Journal of Theoretical and Applied Information Technology
10th February 2013. Vol. 48 No.1
© 2005 - 2013 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A FINANCIAL SERVICES CASE STUDY OF SOA BASED ON CEP

[1] **XIANGSHENG KONG**
[1]Department of Computer & Information, Xin Xiang University, Xin Xiang, P.R.China
E-mail: [1] fallsoft@163.com

ABSTRACT

With the growing amount of information in various domains, real-time retrieval and analysis is the most frequently used operation. The integration of service-oriented architecture (SOA) and Event-Driven Architecture (EDA) known as event-driven SOA (ED-SOA) has been widely adopted for developing various kinds of real-time applications. We describe a generic approach with an ability to perform complex event publishing, querying, analysis, processing and integration with SOA. Finally, a financial services case of SOA based on complex event processing (CEP) is provided in this paper. Our study shows that it is beneficial not only in reducing the development complexity, but also in coping with dynamic changes at all abstraction levels.

Keywords: *SOA; EDA; CEP; Sliding Windows; EPL*

1. INTRODUCTION

SOA is a buzzword and topic for many discussions in nearly every professional journal and conference. SOA is used to describe how distributed services can be reached by middleware[1]. SOA is mainly used to deploy business processes, but it is appropriate to connect every kind of application at the data layer. In a SOA architecture, there is no need to share the whole databases of each company. Background data and working logic are hidden from service requesters. The SOA private layer secures each of the companies' secret data, but makes it possible to create collaborative business processes to extract data from the enterprise application systems [2].

However, a SOA architecture doesn't address all the capabilities needed to respond to the dynamics of real-time business. As enterprises strive to cut costs and improve their responsiveness to customers, suppliers and the world at large, the concept of event-driven design is becoming more widely used. CEP is a relatively new technology for processing and analyzing multiple events from distributed sources, with the objective of extracting useful information from them[3].

Imagine the situation where a real estate broker shows her client a house for sale, matching the preference profile provided by the potential buyer. While the buyer likes the general location of the house, he considers it is unacceptable due to the unexpected traffic noise from a nearby street. Instead of proceeding with the original plan to show another house on the same street, an experienced broker should adjust the plan in light of this additional constraint. The broker connects to the multiple listing service with her mobile device and downloads a newly listed house within minutes of the current location that better satisfies the clients' requirements[4]. He may not be if his SOA implementation doesn't support event processing. Such a scenario can facilitate the introduction of events into SOA.

There are two distinct interactions between SOA and CEP. In the first interaction, the occurrence of an event can trigger the invocation of one or many services. Those services may perform simple functions, or entire business processes. This interaction between events and services is commonly referred to as event-driven SOA. We describe this as a style of SOA. In the second interaction, a service may generate an event. The event may signify a problem or an impending problem, an opportunity, a threshold, or a deviation. Upon generation, the event is immediately disseminated to all interesting parties (human or automated). The interesting parties evaluate the event, and optionally take action[5]. The event-driven action may include the invocation of a service, the triggering of a business process, and/or further information publication/syndication. In this interaction, the service is purely one of many event sources in a broader event-driven architecture.

Currently, the integration of SOA and EDA is known as event-driven SOA (ED-SOA) or SOA 2.0, an extension of SOA to respond to events that occur as a result of business processes. SOA 2.0 will ensure that services do not only exchange messages between them, but also publish events and receive event notifications from others. For this purpose, an Enterprise Service Bus (ESB) will be necessary to process, enrich and route messages between services of different applications[6]. Thus, combining the use of CEP and SOA, we may detect relevant events in complex and heterogeneous systems, i.e., CEP will let us analyze and correlate events in real time SOA 2.0.

2. CEP BACKGROUND

Events can be divided into two types: atomic events and complex events. An atomic event which is defined to be an instantaneous and atomic (happens completely or not at all) occurrence of interest at a point in time represents something that occurs, happens or changes the current state of affairs. For example, an event may signify a problem or an impending problem, a threshold, an opportunity, information becoming available, a deviation etc. Events can be grouped into an event type that give the metadata for events that belong to the same class and include the attributes of these events, and an event type is expressed by an event expression. These attributes can carry information which can be used when a complex event occurs (at a later time) about the action that caused the event to occur.

