Active Complex Event Processing: Applications in Real-Time Health Care

Di Wang, Elke A. Rundensteiner,
Han Wang
Worcester Polytechnic Institute
{wangdi,rundenst,wanghan}@cs.wpi.edu

Richard T. Ellison III
Univ. of Massachusetts Medical School
richard.ellison@umassmemorial.org

ABSTRACT

Our analysis of many real-world event based applications has revealed that existing Complex Event Processing technology (CEP), while effective for efficient pattern matching on event stream, is limited in its capability of reacting in real-time to opportunities and risks detected or environmental changes. We are the first to tackle this problem by providing active rule support embedded directly within the CEP engine, henceforth called Active Complex Event Processing technology, or short, Active CEP. We design the Active CEP model and associated rule language that allows rules to be triggered by CEP system state changes and correctly executed during the continuous query process. Moreover we design an Active CEP infrastructure, that integrates the active rule component into the CEP kernel, allowing fine-grained and optimized rule processing. We demonstrate the power of Active CEP by applying it to the development of a collaborative project with UMass Medical School, which detects potential threads of infection and reminds healthcare workers to perform hygiene precautions in real-time.

1. BACKGROUND AND MOTIVATION

Complex patterns of events often capture exceptions, threats or opportunities occurring across application space and time. Complex Event Processing (CEP) technology has thus increasingly gained popularity for efficiently detecting such event patterns in real-time. For example CEP has been employed by diverse applications ranging from healthcare systems , financial analysis , real-time business intelligence to RFID based surveillance. However, existing CEP technologies [3, 7, 2, 5], while effective for pattern matching, are limited in their capability of supporting *active rules*. We motivate the need for such capability based on our experience with the development of a real-world hospital infection control system, called HygieneReminder, or short HyReminder.

Application: HyReminder. According to the U.S. Centers for Disease Control and Prevention [8], *healthcare-associated infections* hit 1.7 million people a year in the United States, causing an estimated 99,000 deaths. HyReminder is a collaborated project between WPI and University of Massachusetts Medical School (UMMS) that uses advanced CEP technologies to solve this long-standing public health problem. HyReminder system aims to continuously track healthcare workers (HCW) for hygiene compliance (for example cleansing hands before entering a H1N1 patient's room), and remind the HCW at the appropriate moments to perform hygiene precautions - thus preventing spread of infections.

CEP technologies are adopted to efficiently monitor event patterns, such as the sequence that a HCW left a patient room (this behavior is measured by a sensor reading and modeled as "`exit`" event), did not sanitize his hands (referred as "`!sanitize`", where `!` represents negation), and then entered another patient's room (referred as "`enter`"). Such a sequence of behaviors, i.e. `SEQ(exit,!sanitize,enter)`, would be deemed as a violation of hand hygiene regulations.

Besides detecting complex events, the HyReminder system requires the ability to specify *logic rules* reminding HCWs to perform the respective appropriate hygiene upon detection of an imminent hand hygiene violation or an actual observed violation. A condensed version of example logic rules derived from HyReminder and modeled using CEP semantics is depicted in Figure 1. In the figure, the edge marked "*Q1.1*" expresses the logic that "*if query Q1.1 is satisfied for a HCW, then change his hygiene status to warning and change his badge light to yellow*". This logic rule in fact specifies how the system should react to the observed change, here meaning the risk being detected by the continuous pattern matching query *Q1.1*, during the long running query process. The system's streaming environment requires that such reactions be executed in a timely fashion. An additional complication arises in that the HCW status changed by this logic rule must be used as a condition by other continuous queries at run time, like *Q2.1* and *Q2.2*.

We can see that active rules and continuous queries over streaming data are tightly-coupled: continuous queries are monitoring the world while active rules are changing the world, both in real-time. Yet contrary to traditional databases, data is not persistently stored in a DSMS, but rather streamed through the system in fluctuating arrival rate. Thus processing active rules in CEP systems requires precise synchronization between queries and rules and careful consideration of latency and resource utilization.

