Complex Event Processing for Operational Intelligence

A Vitria Technical White Paper
Introduction

Economic conditions, competition, social media-driven customer expectations, and the flood of digitized data are placing unprecedented pressure on organizations to assimilate information and act appropriately and quickly. However, business events have a way of staying a step or two ahead of our ability to capture, analyze, understand, and act upon them. Simply producing reports on a pre-defined schedule is not enough. Decision-makers need real-time alerts and insight so that they may take meaningful action while it still matters.

Operational Intelligence is a new approach for decision-making that allows the optimal response to be taken at the right time. Operational Intelligence provides three key capabilities:

- **Visibility**—the ability to see and access information from a wide variety of sources using a rich and interactive user interface
- **Insight**—the ability to analyze and draw conclusions from multiple real-time and historical data sources as the information changes
- **Action**—the ability to respond in a meaningful way to positively impact business, processes, and customers

Vitria’s M3O platform for Operational Intelligence brings together visibility via rich Web 2.0 dashboards, insight from powerful analytics via Complex Event Processing (CEP), and policy-based action with integrated business process management (BPM).

M3O (which stands for Model, Manage, Monitor, and Optimize) provides these capabilities in a complete, integrated platform for building Operational Intelligence applications in an Event-Driven Architecture, where information systems are modeled from a business event perspective. M3O allows analysts and developers to work collaboratively across the whole of the business event lifecycle.

This paper focuses on the CEP components of M3O that deliver the “Insight” in Operational Intelligence. We first define what is meant by Complex Event Processing, examine the different types of CEP engines, and then explain why Vitria’s unique approach to analyzing real-time XML event data has a number of important advantages over other types of CEP engines. We then explore M3O’s CEP components and how they integrate to provide insight to decision makers:

- **M3O Query Modeler**—the graphical Web 2.0 interface used by analysts and developers to build CEP queries
- **M3O Feed Server**—manages multiple instances and types of real-time feed sources and provides scalability and enterprise quality of service to real-time feeds
- **M3O Analytic Server**—the high-performance CEP engine which executes queries on real-time feeds, creating results ready for visualization in real-time dashboards or for automated, policy-based action and exception management
**What is Complex Event Processing?**

Complex Event Processing is a key Operational Intelligence technology that allows an individual to request an inquiry or analysis once, and then have it continuously evaluated over time against one or many streams of events in a highly efficient manner.

For example, perhaps you want to know the average wait time of customers broken out by region, product requested, and customer type. The underlying data arrives as a series of events in a continuous event stream and computation must be performed on the real-time events in order to calculate the average wait times. In Figure 1 we provide some basic definitions of these concepts.

- **Event** – a message indicating that something has happened
- **Event Stream** or “Feed” – an ordered sequence of events of the same type
- **Complex Event Processing** – matching and transforming source events into result events

Figure 1: Events, Streams, Feeds, and CEP

CEP is a powerful technology for analyzing multiple events over a specific period of time, detecting complex patterns and making correlations. From an Event-Driven Architecture perspective, we can regard CEP as the key technology for processing in real time the potentially high volumes of low-level events and transforming these low-level events into higher-level, aggregated and composite business events for visualization and automated response.

For example, CEP can be used to detect suspicious credit card usage by monitoring credit card activity, as it occurs. It can perform time-series analysis and trending over streams of events, and it can correlate a stream of real-time information with stored and historical data, such as new credit card activity with customer information from a CRM system.

The need to intelligently process large volumes of real-time events exists across a number of domains, including financial trading, risk management, compliance monitoring, Service Level Agreement (SLA) and Key Performance Indicator (KPI) Monitoring, supply chain management, clickstream analysis, cyber security, business process monitoring, logistics, power grid monitoring, infrastructure monitoring, military intelligence, and many others.
SLA Monitoring is an important application area for CEP. It is an important part of most business operations as the speed of business moves to real-time rather than batch-oriented operations. Here, CEP can filter large volumes of events to identify events outside of a specified range, correlate multiple feeds to determine whether diverse events are related to a single issue, aggregate events to determine the extent or seriousness of the issue, and enrich detected issues with customer information to determine which customers will be affected by potential or actual SLA violations.

