Chapter 13 – Complex Event Processing

Matt McClelland

1.0 – Summary

1.1 - Introduction

In today’s world, business enterprises are quickly becoming more and more complex. Different processes take place all over the world and events are flying through the enterprise IT systems. These systems have grown from standalone applications that were able to handle a certain aspect within an enterprise to an enterprise wide IT system that provides a coupling between the different IT applications.

These enterprise wide IT systems are widespread across large enterprises and generate many events that flow through each of the enterprise system layers. These events then feed other applications and/or services which in turn generate new events. Most events that occur in a business enterprise system are simple events that can easily be traced and monitored, but the more complex events - which usually consist of multiple, unrelated simple events - are hard to keep track off. Thus, to tackle this issue and make complex events more meaningful, a new type of event processing is introduced: complex event processing.

1.2 – Complex Event Processing

Complex Event Processing (CEP) is primarily an event processing concept that deals with the task of combining data from multiple sources to infer events or patterns that suggest more complicated circumstances. The goal of Complex Event Processing is to identify meaningful events (such as opportunities or threats) and respond to them as quickly as possible. CEP employs techniques such as detection of complex patterns of many events, event correlation and abstraction, event hierarchies, and relationships between events such as causality, membership, and timing, and event-driven processes [2].

The thought behind Complex Event Processing is based on the observation that in many cases actions are triggered not by a single event, but by a complex arrangement of events, happening at different times, and within different contexts. It is primarily used to predict high-level events likely to result from specific sets of low-level factors and is also used to identify and analyze cause-and-effect relationships among events in real time, allowing personnel to proactively take effective actions in response to specific scenarios [1].

2.0 – Use Cases/Purposes:

As you may already know, Complex Event Processing is quickly becoming one of the most popular emerging technologies in the IT world and is more and more frequently being used by various businesses for building and maintaining complex information systems such as the following:

- Business Activity Monitoring (BAM)
- Business Process Management (BPM)
- Enterprise Application Integration (EAI)
- Event-Driven Architectures (EDA)
In the next few sections, we will explore and briefly introduce each of the above systems and explain what role Complex Event Processing is currently fulfilling within each.

2.1 – Business Activity Monitoring

Business Activity Monitoring (BAM), also called business activity management, is the use of technology to proactively define and analyze critical opportunities and risks in an enterprise to maximize profitability and optimize efficiency [7]. It is most often found in the form of supportive tools that give insight into the business performance and can also help in finding possible bottlenecks. BAM consists of three main steps: collecting data, processing data, and displaying the results. Complex Event Processing is a very welcome addition to a BAM interface because of its ability to detect complex situations that occur in a large enterprise, and thus can help populate BAM reports and dashboards with even more complex and useful information, thus giving the business a deeper understanding and better perspective of what is truly going on within their enterprise.

Given in Figure 2.1 below is an example of a BAM Dashboard supplied by Oracle [3]:

![Figure 2.1: Oracle BAM Dashboard [3]](image)

2.2 – Business Process Management

Business Process Management (BPM) is a systematic approach to improving an organization’s business processes that intersects the fields of both Business Management and Information Technology [7]. BPM activities seek to make business processes more effective, more efficient, and more capable of adapting to an ever-changing environment. BPM is all about business processes that, among others, consist of organizations, humans, and systems. Most BPM’s consist of at least the following three phases: process design, execution, and monitoring [7]. While business management field provides the knowledge to design the business processes, the IT field provides the technology to execute them. Complex Event Processing can aid BPM’s
by detecting inefficiencies in their design and workflow, which in turn results in better processing and decision making across the enterprise.

The BPM model on the next page shows how business processes cut across organizational and system boundaries [4]:

![Business Processes across Product Divisions and Systems](image)

**Figure 2.2:** Business Processes across Product Divisions and Systems [4]

### 2.3 – Enterprise Application Integration

Enterprise Application Integration (EAI) is a business computing term for the plans, methods, and tools aimed at modernizing, consolidating, and coordinating the computer applications in an enterprise. Today’s enterprises already have many different types of applications, including: CRM (Customer Relationship Management), SCM (Supply Chain Management) and BI (Business intelligence) applications [7]. Much information and knowledge is stored in these systems and a lot of money has been spent on them. EAI can be seen as a method to link these legacy applications and combine them with new applications. With EAI, the data in different systems can also be kept consistent. A model of an EAI is shown for clarification in Figure 2.3 on the next page:
2.4 – Event-Driven Architectures

