 13 sirve para ilustrar el funcionamiento de una ventana con tiempo, para este caso se asume una consulta simple:

select * from Withdrawal.win:time_batch(4 sec)

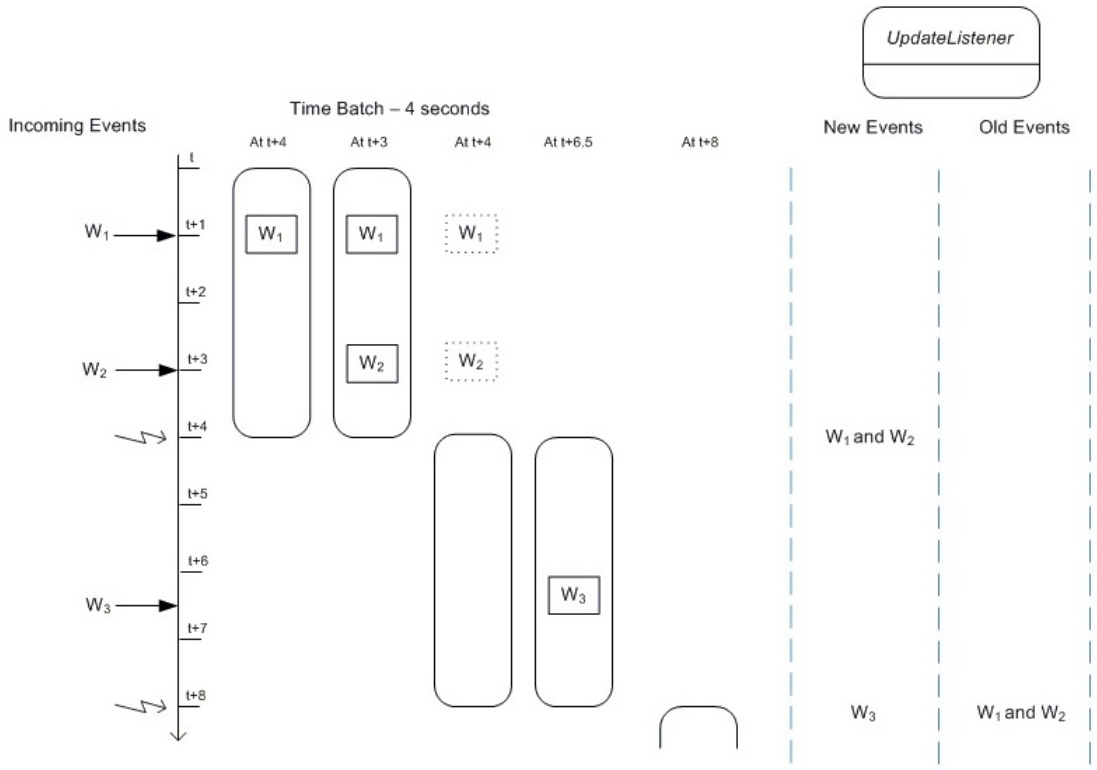


Figura 13: Ejemplo de ventana con tiempo por lotes
[14]

[14] Imagen Obtenida de [13]

La actividad ilustrada en el diagrama:

1. En el tiempo t + 1 segundos un evento W1 llega y entra al lote. No se llama al update listener para informar la ocurrencia.

2. En el tiempo t + 3 segundos un evento W2 llega y entra al lote. No se llama al update listener para informar la ocurrencia.

3. En el tiempo t + 4 segundos el motor procesa el lote de eventos y comienza un nuevo lote. El motor reporta los eventos W1 y W2 a los update listener.

4. En el tiempo t + 6.5 segundos un evento W3 llega y entra al lote. No se llama al update listener para informar la ocurrencia.

5. En el tiempo t + 8 segundos el motor procesa el lote de eventos y comienza un nuevo lote. El motor reporta el evento W3 a los update listener. El motor reporta los eventos W1 y W2 como datos antiguos (lotes antiguos) a los update listener.

5.7. Representación de eventos

Un evento es un registro inmutable de un acontecimiento pasado de un cambio de acción o estado. Las propiedades de eventos capturan la información del estado para un evento. En Esper, un evento puede ser representado por cualquiera de los siguientes objetos subyacentes de Java:

Java Class	Descripción
Java.lang.Object	Cualquier Java POJO(plain-old java object) con métodos getter siguiendo las convenciones de JavaBean. Cabe destacar que las clases heredadas de java que no siguen las convenciones JavaBean también pueden servir como eventos.
Java.util.Map	Mapa eventos son implementados de la clase java.util.Map interfaz donde cada entrada de correlación es un valor de la propiedad
Object[] (vector de objecto)	Matriz de objetos, son las matrices de objetos en el que cada elemento de la matriz es un valor de la propiedad
Org.w3c.dom.Node	Documento XML del modelo de objetos (DOM)
Org.apache.axiom.om.OMDocument or OMElement	XML Streaming API para XML (Stax). Axiom Apache, proporcionado por el paquete EsperIO.
Application classes	Plug-in representación de eventos a través de la API de extensión

Tabla 3: Representación de Eventos en Esper

Esper ofrece múltiples opciones para la representación de un evento. No hay necesidad absoluta para crear nuevas clases Java para representar un evento.

