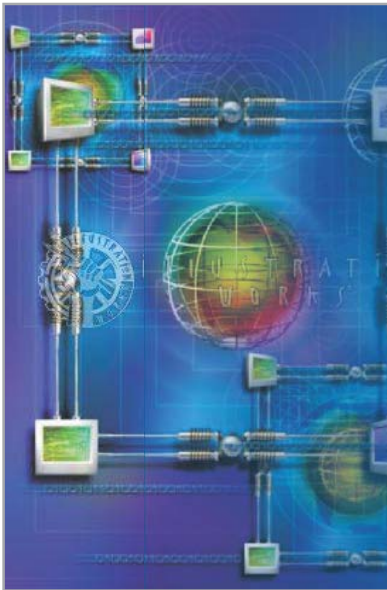




Think Beyond. Think Mphasis.

WHITE PAPER



Complex Event Processing

Lokhi Prasad Deori

Technical Architect, Integration Practice
Mphasis

August 2010

Abstract

Enterprises are now more complex than ever. A number of processes take place all over the world and events occur successively 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 spread across large enterprises and generate many events that flow through the enterprise system layers. The events feed other applications or services which in turn generate new events. Complexities such as these have resulted in event-clouds that hang within an enterprise. Due to these event-clouds, the event-flow of an enterprise IT system lacks transparency and is difficult to comprehend.

A new concept is arising that can address this issue: Complex Event Processing (CEP). With CEP it is possible to correlate events and detect complex situations. Beginning with a short introductory chapter; chapter 2 introduces a general CEP language with other technologies. Chapter 3 introduces commonly used CEP design patterns and chapter 4 talks about TIBCO Business Events, which implements Complex Event Processing.

Table of Contents

1 Introduction	4
1.1 Overview	4
1.2 Glossary	4
1.3 Web References	4
2 Context	5
2.1 Purpose of CEP	5
2.2 CEP and Event-Driven Architectures	5
2.3 CEP and Enterprise Service Bus	5
2.4 CEP and Business Process Management	6
2.5 CEP and Business Activity Monitoring	6
2.6 CEP and Service Oriented Architecture	6
2.7 Enterprise system layers	7
3 Commonly Used Design Patterns	7
3.1 Filtering	7
3.2 In-memory caching	8
3.3 Aggregation over windows	8
3.4 Database lookups	9
3.5 Database Writes	9
3.6 Correlation (Joins)	9
3.7 Event pattern matching	10
3.8 State machines	10
3.9 Hierarchical Events	11
3.10 Dynamic Queries	11
4 TIBCO Business Events	12
4.1 Overview	12
4.2 Capturing Events	12
4.3 Rule Based Engine	12
4.4 Event Pattern Matching	12
4.5 Static Modelling	13
4.6 Dynamic Modelling – State Machines	13
4.7 Database Writes and Lookups	13
4.8 Query Windows	14
4.9 Correlation of Events	14
4.10 Object Management	14
4.11 Dynamic Query of Events	15
4.12 Distributed Cache	15
5 Conclusions	15
5.1 Key capabilities	15
5.2 Benefits	15
5.3 Recommendations	16

Index of Figures

Figure 1: Event-Driven Architecture vs. Complex Event Processing.....	5
Figure 2: CEP, EDA and SOA.....	5
Figure 3: CEP with ESB.....	5
Figure 4: CEP with BAM.....	6
Figure 5: SOA.....	6
Figure 6: CEP & SOA.....	7
Figure 7: Typical enterprise system layers.....	7
Figure 8: Filtering.....	7
Figure 9: Event Window Caching.....	8
Figure 10: Aggregation over windows.....	8
Figure 11: Database lookups.....	9
Figure 12: Database Writes.....	9
Figure 13: Correlation (Joins).....	9
Figure 14: Event pattern matching.....	10
Figure 15: State machines.....	10
Figure 16: Hierarchical Events.....	11
Figure 17: Dynamic Queries.....	11
Figure 18: Channels and Destinations.....	12
Figure 19: Rule Editor.....	12
Figure 20: Run to completion Cycle.....	13
Figure 21: Concept Relationships.....	13
Figure 22: State Machines.....	13
Figure 23: Explicit Windows.....	14
Figure 24: Object Management.....	15
Figure 25: Query Structure.....	15
Figure 26: Distributed Cache.....	15

1 Introduction

1.1 Overview

The concept of complex event processing evolved in the 90s (1989 – 95), and the term was coined by Prof. David Luckham (<http://complexevents.com>)


“Complex event processing is a technology for extracting information from message-based systems.”

A complex event processing (CEP) system enables organizations to process distributed business events and identify opportunities or threats. Business events may be tracked individually, such as a stream of stock trades, or correlated with other events, producing derived or “complex” events often called “situations.”

CEP is a technology for low-latency filtering, correlating, aggregating, and computing on real-world event data. It is an emerging network technology that creates actionable, situational knowledge from distributed message-based systems, databases and applications in real-time or near real-time.

Complex Event Processing software allows you to process and analyze multiple streams of high-volume, high-speed business and system events, and to uncover opportunities and threats as they happen – not after the fact. It can be applied to extracting and analyzing information from any kind of distributed message-based system.

1.2 Glossary

Name	Description	Name	Description
BAM	Business Activity Monitoring	SQL	Structured Query Language
BE	TIBCO BusinessEvents	TTL	Time To Live
BI	Business intelligence	XML	Extensible Mark-up Language
BPM	Business Process Management		Complex Event Processing Engine
BPMG	Business Process Management Group		In-Memory Cache with Windows
BW	TIBCO BusinessWorks		Incoming, Outgoing, or Intermediate Event Streams
CEP	Complex Event Processing		A Continuous query, registered with the process
CPU	Central Processing Unit		A relational database
EDA	Event-Driven Architecture		
EQL	Event Query Language		
DSL	Domain Specific Language		
ESB	Enterprise Service Bus		
ERP	Enterprise Resource Planning		
ESP	Event Stream Processing		
FIFO	First In First Out		
JVM	Java Virtual Machine		
KPI	Key Performance Indicators		
OM	Object Management		
QL	Query Language		
PUB / SUB	Publish / Subscribe		
RMS	Rules Management Server		
RTC	Run to Completion Cycle		
SOA	Service-Oriented Architecture		

1.3 Web References

- <http://www.coral8.com/developers>
- http://en.wikipedia.org/wiki/Rete_algorithm
- <http://www.ibm.com/developerworks/webservices/library/w-ovr/>
- <http://www.tibcommunity.com/message>
- <http://www.tibcommunity.com/communities.jspa>
- <http://www.tibco.com/software/complex-event-processing/businessevents/default.jsp>
- <http://www.tibco.com>
- <http://www.uml.org>

2 Context

2.1 Purpose of CEP

CEP can be used to serve many purposes. The most relevant purposes of CEP are given below with a brief introduction:

- Event Driven Architectures
- Enterprise Application Integration
- Business Process Management, and
- Business Activity Monitoring

2.2 CEP and Event-Driven Architectures

Event-Driven Architecture (EDA) is a software infrastructure that by nature is very loosely coupled. The main idea behind EDA is that a large software system consists of many small components that have their own unique functions. The communication between the components is done through events. An event can be seen as a notification, which tells other components that a certain business event has occurred. As events are very important within an Event-Driven Architecture, the handling and routing of these events is critical.

