# Event Correlation and Pattern Detection in CEDR

Roger S. Barga[1] and Hillary Caituiro-Monge[2]

[1] Microsoft Research, Microsoft Corporation,
One Microsoft Way, Redmond WA 98052
barga@microsoft.com

[2] UC Santa Barbara, Computer Science Department,
Santa Barbara, CA
hcaituiro@cs.ucsb.edu

## Introduction

*"By 2008 event processing will be mainstream, with most new business systems in large companies set up to emit vast amounts of event information. Applications are going to start to get very chatty..."* *David McCoy, Gartner Group 2005*

Most business processes today are event driven – they receive and react to events, and in turn create new events which are sent to other processes. In today's fast-paced business environment organizations need to detect and respond to critical business events in real-time. As noted in recent reports by Gartner [1][2], a growing trend is to build enterprise systems that respond instantly to significant business events – *latency is the enemy*. Early examples are event-driven business intelligence (BI) systems and business activity monitoring (BAM) systems [3]. A challenge to the success of event-driven enterprise systems is the ability to bridge the gap between raw events the system receives and the critical *business events* to which the enterprise needs to respond [4].

A simple example is a distributed attempt to break into a computer network. While a single computer can be configured to lock out an account when a number of failed logon attempts are detected, there is currently no protection from a distributed attack to break in using a number of different computer systems. For this, the distributed attack is only identified when an unusual number of failed logon attempts for the same user across different machines within a specified time interval are detected.

Another simple example is a flawed software upgrade. Systems management teams typically roll out major operating system upgrades overnight to machines across the company, first installing the service pack on each system, then issuing a system shutdown, followed by a restart command. After waiting a period of time, say five minutes, if a machine fails to restart an event needs to be raised indicating a possible failure in the upgrade for the computer system. If a pattern across these failures can be detected, such as the machine type or other software installed on the machine, the software upgrade roll out can be altered to avoid these machines.

But high-level events signifying a distributed security attack or error in the software upgrade have to be inferred from explicit system events generated by individual computers distributed across the corporate network. This depends on an ability to

*correlate events* and recognize *complex patterns* of events involving timing and relationships between the events. We can easily understand a "security attack" or "install failure" when it is pointed out in the event log. But it isn't actually generated, and that's the heart of the problem. It is a *virtual event* signifying a real activity. It has to be recognized from the morass of events being raised by computers across the network. Recognizing or detecting a significant group of lower-level events from among all the enterprise event traffic, then creating a single event that summarizes in its data their significance is called *event correlation.* Today individual applications must consume and process all events to decide when a complex event has occurred. This requires the application process and manages state from events streaming by, resulting in substantial programming overhead, ad-hoc development efforts and communication traffic. This functionality should be offered as a middleware runtime service.

Event correlation services will play an important role in constructing future distributed enterprise applications that can immediately react to critical events. It also promises a powerful technology for monitoring and managing the enterprise, allowing users to simply register patterns of events and the runtime will filter and monitor the event streams, manage event state and perform correlation and detection. Event correlation depends on technology for recognizing patterns of events in large amounts of lower-level event traffic, in real-time. It also depends on the ability to express patterns consisting of multiple events together with their common data and timing. And, if implemented properly, provides the ability to track lower-level events that were aggregated to create a high-level event – so-called *drill down* diagnostics for tracking vertical causality.

In this demonstration we present an event correlation and pattern detection system called CEDR, short for **C**omplex **E**vent **D**etection and **R**esponse that includes a number of features for complex event processing, such as:

***Effectiveness*** – CEDR provides a declarative language for registering event patterns that uses event instances and correlation as building blocks. The language follows a separation of concerns, supporting orthogonal and composable aspects. The result is simple patterns are easy to register, while programmers have the means to express detailed event correlation conditions and complex event patterns.

***Context*** – CEDR defines a *detection window* or *lifespan* to precisely specify the interval during which detection is relevant for a complex event. The window can be a set of events, or based on internal events such as time of the day or date, etc.

***Temporal Intervals*** – Our current implementation of CEDR includes a number of high level event composition operators. While the specific operators are not unique to CEDR, their ability to correctly handle temporal intervals is novel. This ability to reason about events and their relationship to temporal intervals, such as the interval of time during system failure or security breach, is necessary in order to identify and detect certain patterns of interest (questionable funds transfers, etc). It also enables CEDR to process patterns that include both primitive events and complex events, since complex events have a valid interval (start, end time) instead of a single-valued timestamp.