A complex event often happens in a continuous time interval, which is assigned by users (called case 1) or abstract directly from atomic events[7].For example, "Mary is getting coffee" can be extracted from a series of atomic events "Mary is in her office", "Mary is in coffee room", "Mary is in her office", and so on. Complex events are generated by composing atomic or other complex events using a set of event detection operators.

An event stream is defined to be a linearly ordered (often by time) sequence of events forms Event Stream. An event cloud is defined to be a partially ordered set of events form Event Cloud.

For example,seting of all stock trades for BIDU within a 5 minute time window is an Event Stream. While all Stocks sold in a business day is an Event Cloud. Event Cloud enables users to search for business events and patterns of business events within a repository for historical events. We consider this repository as a "cloud of events", which is used for searching and analyzing purposes (shown in Figure 1)[8].

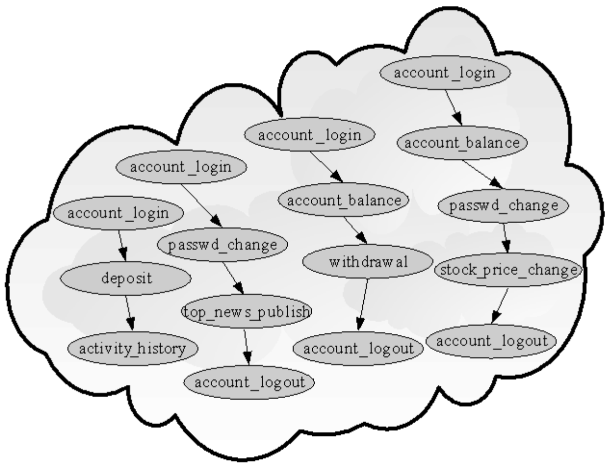


Figure 1. Event Stream & Event Cloud

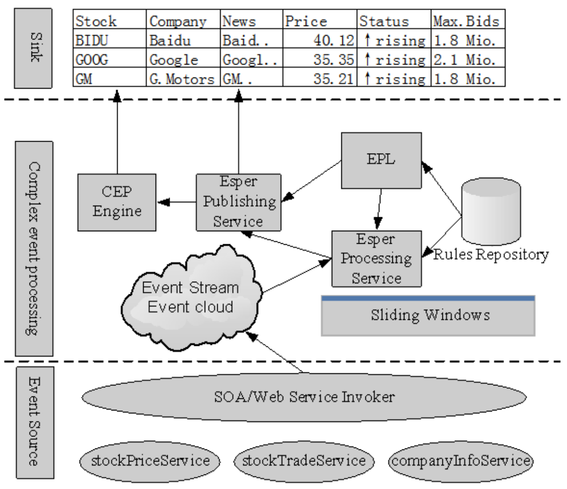
CEP uses relations between multiple events to derive higher level information using event patterns that are executed upon all available events (the event cloud)[9]. The pattern matching can be seen as inverted database queries: instead of performing different queries on a static dataset CEP performs static queries on dynamic data in order to perform matching on new events as quickly as possible.

Event Stream Processing (ESP) focuses on processing event objects that are ordered in time. These event objects are received in a data stream which can be of infinite size. Therefore specialized techniques like data views that allow only a certain length of the event stream to be subject to queries (like time views and length views) have been developed. Because these time-ordered data streams are members of the whole event cloud, ESP can be seen as a subset of CEP. An event processor is an application that performs operations on event objects, including creating, reading, transforming, aggregating, correlating or removing them.

Esper is a component for CEP and ESP applications. There are two implementations of Esper, Esper for Java and NEsper for .NET. Both supply an API to access the engine features, such as deploying queries, sending events into the engine and retrieving events out of the engine, in their respective language. Esper and NEsper enable rapid development of applications that process large volumes of incoming messages or events. Esper and NEsper filter and analyze events in various ways, and respond to conditions of interest in real-time. For Esper, events can be instances of java.util.Map, org.w3c.dom.Node (Java representations of XML documents), or other POJO.

