The Conceptual View of StreamEPS.

(revised December 2020)

F. Appiah. *Member, IEEE*

Abstract—These Event processing systems are much used in wide variety of applications in the processing of large stream of events. The most distinguished of applications is the time-series data management system with timely processing to identify trends, pattern matches and forecast future values. The complexity of event information, coupled with the fact that historical event data is being kept in the database, requires the use of an event processing model that provides the user with high-level abstractions. In this paper, I survey the StreamEPS to help developers and researchers alike to understand the conceptual view and processing of the event processing software system. StreamEPS forms part of Complex EventProcessing (CEP), Data Stream Management System (DSMS) and Information Flow Processing (IFP) domain.

Index Terms—architecture, EPS, StreamEPS, conceptual view, event, stream, data

xxxx-xxxx/0x/\$xx.00 © 200x IEEE Published by the IEEE Computer Society

- F. APPIAH is with the Kwame Nkrumah University of Science and Technology, Ghana. E-mail: appiahnsiahfrank@gmail.com.

1 INTRODUCTION

StreamEPS[1] is an innovative open source Java Event Stream Processing(EPS) platform or system that provides an engine for processing segment oriented, temporal-oriented, state-oriented and spatial-oriented event stream. It is to help develop and maintain a large number of distributed computer applications that requires the processing of continuous unbounded streams of event data and timely processing of information. Examples of such systems includes a mobile payment system, an intrusion detection system, and applications performing on-line analysis of stock prices to identify trends, pattern matches and forecast future values. It provides support to other processing systems via processing element adapter, PE. It supports distributed, scalable, partially fault-tolerant properties. This event processing system is modeled on the event processing model depicted by [2], which has core base supporting the below features and feature future list:

- Classic event operators including conjunction, disjunction, negation.
- Modal event operators including always, sometimes.
- Event aggregation for count, avg, sum, min, max, mode.
- Pattern matching for classic event operators (logic, modal) and aggregation functions; avg, min, max etc.
- Event Filtering for event content-filter, event

header-filter and event type-filter using a data structure; comparison value-set, range value-set etc.

- Enrichment, projection and translation of events from stream.
- Temporal-based grouping including fixed and sliding interval, fixed and sliding event interval
- Spatial-based grouping including fixed location, entity distance location and event distance location will be supported.

The processing model of CEP's is based on the concept of event notifications using the pattern processing style either a pull-style or a push-style. In DSMS, standing queries are executed to provide answers to event data streams upon the arrival of events. According to [3], their paper describes an Information Flow

Processing (IFP) engine a tool capable of timely processing large amount of information as it flows from the peripheral to the center of the system. Conversely, a software system is said to be event-based if its parts interact primarily using event notification[4] to actuate events from external sources to detect patterns in the event

stream. A system designed with StreamEPS provides easy build, test and maintainable code structure for any project. An event-based system reduces the overall complexity of a system.

StreamEPS provides the traditional publish subscribe pattern with extended functionality for detecting the

occurrence or matching of complex pattern events as defined by CEP to increase the computational power of subscription languages.

In this paper, a thorough look of the overview of event processing systems, a conceptual and functional architecture of the StreamEPS. In the conceptual architecture, we will define a basic model for the event processing system engine and the functional architecture will provide the functional decisions made to address some issues in such systems.

2 OVERVIEW OF EVENT

PROCESSING SYSTEMS.

A general model for an event processing system (EPS) engine is shown in Figure 1. An event is an occurrence within a particular system or domain with an associated action. It is something that has happened, is eventually as having happened in that domain of an application or always happening in the domain of an application. The word event is also used to mean a programming entity that represents such an occurrence in a computing system. An event processing system is a general-purpose system and event queue and distribution mechanism. It maintains transaction logs for a comprehensive audit trail of event stream. At the processing server, the events are logged and queued and if necessary persisted. An event system basically has three central entities namely the standing queries or rules, producer and consumer. A producer normally pulls or pushes events received from a stream either asynchronously or not. The consumer receives events after the occurrence of an aggregation, filter or pattern event detection using basically the publish-subscriber model or the pull-push model used much in systems for processing events and this is adopted in the StreamEPS.

There are several forms of event[6] used in the design of StreamEPS which includes database event, primitive event, schedule event, encrypted event and stream event. Synchronously means that a subscriber, a receiver or a consumer explicitly fetches the message from the destination by calling a method. The method can block until a message arrives, or it can time out (if Time-to-Live (TTL) is set) if a message does not arrive within a specified time limit. Alternatively, an EPS may decide to implement a discard policy that tells it when to throw away events. On the other hand, asynchronously means: a subscriber need not wait for the delivery of the event. Whenever the event arrives, the server forwards it to the consumers that have registered for that event. The consumers do not have to wait for the delivery of the event. The measure of the strength of association established by a connection from one module(component) to another is greatly reduced by a factor. One can also associate the other pairwise of coupling, cohesion[5] which measures the degree of connectivity among the elements in the same-single module. Minimizing size and coupling of component is of utmost concern in the design of event processing systems. Using one or more Builders

to create all the classes of a system, you could load a great deal of coupling from the classes that do the heavy lifting in a system. Queuing and distribution are two separate operations. Multi-threading helps improve the performance of event processing systems. Notification services

defined in a model unlike events may implement policies that specify the order in which events are delivered to the receiver including:

- Any order
- First-In-First-Out
- Last-In-First-Out
- Deadline
- Business Priority.