In the Utilities industry, filtering can be used to identify meter readings from customers participating in Demand Response programs, which can then be correlated and aggregated to determine if customers’ energy curtailment is on track, while enrichment from reference data sources can be used to identify the economic impact of energy curtailment programs in real time.

Table 1 lists the most important fundamental use cases for Complex Event Processing and how they might be applied in some of the domains listed above. These are the CEP building blocks which, when coupled with the ability to perform these functions across vast amounts of data in real time, provide the basis for building complex Operational Intelligence applications.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Description</th>
<th>Example Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filtering</td>
<td>Select events that match user-specified criteria using comparison expressions</td>
<td>SLA Monitoring: detect when response time exceeds a target value (KPI threshold or limit) for each region</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Trading: filter high volume trades</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Power Networks: capture abnormal sensor readings</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clickstream Analysis: capture orders from a specific set of IP addresses by geography</td>
</tr>
<tr>
<td>Correlation</td>
<td>Join events from different feeds based on common attributes and/or expressions</td>
<td>SLA Monitoring: join multiple product order feeds over sliding time- or count-based windows</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Exception Management: correlate information from multiple systems involved in a transaction or business process</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cyber Security: correlate intrusion attempts from multiple intrusion detection systems, log files, etc.</td>
</tr>
<tr>
<td>Use Case</td>
<td>Description</td>
<td>Example Application</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Aggregation</td>
<td>Compute various statistics from event data over time- and count-based sliding and jumping windows, including Count, Average, Sum, Min, Max</td>
<td>SLA Monitoring: compute average response over all regions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Network Management: compute one-minute histogram of average, minimum, and maximum throughput</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clickstream Analysis: compute the number of visitors who click on a particular link within a specified time interval</td>
</tr>
<tr>
<td>Event Pattern</td>
<td>Detect sequential flow of state-changes of one or more event streams over time (i.e., generate an alert if Event A occurred, and then Event B occurred within 60 seconds)</td>
<td>SLA Monitoring: determine whether linked SLA metrics are affected by one going out of bounds</td>
</tr>
<tr>
<td>Pattern Matching</td>
<td></td>
<td>Fraud Management: detect suspicious linked transactions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Trading: detect “wash trading” patterns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clickstream Analysis: how long did the custome take to click “Buy” after opening an advertisement page?</td>
</tr>
<tr>
<td>Enrichment</td>
<td>Merge reference data from external DBMS systems into analytics models to provide full business context for underlying events</td>
<td>SLA Monitoring: historical comparison (i.e., network performance last month); retrieve detail information from master records (look up customer details using customer ID)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Retail: look up product data from RFID tag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cyber Security: retrieve existing intrusions from a particular IP address</td>
</tr>
<tr>
<td>Multi-dimensional</td>
<td>Compute various statistics from the event data broken down by one or more attributes (dimensions)</td>
<td>SLA Monitoring: compute average response per customer level, per partner type, per region</td>
</tr>
<tr>
<td>Analysis</td>
<td></td>
<td>Retail: compute sales volume per product, per region</td>
</tr>
<tr>
<td>Situational Analysis</td>
<td>Overlay contextual information (such as geo-spatial information) on event data</td>
<td>SLA Monitoring: visualize SLA violations by geography</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Logistics: mash up delivery fleet locations with Google Maps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Military Intelligence: monitor movements of people of interest</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Network Management: provide “heat maps” of network throughput to highlight bottlenecks</td>
</tr>
</tbody>
</table>

Table 1: Sample CEP Use Cases and Applications
It is essential that any Complex Event Processing system can efficiently handle XML since most real-time feeds will contain XML messages.

The Importance of XML in Complex Event Processing

Different Types of CEP Engines

There are two major types of CEP engines on the market:

- Query-based—applies database-like queries against event streams
- State-based—represents expected sequences of events as finite state machine models

Query-based CEP engines are the most general-purpose and widely used CEP engines and they cover all the CEP use cases listed in Table 1. In contrast, state machine-based CEP engines are designed to excel at sequence and pattern matching with high transaction volumes over long time durations, but are less flexible than query-based engines in addressing the full range of CEP use cases. Implementing aggregations, correlations, time-series analysis, and so on in state-based engines generally requires relatively complex programming in a proprietary event programming language, rather than using relatively simple declarative queries.