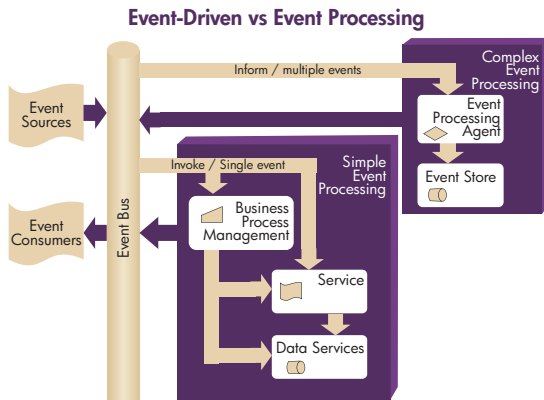


Figure 1 : Event-Driven Architecture vs. Complex Event Processing
[Figure source: TIBCOsummitBEVincent.pdf]

CEP correlates multiple messages within given time frames. EDA is an architectural approach to model information systems from a business event perspective. EDA differs from SOA by its focus. CEP is a technique to process message streams. These messages do not need to represent business events. A business event is something that happens (change of state) when your business has planned to react in a pre-defined manner. A business event is represented by a message, but not all messages are representations of business events. CEP is about messages, EDA is about business events.

EDA is CEP at the business level. The business can be seen as a complex event processor which holds states, reacts on state changes, and correlates business events. CEP can be used to implement EDA; and is a powerful addition to EDA, as it can detect complex situations in real-time. SOA puts services at the centre of the model and EDA does so with business events. The SOA-approach tends to result in a synchronous communication style and the EDA-approach in an asynchronous communication style.

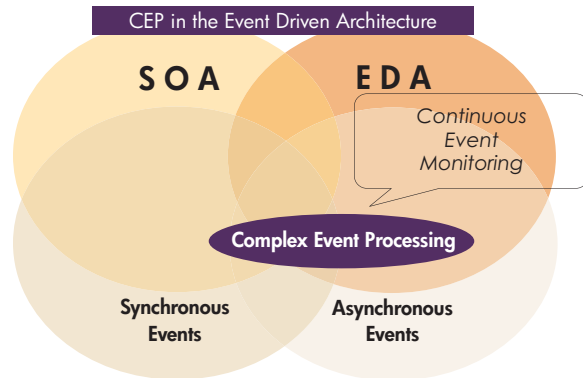


Figure 2 : CEP, EDA and SOA
[Figure source: TIBCOsummitBEVincent.pdf]

2.3 CEP and Enterprise Service Bus

To integrate old and new, Service-Oriented Architecture (SOA) needs an infrastructure that can connect any IT resource, irrespective of its technology or where it is deployed. To be flexible, it needs an infrastructure that can easily combine and re-assemble services to meet changing requirements without disruption. And to be dependable, it needs an infrastructure that is robust and secure. The preferred solution for this infrastructure is the Enterprise Service Bus (ESB).

An ESB is a software infrastructure that simplifies the integration and flexible re-use of business components within a Service-Oriented Architecture. An ESB provides a dependable and scalable infrastructure that connects disparate applications and IT resources, mediates their incompatibilities, orchestrates their interactions, and makes them broadly available as services for additional uses.

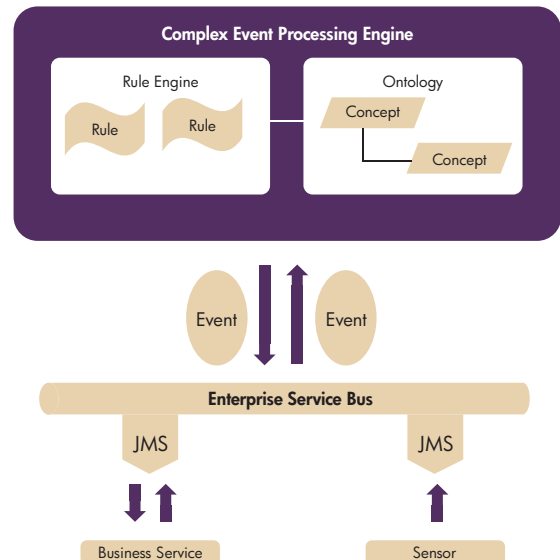


Figure 3: CEP with ESB

ESB provides the communication backbone to federate event-driven services in a loosely coupled fashion as shown in Figure 3. The CEP engine continuously evaluates changes in the state against patterns of interest that impact the business. However, CEP is unlike conventional event processing technologies, in that it treats all events as potentially significant and records them asynchronously. Applications that are appropriate for CEP are event-driven, which implies some aspects of real-time behaviour. To be more specific, the typical CEP application area can be identified as having some aspects of 'situation awareness', 'sense and respond' or 'track and trace', aspects which overlap in actual business situations.

2.4 CEP and Business Process Management

Business process management (BPM) is the automation and coordination of the countless assets and tasks that make up your business processes. There are many types of business processes. They can involve people and IT systems, be internal or external to organizations, be easily modelled and repeatable or vary for each situation. In most business environments, they are constantly changing, based on unique business events and conditions.

The BPM technologies are used to define and coordinate long running, stateful, system-to-system integrations. They use Complex Event Processing technology to handle other stateful interactions, where there's a need for near real-time processing.

2.5 CEP and Business Activity Monitoring

Business Activity Monitoring (BAM) refers to the aggregation, analysis, and presentation of relevant and timely information about business activities across your extended enterprise. BAM provides more accurate information about the status and results of operations, processes, and transactions so you can make better decisions, quickly address problem areas, and reposition your organization to take full advantage of emerging opportunities.

It is a supportive tool to give insight in the business performance and can help find possible bottlenecks. BAM consists of three main steps: collecting data, processing data, and displaying the results. However, it has been limited to real-time dashboard applications. Basically, it displays visual real-time graphs and charts based on Key Performance Indicators (KPIs). It lacks Complex Event Processing (CEP) capabilities that businesses require. Dashboards are just a tiny segment of BAM. Current BAM alone cannot meet the business requirements. For example, dashboards show only snapshots of information. It does not indicate what transitions have taken place. In addition, managers do not monitor them at all times and hence vital events can be missed. It is possible that BAM combined with CEP is a better framework.

Complex Business Activity Monitoring with Event Processing (CAMEP) combines the two as a single framework. Rosella Analytic Platform is an integrated end-to-end developer environment which provides all the needed ingredients for CAMEP.

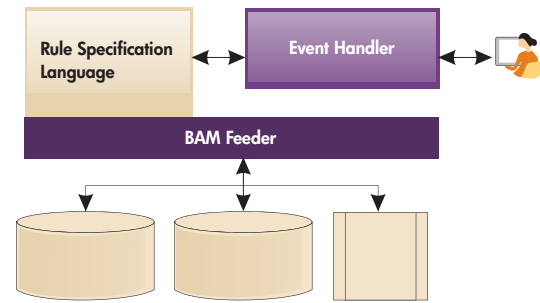


Figure 4 : CEP with BAM

Figure 4 shows the sample architecture of CAMEP. Event Handler: Extensible event handlers can meet any business requirement. The BAM feeder collects information from operational databases, data warehouses, data marts, and from other sources, and feeds it into Rule Specification Language. The Rule Specification Language evaluates rules and invokes event handlers to generate business events. Rules are developed by business users and analysts.