5.8. Propiedades del evento

Propiedades de eventos capturan la información de estado para un evento. Las propiedades de eventos permiten diversos tipos, según sea la implementación del evento. Estas pueden ser de tipo simple, así como indexado, mapeada y propiedades anidadas. La siguiente tabla muestra los diferentes tipos de propiedades y su sintaxis en una expresión evento. Esta sintaxis permite a los estados a consultar profundas objetos JavaBean gráficas, estructuras XML y eventos de mapa.

Tipo	Descripción	Sintaxis	Ejemplo
Simple	Una propiedad que tiene un único valor que puede ser recuperado.	nombre	sensorID
Indexada	Propiedad indezada almacena una colección ordenada de objetos (todos del mismo tipo) que se puede obtener acceso mediante un número entero de valor no negativo.	nombre[indice]	sensor[0]
Mapeada	Esta propiedad almacena una colección de objetos con llave (todos del mismo tipo)	nombre("llave")	sensor("luz")
Anidada	Las propiedades anidadas, son aquellas que viven dentro de otra propiedad de un evento	nombre.nombreAnidado	sensor.valor

Tabla 4: Propiedades de eventos en Esper

Cabe destacar que las combinaciones son también posibles. Por ejemplo, una combinación válida podría ser: persona.direccion('casa').calle[0] .

5.9. Propiedades dinámicas de eventos

Las propiedades dinámicas son propiedades de eventos que no tienen que ser conocidas en tiempo de compilación. Estas propiedades se resuelven en tiempo de ejecución, ya que no se puede saber el total de estás, según sea el modelo a representar.

La idea detrás de las propiedades dinámicas es que para un determinado evento de representación subyacente, no siempre se sabe todas las propiedades de antelación. Un evento subyacente puede tener propiedades adicionales que no se conocen en tiempo de compilación, que se quiere consultar sucesivamente. El concepto es especialmente útil para los eventos que representan modelos orientados a objetos de dominio.

La sintaxis de las propiedades dinámicas consiste en el nombre de propiedad y un signo de interrogación. Indexadas, las propiedades asignadas y anidadas también pueden ser propiedades dinámicas:

Tipo	Sintaxis
Simple	nombre?
Indexada	nombre[index]?
Mapeada	nombre(llave)?
Anidada	nombre?.propiedadNombre

Tabla 5: Propiedades dinámicas de eventos en Esper

Las propiedades dinámicas siempre devuelven el java.lang.Object tipo. Asimismo, las propiedades dinámicas devolver un null como valor si la propiedad dinámica no existe en eventos procesados en tiempo de ejecución.

5.10. Declaración de eventos en Esper

Las clases de java son una buena alternativa para representar los eventos; sin embargo la representación de eventos en XML también pueden ser una buena elección dependiendo de la arquitectura de los requerimientos.

Suponga que hay un NewEmployeeEvent clase de evento, como se muestra a continuación. Las propiedades mapeadas e indexadas en este ejemplo retornan objetos Java, pero también puede devolver tipos de lenguaje Java primitivos (como int o String). El tipo de objeto Addres y Employee pueden tener propiedades que están anidados dentro de ellos, como el nombre de una calle en la Address de un objeto o nombre del empleado en el Employee objeto.

```
public class NewEmployeeEvent {
        public String getFirstName();
        public Address getAddress(String type);
        public Employee getSubordinate(int index);
        public Employee[] getAllSubordinates();
}
```

Las propiedades simples de eventos requieren un método captador que devuelve el valor de la propiedad. En este ejemplo, el getFirstName método getter devuelve el firstName propiedad de evento de tipo String.

En el caso de la declaración de eventos en XML, utilizando un ejemplo similar. Cabe destacar que la representación en POJO puede ser considerada equivalente para el propósito de este ejemplo:

```
<mediaorder>
  <orderId>PO200901</orderId>
   <items>
    <item>
      <itemId>100001</itemId>
      <productId>B001</productId>
      <amount>10</amount>
      <price>11.95</price>
    </item>
   </items>
   <books>
    <book>
      <bookId>B001</bookId>
      <author>Heinlein</author>
      <review>
       <reviewId>1</reviewId>
       <comment>best book ever</comment>
      </review>
    </book>
    <book>
      <bookId>B002</bookId>
      <author>Isaac Asimov</author>
    </book>
   </books>
</mediaorder>
```

5.11. Configuración del motor de Esper

La interfaz EPServiceProvider, de la API representa una instancia del motor. Cada instancia de un motor de Esper es completamente independiente de otras instancias del motor y tiene su propia interfaz administrativa y de ejecución.

Una instancia de la máquina Esper se obtiene mediante métodos estáticos de la clase EPServiceProviderManager. El método getDefaultProvider y getProvider(String providerURI) devuelven una instancia del motor de Esper. El último puede ser utilizado para obtener las instancias múltiples del motor para diferentes valores de proveedor de URI. El EPServiceProviderManager determina si el proveedor URI coincide con todos los valores anteriores URI proveedor y devuelve la instancia del mismo motor para el mismo valor del proveedor URI. Si el proveedor URI no se ha visto antes, se crea una instancia del motor de nuevo.