3. A FINANCIAL SERVICES CASE STUDY OF SOA BASED ON CEP

We consider a scenario from the financial computing domain, in which Web services provide live data about companies and stock prices. The aim is to combine the information in an XML document that is actively updated when the underlying data change. Figure 2 illustrates, on a high level, how data and events are received and processed.



Stock	Company	News	Price	Status	Max.Bids
BIDU	Baidu	Baid..	40.12	↑rising	1.8 Mio.
GOOG	Google	Googl..	35.35	↑rising	2.1 Mio.
GM	G.Motors	GM..	35.21	↑rising	1.8 Mio.

Figure 2. A Financial Services Case Study Of SOA Based On CEP

(1) Sliding Windows

Stream of events that arrives to CEP engine is by definition infinite. Performing any kind of group operations on such data structure is impossible. Sliding windows solve this problem by selecting a limited subset of incoming events. The criteria by which events are put into windows can be divided into two categories (shown in Figure 3).

Time-based — events are selected by the time of their creation (arrival to CEP engine). For example, a window may contain only events from last 3 minutes. If at some point any event turns out to be older than specified time period is removed. Simultaneously, new events that have just arrived are putting into window. Therefore, time window "slides" through event steam with regard to time.

Size-based — windows that have limited event capacity. If window is full and new event arrives, the oldest of events is removed to make room for the new one. Thus, a window slides with regard to size.

Each window can be additionally filtered by arbitrary condition. For example, one may want to have only events from last 4 minutes that come from producer with given name.

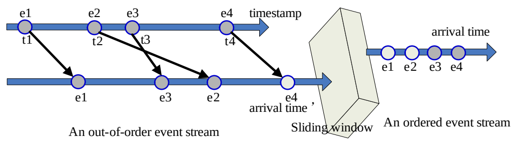


Figure 3. Sliding Windows Principal

(2) Esper Publishing Service

The Esper publishing service will receive an asynchronous notification. Then, the publishing service will register this type into the Esper runtime engine and retrieve a sender for this type to be used[10]. Analogously, whenever the Esper publishing service receives an unregistered notification from the registry, it will unregister the type from the Esper runtime engine and remove the sender for this type, so it cannot be used anymore for publishing. Figure 4 shows a UML sequence diagram for publishing events using Esper. Whenever an event is published, the Esper publishing service will first find a suitable sender for the event's type. Then, the sender will be used to send the event into the appropriate event stream.

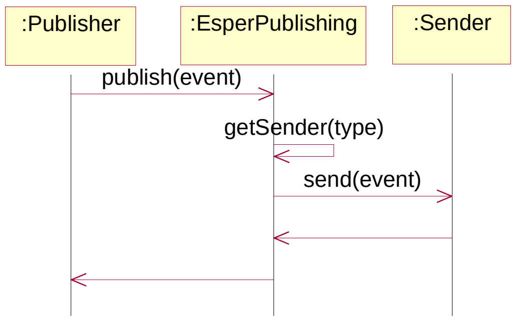


Figure 4. Esper Publishing Service Sequence Diagram

(3) Esper Event Programming Language (EPL)

EPL processing by executing continuous queries on event streams is used to define complex events, similar to the concepts known from active databases[11]. These query languages execute operations similar to SQL, including:

SELECT (Select event types, attributes of an event in the event stream)

WHERE (Define conditions for the events that should fulfill the query)

AGGREGATION (Min, Max and other aggregations known from SQL are available)

JOIN (Similar to our definition of event correlation, events can be joined via their attributes)

TIMEWINDOW (The queries are executed against the events in a specific sliding time window)

For example:

```
select avg(price)

from StockTickEvent.win:time(300)

where StockTickEvent.symbol='IBM';

select symbol, avg(price) as averagePrice

from StockTickEvent.win:length(100)

group by symbol;
```

The first query returns the average price of all IBM stock tick event within the last 300 seconds (with a sliding time window). The second query returns the average price per symbol for the last 100 stock ticks.