3 THE ARCHITECTURE OF STREAMEPS.

StreamEPS is an event processing system made up of 8 major components namely the engine, receiver, EPS producer, forwarder, decider, event producer, event channels and the store as shown in Figure 2. The engine receives such events as input and processes them, as soon as they are available, according to a set of processing rules or standing queries, which specify how to filter, combine and aggregate different streams of event, event by event, to generate new subset of events, which represent the output of the engine for the Client Terminal. The engine for the event processing system supports queued events, asynchronous and synchronous dispatch of events via an indicator set by the developer.

The EPS engine is pre(post)-process aware that means that the event can pre-process before on received and post-processed before sending the event to the processing pipe defined by the engine builder. The queuing of events is simple. Whenever a new event comes in, it is indexed(map-indexed) into the queue or deque (called "deck") using its random ID generated with the event received time stamp and then put at the front of the queue.

After putting the event into the queue, the sequence count of the queue is reduced by a factor of 1 until 0, then the event queue is sent for further processing. Thus the events put into the queue are now available for distribution. The queue is swept/removed in order to distribute the event. Similarly, an event is canceled, the event is removed from the queue. The canceling of the event depends on its cancellation policy. The expired events are removed when the queue is swept after the time elapse. Notifications are basically signals sent from one part to another, in response to an event.

3.1 Functional Architecture

An event channel is a proxy to the event producer of the client and client output terminal which comprises of the a list of channel input terminals and channel output terminals. Events delivered to the engine receiver is based

on a meta-data/meta-content or a defined predicate expressions which is used as a set of criteria. Event notification unlike publish/subscribe style adheres to a contract/interface and uses the basic notion of an event-sink of event-based systems to request asynchronous delivery of specified events from an event source. The functional leverage of an advanced form of callbacks using both the pull-style model and push-style model is not overlooked in the software design of the StreamEPS. A specified two-callback constructs used in the design are the `onContextReceive` and `pushContextReceived`. A callback is a function that is called when ever or where ever a new event is received from an event producer. The push model allows the event producer to source events from the external world and push them into the event channel which is delivered to interested and registered client terminals or consumers. A parameterized interface is an important design pattern in the requirement of different context partitions because of the need of context substitution for the engine processing. The context partitions employed in the design of the StreamEPS are segment-oriented, temporal oriented, state-oriented and spatial-oriented[7]. The consumer here is client output terminals which can request for an event from a stream event channel which in turn will request the event from the client event producer. In order to improve the performance of the execution of event streams in the system, both soft and weak references are used as provided by Java in addition to the reference queue. The reference queue is used to create a situation where or when a certain event object or set of event objects not reachable only via reference notification. The soft references are used to create a fine-grained cache of event collections that are not known if advance whether it will be needed or re-used. Conversely, in the design of StreamEPS, weak references are used to create a reference to an immutable or not changing major component of the event processing system engine.

Several number of patterns were used for the inter-communication in the functional design of the engine for the StreamEPS, notably are the flyweight, interpreter and chain of responsibility. The flyweight pattern which is employed has the potential to increase performance but is sometimes counter intuitive to Object-Oriented design.

Chain of responsibility reduces communication and connections between objects. Within the design of the StreamEPS, using abstraction was to avoid "stove pipe"; which means event data is not tightly coupled to each application.

4. FIGURES

Figure 1: A general model of event processing system engine

Figure 2: The conceptual architecture for StreamEPS.

5. CONCLUSION

This paper provided an introduction to the functional components in the java open source event processing system, StreamEPS including the aggregate, filter, combine, extractor etc. An overview of a general event processing engine with reference to StreamEPS. There was a discussion on the number of notification services model that is employable in the design of an event processing engine. Of common use is the First-In First-Out, queue used to implement policies that specify the order of received of events in the engine. Finally, the architecture of

StreamEPS, which the functional components made up of 8 software components figured in the conceptual architecture was discussed. There was a mention of the two forms of callbacks used in the pull-style model and push-style model. The other important point to make in the software design and architecture of StreamEPS is the use of the design pattern namely the flyweight, interpreter and chain of responsibility for performance and flexible implementation.

6. REFERENCES

- [1] StreamEPS, The Stream Event Processing System. (Accessed: December, 2011) <http://github.com/fanhubgt/StreamEPS>.
- [2] Ophzer Etzion and Peter Niblett, Event Processing In Action. Manning Publications Co. 2011.
- [3] Gianpaolo Cugola and Alessandro Margara. Processing Flows of Information: From Data Stream to Complex Event Processing. Dip. di Elettronica e Informazione Politecnico Di Milano, Italy. ACM Journal Name, Vol. V, No. N, Month 20YY, Pages 1--70.

- [4] Ted Faison. Event--Based Programming Taking Events to the Limit. Apress 2006.
- [5] Wayne Stevens, Glenford J. Myers, and Larry Constantine, Structure Design, IBM Systems Journal, May 1974.
- [6] Ajit K. Patankar and Arie Segev. An Architecture and Construction of a Business Event Manager.
- [7] Opher Etzion and Nir Zolotorvesky. Spatial Perspectives in Event Processing. IBM Haifa Research Lab, Haifa University, Haifa, Israel. Springer-Verlag. Buchmann Festschrift, LNCS 6462, pp. 85--107, 2010.

Frank Appiah He is a holder of Bsc(Hon) from Kwame Nkrumah University of Science and Technology in 2018, Msc in Advanced Software Engineering from King's college London in 2010 and PhD in computer science and engineering from both KCL (2012/2014) and KNUST (2014) respectively. Frank Appiah has professional certificates in Management and engineering since 2011. He developed StreamEPS - Stream Event Processing System in 2011 which is

hosted at Github. This work is a PhD thesis at KNUST, Department of Computer Engineering, Kumasi, Ghana.