El siguiente código ejemplifica la secuencia de uso típica del motor de Esper:

```
// Configurar el motor, esto es opcional
Configuration config = new Configuration();
// carga la configuración desde un archivo
config.configure("configuration.xml");
// realizer ajustes de configuración adicional
config.set....(...);
// obtiene una instancia del motor
EPServiceProvider epService =
EPServiceProviderManager.getDefaultProvider(config);
// Opcionalmente, utilizar el initialize si
//la instancia del motor mismo ha sido iniciado
epService.initialize();
// Opcionalmente, realiza los cambios de
//configuración en tiempo de ejecución
epService.getEPAdministrator().getConfiguration().add...(...);
// destruir la instancia del motor cuando
//no se necesita, se liberan recursos
epService.destroy();
```

5.12. Clase Configuración

Una instancia de com.espertech.esper.client.configuration representa todos los parámetros de configuración. La clase Configuration se utiliza para construir un objeto EPServiceprovider, la cual proporciona las interfaces de administración y tiempo de ejecución de una instancia del motor Esper. Con esta clase, se puede obtener una configuración creando instancias de ella directamente y añadir o establecer valores en ella. La Configuration se pasa un objeto de tipo EPServiceProviderManager para obtener un motor de Esper configurado.

Un ejemplo de la configuración, se presenta a continuación:

```
Configuration configuration = new Configuration();
configuration.addEventType("PriceLimit",
PriceLimit.class.getName());
configuration.addEventType("StockTick", StockTick.class.getName());
configuration.addImport("org.mycompany.mypackage.MyUtility");
configuration.addImport("org.mycompany.util.*");

EPServiceProvider epService =
EPServiceProviderManager.getProvider("sample", configuration);
```

6. Simulación de Flujo de Acciones en la Bolsa

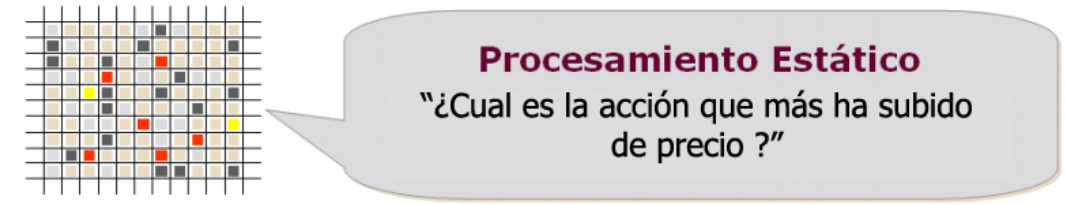
En este último capítulo abarcaremos todo lo que es el caso de estudio, partiendo por una pequeña reseña de este, continuando con los modelos de clase y de secuencia de la aplicación desarrollada. Primero veremos el diagrama de clase para posteriormente, presentar dos diagramas de secuencia, que ejemplifican el funcionamiento del motor de Esper y la generación de eventos en este. Finalmente, explicar la aplicación que se implementó sobre el flujo de acciones en la bolsa.

6.1. Caso de Estudio

Dentro de los problemas cotidianos de las empresas, existen casos en los que un retraso o latencia de pocos segundos puede resultar en un grave perjuicio económico, organizativo o incluso vital. Por lo cual, plantearemos un caso de estudio que requiere de respuestas en tiempo real para poder monitorear los flujos de datos en un tiempo determinado.

6.2. Flujos de datos en la Bolsa de Valores

El caso a estudiar consiste en la problemática de obtener información de las transacciones realizadas en la bolsa de valores en tiempo real, se requiere monitorear constantemente los flujos que se generan a partir de un cambio de estado en una acción.



6.2.1. Declaración de Evento

Se ha elaborado un tipo de evento que asimila las características de una transacción de una acción, la cual se compone por su nombre identificador «nemo», el valor de este «precio» y una marca de tiempo que entrega la hora con los milisegundos; aquello corresponde al momento exacto en donde se realizó la transacción «marcaDeTiempo». A continuación se muestra el código utilizado, el cual es representado mediante un POJO(acrónimo de Plain Old Java Object):

```
package evento.modelo;
import java.util.Date;

public class Accion {
        private String nemo;
    private Integer precio;
    private Date marcaDeTiempo;

    public Accion(String s, Integer p, long t) {
        this.nemo = s;
        this.precio = p;
        this.marcaDeTiempo = new Date(t);
    }
    public Integer getPrecio() {
        return precio;
    }
    public String getNemo() {
        return nemo;
    }
    public Date getMarcaDeTiempo() {
        return marcaDeTiempo;
    }
    @Override
    public String toString() {
        return "Precio: " + precio.toString() +
                    " Tiempo: " + marcaDeTiempo.toString();
    }
}
```

6.2.2. Consultas

A continuación se explicaran algunos de los statments utilizados en el plan de consultas, describiendo su funcionamiento y graficando el resultado que estos entregan.

```
select * from Accion(nemo= 'BANVIDA').win:time(1 min)
```

Lo que se obtiene al ejecutar la consulta anterior es un aviso, el cual se genera apartir de la ejecución de una transacción de una acción, en este caso BANVIDA, durante períodos de un minuto. En caso de que se generen un número finito de eventos, y que en ningún momento se realice una transacción de la acción BANVIDA, el statement que fue asignado a la consulta no levantará un evento de la misma. Por lo tanto, su funcionamiento es similar a los Trigger de las bases de datos relacionales, cuya diferencia, es la gran cantidad de datos simultáneos y en tiempo real que puede monitorear. Ahora bien, si el statement recibe un listener de parte de la consulta, se puede llevar a cabo una decisión con respecto a estos, en este caso nosotros hemos generado un gráfico donde se pueden apreciar los cambios que sufre el valor de la acción durante un periodo de tiempo determinado.