***Filtering*** – CEDR dynamically filters the event stream, depending on patterns that are currently active and can fine tune filtering based on specific event instances required to match active patterns. CEDR can "push" event filters to client agents where the events originate, reducing events flowing over the network and improving server performance.

***Abstraction*** – CEDR provides the means to create new events whenever a pattern is detected. These new events can be constructed to contain information contained in events that matched the pattern. This is called *event pattern abstraction* and is a necessary step toward supplying higher level views of patterns of business activity.

***Detection*** – CEDR pattern detection uses an optimized algorithm based on nondeterministic finite automaton (NFA). The algorithm is able to naturally deal with temporal correlations between event instances and allows the detector to "back out" event instances that can no longer be used to detect a pattern.

***Priorities and Guarantees*** – CEDR supports *pattern priorities* to order processing of pattern detection. During event bursts CEDR prioritizes computation on critical or priority patterns. And rather than drop event instances or resort to event sampling, which reduces detection accuracy, CEDR collects event instances for low priority patterns and defers detections until rates return to normal. This provides a *deferred detection guarantee* with for low priority patterns without a loss of accuracy.

***Causality Tracking*** – CEDR produces a causality chain for complex events that allows the user to drill down from a high-level event to the set of lower-level event instances that "caused it". This causality chain makes lots of questions about "what happened" and "what is happening" much easier to answer.

Together these features offer exceptional functionality and control over event pattern specification and detection. It should be emphasized that while some of these features have parallels with topics that have been studied in isolation in the past, it is both their combination and implementation in an actual middleware service that makes CEDR interesting and unique.

## What Will Be Demonstrated?

The system demonstration will include an illustration of the CEDR event pattern specification language using the GUI, CEDR detection performance and event monitoring tools that display in real-time the current state and performance of the runtime system, and the execution of sample event correlation and complex pattern detection applications.

One application we will demonstrate is an operations management application that was developed for in-house use by Microsoft Operations Management. The application is fed simulated event streams representing systems events from hundreds of computer systems, ranging from user attempts to logon to the system, software installation on the system to reports of hardware and software failures. CEDR's event correlation and pattern detection capabilities are exercised to show that it is able to detect virtual events from the event instance stream, which will reach hundreds of thousands of events per second, and deliver alerts (complex events) to the user.

Our demo will show the following items:

- Definition of primitive events and registration of patterns to detect using the CEDR GUI, demonstrating how event monitoring applications can be written on-the-fly;
- Event generation tool that simulates events from hundreds of computer systems, before the CEDR detection system is brought online;
- Run-time reaction to event instances using CEDR real-time monitoring tools, such as pattern lifetimes becoming active, the event filter being dynamically updated to allow new event instances to pass, event instances collected for patterns and finally detected.
- Creation of new complex events based on user defined patterns;
- Causality tracking, which is a post-mortem graphical representation (drill down) of complex patterns using event instances and detection records stored in SQL Server.
- After the initial demonstration we adjust the event generation rates to flood the CEDR detection for short bursts. First we will assign priorities to selected complex event patterns and then illustrate that during these event bursts that high priority patterns are detected in real-time, while the lower priority patterns are detected later when the event arrival rate returns to normal.

Our presentation will also include accompanying posters that provide context for both the demonstration and implementation of CEDR and illustrate the following aspects of our work:

- System architecture of CEDR, with illustrations of the key functional units (dynamic event filter, pattern definition manager, detection processor, etc), along with primary data structures.
- An illustration of the separation of concerns in the design of the pattern language of CEDR, with examples that demonstrate the composability of the aspects for patterns of varying complexity.
- List of high level event operators and their definitions, along with the list of the core event algebra operators used to implement these higher level operations. A vignette outlines interval semantics for selected operators for discussion.
- A poster summarizing related work that includes a 'gap analysis' to identify where previous work is either lacking or additional effort is required, i.e., the gaps. Areas where CEDR leverages existing work and contributes (fills a gap) are clearly marked.

## Selected References


1. Schulte, Roy "The Growing Role of Events in Enterprise Applications", Gartner Group Report, July 2003.

2. Schulte, Roy and Natis, Yefim "Event-Driven Architecture Complements SOA", Gartner Group Report, July 2003.

3. Walker, Diana "The Instantly Responsive Enterprise", Information Age, June 2004, pp 20-24.

4. Luckham, David "The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems", Addison Wesley Publishers, 2002.