Event-Driven Architecture (EDA) is a software infrastructure that by nature is very loosely coupled. The main idea behind EDA’s is that a large software system consists of many small components that all have their own functionality. The communication between the components is done using events, which under these circumstances can be seen as a notification, which tells the rest of the components when a certain ‘job’ is done [7]. Because events are very important within an Event-Driven Architecture also the handling and routing of events is very important. Complex Event Processing is a very powerful addition to EDA, because it has the ability to detect complex situations in real-time. A model of an EDA is given below in Figure 2.4:
3.0 – Architectures

There are several different architectures that arise in Complex Event Processing. As you will see, each architecture has its own unique component to add to the basic CEP Process seen in Figure 3.1 below; however, the manner in which this is done varies greatly. In this section, we will identify the kinds of variations you can expect to see and present a number of well known architectures, each of which attempts to address a common business challenge.

The core CEP process (Figure 3.1) usually follows the same design as outlined below. Some event is sensed, analyzed in the context of some reference data to determine whether something of business interest has occurred, and some decision is made about what the nature of the response should be [8]. However, despite the fact that the core process is always the same, there are many different architectures that seem to arise for Complex Event Processing.

![Figure 3.1: Core CEP Process [8]](image)

In the next few sections, we will go into detail about a few of the most widely used and well known CEP architectures as well as the business problems each attempts to solve.

3.1 – Condition Detection

The most basic architecture you will encounter in CEP Architectures is the Threshold Detection Model (Figure 3.2). In this pattern, a component performs some form of an observable action, which then either is or is not triggered as an event. If an event is triggered, then the threshold detection component compares a value conveyed by the event to a threshold value and if the event value exceeds the threshold value, a business event is generated announcing this condition. A model of the Threshold Detection Architecture is shown for clarification in Figure 3.2 below:
Figure 3.2: Threshold Detection Model [8]

When using the above model, the location of the threshold value must also be taken into account. One alternative is to permanently fix the threshold value in the analysis component. Another option is to make it a piece of the contextual information that is looked up by the condition detector component, either when it starts or each time an event is triggered [8].

The more general form of the Threshold Detection Architecture is the Condition Detection Architecture shown in Figure 3.3. In Condition Detection Architecture, the detected condition is defined by a number of values that define the boundaries of the condition being recognized, such as if the triggered event takes place at a certain time or place. The information considered in this analysis is usually a combination of event and contextual data. If the condition is detected, then a business event is generated announcing the existence of the condition [8]. An example of the Condition Detection Architecture is provided in Figure 3.3 on the next page:
When using the Condition Detection Architecture, the sources of the parameters defining the boundary conditions and the contextual data required to detect the condition must also be considered, along with the possible need to change some of these values at runtime. The design effort required to provide access to information originating in other systems and make it efficiently available is often a major challenge for CEP Architectures.

One thing that should be noted for the Condition Detection Architecture is that the reference data being used is not modified by the processing of events and therefore does not reflect prior history. The only state information being used is the information found at the time an event was triggered. Although this makes the condition detector stateless, and therefore easy to scale, it does not account for conditions in which prior events may be needed.

### 3.2 – Situation Recognition

The Situation Recognition Architecture on the surface looks a lot like the Condition Detection Architecture; however, there is one major difference to note. In the Situation Recognition Architecture, the context data used to identify a situation when the triggering event arrives now contains historical information about previously processed events [8]. Many of the triggering events that arrive do not result in a business event, but their occurrence results in the modification of the context data which in turn helps provide the context for each of the subsequent events that arrive. Provided in Figure 3.4 below is an example of a Situation Recognition Architecture:

![Figure 3.4: Situation Recognition Architecture](image)

### 3.3 – Track and Trace

The Track-and-Trace Architecture (Figure 3.5) is a special case of the Situation Recognition Architecture. The most notable difference between these two architectures is that the Track-and-Trace Architecture includes a model of the expected process and the state of an
existing instance of the process. If the triggering event marks the beginning of a new event execution, an initial process state is created for the event. For other events, information in the event is used to locate the state of the process already being executed (there may be many instances of the process being executed at any given point in time). Once the current state has been identified, the process model is then used to interpret the triggering event in the context of that state [8].

This type of analysis is appropriate for monitoring any type of unmanaged process, for example, tracking a package from an initial location to a destination. Tracking your luggage from the time you drop it off until the time you pick it up at the baggage carousel at your final destination is another.

In general, this approach works well for monitoring any process in which there is a hand-off of responsibility from one participant to another. You give your luggage to the counter agent, who then hands the bag over to the conveyor as a means of handing off responsibility to the baggage handlers. The process continues until the final hand-off, which begins when the baggage handler at your final destination places the bag on the conveyor leading to the baggage carousel and ends when you pick up your luggage.