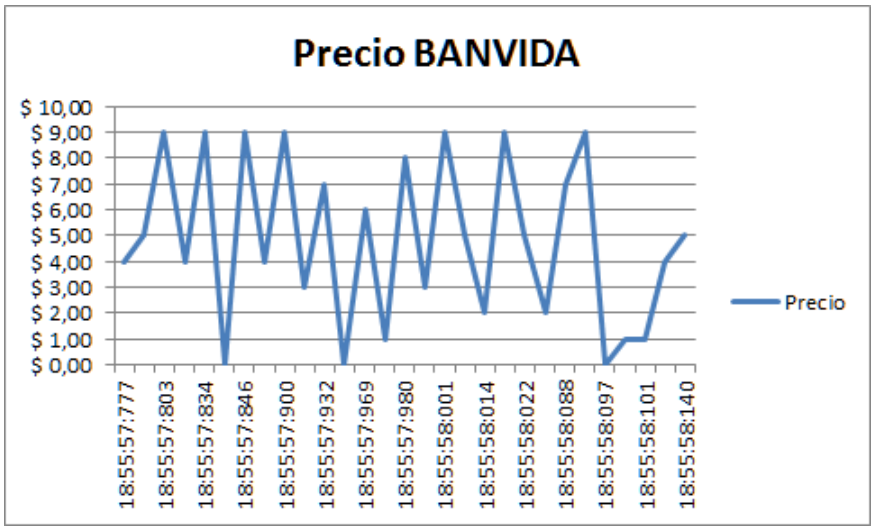


Figura 14: Variación de Precio

Los datos del grafico corresponden a 10000 eventos en el periodo de tiempo de 18:55:57 hasta las 18:55:58. Cabe destacar que los eventos

generados, fueron de forma aleatoria de un total de 382 distintos Nemos (identificador de las acciones), cuyos precios también fueron generados aleatoriamente con una escala de 1 a 10. Por lo tanto, los eventos capturados por el statement encargado de la consulta anterior, solo recibió 29 eventos que cumplian con la regla descrita en la consulta.

En cuanto a los datos obtenidos de la consulta continua, se puede determinar que el precio de la acción BANVIDA cambia bruscamente su precio en un período corto de tiempo, el cual sube y baja rápidamente. Si fuera un caso real, los accionistas estarían alarmados por las variantes de ésta y seria muy probable que su compra/venta fuera congelada.

```
select avg(precio) as avgPrecio, " +
"marcaDeTiempo from Accion(nemo= 'TELCOY').win:time(20 sec)"
```

La consulta anterior captura el precio promedio de la acción TELCOY más su marca de tiempo, que refleja el instante que se realiza la transacción, la cual se desplaza en una ventana de tiempo de 20 segundos. Al igual que la consulta anterior, cada vez que sea capturado un evento que correspondan a esta consulta, se obtendrá los datos que se esperan según la definición de la consulta.

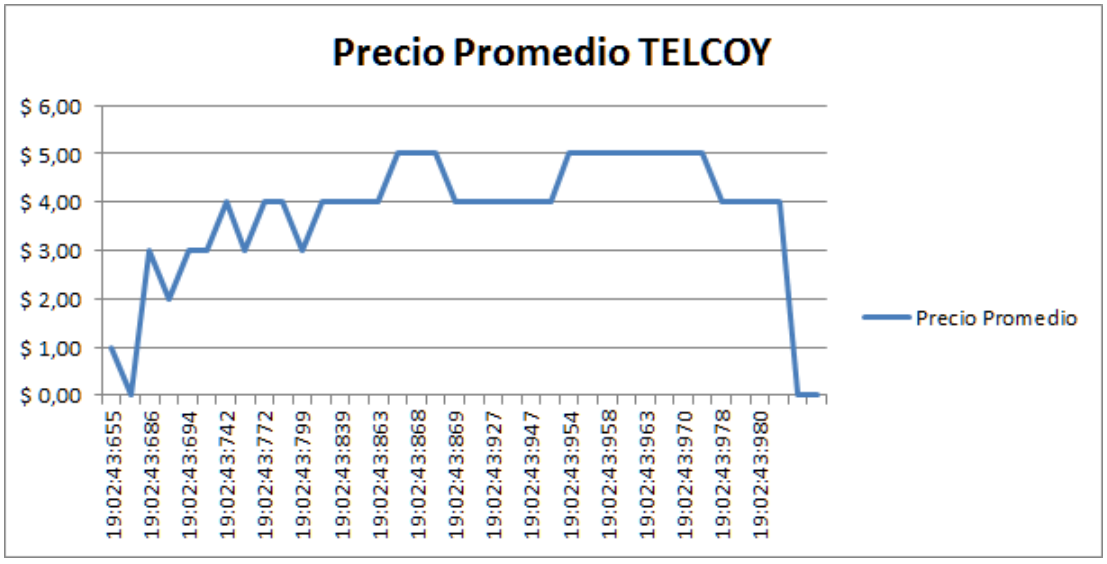


Figura 15: Promedio del Precio

Los datos del grafico corresponden a 10000 eventos en el periodo de tiempo de 19:02:43 hasta las 19:02:44.