Limitations of Existing CEP Technology. In summary, the following active functionalities are needed by many event stream applications, but not supported by the existing

Proceedings of the VLDB Endowment, Vol. 3, No. 2
Copyright 2010 VLDB Endowment 2150-8097/10/09... $ 10.00.

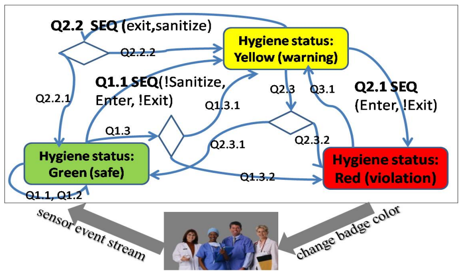


Figure 1: Condensed HCW Hygiene Status Logic

CEP technology.

(1) *Build-in support for declaring a continuous query's status, condition and action.* In existing CEP systems, a CEP query is running in a passive way not able to react to the changes happening during the long-running query process. Those changes may include semantic information being updated, risks or opportunities being detected, and generally any condition critical for solving real-world problems.

Nowadays, business intelligence products [9, 10] tend to tackle the problem by providing an external software layer on top of CEP systems. This solution reaps bad performance as efforts have to be made for crossing the border between the CEP engine and the application layer repeatedly for the handling of possibly a single rule. Moreover, the upper layer lacks the access to the CEP kernel, which limits the opportunity for fine-grained optimization. All these limitations prevent this hard-coded "application-logic-on-top-of-CEP" solution from achieving near-real responsiveness required by most real-time event-based applications.

(2) *Active semantic binding on input data.* Combining semantic knowledge about the application space with incoming stream data is important in CEP applications for extracting meaningful information from the raw data. Beyond the need of static semantic knowledge, there is a need to change the semantic information based on the outcome of queries, and to make the most updated semantic knowledge both readable and writable by queries in real-time.

Proposed Solution: Active CEP. We are the first to extend complex event processing with build-in logic rule support. In other words, we embed active rule semantics directly within the CEP context to build an ***active CEP infrastructure***. The active rule components are *integrated* into the CEP kernel, which allows fine-granularity event notification, deterministic rule behavior enforcement and optimized rule processing.

Our Contributions. (1) We propose an Active CEP model and associated Active CEP rule language for the specification and semantics of Active CEP. This significantly extends the existing CEP model to meet the needs of diverse event stream based applications. (2) We provide an infrastructure to implement the Active CEP model which integrates the active rule extension into the CEP kernel. This approach is in sharp contrast to existing CEP based systems that build an external software layer on top of the CEP engine. (3) We develop a large suite of optimization techniques that achieve scalable and efficient Active CEP execution. (4) We show the feasibility of our proposal by implementing all the techniques above within the full-fledged WPI/HP CEP engine [4]. We demonstrate the effectiveness of our Active CEP through a real-world application, the HyReminder system, which is being deployed into Intense Care Units at the UMass Memorial Hospital.

2. ACTIVE CEP SYSTEM ARCHITECTURE

Figure 2 illustrates the system architecture of our ACEP system, where three core components, i.e. semantic repository, complex event processor and rule manager, are coherently connected.

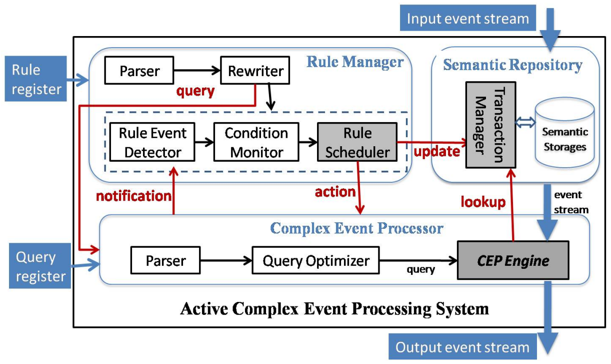


Figure 2: Active CEP System Architecture

Semantics Repository is responsible for maintaining the contextual knowledge model that captures relevant information about the environment into which the application is deployed. The semantic knowledge in the semantic repository can be either *static* (e.g. the physical layout of the hospital such as the position of rooms and sanitizers) or *dynamic* (such as the hygiene performance status of a HCW that is dynamically updated based on his behaviors over time).