The challenge most Track-and-Trace Architectures face is finding appropriate evidence of progress for keeping track of the progress state. In some circumstances, it may simply not be possible to find the information needed to track an event’s lifespan. When this occurs, you may want to implement the degree of tracking that is supported by the available evidence and begin...
an initiative that will eventually provide more detailed evidence of progress [8]. In the next section, we will introduce one such architecture that looks to aid in this pursuit.

### 3.4 – Business Process Timeliness Monitor

The Business Process Timeliness Monitor (Figure 3.6) is an extension of the Track-and-Trace Architecture that looks to address the absence of an expected event within some period of time. While you can certainly apply this approach to recognizing that an overall process did not complete on time, the greatest benefit comes from recognizing that some intermediate event did not occur on time, and thus the overall process is in jeopardy of being late. The recognition can be used to trigger an action that will correct the course of the overall process and get it back on track for an on-time completion [8].

![Figure 3.6: Business Process Timeliness Monitor](image)

One thing to note about the Timeliness Monitor Architecture is that detecting the absence of an event requires the establishment of a service-level agreement specifying the maximum amount of time it should take for the process to complete or remain in each intermediate state. When the state machine monitoring the process is started or a particular intermediate state is entered, a timer is started. When the overall process completes, or the intermediate state is exited, the corresponding timer is stopped. However, if the timer expires before the process completes or the intermediate state is exited, a timeout event is generated indicating that some expected event did not occur [8].

In recognizing this situation, it is the expiration of the timer that serves as the trigger for the analysis. Some introspection of the state machine may be required to identify which events did not occur, but the larger design requirement is to determine which parties should be notified when this situation arises and what actions those parties are going to take to get the overall process back on track.
3.5 – Situational Response

All of the architectures that we have discussed up to this point have had one characteristic in common – they simply recognize that some condition exists and announce that fact with an event. However, in some situations there is an additional challenge in determining what the appropriate response ought to be and thus a need arises for a Situational Response Architecture (Figure 3.7).

In a Situational Response Architecture, further analysis is required, generally to focus the actions on achieving specific business objectives. Reference data, often containing historical information, is required for the analysis. The result of the analysis is generally one or more directives to actually perform the identified actions [8].

Consider the case in which there is some form of perishable product being sold: fresh produce and meat, seats on a plane, or hotel rooms—anything that becomes worthless if not sold by some point in time. The desired business strategy is to dynamically set the price of the product based on the remaining inventory and the time remaining before the product becomes worthless. The situation being responded to in these cases is the presence of a potential consumer for the perishable product [8].

One approach could be to track the rate at which the product is selling versus the cost of the product. Then, the offering price for the product could be adjusted dynamically, which in turn would require Complex Event Processing to do the dynamic price adjustments as consumers shop and as commodity inventories change [8].

3.6 – Decision as a Service

In the Decision-as-a-Service Architecture (Figure 3.8), the logic necessary to make a decision is factored into a separate component. The service consumer gathers all relevant current state input data for the decision and passes it to the service and the decision service computes the output data from the input data, which reflects the decision results. Given below in Figure 3.8 is one such example of what a Decision-as-a-Service Architecture would contain:
The value of the Decision-as-a-Service Architecture is that it simplifies the maintenance of both the consumer and decision service. In particular, it allows the implementation of the service – or the business rules – to be updated without requiring a modification to the service consumer [8].

4.0 – Conclusion

Complex Event Processing has an enormous impact on how companies can make strategic use of Big Data. Using the architectures highlighted in this chapter, companies are able to process and analyze data in real time and gain immediate insights, whereas in the past these key findings may have never been noticed. With CEP approaches, companies can stream data and leverage a business process engine to apply business rules to the results of that streaming data analysis immediately. The opportunities to gain insights that can lead to new innovations and new solutions that may otherwise have never been possible is the foundational value that Complex Event Processing brings to table and through the CEP design approaches and architectures outlined throughout this chapter, these breakthroughs are now possible.
SOFTWARE ARCHITECTURES

References

[1]. SearchSOA: Complex event processing (CEP); Date Accessed: April 13, 2014; http://searchsoa.techtarget.com/definition/complex-event-processing


[6]. Event-Driven Architecture Overview; Date Accessed: April 20, 2014; http://www.elementallinks.com/2006/02/06/event-driven-architecture-overview/#sthash.T0dTMVSW.dpbs


[8]. Brown, Paul; Architecting Complex-Event Processing Solutions with TIBCO; Publisher: Addison-Wesley Professional; September 21, 2013