Most query-based stream processing systems use a derivative of SQL as their query modeling language. This is understandable given the familiarity most developers have with SQL. However, rather than using SQL as its modeling language, the Vitria M3O Analytic Server is a query-based CEP engine specifically designed to process real-time XML event streams and uses XQuery as its query modeling language. What are the advantages of using an XML query language over the relational SQL query language?

- Relational tables are flat, whereas most enterprise data (including “business objects”) are naturally represented and passed as hierarchically structured objects, often several levels deep. XML is a natural fit for representing this hierarchical structure.
- Relational tables are rigidly uniform, while XML data tends to be more highly variable. XML is much more adept at handling structural variations, typing variations and missing data, than relational structures, and these situations are very often the norm in rapidly evolving systems.
- Relational data is naturally unordered, while order often has a very important meaning in XML data (particularly for document data).
- Relational tables have relatively static schemas that can be difficult to evolve, while XML schemas tend to be more extensible, and the “self-describing” nature of XML blurs the data/metadata distinction.

These are many of the same “impedance mismatch” arguments made for the advantages of object and native XML persistent storage engines and databases over the more familiar relational database management systems. However, while the relational model remains ubiquitous for data at rest and has relegated native XML persistence to relatively specialized applications, the case for data in motion is...
very different. Today, most business data exchanged between systems is expressed in XML format, both inside and across the firewall, taking advantage of XML's platform independence, its self-describing nature, and its extensibility. The mainstream adoption of the Web Services model to implement Service Oriented Architectures has rapidly accelerated this trend.

With so much data in motion expressed in XML it is essential that any Complex Event Processing system can efficiently handle XML since most real-time feeds will contain XML messages. M3O Analytic Server is a query-based CEP engine that uniquely combines the ability to query XML event streams with SQL queries for enrichment from stored data.

**StreamXQuery™: Efficient Queries on Real-time XML Data**

XQuery is the standard query language for XML data and documents and is analogous to the SQL language for relational queries. The XQuery language is rich enough to support navigation within an XML input document, the combining of data from multiple XML inputs, and the generation of new XML structures from one or more XML inputs.

The most important expression in XQuery is the FLWOR (pronounced “flower”) expression, which is analogous to SELECT-FROM-WHERE-ORDER BY queries in SQL. The general form of the FLWOR statement is:

\[
\text{for } \text{variable1 in collection1, variable2 in collection2, ... } \\
\text{let } \text{variable10 := value10, variable11 := value11, ... } \\
\text{where } \text{predicate} \\
\text{[group by partition-expression] } \\
\text{order by element-name} \\
\text{return result-expression}
\]

**Example 1: XQuery FLWOR Expression Structure**

For data handling, XQuery has much of the richness of SQL and more—it includes support for sub-queries, union, intersection, difference, aggregate functions, sorting, existential and universal quantification, conditional expressions, user-defined functions, and static and dynamic typing, in addition to various constructs to support document manipulation, such as query primitives for element order-related operations.

StreamXQuery is Vitria’s implementation of the XQuery language in the M3O Analytic Server. The implementation is designed and tuned specifically to handle continuous real-time streams of XML events. The StreamXQuery implementation provides the following enhancements to the base XQuery language:

- An extension function (i.e., part of the XQuery language syntax) for defining event windows
- A rich set of extension functions for CEP uses cases including summing; computing minima, maxima, averages; performing SQL queries; comparing aggregations; returning system information
• Geospatial extension functions that include computing the distance between two points; checking whether a point is inside a circular, rectangular, or polygonal geo-fence; and returning the latitude, longitude, or altitude of a given point
• Statistical extension functions to compute linear regression, standard deviation, and variance
• The addition of the group by clause in the FLWOR syntax for grouping results

StreamXQuery™ Windows

In order to perform meaningful analytics on a continuous stream of events, the stream must first be bounded in order to create a collection of scoped events upon which query processing may be performed. In StreamXQuery, the stream is bounded by windows. A window is analogous to a table in a relational query—it represents the bounds of the data set.