Cabe destacar que los eventos generados, fueron de forma aleatoria de un total de 382 distintos Nemos (identificador de las acciones), cuyos precios también fueron generados aleatoriamente con una escala de 1 a 10. Por lo tanto, los eventos capturados por el statement encargado de la consulta anterior, solo recibió 36 eventos que cumplian con la regla descrita en la consulta.

Con respecto a los resultados demostrados mediante el grafico de la figura 15, en un caso real, la empresa que está interesada en la consulta puede tomar las medidas pertinentes por la fuerte caída del promedio del precio en un instante determinado, entre otras cosas. A la vez, se puede observar que la acción fue en aumento, la cual se mantuvo en un lapso de tiempo.

6.3. Modelo de Clase

A continuación se presenta el diagrama de clases generado para la aplicación web que permite, generar eventos y capturarlos mediante los listener del motor de flujos de datos, Esper.

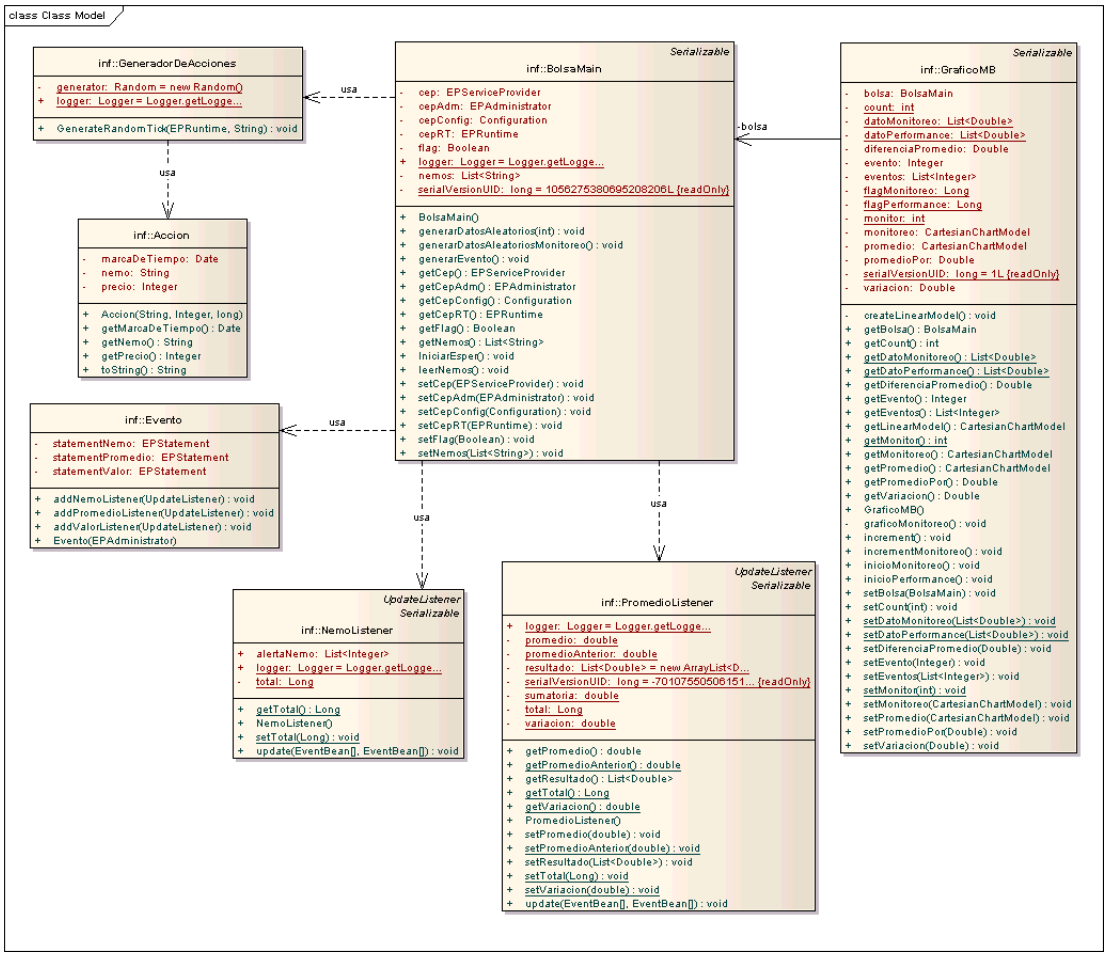


Figura 16: Modelo de Clases

6.4. Diagrama de Secuencia Instancia Motor de Esper

Se presenta el diagrama de secuencia, donde se obtiene la instancia de la clase que configura el motor de Esper. La clase BolsaMain.java importa los componentes necesarios para poder instanciar la configuración y administración propia de la API proporcionada por el motor de flujos de datos.