Complex Event Processor supports long-running pattern matching queries with negation, nested query and spatio-temporal features over event streams [4]. The processor in our system also handles CEP queries that integrate relational database lookup, typically access to the semantic repository. Queries issued by the application are rewritten by the query optimizer using plan based cost estimation, and then passed to the CEP engine for execution.

Rule Manager is tightly integrated with the CEP processor. The rule rewriter co-works with the query optimizer to convert a rule into an alternate, potentially more economical form. Rule event detection is implemented by placing the checking and notification code inside the kernel of the CEP Processor. The rule scheduler executes a triggered rule's action, which directly affects the semantic repository and the CEP processor.

3. ACTIVE CEP TECHNOLOGY

3.1 Semantics

Data Model. Our abstract semantics is based on two data types: (1) An *event stream* S is an unbounded sequence of events; while an *event* represents an instantaneous occurrence of interest [6], and is assigned a timestamp from a discrete ordered time domain. An event contains the name of its event type (defined in a schema) and a set of attribute values. (2) A *relation* R is a static bag of tuples that all conform to the same schema associated with R.

All data in the CEP system constitute the *CEP system state*. Formally, let S be the domain of *CEP system states*. If s is a state in S, the $s = \{e_1, e_2, ..., e_n; t_1, t_2, ..., t_m\}$ where e_i is an event tuple in event streams and t_i is a relational tuple in relations.

Operators. We adopt the syntax for CEP queries commonly used in the literatures, [4], [7] and [3] [1]:

```
PATTERN (event-pattern) ON event-stream
[WHERE qualification]
[WITHIN window]
RETURN <output-specification>
```

The `event_pattern` describes an event pattern to be matched over the input `event-stream`. For example, `PATTERN(A,B)` will find instances of event `B` that follow event `A`. The `qualification` defines value constraints on the event pattern. The `window` describes the time window during which events that match the pattern must occur. The `output-specification` specifies the construction of complex events being output into the output stream. PATTERN is considered as a high-level operator over event streams. An event tuple arrival is modeled as an APPEND operator that inserts the event instance into the event stream.

Operators on *relations* include a subset of DML operators defined in SQL: SELECT, INSERT, DELETE and UPDATE.

All operators being executed on the CEP system compose the *CEP system changes*. Formally, let Δ be the domain of *CEP system changes*. If δ is a set of changes in Δ, then $\delta =$ [*Pattern, Append, Select, Insert, Delete, Update*]. Namely δ describes the operations on s.

Active Rules. Active rules in our system must be able to respond to complex events being detected by CEP queries, semantic information being updated and more generally to any possible CEP system change. Hence the semantics for an Active CEP rule is defined as a function that maps a CEP system change and a CEP system state into the new CEP system state that results from processing those rules.

Formally, let R be the domain of *rules*. If r is a rule in R, then r is a function that takes as arguments a CEP system change δ and a CEP system state s. It returns a boolean value, a new set of changes and a new system state; that is: $r : \Delta \times S \rightarrow \{true, false\} \times \Delta \times S$

3.2 Active CEP Rule Definition

We now present our proposal of a declarative CEP rule language implementing the semantics described above. The syntax of defining a rule in our Active CEP system is:

```
CREATE [OR REPLACE] RULE <rule-name>
{BEFORE|AFTER} {APPEND|RETURN} ON <stream-name>
[REFERENCING NEW AS <new-tuple-name>]
[FOR EACH TUPLE]
[WHEN <trigger-condition>]
<action-body>
```