2.6 CEP and Service-Oriented Architecture

A Service-Oriented Architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. And hence some means of connecting services to each other is required.

SOA Collaborations: The following diagram illustrates the entities in a service-oriented architecture that collaborates to support the 'find, bind and invoke' paradigm.

Service Consumer: The service consumer is an application, a software module or another service that requires a service. It initiates the enquiry of the service in the registry, binds to the service over a transport, and executes the service function. The service consumer executes the service according to the interface contract.

Service Provider: The service provider is a network-addressable entity that accepts and executes requests from consumers. It publishes its services and interface contract to the service registry so that the service consumer can discover and access the service.

Service Registry: A service registry is the enabler for service discovery. It contains a repository of available services and allows for the lookup of service provider interfaces to interested service consumers.

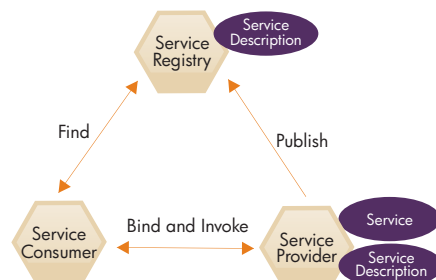


Figure 5 : SOA [Figure source: <http://www.ibm.com/developerworks/webservices/library/w-ovr/>]

SOA and Event Processing – Putting It All Together

Possibly, both SOA and Event processing may help an optimized business and when combined, can create extreme value to business operations. SOA and event processing also help expose business information that is otherwise locked in application silos and databases.

On a functional level, SOA and events need each other. SOA can benefit from events when it comes to building an actual event-driven application, be it large or small. For example, an event can trigger the launching of a service or string of services to solve a business problem. Modelling event-processing as services is also important because events and CEP applications benefit from the business objectives of an SOA. In fact, services can be used in an event-driven application at practically every functional step in the architecture. In a similar manner, an SOA powered with events facilitates agile, adaptive business processes that can respond to ever changing opportunities and threats. The outcome of a business service or orchestration of services is often another business event.

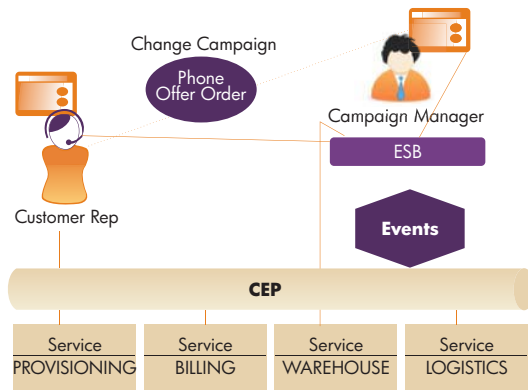


Figure 6 : CEP & SOA

[Figure source: wp-event-driven-soa_tcm8-803.pdf]

2.7 Enterprise System Layers

A typical enterprise system consists of several layers. These layers provide a level of abstraction, making integration of different systems easier. An example of enterprise system layers is found in Figure 7 below.

The two top layers are the most abstract layers. Monitoring and control provides user interfaces for the users to work with and gives them control over the system. In the Business Processes layer the different processes are defined. The processes often cut across system and organizational boundaries. The ESB provides the connection to the different (often legacy) applications and makes integrating these applications possible.



Figure 7 : Typical enterprise system layers

3 Commonly Used Design Patterns

A design pattern in architecture and computer science is a formal way of documenting a solution to a design problem in a particular field of expertise.

Some fundamental CEP design patterns that appear repeatedly in CEP applications are listed below, in order of complexity from the simplest to the most sophisticated. These basic patterns may be thought of as building blocks that can be combined to create a complete application.

3.1 Filtering

Context

Filtering Events Based on Event Attributes

Problem

Filtering is ubiquitous in CEP applications. Here are some examples: A filter may be used to filter out all trades where the volume is too small, or all trades that do not refer to particular stock symbols. A filter may be used to capture the trades that originate from a certain set of IP addresses. Filtering of events may be used to capture sensor readings where values fall outside of the normal range.

Solution

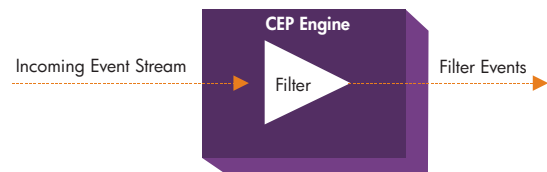


Figure 8 : Filtering

[Figure source: <http://www.coral8.com/developers/documentation.html>]

Figure 8 depicts a simple filter query. The query subscribes to one stream, evaluates a specified logical condition based on event attributes, and, if the condition is true, publishes the event to the destination stream. For example, an application monitoring a stream of purchase orders may filter out all orders where the condition is Priority != 'High' and Amount < 100000.

This example presents the simplest kind of filter, where events are evaluated one by one, and where the query condition involves only the attributes of one event. It is also possible to construct other filters that are more complex. For example, filters that compare events to other events in the same stream, or in another stream, or compare events to a computed metric.

For instance, a filter might capture orders where the purchase amount is larger than the previous purchase amount, or purchase amounts that are larger than the average for the previous day. Such more complex queries are discussed later in this document.

3.2 In-memory caching

Context

Caching and Accessing Streaming and Database Data in Memory

Problem

In-memory caching is used in every non-trivial CEP application. Here are a few examples - In a trading application, the cache may hold the values of recent trades, recent orders, or recent news events, coupled with the relevant historical and reference information. A typical application may hold the recent clicks and searches performed by users, coupled with the relevant historical and reference information.

Solution

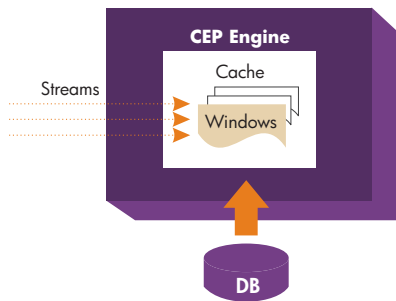


Figure 9 : Event Window Caching

[Figure source: <http://www.coral8.com/developers/documentation.html>]

Figure 9 shows caching and accessing streaming and database data in Memory. In-memory data caching is the foundation of most CEP design patterns. The cache typically stores two kinds of data:

Recent events from one or more streams: Recent events are typically stored in windows. A window is an object, similar to an in-memory database table. However, a window can manage its state automatically, by keeping and evicting certain events according to its policy. For example a window policy might specify: KEEP 1000 ROWS PER ID. This window maintains 1000 rows for each ID value and expire old rows, as necessary.

Data from one or more database tables: Just as streaming events can be cached in memory; it often makes sense to cache data from a relational database, so that different kinds of operations may be performed on this data more efficiently. This cache is typically managed according to the Least Recently Used (LRU) algorithm, or by explicit invalidation.