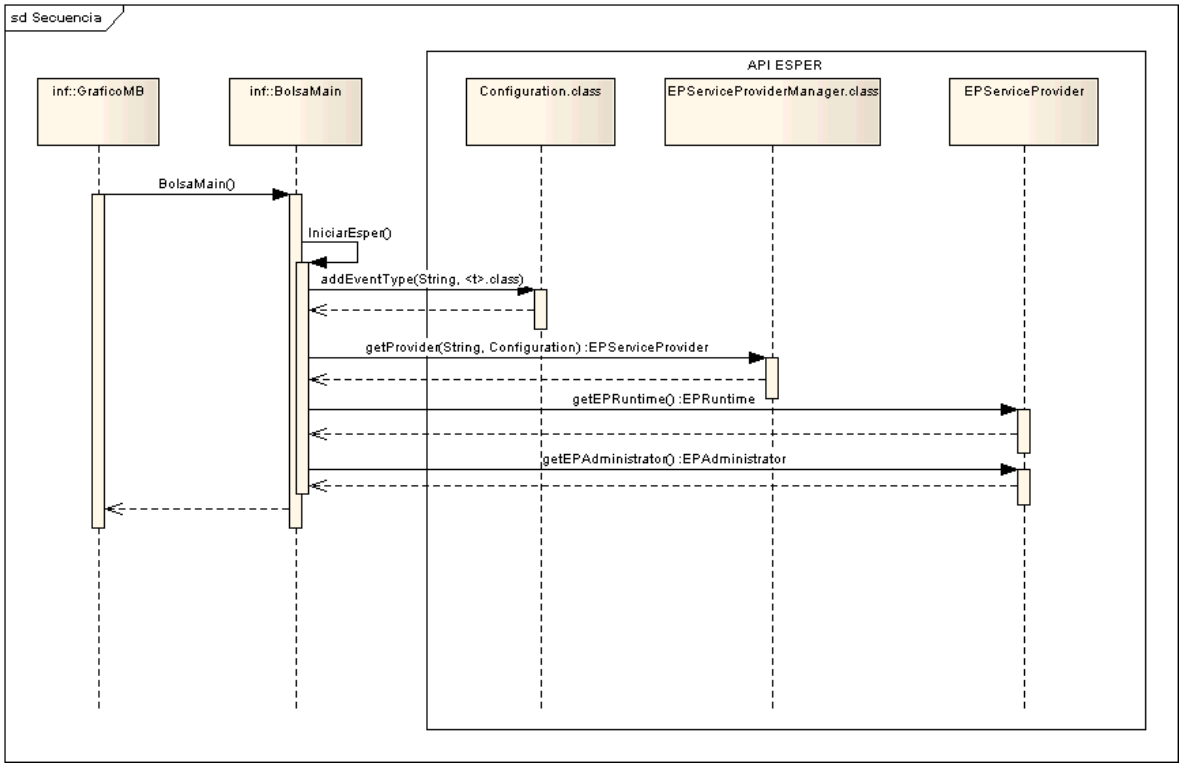


Figura 17: Diagrama Secuencia Motor de Esper

6.5. Diagrama de Secuencia Generación de Eventos

La secuencia para generar y posteriormente capturar los eventos por el motor de Esper, se señalan en el siguiente diagrama de secuencias, el cual constituye el flujo completo desde que el usuario de la aplicación envía una petición para lanzar una cantidad de eventos limitados.

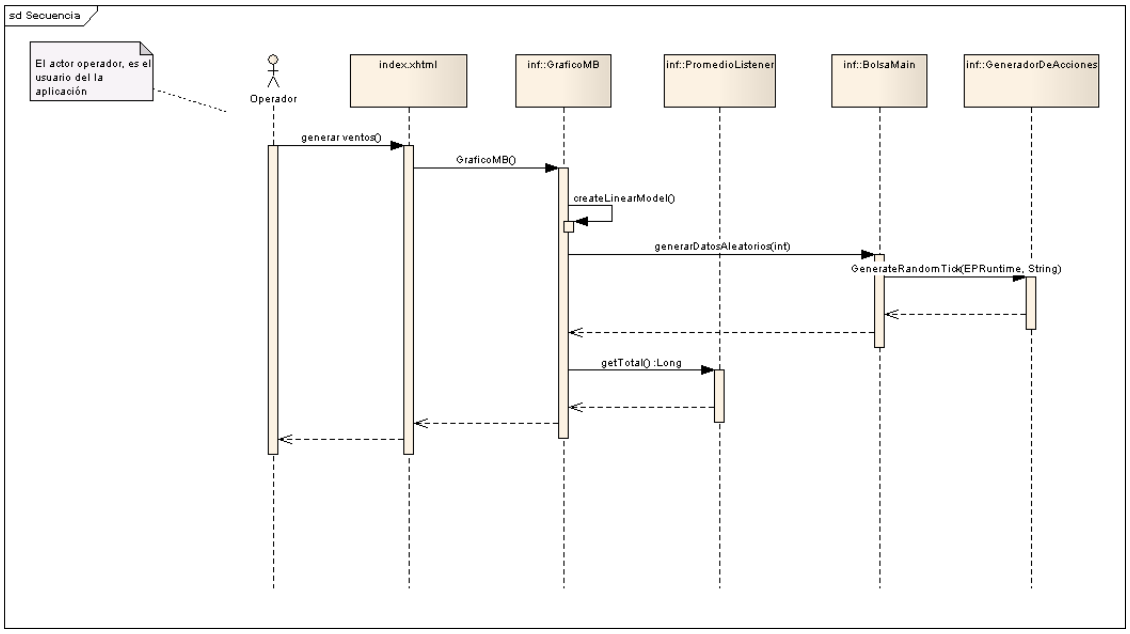


Figura 18: Diagrama Secuencia Generación de Eventos

6.6. Aplicación: Simulación de Flujo de Acciones en la Bolsa

A continuación se presenta el listado de tecnologías utilizadas para construir la aplicación web, donde se implementará el motor de flujos de datos, Esper.