Each event in a stream carries a timestamp. In Table 2 we show how the M3O Analytic Server guarantees that events are processed strictly in time order according to their timestamps and also how M3O Analytic Server supports time-based windows based on this timestamp data as well as count- and attribute-based windows.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time-based</td>
<td>Indicates that only events entering M3O Analytic Server in a past time period as specified will be processed. The time range can be expressed in Day, Hour, Minute, or Second units. A trailing time-based event window takes a time interval $\omega$ as a parameter. At time $\lambda$, the event window contains all events with timestamps between $\lambda - \omega$ and $\lambda$.</td>
</tr>
<tr>
<td>Count-based</td>
<td>Process only the specified number of latest events each time. An event range window takes an integer $N &gt; 0$ as a parameter. At time $\lambda$, the event window contains a maximum of $N$ events having the latest timestamps $\leq \lambda$.</td>
</tr>
<tr>
<td>Attribute-based</td>
<td>For each distinct value of the attribute (i.e., customer ID or region name), store in the window the most recent event having that value. Compound attributes are supported.</td>
</tr>
</tbody>
</table>

Table 2: Window Types

Both time- and count-based windows can be either sliding or jumping. With sliding windows, events move into and out of the window as time or the number of events progresses. With jumping windows, events fill the window until the upper time- or count-based boundary is reached. They are then discarded, at which point a new window starts filling up with new events as they arrive.

Windows are described in StreamXQuery queries by using the XQuery extension function $qs:window()$. The signature of this function is:

$$qs:window($sn as xs:string, $wn xs:string?)$$
as document-node()∗
The argument $sn is the registered stream name, $wn is the window specification. Based on the window specified in the qs:window() function, an event stream is piped to the StreamXQuery processor in M3O Analytic Server which creates a node collection of events that fall into the window. This collection is then iterated over by the for statement of the query’s FLWOR expression.

Some example count- and time-based window specifications are described in Table 3.

<table>
<thead>
<tr>
<th>Window Specification</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>sliding size 10</td>
<td>Count-based sliding window size 10 without “step” implies “step 1”</td>
</tr>
<tr>
<td>sliding size P1D</td>
<td>Time-based sliding 1 day window without “step” means window moves for each new timestamp (that is, the step is variable)</td>
</tr>
<tr>
<td>sliding size P1D step P1M</td>
<td>Time-based sliding window size 1 day with step 1 minute</td>
</tr>
<tr>
<td>sliding size P1D step P1D start 2009-10-01T00:00:00Z</td>
<td>Time-based jumping window (size == step), starting at midnight 1st October 2009</td>
</tr>
</tbody>
</table>

Table 3: Window Specification Examples

**A StreamXQuery™ Example**

Putting all of this together, the following example illustrates a StreamXQuery query that performs analysis on a real-time feed of stock trades (StockFeed) using a sliding window one minute in length with step one second. The events are filtered for trade volumes greater than 5000 and grouped by stock ticker symbol. The result is an XML document that contains each symbol name and a computation of the aggregate total amount (price times volume) as the window slides.

```xml
declare namespace ns1 = "http://schema.vitria.com/Event/StockTrade";
declare option outputstream "StockFeed";

for $v in qs:window('StockFeed',
    'sliding size P1M step P1S start 1970-01-01T00:00:00Z')
where $v/ns1:StockTrade/Volume >= 5000
group by $v/ns1:StockTrade/Symbol
return
  <Report>
    <Symbol>
      {fn:string($v/ns1:StockTrade/Symbol)}
    </Symbol>
    <TotalPrice>
      {qs:sum((fn:number($v/ns1:StockTrade/Price) *
                 $v/ns1:StockTrade/Volume ))}
    </TotalPrice>
  </Report>
```

Example 2: Stock Feed StreamXQuery™ Example Query
Advantages of StreamXQuery™ over SQL-based Query Engines

Vitria’s approach to query-based CEP is unique due to its use of StreamXQuery rather than taking the more common SQL-based approach. This gives M3O Analytic Server the ability to query complex XML types on the fly directly without the need to first marshal the XML data into tabular data structures. By performing incremental logic on each event, the number and size of computationally expensive joins is reduced.