Note: Many applications require an In-memory cache to be persistent. This means that, if a machine that hosts the CEP engine fails, the data kept in windows is not lost. This functionality is even more important when the window holds not just the last few seconds, but minutes, hours, days, and even weeks worth of events.

3.3 Aggregation over windows

Context

Computing Statistical Metrics over Various Kinds of Moving Windows

Problem

Windows-based computations are used in a wide variety of applications. For example: It is often necessary to compute 'one minute bars' the average, maximum, minimum, and/or closing price within each one-minute interval. There might be a requirement to compute the number of visitors who click on a particular link within a specified time interval. Applications may compute maximum and minimum CPU usage, memory, and Disk I/O utilization for each machine, within a specified time interval.

Solution

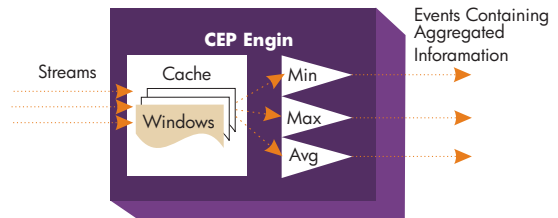


Figure 10 : Aggregation over windows

[Figure source: <http://www.coral8.com/developers/documentation.html>]

This pattern does not merely keep events in memory, but uses the stored values to compute various statistics. A typical example here would involve computing a running average over a sliding window.

This design pattern comes in quite a few flavours, differing along the following dimensions:

The kinds of aggregators computed: These include running averages, sums, counts, minimum, maximum, standard deviation, user-defined aggregators, and so on.

Windows: The kinds of windows used include Time-Window, Sliding Window and Tumbling Window that keep the specified number of largest or smallest elements, and so on. The Time-Window specifies how long an event remains in the window and the expiry time is calculated from the start time. The Sliding Window maintains a queue of a specified size, into which entities flow. When the queue is full and a new event arrives, the oldest event in the queue is removed from the window (FIFO). A Tumbling Window has a specified queue size - specified as a certain number of entities, and empties each time the maximum size is exceeded. Emptying the window completes one cycle. The lifetime of an event in the window, therefore, is one cycle.

Output frequency: Continuous vs. Periodic. In the case of continuous output (also called "tick-by-tick" output) each incoming event updates the calculated expression, and an output event is produced. With periodic output, the calculated expression is updated continuously, but is published only periodically, for example, every ten seconds. Note that, in both cases, the expression is computed incrementally, that is, the entire window is not rescanned on each incoming event.

3.4 Database Lookups

Context

Accessing Databases to Retrieve Historical or Reference Context for Incoming Events

Problem

A trading application may look up historical price for a stock, or certain information about an order, or certain rules and regulations stored in a database. RFID Application: An application may look up information about a palette or case, identified by its tag ID, or information about the reader that reported the tag. An application may also check where the object should be located, according to the plan stored in the database, and compare this location to the actual location of the object.

Solution

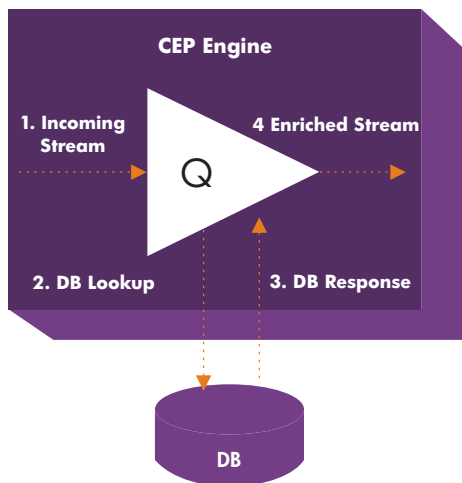


Figure 11 : Database lookups

[Figure source: <http://www.coral8.com/developers/documentation.html>]

While there are applications that deal exclusively with real-time events, most useful applications refer to historical data or reference data to enrich the incoming events. Figure 11 shows how an event comes into the system. The engine issues an SQL request to the database and passes a key (from the event) as a parameter to the database query and the database returns a result. The engine combines the result with data from the event, and forwards the enriched event to the next query for further processing.

3.5 Database Writes

Context

Sending Raw or Derived Events to a Relational Database

Problem

This design pattern cuts across a wide range of applications, such as: Trading - Writing 'one minute' bars (maximums, minimums and the closing price for each one-minute interval) into the database. Click-stream Analysis - Storing the raw click-stream history, together with derived data, in the database. Network Security - Storing new relevant security events in the database.

Solution

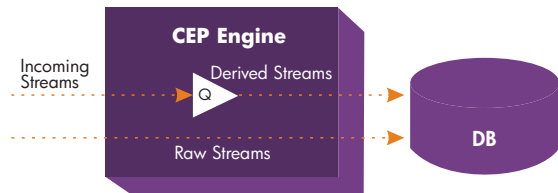


Figure 12 : Database Writes

[Figure source: <http://www.coral8.com/developers/documentation.html>]

A relational database can manage very large volumes of data for very long periods of time, and it also supports a number of interfaces that other applications can use to retrieve the data. This design pattern illustrates the complementary nature of databases and CEP engines.

Note: If the database must store large volumes of events, this design pattern may call for a number of advanced techniques, such as batching, asynchronous writing (to avoid blocking), queuing (to handle spikes), concurrent writes, writing via native database interfaces, and so on.

3.6 Correlation (Joins)

Context

Joining Multiple Event Streams

Problem

Joins and correlation of events are the most sophisticated CEP applications. Here are some examples: Trading: Correlating information from multiple exchanges to find arbitrage opportunities. In Business Process Monitoring: Correlating information from multiple systems that participate in a business process to manage and track exceptions. Network Security: Correlating information across different security devices and applications for sophisticated intrusion detection and response.

Solution

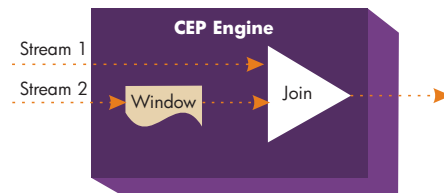


Figure 13 : Correlation (Joins)

[Figure source: <http://www.coral8.com/developers/documentation.html>]

While simple applications often look at just one stream at a time, most advanced applications must look at and correlate events across multiple streams. A join in a CEP application shares many characteristics with a join in SQL. In CEP, a join between two data streams necessarily involves one or more windows. Streams do not store events, but pass events to, from, and between queries. To perform a join, it is necessary to store some events in memory, to wait for events on the corresponding stream. This is what a window does. The above diagram depicts a stream to window join. Events

arriving in Stream 2 are stored in the window. Events arriving in Stream 1 are joined with events stored in the window, and the matching pairs are published by the join.

3.7 Event Pattern Matching

Context

Complex Time-Based Patterns of Events across Multiple Streams

Problem

Event patterns occur naturally in situations where complex behaviour is tracked, such as: Fraud Detection: Fraud patterns are often described as a sequence of events, in one or more streams. For example, in financial services, many fraud patterns involving traders and brokers include events such as the broker taking an order from the customer and emailing the trader, the trader issuing a certain trade and perhaps calling another trader, the other trader waiting for certain market events then issuing another transaction, and so on. In Business Process Monitoring many instances of business process failures may be described as patterns. For example, an application may initiate a certain sequence of steps, some of which complete normally, while others encounter problems because of another application.