- **Java EE 6**: Java Enterprise Edition, es un estándar industrial para aplicaciones informáticas empresariales, perteneciente a la plataforma de Java, que permite desarrollar y ejecutar software implementados en el lenguaje de programación Java.
- **JSF 2.0**: JavaServer Faces, es un framework que permite construir del lado del servidor interfaces de usuario. La API de JSF, otorga herramientas para desarrollar aplicaciones web basadas en el patrón MVC (Modelo Vista Controlador). Esta tecnología se basa en componentes y eventos generados por el usuario a través de un navegador web a una página en particular, los cuales son manejados en el lado del servidor por componentes javas.
- **Tomcat 6.0.35**: Es una implementación de código abierto desarrollado en java, que permiten trabajar con las especificaciones Java Servlet y JavaServer Pages. Tomcat, tiene como función principal servir como un contenedor de servlets.
- **Maven 3.0.4**: Gestor de proyectos de software que permite la construcción de proyectos. Destaca por su modelo de configuración basado en un archivo XML, para la construcción de proyectos.

Esta aplicación permite mantener un monitoreo sobre el flujo de datos que existe en el sistema, los cuales están basados en la bolsa de valores de Santiago, en este caso se utilizara como filtro la acción TELCOY para poder analizar el comportamiento de esta en tiempo real.

En la parte gráfica de la aplicación se pueden apreciar 3 valores:

- **Variación**: Corresponde a la variación porcentual entre el ultimo promedio y el actual.
- **Promedio anterior**: Obtenido con los datos entregados por el listener, previamente configurado en el motor de Esper, hasta un ciclo antes del promedio actual.
- **Promedio**: Indica el promedio actual.

Además se pueden observar dos gráficos lineales, los cuales reflejan los resultados que se generan a partir de dos consultas realizadas sobre un mismo evento, pero de distinto tipo. La primera es realizada mediante una ventana de tiempo, mientras que la segunda es una ventana por lote.

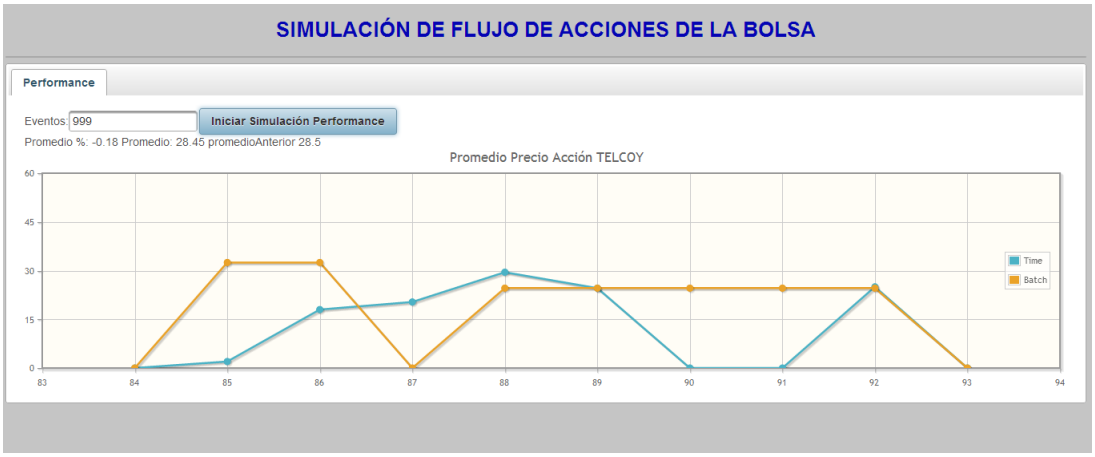


Figura 19: Interfaz Aplicación

Como se puede ver en la figura 19, los gráficos son distintos, pese a que los listener capturan el mismo evento. Esto es producto de la implementación de dos tipos de ventana de tiempo en los statement configurados.

En la figura 20 se aprecia un indicador el cual irá variando según el comportamiento que tenga el precio de la acción el que contiene diferentes colores que indican que tan crítico es precio en el tiempo y poder tomar alguna decisión en tiempo real, el rojo expresa un valor muy bajo y el verde un valor alto.

Figura 20: Interfaz Aplicación

7. Conclusión

A partir de lo expuesto en el presente trabajo, se puede concluir que el caso de estudio desarrollado, nos entrega una visión más amplia de lo que respecta al funcionamiento de la arquitectura e infraestructura de software de un sistema administrador de flujos de datos. Cabe destacar que los motores de consultas continuas, son imprescindibles a la hora de requerir información oportuna para la toma de decisiones dentro de una empresa y/u organismo. El mecanismo utilizado en Esper y en general de los DSMS, para capturar eventos de interés, es sin duda una funcionalidad que destaca a estos sistemas. Es importante señalar que la recopilación, análisis y presentación de la información acerca del monitoreo de datos, mediante sistemas administradores de flujos de datos, ha sido esencial para obtener los fundamentos teóricos de los principios de flujos de datos, arquitectura y principales características de los DSMS.