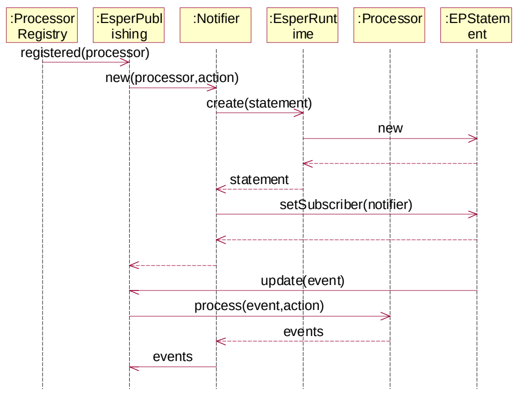


Figure 5. Esper Processing Service Sequence Diagram

(4) Esper Processing Service

Esper Processing service intercepts events that require processing from event streams and delivering them to appropriate processor. Thus, it will create an EPL SELECT statement to select context events that require processing from event streams[12]. Figure 5 shows a UML sequence diagram for delivering events to processor using EPL SELECT statements and notifies.

4. CONCLUSIONS

SOA and event processing are both required for an optimized business and, when combined, can create extreme value to business operations. SOA and events need each other. SOA can profit from events when it comes time to build an actual event-driven application, large or small. In fact, SOA services can be used in an event-driven application at practically every functional step in the architecture.CEP extracts and creates value by identifying threats and opportunities from distributed enterprise events.

REFRENCES:

[1] D. Schilberg, A. Gramatke, K. Henning. "Semantic Interconnection of Distributed Numerical Simulations Via SOA", Proceedings World Congress on Engineering and Computer Science 2008, vol. 5, pp. 894-897, 2008.

[2] Martinek P., Szikora B. "Detecting semantically related concepts in a SOA integration scenario", pp. 117-125, 2008.

[3] M. R. N. Mendes, P. Bizarro, and P. Marques. "A framework for performance evaluation of complex event processing systems", pp. 313-316, 2008.

[4] H.Wang, X.; Gu, T.; Zhang, D. Q.; and Pung. "Ontology based context modeling and reasoning using owl", Proceedings of Workshop on Context Modelling and Reasoning(CoMoRea'04), pp. 18–22, 2004.

[5] B. Michelson, "Event-driven architecture overview", Patricia Seybold Group, 2006.

[6] Juan Boubeta-Puig, Guadalupe Ortiz, and Inmaculada Medina-Bulo. "An Approach of Early Disease Detection using CEP and SOA", SERVICE COMPUTATION 2011: The Third International Conferences on Advanced Service Computing, pp. 143-148, 2011.

[7] Chunjie Zhou and Xiaofeng Meng. "Complex Event Detection in Pervasive Computing", The Third SIGMOD PhD Workshop on Innovative Database Research (IDAR2009), pp. 1-6, 2009.

[8] Szabolcs Rozsnyai, Roland Vecera, Josef Schiefer and Alexander Schatten. "Event Cloud - Searching for Correlated Business Events", Proceedings of the 9th IEEE International Conference on ECommerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007), pp. 409–420, 2007.

[9] Thomas Everding and Johannes Echterhoff. "Event Processing in Sensor Webs", Proceedings of Geoinformatik 2009, vol. 35, pp. 11–19, 2009.

[10] ChenWang, Martin de Groot, and Peter Marendy. "A service-oriented system for optimizing residential energy use", 7th International Conference on Web Services, pp. 735–742, 2009.

[11] S. Rozsnyai. "Efficient indexing and searching in correlated business event streams", Master's thesis, Technical University Vienna, 2006.

[12] Jay Budzik and Kristian J. Hammond. "User interactions with everyday applications as context for just-in-time information access", 5th International Conference on Intelligent User Interfaces, pp. 44–51, 2000.