Solution

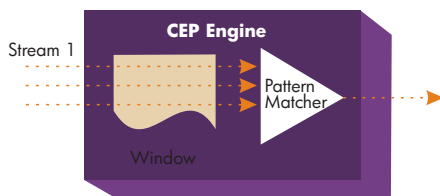


Figure 14 : Event pattern matching

[Figure source: <http://www.coral8.com/developers/documentation.html>]

The continuous joins discussed in the Correlation pattern are quite powerful, but some tasks make the use of multiple joins very cumbersome. Suppose we want to be notified if within a 10 minute interval an event A occurs, followed by event B, followed by either event C or D, followed by the absence of event E, with all events relating to each other in some way. While such an event pattern can be tracked with a combination of inner and outer joins, it is often desirable to have a more direct way of expressing such time-based relationships. Figure 14 depicts a similar pattern with four events in three streams. Most interesting event patterns involve a number of relationships among events such as:

A followed by B: Event B occurs after event A. **A and B:** Both events A and B occur, in either order. **A or B:** Either A or B (or both) occur. **Not A:** Event A does not occur.

Some of the most interesting patterns involve "negative" events, in which the pattern-matching criteria are met when a specified event does not occur within the specified time interval. For example, when tracking a process based on requests and responses, it may be important to know when a response does not occur within a specified time period from the request, or when a response occurs without first being preceded by a request.

3.8 State Machines

Context

Modeling Complex Behaviour and Processes via State Machines

Problem

Finite state machines are applicable wherever an application tracks complex behaviour and processes. Here are some examples: Order processing: In many businesses, order processing is highly complex as the order goes through many stages or states. For example - a request for a price quote is issued, the order is received, then approved and fulfilled and so on. The number of states sometimes reaches into hundreds or even thousands. Tracking User Behaviour on a Web Site: User behaviour on a web site may be described as a series of transitions between states (first-time user, researching products, looking for a product, trying to place an order, order completed, and so on). Tracking and responding to these transitions can often be accomplished with an Finite State Machine (FSM).

Solution

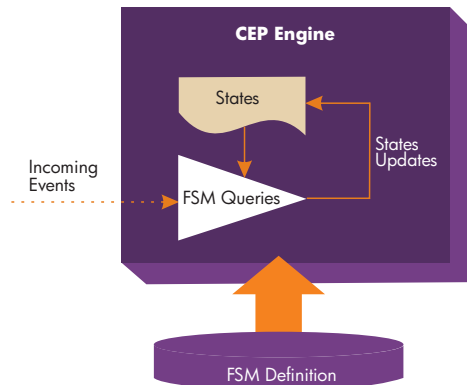


Figure : State machines

[Figure source: <http://www.coral8.com/developers/documentation.html>]

State machines are used in a wide variety of applications, where complex behaviour and processes need to be modelled and tracked. A simple FSM defines a set of states for a process, together with events that define transitions from one state to another.

A few important considerations when designing a finite state machine in a CEP environment are:

- It is good design practice not to hard-code the definition (the possible states and transition rules) of the finite state machine, but to keep it in a relational database, where it may be easily inspected and modified.
- While tracking multiple processes, a finite state machine typically tracks not one, but many processes, each identified by a process ID. If state persistence is enabled, the state of each process is stored in memory and backed up on disk. In an ideal world - all processes would always be in a valid state, events would never be lost, and no illegal transitions would occur. But the real world is far more complex. A production finite state machine is usually designed to cope with and recover from these failures.

3.9 Hierarchical Events

Context

Processing, Analyzing, and Composing Hierarchical (XML) Events

Problem

Hierarchical events are applicable wherever complex behavior and processes are tracked. For example: RFID Applications: An RFID-tagged pallet may contain a number of RFID-tagged cases, each of which may contain a number of RFID-tagged items. Hierarchical events are typically necessary to model such containment directly and hierarchical events often arise in these applications.

Solution

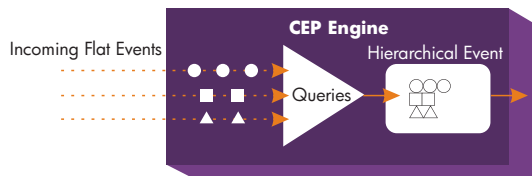


Figure 16 : Hierarchical Events

[Figure source: <http://www.coral8.com/developers/documentation.html>]

Most simple CEP applications analyze flat events. A flat event is similar to a row in a database table - it has a fixed number of fields corresponding to the columns of the table. Flat events provide sufficient functionality for many applications, but other applications deal with events that are more complex. For Example, a Purchase Order event may contain a list of the ordered items. Such hierarchical events appear more often in CEP applications, especially with the rise of XML and SOA.

Figure 16 depicts the design pattern where a complex event is created out of a number of simple events. The following operations may be performed on hierarchical events:

Decomposing Hierarchical Events: A complex, hierarchical event may need to be decomposed into simpler events. For example, it may be necessary to know the items that make up a purchase order.

Correlation Across Hierarchical Events: It may be necessary to know, for example, if two or more Purchase Orders contain the same item.

Composing a Hierarchical Event: For example, it may be necessary to compose a Purchase Order from a list of items.

3.10 Dynamic Queries

Context

Submitting Parameterized Queries, Requests, and Subscriptions Dynamically

Problem

Dynamic queries come up in many applications, especially those that involve large numbers of business users. Here are some examples:

In Trading Environments - every trader can enter their subscriptions

Enterprise Portals: Every user of an enterprise portal, from the CEO down, can subscribe to different queries. Again, parameterization is important here. While the CEO may register interest in incoming purchase orders over one million dollars, a sales manager may want to know about all purchase orders for their territory.

Fraud Detection and Other Machine Learning Applications: Machine learning applications in a CEP environment must dynamically adjust both the queries and parameters, in response to ever-changing external conditions.

Solution

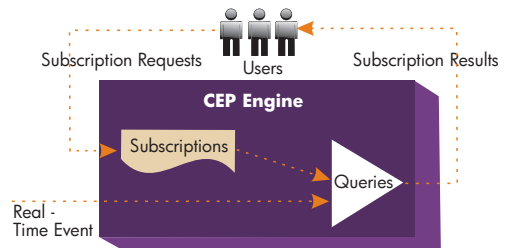


Figure 17 : Dynamic Queries

[Figure source: <http://www.coral8.com/developers/documentation.html>]

Many CEP discussions revolve around the subject of continuous queries, i.e., queries registered with a CEP engine and triggered by the arrival of data. Dynamic queries come in several flavours:

Dynamic Registration of Continuous Queries: In many applications, the ability to register continuous queries programmatically, without restarting the server, is important.

Request/Response Queries: Request/response queries analyze streaming data, but these queries return results only upon explicit request from a user. Often the query itself is pre-registered with the engine, and its execution is triggered by a message on a separate "request" stream.