Este proyecto permite aclarar y comprender los distintos objetivos que poseen los sistemas de gestión de flujos, tales como el análisis de grandes volúmenes de datos que circulan y/o trabajan en diversas aplicaciones en tiempo real. Estos sistemas se han enfrentado al problema de procesar flujos de forma continua, debido a que los sistemas administradores de bases de datos convencionales no pueden trabajar bajo mencionadas características, puesto que funcionan a través de operadores relacionales tradicionales, que son definidos para relaciones finitas. Un factor relevante de los sistemas DSMS es el procesamiento de datos, el cual consiste en formar ventanas deslizantes sobre flujos de datos. De esta forma, los flujos de datos se pueden representar como relaciones finitas en un determinado tiempo, lo cual permite que la ventana transite por el flujo. De tal manera, el contenido de la relación se cambia para reflejar los datos que pasan en un instante específico.

Cabe destacar que el concepto de consultas continúas, inducidas a flujos de datos, que circulan en tiempo real, por lo general, en grandes volúmenes, cambia los paradigmas tradicionales con los que se acostumbra a trabajar y estudiar datos. Debido a que los datos estáticos y finitos, se volvieron comunes para todas las áreas que requerían administrar su información, se volvió imprescindible realizar trabajos que dieran como resultado, herramientas y/o técnicas para poder monitorear los flujos de datos. Por lo general, son flujos que transitan por las aplicaciones, principalmente nuevas, como lo son las redes de sensores, estudio del estado de pacientes en los hospitales, entre otras.

Según los lineamientos propuestos y definidos en el marco de este proyecto, se recolecto información sobre los sistema administrador de flujo

de datos y lenguaje de consultas de los DSMS ESPER, STREAMS y IBM InfoSphere Streams[29]. Estos son necesarios para tener una visión de las distintas estructuras y funcionalidades que brindan los distintos DSMS, de esta forma poder implementar un sistema de consultas sobre un flujo de información, que cumpla con los requisitos del sistema requerido.

Referencias

[1] D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, J. h. Hwang, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, Y. Xing, R. Yan, and S. Zdonik. Aurora: A data stream management system. In *In ACM SIGMOD Conference*, page 666, 2003.

[2] Nauman A. Chaudhry Kevin Shaw Mahdi Abdelguerfi. *Stream Data Management*. Editorial Springer, 2005.

[3] Amit Ahuja. *Stream Load Shedding - Dynamically Managing Channel Capacity*. Bertrams Print On Demand, 2008.

[4] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R Motwani, U. Srivastava, and J. Widom. Stream: The stanford data stream management system. Technical Report 2004-20, Stanford InfoLab, 2004.

[5] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The cql continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, June 2006.

[6] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, pages 1–16, New York, NY, USA, 2002. ACM.

[7] Shivnath Babu. Adaptive query processing in data stream management systems. ., 2005.

[8] Shivnath Babu and Jennifer Widom. Continuous queries over data streams. *SIGMOD Record*, 30(3):1 – 12, September 2001.

[9] Don Carney, Uğur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Monitoring streams: a new class of data management applications. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 215–226. VLDB Endowment, 2002.

[10] Badrish Chandramouli, Mohamed H. Ali, Jonathan Goldstein, Beysim Sezgin, and Balan Sethu Raman. Data stream management systems for computational finance. *IEEE Computer*, 43(12):45–52, 2010.

[11] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R. Madden, Fred Reiss, and Mehul A. Shah. Telegraphcq: continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 668–668, New York, NY, USA, 2003. ACM.

[12] Chuck Cranor, Yuan Gao, Theodore Johnson, Vlaidslav Shkapenyuk, and Oliver Spatscheck. Gigascope: high performance network monitoring with an SQL interface. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, page 623, New York, NY, USA, 2002. ACM Press.

[13] Esper. http://esper.codehaus.org/.

[14] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2 edition, 2008.

[15] Johannes Gehrke and M. Hellerstein. Guest editorial to the special issue on data stream processing. *The VLDB Journal*, 13(4):317–317, December 2004.

[16] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Quicksand: Quick summary and analysis of network data, 2001.

[17] Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, June 2003.

[18] Lukasz Golab and M. Tamer Özsu. *Data stream management*. Morgan & Claypool, New York, NY, USA, 2010.

[19] Java. http://www.oracle.com/technetwork/java/javaee/overview/index.html.

[20] JSF. http://javaserverfaces.java.net/.

[21] Morten Lindeberg. Data stream management system, introduction - concepts and issuess. slide 6, Septiembre 2008.

[22] Alessandro Margara and Gianpaolo Cugola. Processing flows of information: from data stream to complex event processing. In *Proceedings of the 5th ACM international conference on Distributed event-based system*, DEBS '11, pages 359–360, New York, NY, USA, 2011. ACM.

[23] Maven. http://maven.apache.org/what-is-maven.html.

[24] Oracle. Cep cql language reference http://docs.oracle.com/.

[25] Minos Garofalakis Johannes Gehrke Rajeev Rastogi. Data stream management: Processing high-speed data stream. Stanford University, Usa, julio 2006.

[26] P. Bonnet J. Gehrke P. Seshadri. Towards sensor database systems. in proc. int. conf. on mobile data management. Computer Science Department, Upson Hall Cornell University, Ithaca, NY, USA, Enero 2001.

[27] StreamBase. http://www.streambase.com/products/streambasecep.

[28] Tomcat. http://tomcat.apache.org/.

[29] Chuck Ballard Daniel M Farrell Mark Lee Paul D Stone Scott Thibault Sandra Tucker. *IBM InfoSphere Streams Harnessing Data in Motion*. Redbooks, Septiembre 2010.