For example, to express the logic rule described in the HyReminder application in Section 1, we can first define the CEP query, named Q1-1, and then define an active rule, named R1-1, on query Q1-1's output, as shown below:

```
CREATE QUERY Q1-1 ON STREAM sanitize, enter, exit
PATTERN SEQ(!sanitize, enter, !exit)
WHERE [workerID] AND
time-distance(sanitize, enter) < 2min AND
time-distance(enter, exit) < 10sec AND
'green'=(SELECT status FROM workerStatus
  WHERE workerID=enter.workerID)
RETURN enter.workerID, enter.location

CREATE RULE R1-1
AFTER RETURN ON Q1-1
REFERENCING NEW AS newTuple
FOR EACH TUPLE
WHEN newTuple.location = 'H1N1 room'
BEGIN
  UPDATE workerStatus SET status = 'yellow'
  WHERE workerID = newTuple.workerID
END
```

[1] Our CEP engine [4] can also support advanced pattern features such as negation and Kleene closure. Due to space limit, we omit those advanced features in this work.

3.3 Integrated Rule and Query Execution

For query processing, our Active CEP system employs a NFA-based approach for complex event sequence construction (as in [3, 4]). It adopts customized optimizations that push predicates and window constraints down into the sequence construction. For query predicates containing lookup on relations, we execute the embedded query for each tuple that needs to be evaluated. This simple strategy makes sure the value retrieved from the relation is the most updated one, but brings large I/O costs. In next section we will present an optimized approach for improving the performance. When an active rule is defined on the CEP query, each matched complex event result emitted from the CEP query engine is passed to the rule manager right away. Such build-in-kernel mechanism is crucial for the rule manager to react in a timely fashion. The rule processing pipelines the event detection, condition verification, rule scheduling and action execution, as described in Section 2.

3.4 Optimization

We tackle two salient challenges in event-stream-centric rule processing: *large amount of triggering events* and *fluctuating semantic information* that is frequently accessed by queries and updated by rule actions over time. Since stream-based applications have strict performance requirements, solving the above issues is of paramount importance.

Rule Rewriter. An important optimization for preventing unnecessary events from being passed to the rule manager is to evaluate the rule conditions as early as possible.

In the example rule `R1-1`, the output tuples of query `Q1-1` will be passed to rule processing, but only very few of the tuples will end up triggering the rule to execute its action (suppose the rule condition `newTuple.location='H1N1 room'` is selective). Intuitively, it would be much more efficient to evaluate the condition in the query first. Hence we can remove the condition defined in the rule and add it into the query's qualification list. By doing this we have two CEP queries: the original query with no modification (`Q1-1`), and the query with newly-added qualification (named `Q1-1-R`). And the rule is rewritten to be defined on `Q1-1-R`. These two queries will be run in the CEP engine, assuming the output of the former is still consumed by some other rules or returned to end users.

The overhead of the above rewriting algorithm would be the extra new query registered in the CEP engine. Our system hence performs a *cost-based* analysis for each rule before executing the rewriting. The detailed cost analysis can be found in our technical report.

Caching sub-query result. Fetching the result of a sub-query in advance from a data source outside the CEP engine and caching it for the query's lookup, instead of executing the sub-query for every (qualified) tuple, reduces the query evaluation cost. Hence we propose to create a *Dynamically Cached Index* (DCI) for multiple CEP query conditions that have the same *parameter-list*. The parameter-list is composed of: (1) the target attribute of the relation to access in the outside source, (2) the attribute of the event used as an argument for the sub-query, (3) the value of (2). DCI is hash-based with the key as the parameter-list and value as the according sub-query result.

A *cost-based* analysis will be applied to DCI. Generally speaking, DCI is preferred when many sub-queries share the same contents and/or the number of updates on the outside relations is smaller compared to the total number of event tuples processed.