Subscription Queries: Subscription queries are similar to request/response queries, as these queries are instantiated by an explicit command. However, while request/response queries produce a single response immediately, subscription queries register interest in certain events, and the responses are streamed to subscribers. Figure 17 illustrates how the engine keeps a list of subscriptions from users, and dispatches the results of queries to the right subscriber.

4 TIBCO BusinessEvents

4.1 Overview

TIBCO implements the concept of CEP through TIBCO BusinessEvents Software. The below given list describes some of the features of TIBCO BusinessEvents. This section describes how the mentioned features are realized against the design patterns mentioned in chapter.

- Definition of Channels for capturing events – section 4.2
- Ability to filter events, do pattern matching of events, declarative programming, forward chaining – sections 4.3 & 4.4
- Mapping of events to business objects and define and relationships between them – section 4.5
- Modelling of Events state transitions with timing expectations showing the expected lifecycle and behaviour of objects and analyze events - State machines – section 4.6
- Capability to write events into database, maintain history of events and lookup for events from dataset – section 4.7
- Ability to aggregate event patterns into higher order event patterns and computing statistical metrics over various kinds of moving Windows – section 4.8
- Ability to correlate events and join multiple Event Streams - section 4.9
- Ability to Cache Objects and make Dynamic query on cached objects. Ability to maintain object in Distributed object cache – section 4.10, 4.11 & 4.12.

4.2 Capturing Events

In TIBCO BE, events enter and leave the system through channels. These channels represent physical connections to resources. BE has four types of resources namely, Rendezvous Daemon, Enterprise Message Service Bus, Local Channels and Custom Channels. Destinations in a channel represent listeners to messages from that resource, and they can also send messages to the resource. Figure 18 shows capturing of events through channels.

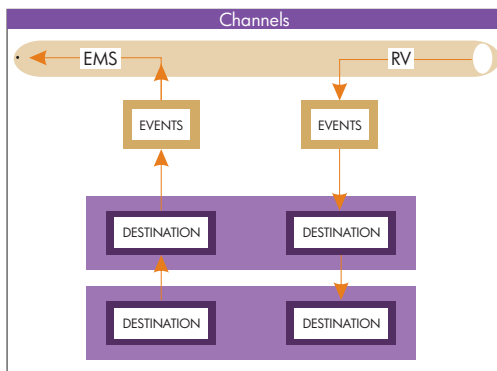


Figure 18 : Channels and Destinations

4.3 Rule Based Engine

Rules are one of the core components of TIBCO BE. Rule is a statement with a condition and an action (if-then).

Example: If the temperature is greater than 75 degrees, turn on the air conditioning. If the order is more than \$1000, grant gold status to the customer. These rules are declarative and not procedural. In its core TIBCO BE is a rule engine that receives events, applies rules to events, transforms events, and sends out events. Rules can filter, correlate and aggregate events by applying constraints and threshold boundaries. Rules engine is based on industry-standard RETE protocol for familiarity and stability, but has been recompiled and tuned to support simultaneous application of thousands of rules to millions of events.

TIBCO BE pre-processor is a rule function that processes incoming messages before BusinessEvents transforms them into Events. For example, a pre-processor might filter the messages so that only certain ones are used as events. Pre-processors are multi-threaded and you can choose from various threading and queue options, as appropriate, to handle the work load.

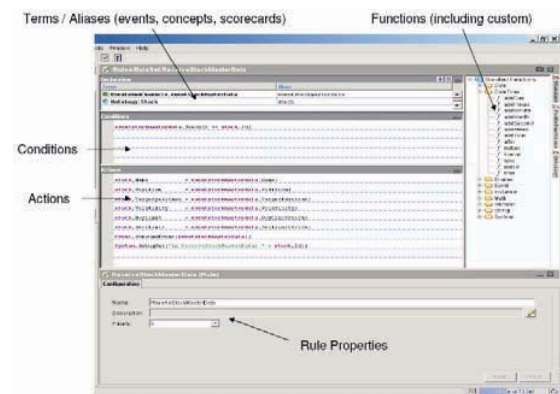


Figure 19 : Rule Editor

4.4 Event Pattern Matching

Information from enterprise applications and other sources flows into BE through channels as messages. BE transforms messages to events, based on event definitions (event types) and asserts them into working memory. All the rules whose conditions match information in the events are assembled into a rule agenda and the first rule executes. Rule actions create and modify the facts, which are processed into a structure known as a Rete network, an in-memory network of objects based on the Rete algorithm which enables fast matching of facts with rule dependencies.

BE uses a forward-chaining inference engine, based on the Rete algorithm. Every time the facts change due to rule actions or the arrival of new information, the inference engine updates the rule agenda. The process goes on until there are no more changes to process. This is known as a run to completion cycle or RTC.

The flowchart below illustrates one run to completion (RTC) cycle, which generally begins when an external action

causes changes to working memory. One RTC cycle ends when there are no more rule actions to execute as a result of the initial change. This is also known as forward chaining, or inferencing. During one RTC no new external actions can affect the working memory. An RTC composes of one or more conflict resolution cycles. A conflict resolution cycle begins when BusinessEvents builds (or refreshes) a rule action agenda, which is used to determine which rule action to execute, and ends when a rule action is executed (or the agenda is empty). If the rule action changes working memory, another conflict resolution cycle begins.

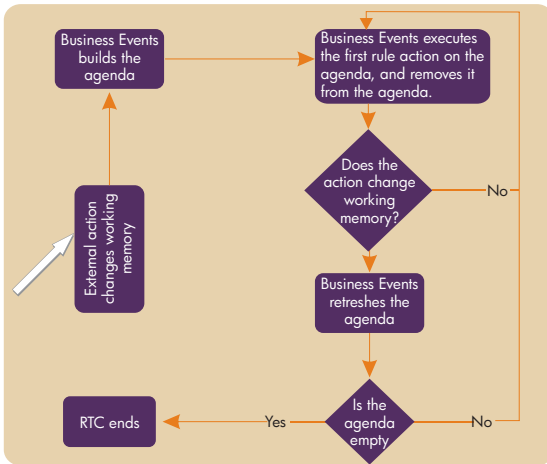


Figure 20 : Run to completion Cycle
 [Figure source: <http://power.tibco.com/pub/lib/TIBCOBusinessEvents>]

4.5 Static Modelling

Any event that is captured through channels can be mapped to Concept. Concept is an abstract entity similar to the object-oriented concept of a class. It is a description of a set of properties that, when grouped together, create a meaningful unit. Concepts can be organized in a hierarchical structure. Concepts can have inheritance, containment and reference relationships with other concepts.

Example: Department, employee, purchase order, and inventory item are all concepts. The term concept type refers to the definition and the term concept instance refers to the actual object. Figure 21 depicts static modelling in BE.