This is illustrated in Figure 2, where in order to determine the Incomplete Orders by Customer & Product query over the last two minutes, a SQL-based engine would have to:

- Normalize the three event feeds into four tabular data structures. The complex XML event of Orders & Line Items would need to be flattened into two tabular structures to handle the nesting in the XML document.
- Perform nine joins—including a join against the Order and Order Line Items tables that were created to flatten the Order & Line Items document, as well as eight other joins, continually looping back to each table.

This is a computationally expensive approach, especially as the number and complexity of the events grows. In contrast, the StreamXQuery-based M3O Analytic Server executes as follows:

- Queries simple and complex XML types directly without needing to normalize
- Performs queries and calculations incrementally on result sets, limiting the total number of joins to just three in this example
The M3O Analytic Server can also query data persisted in relational databases and combine that with real-time streamed data to enrich the streamed data and to provide trending and comparisons against historical information.

**M3O®: A Modern Architecture for CEP**

Complex Event Processing sits at the heart of the M3O Operational Intelligence architecture. In Figure 3 we see that events can come from a variety of sources, including transports such as JMS, Web Services, relational databases, and web-based feeds such as RSS. The diagram also illustrates that business process execution engines such as M3O Business Process Server (or even third-party process engines) can also provide process execution events as a feed so that those events can be correlated and visualized with events from other feeds.

These event sources are normalized as feeds by the M3O Feed Server and the feeds are streamed into the M3O Analytic Server (the CEP engine) for computation. The M3O Analytic Server then performs in-memory operations against the feeds, writing results to the repository and archives for later auditing and analysis, painting real-time graphical dashboards for human decision making, and initiating actions in the form of outbound alerts or the execution of event resolution business processes in the M3O Business Process Server.

The M3O Analytic Server can also query data persisted in relational databases and combine that with real-time streamed data to enrich the streamed data, and to provide trending and comparisons against historical information.

**M3O® Query Modeler**

The M3O Query Modeler is the graphical environment in which analysts and developers model StreamXQuery and SQL queries using drag-and-drop and pick-list selection techniques. The graphical UI allows users to quickly build queries using either the graphical techniques or by manually editing queries, depending on the user’s familiarity with XQuery and SQL. Queries are stored in a shared, version-
controlled repository which promotes collaboration while providing control over the versions of the query models deployed in the M3O Analytic Server runtime.

Figure 4 illustrates a graphical XQuery query being constructed in an SLA management application. In the interface, users wire the left-hand side, which contains feeds and the data on those feeds represented by their XML Schema, to the right-hand side which contains the query result along with the window specification and any grouping constructs. The wiring includes examples of intermediate functions being applied to the data as it is mapped from source feed to result schema. Functions are added to the wiring using drag and drop from a palette that includes functions for performing aggregations, numeric calculations, date-time, geospatial, and custom processing.

![Figure 4: M3O® Query Modeler—Graphical XQuery Modeling](image)

Queries are stored as modular components which can be wired together into Event Processing Networks in the graphical interface. In this way, sub-queries can be reused in the same or multiple projects. Figure 5 shows source and intermediate feeds being wired together with queries in an Event Processing Network used to create a queue backlog query in the SLA management application.

![Figure 5: Wiring an Event Processing Network](image)
M3O Feed Server

M3O Feed Server is the architectural component dedicated to managing event sources, or feeds. To ensure optimal scalability and extensibility, the management of feeds is separated from the event analysis performed in the M3O Analytic Server.

M3O Feed Server provides a robust, enterprise-grade quality of service for feeds to support subscriber recovery, event archive, snapshots, scalable deployment, and collaborative sharing of feeds across multiple projects. It provides connectivity to traditional and non-traditional enterprise sources, including JMS, RSS, Web Services, and relational databases.