4. SOFTWARE DEMONSTRATION

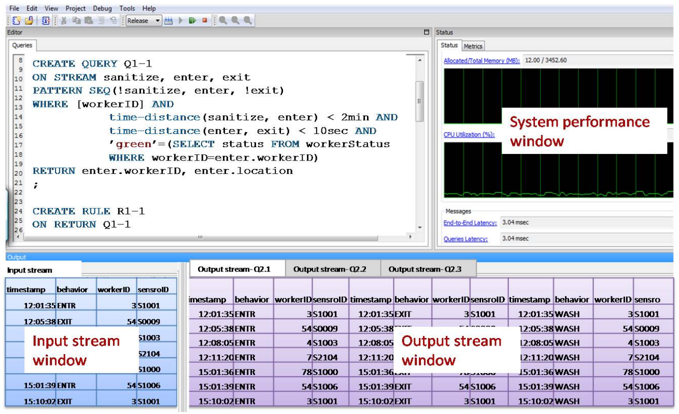
The HyReminder implemented by using Active CEP technology will be deployed in Intense Care Units at UMass Memorial Hospital. We use this application scenario to demonstrate how Active CEP solves real-world problems.

Datasets, Queries and Rules. Referring to the hand hygiene regulations applied in most US hospitals [1], we create CEP queries and active rules for HyReminder using the following methodology: (1) model the specific sequence of HCW behaviors using PATTERN queries; (2) model the HCW's hand hygiene performance with three status, namely compliance (or safe, or "green"), intermediate (or warning, or "yellow") and violation (or "red"); (3) model the logic of a HCW status transitions, namely a certain sequence of behaviors leads the worker to another status as stipulated in the hand hygiene regulations, using active rules.

Application Interfaces. In our demonstration, audiences will be able to see and perform the following features.

A. Define and View Rules and Queries

The internal interface enables the system administrators to specify and debug queries and rules. As shown in Figure 3, the system administration interface offers the window for editing queries and rules in textual form. It also displays the input event streams, output streams for each query and the real-time system performances. The audiences will get an insight view of our Active CEP language. Also, they can submit his/her own queries and test them against both live and persistent event data.



```
CREATE QUERY Q1-1
ON STREAM sanitize, enter, exit
PATTERN SEQ(!sanitize, enter, !exit)
WHERE [workerID] AND
            time-distance(sanitize, enter) < 2min AND
            time-distance(enter, exit) < 10sec AND
            'green'=(SELECT status FROM workerStatus
            WHERE workerID=enter.workerID)
RETURN enter.workerID, enter.location
;

CREATE RULE R1-1
ON RETURN Q1-1
```

Allocated/Total Memory (MB): 12.00 / 3452.60

CPU Utilization (%):

End-to-End Latency: 3.04 msec

Queries Latency: 3.04 msec

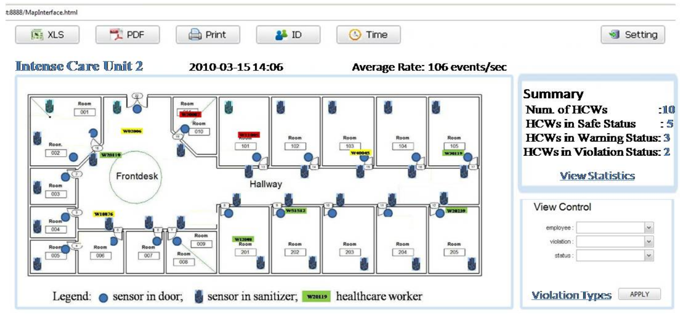
timestamp	behavior	workerID	sensroID
12:01:35	ENTR	3	S1001
12:05:38	EXIT	54	S0009
[illegible]	[illegible]	[illegible]	S1003
[illegible]	[illegible]	[illegible]	S2104
[illegible]	[illegible]	[illegible]	S1000
15:01:39	ENTR	54	S1006
15:10:02	EXIT	3	S1001