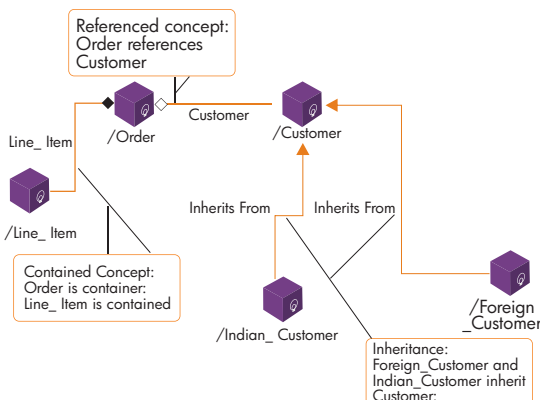


Figure 21: Concept Relationships

4.6 Dynamic Modelling – State Machines

TIBCO BE software allows modelling the life cycle of Concept Instances, i.e., for each instance of a given concept, one can define what states it will pass through and how it will transition from state to state based on rules that apply. Each state model begins with a start state and ends with one or more end states. Between the start and end states may be simple, composite, and concurrent states connected by transitions. Figure 22 displays a simple state model that shows a state model for processing an order. The concept model is based on standard UML Class and Class Diagram principles.

Some of the features of BusinessEvents State Machines are Used for describing tasks, sequences of tasks (processes), and expected outcomes or 'states' of tasks and processes. A state machine captures and stores in an in-memory database the status of events relative to causes, roles and expected behaviour for instant correlation against other events. Data can persist for any length of time depending on how long an event is relevant.

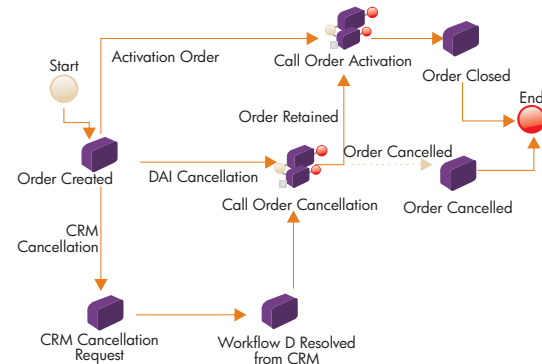


Figure 22 : State Machines

4.7 Database Writes and Lookups

One Database table or view, maps to one BE Database concept definition. TIBCO Database concepts are BE concepts that are created by mapping tables or views from a database to BE. A row in the table or view maps to one database concept instance. They enable performing of database operations such as insert events, update events, delete and retrieve historical or reference context for incoming events. Some of the Database Concept features of BE are Introspect Metadata (Application and System) and generate Ontology concepts/events for Tables and build Relationships. Application services are exposed as BW web services, Rule Functions for CRUD Operation based on the Application Metadata, Provide hooks/use hint for Queries as per Application Metadata, Sequences, Referential constraints are also read as part of Application Metadata. DB Concept can contain other DB Concepts, building an Object mode.

4.8 Query Windows

In BE, a window is a boundary for analyzing data streams. It is a container in which events and concepts (section 4.5) are held and processed by the query. Entities enter and leave the window as determined by the window type and how it is configured. One query can contain multiple windows, and the contents of these windows can be analyzed and compared. Windows can be divided into two basic types, explicit and implicit.

With implicit windows, the lifetime of the entities themselves control the lifecycle of the entities. Implicit windows process changes, additions, and deletions affecting the specified entities until the query ends.

Example of Implicit Windows:

```
select count (*) from /EventA evt group by 1.
```

The example shows a group by 1 clause. This is a dummy group, required because aggregate functions, count (*) in this case, require a group by clause.

Suppose for the first 10 minutes after the query statement is executed, 100 events are created in quick succession, every time the query receives a new event notification, the count goes up progressively until it stabilizes at 100.

Suppose thirty minutes later, 50 of those 100 events are consumed by a rule or expire because of their TTL settings, the events are deleted from the cache. The query receives deletion notifications and the query output, count (*), changes until it drops and stabilizes at 50.

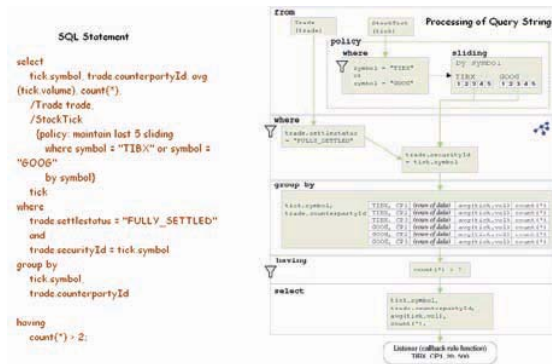


Figure 23 : Explicit Windows
[Figure source: <http://power.tibco.com/publish/TIBCOBusinessEvents>]

Explicit windows (sliding, tumbling, and time windows) define the window boundary, i.e., a condition that limits the lifecycle of the entities in the window.

Example of Explicit Windows:

In this example the SQL clauses are formatted to make each clause distinct. Figure 23 shows the SQL clauses and order in which the SQL String is processed from (including stream clause), where, group by (including having), select. Of special interest is how the where clause in the stream policy operates with the main where clause; and how the stream policy can create multiple windows.

4.9 Correlation of Events

TIBCO BE allows abstracting and correlating meaningful business information from the data flowing through information systems and take appropriate action using business rules. The event correlation is performed using simple BE rules (section 4.3). Using a rules dependency set, BE evaluates the rule conditions such as:

- Filters, i.e., conditions that only involve one scope element (object)
- Equivalent join conditions, i.e., conditions that compare two expressions using == or != where each expression involves one (different) object from the scope
- Non-equivalent join conditions, i.e., conditions involving two or more scope elements other than equivalent joins.

Joins can be performed between two or more entities in a query. Example below describes a sample correlation of events.

```
select thik.symbol,
sum(tick.price) * 1000 / count (*)
avg(tick.volume),
count(*)
t.counterpartyId
from / Trade t, / Stock Tick {policy: maintain last 1 sliding where
symbol = "TIBX"} tick
where t.securityId = "TIBX"
and t.settlestatus = "FULLY_SETTLED"
and t.securityId = tick.symbol
group by tick.symbol, t.counterpartyId
having count(*) > 2;
```

Two entities Trade and StockTick are joined by matching their respective security Id and symbol. But the query also places the restriction that only TIBX trades and stock ticks are of interest and only if the trade's settlement status is FULLY_SETTLED. These filters appear before the actual join expression.

4.10 Object Management

As described in section 4.5 events can be mapped to Concepts. Concept types are descriptive entities similar to the object-oriented concept of a class. At runtime, Rules can create instances of concepts. For example, when a simple event arrives, a rule can create an instance of a concept using values present in the event. Rules can also modify existing concept instance property values.

Object management refers to various ways that BE can manage the state of ontology object instances created by each Rete network (inference agent). The objects once created can be enabled to be available for reuse, either "in memory", cache or in databases.

In Memory: Concept instances are managed in memory by standard JVM features. This basic option does not provide data recovery in case of system failure. The working memory on each system is not synchronized. However, fault tolerance of the engine process is available.

Cache: Using cache clustering technology, object data is kept in memory caches, with redundant storage of each object for reliability and high availability.

Persistence: Object data is periodically written to a data store on disk. This option enables recovery of objects from the persisted state.