M3O Feed Server has a plug-and-play, distributed architecture. The architecture can scale to manage varying and unpredictable load volumes by increasing and decreasing the number of Feed Server instances as required. Using this approach, M3O Feed Server can stream millions of real-time events among multiple publishers and consumers.

In addition to standardized feed connectivity, M3O Feed Server provides enterprise quality of service (QoS) to unreliable feed sources, including durability, scalability, and availability of high volume feeds in order to facilitate real-time analysis. Guaranteed message delivery ensures that subscribers receive events in proper event order and that no duplicate events are transmitted.

In addition to supporting Quality of Service, the Event Archive provides the ability for analysts to conduct forensic what-if analysis of feeds with interactive, point-in-time playback of events from the archive. This analysis can be used in an SLA management use case, for example, to allow the analyst to understand past performance so that SLA thresholds can be intelligently recalibrated.

M3O Feed Server supports the creation of Snapshots that effectively give the analyst the ability to create on-the-fly data marts from events persisted in the Event Archive. Snapshots can be used to compare current performance with historical performance to detect trends and to understand deviations from past performance. Data interoperability with traditional BI systems can be used to enhance existing BI systems with snapshots from real-time feeds.

Summary windows maintain the state and history of a feed and allow that history to be instantly replayed. Analysts are able to access and play back a feed state immediately, even if the connection to the feed is broken or if they are a new subscriber and need to catch up with the feed history in their dashboards.

M3O Feed Server provides federated collaboration of feeds across multiple organizational groups and business units. By providing access to shared feed instances, M3O Feed Server promotes collaboration and consistency together with the highest quality of service.
M3O® Analytic Server

M3O Analytic Server is the CEP core of the M3O Operational Intelligence platform. It manages multiple concurrent inbound feeds from the M3O Feed Server, performs StreamXQuery queries on those feeds, and generates a set of outbound result feeds as illustrated in Figure 6.

A result feed contains output (XML) events resulting from StreamXQuery query execution. A result feed can be used in several ways:

- As an input to further downstream queries (modular sub-query components can be graphically assembled using the Query Modeler into query networks)
- As a channel for real-time event visualization in M3O dashboards
- By M3O Policy Manager, so that events in the feed can be dispatched to business processes in the M3O Business Process Server for handling and workflow in a BPMN business process
- By any subscriber application (via an outbound M3O feed)

M3O Analytic Server continuously executes each query defined in the runtime upon arrival of a new event into any of the windows a query references. Event processing is implemented as a sliding window algorithm where events are processed in chronologically ascending order based on their timestamps across all windows in a query. Such query-wide event process ordering is crucial to producing results that accurately reflect the context at the specified point in time. M3O Analytic Server buffers inbound feeds persistently until the events on those feeds are processed, thereby guaranteeing that no events are lost before processing.
Incremental query processing is an internal optimization technique of the M3O Analytic Server query processor that minimizes the computational overhead of continuous query execution. M3O Analytic Server uses the incremental calculation algorithm as much as possible, particularly when computing aggregations.

M3O Analytic Server provides a wide array of powerful analytical and calculative features that enable users to analyze, understand, and act upon information in real time. These features are summarized in Table 4.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real-time event capture, filtering, pattern matching and detection, and aggregation</td>
<td>Detect patterns and trends in real time&lt;br&gt;Correlate real-time and historical data&lt;br&gt;Detect and capture specific business events&lt;br&gt;Filter significant events&lt;br&gt;Aggregate and enrich information</td>
</tr>
<tr>
<td>Real-time predictive analytics, trending, forecasting, and what-if analysis</td>
<td>Compute trend lines in real time and explore deviations&lt;br&gt;Conduct forensics of real-time events&lt;br&gt;Recalibrate Service Level Agreements</td>
</tr>
<tr>
<td>Moving averages over temporal conditions</td>
<td>Analyze data over various time windows&lt;br&gt;Easily adjust time windows and see analysis immediately&lt;br&gt;Recalculation on the fly</td>
</tr>
<tr>
<td>Multi-dimensional analysis of continuous real-time feeds</td>
<td>View and analyze data along a number of different dimensions&lt;br&gt;Control data altitude through drill-down</td>
</tr>
<tr>
<td>Snapshots</td>
<td>View incremental information and provide additional event processing to further analyze the data and take real-time actions&lt;br&gt;Analyze relevant time dimensions and historical context</td>
</tr>
<tr>
<td>Geospatial analytics</td>
<td>Situational Awareness: deduce and analyze geographical relationships between people and entities&lt;br&gt;Define by specified geofence (circle, rectangle, polygon) or by entity (including entry and exit of geofence)&lt;br&gt;Track proximity of key objects&lt;br&gt;Mash up and display on Google Maps</td>
</tr>
</tbody>
</table>
### Table 4: M3O® Analytic Server Features and Capabilities