imestamp	behavior	workerID	sensroID	timestamp	behavior	workerID	sensroID	timestamp	behavior	workerID	sensro
12:01:35	ENTR	3	S1001	12:01:35	EXIT	3	S1001	12:01:35	WASH	3	S1001
12:05:38	ENTR	54	S0009	12:05:38	[illegible]	[illegible]	[illegible]	12:05:38	WASH	54	S0009
12:08:05	ENTR	4	S1003	12:08:05	[illegible]	[illegible]	[illegible]	12:08:05	WASH	4	S1003
12:11:20	ENTR	7	S2104	12:11:20	[illegible]	[illegible]	[illegible]	12:11:20	WASH	7	S2104
15:01:36	ENTR	78	S1000	15:01:36	[illegible]	[illegible]	[illegible]	15:01:36	WASH	78	S1000
15:01:39	ENTR	54	S1006	15:01:39	EXIT	54	S1006	15:01:39	WASH	54	S1006
15:10:02	ENTR	3	S1001	15:10:02	EXIT	3	S1001	15:10:02	WASH	3	S1001

Figure 3: System Administration Console

B. Real-time Rule Execution Monitoring

We provide the real-time monitoring console that displays the current hygiene compliance state of every HCW in the specified ICU for the head nurses to supervise . The map-based monitoring window chronologically displays each worker as a moving object in the ICU map (Figure 4). The moving object is indicated by different colors to represent the worker's most updated status, determined by the active rules defined in the system. Moreover, real-time statistics about the hand hygiene violations can be accessed by specifying conditions in the "view control" panel. Our audiences can explore the particular active rules associated with a HCW's status by clicking the HCW's icon. And audiences can for instance modify some parameters of the active rules and then view any immediate changes in terms of HCW's status transition in real-time.



Summary
Num. of HCWs :10
HCWs in Safe Status : 5
HCWs in Warning Status: 3
HCWs in Violation Status: 2

Figure 4: Real-time Hand Hygiene Monitoring

C. Complex Event Analysis and Report

We also provide a suite of graphic Complex Event Processing and Analysis tools for biostatisticians to conduct clinical research. The audience can use the tool to see the trends of hygiene performance over time, navigate the query results, aggregate the results and interactively re-run the queries for different time ranges.

5. ACKOWLEDGEMENTS.

This work is partially supported by NSF-IIS #0917017 and by UMMS-WPI CCTS Collaborative Grant 2009/2011 for a collaboration between WPI and Univ. of Massachusetts Medical School. We are grateful to HP Labs for access to their CHAOS stream engine.

6. REFERENCES

[1] J. M. Boyce and D. Pittet. Guideline for hand hygiene in healthcare settings. *MMWR Recomm Rep.*, 51:1–45, 2002.
[2] A. Demers et al. Cayuga: A general purpose event monitoring system. *CIDR*, pages 412–422, 2007.
[3] E. Wu et al. High-performance complex event processing over stream. *SIGMOD*, pages 401–418, 2006.
[4] M. Liu et al. E-cube: Multi-dimensional event sequence processing using concept and pattern hierarchies. *ICDE Demo*, pages 1097–1100, 2010.
[5] R. S. Barga et al. Consistent streaming through time: A vision for event stream processing. *CIDR*, pages 363–374, 2007.
[6] S. Chakravarthy et al. Composite events for active databases: Semantics, contexts and detection. *VLDB*, pages 606–617, 1994.
[7] Y. Mei et al. Zstream: A cost-based query processor for adaptively detecting composite events. *SIGMOD*, pages 193–206, 2009.
[8] U.S. Centers for Disease Control and Prevention. Estimates of healthcare-associated infections. `http://www.cdc.gov/ncidod/dhqp/hai.html`.
[9] Aleri Inc. Aleri cep. `http://www.aleri.com/`.
[10] StreamBase Inc. Cep product. `http://www.streambase.com/`.