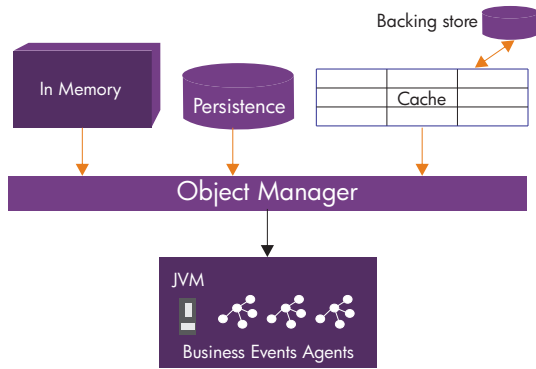


Figure 24 : Object Management
 [Figure source: <http://power.tibco.com/pubslib/TIBCOBusinessEvents>]

4.11 Dynamic Query of Events

Using query language, Concepts (section 4.5) and Events in the cache can be queried, using an SQL-like for performance reasons. TIBCO BE achieves querying of concepts and simple events through Query Agents. These agents are stateless in nature introspect and monitors BE working memory. The query agent's local cache stores cache data locally for efficient reuse. The local cache listens to and synchronizes the locally stored entity instances with those in the main cache, so that the local cache stays up-to-date. Queries are executed in Query Agents language that has read-only access to the underlying objects and events in the cache. BE has types of queries are available, snapshot queries and continuous queries.

Snapshot Queries: Snapshot queries return data from the cache as it exists at a moment in time. A snapshot query returns a single, finite collection of entities that exist in the cache.

Continuous Queries: Continuous queries collect data as objects are added, deleted, or modified in the cache. Continuous queries work on data streaming through the query. Continuous queries continue to gather and return data when notified of changes, until you stop the query.

Figure 25 shows the BE Query structure similar to the structure of a SELECT statement in SQL, and it has parallels with the structure of a BusinessEvents rule.

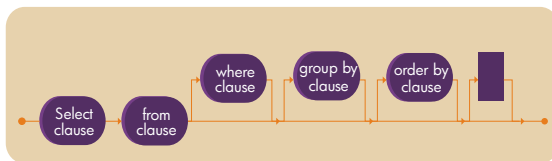


Figure 25: Query Structure
 [Figure source: <http://power.tibco.com/pubslib/TIBCOBusinessEvents>]

4.12 Distributed Cache

In BE, with Cache OM (section 4.10), multi-engine features (engine concurrency) are available. These features provide distributed Cache (Deployment) for High Performance & Scalability improvements. In a distributed cache, cached object data is partitioned between the nodes (JVMs) in the cache cluster for efficient use of memory. Multiple Engines may share the cached information. Nodes and Engines may run across multiple machines which eliminates issues with high JVM unbounded memory growth. No two nodes are responsible for the same item of data. By default, one backup of each item of data is maintained, on a different node. One can configure more backups of each object to be kept on different nodes to provide more reliability as desired (and can disable maintenance of backups). Cache OM uses cache clustering to provide data failover and fallback.

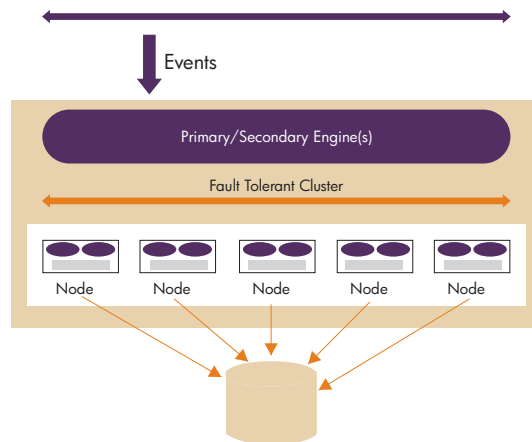


Figure : Distributed Cache
 [Figure source: *Open the Door to Event Processing.pdf*]

5 Conclusions

5.1 Key Capabilities

To summarize, Event Processing and in particular TIBCO BE provides the capability for businesses to track, trace, and correlate events, to process events for trends and patterns and to predict the impact of identified situations.

This helps take immediate action to prevent or minimize damage from threats to businesses. Eg. a customer not receiving a product on time resulting in a cancelled order. The same is true on the upside. A positive situation can be capitalized upon rapidly and effectively. Eg. an emerging trading pattern or customer up-sell opportunity. Situations can also be analyzed to improve the underlying business processes and applications.

5.2 Benefits

The key benefit CEP provides to businesses is to accelerate responses to threats and opportunities by automatically identifying obscure but important relationships between seemingly unrelated events before they result in situations that impact customer experience or the bottom line. CEP improves resource allocation and problem resolution by helping organizations prioritize situations that require the most urgent attention. It does so based on a sophisticated analysis of likely outcomes and secondary or indirect impacts.

5.3 Recommendations

When do you need CEP?

The best strategy is to ask that question now and identify the first CEP projects. Almost every enterprise application can take advantage. Here are some examples:

Insurance Domain: Real-time policy evaluation

During on-line applications and interaction with agents, are there events and risk indicators that change the application risk profile?

Fraud Detection: Correlating fraud-indicator events in real-time at all stages in the claims value chain.

Investment and/or liability situation assessment: For real-time reports of insurers' investment and risk portfolios to maintain solvency and ensure optimum investment potential of the investment funds.

Telecommunications: Telecommunications firms can monitor their service delivery infrastructure, detect network events that signify specific services and capture activities so that the services can be properly billed. A service provider can monitor, analyze, and act on information emitted by network elements and do it in real-time. Network operations centers (NOCs) can monitor both raw network activity and the real-time analytics.

Network Security: Correlating information across different security devices and applications for sophisticated intrusion detection and response.

Automate and Monitor Processes across the 'value chain':

Model a value chain as a state model, and correlate events in various ways to support business activity monitoring - including monitoring BPM.

Contact us

USA

460 Park Avenue South
Suite #1101, New York
NY 10016, USA
Tel.: +1 212 686 6655
Fax: +1 212 686 2422

UK

88 Wood Street
London EC2V 7RS, UK
Tel.: +44 20 85281000
Fax: +44 20 85281001

Australia

410 Concord Road
Rhodes, NSW 2138, AUS
Tel.: +61 290 221 146,
Fax: +61 290 221 134

INDIA

Bagmane Technology Park
Byrasandra Village,
C.V. Raman Nagar
Bangalore 560 093, India
Tel.: +91 80 4004 0404,
Fax: +91 80 4004 9999

Mphasis and the Mphasis logo are registered trademarks of Mphasis Corporation. All other brand or product names are trademarks or registered marks of their respective owners. Mphasis is an equal opportunity employer and values the diversity of its people. Copyright © Mphasis Corporation. All rights reserved.

About Mphasis

Mphasis is a leading Applications, Business Process Outsourcing, and Infrastructure services provider. The company delivers real improvements in business performance for clients through a combination of technology knowhow, domain, and process expertise. With currently over 37,000 people, Mphasis services clients in Financial Services, Communications, Media & Entertainment, Healthcare & Life Sciences, Manufacturing, Transportation & Logistics, Retail & Consumer Packaged Goods, Energy & Utilities, and Governments around the world. To know more, visit www.mphasis.com