<table>
<thead>
<tr>
<th>Feature</th>
<th>Capability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear regression</td>
<td>Statistical functions using linear squares algorithm for best-fit line analysis to compute: Slope, Intercept, Standard Deviation, Variance, Regression (r²), DateTime-to-Milliseconds, DateTime-to-Seconds, Milliseconds-to-DateTime, Seconds-to-DateTime</td>
</tr>
<tr>
<td></td>
<td>Provides foundation for dashboard widgets to display linear regression of real-time data series, layering with SLA thresholds, and predictive forecasting of SLA violation times</td>
</tr>
</tbody>
</table>

**Presentation Manager and Open Query API**

The presentation manager provides the interface between queries created in the M3O Query Modeler and their execution in the M3O Analytic Server. The presentation manager also provides the Open Query API that allows partners and customers to use custom built or commercial off-the-shelf (COTS) Flex or Java-based applications to interface with and push queries into M3O Analytic Server. The API enables non-Vitria Flex and Java clients to access M3O Analytic Server to programmatically create new queries or modify existing queries, define result destinations, and so on.

**Scaling Complex Event Processing in M3O®**

Performance is crucial in real-time computing and analytics. Equally important is the ability to scale performance in line with event patterns in order to deliver continued performance over time.

Event sources, event volumes, and analytic complexity may scale in line, but more likely will scale independently depending on the evolution of the business. For example, we can imagine a scenario with hundreds of event sources and millions of events, and yet very simple analytical computations. Conversely, we can also envision situations with very few sources and relatively low event volumes that nonetheless require intensive calculations, or perhaps relatively long computational time windows, thereby requiring the server to hold events in memory for extended periods.

Vitria’s architectural approach allows for flexible scalability by separating the M3O Feed Server from the M3O Analytic Server and enabling clustering on each independently.

The Open Query API enables non-Vitria clients to access M3O Analytic Server to programmatically create new queries, modify existing queries, start and stop queries, and define query result destinations.
Vitria’s architectural approach allows for this kind of flexibility by separating the M3O Feed Server from the M3O Analytic Server and enabling clustering on each independently. This flexible scaling approach is illustrated in Figure 7.

![Figure 7: Flexible Scaling of M3O Feed Server and M3O Analytic Server](image)

**Conclusion**

As we have shown, Complex Event Processing forms the cornerstone of Operational Intelligence. Vitria’s M3O Operational Intelligence platform provides a fully-integrated suite that combines a scalable CEP engine specifically designed for handling XML events efficiently together with a modern, Web 2.0 interface for both design-time modeling and run-time visualization, and a business process engine to enable automated responses. By providing the real-time insight into current business operations, M3O Query Modeler, M3O Feed Server, and M3O Analytic Server enable important information to be analyzed, visualized, and acted upon when it matters most.

**About Vitria**

Vitria Technology, Inc. is the industry’s leading Operational Intelligence company. Our innovative Operational Intelligence solutions empower customers to analyze business activities in process and take real-time action. The result is better decisions when they matter most—before opportunities have faded or problems have escalated. With a rich heritage as a pioneer of BPMS, Vitria’s award-winning solutions provide the backbone for many Global 2000 companies’ mission-critical business processes. Vitria has customers in North America, South America, Europe, Asia, and Australia.