Events and Streams: Harnessing and Unleashing Their Synergy![*]

Sharma Chakravarthy[†]
IT Laboratory and the CSE Department
The University of Texas at Arlington
sharma @ cse.uta.edu

Raman Adaikkalavan
CIS Department
Indiana University South Bend
raman @ cs.iusb.edu

ABSTRACT

One of the purposes of this paper is to demonstrate the ubiquitous nature and relevance of simple and complex events in situation monitoring and other unforeseen applications. This paper retrospectively analyzes the progress of event-based capability and their applicability in various domains. Although research on event-based approaches started in a humble manner with the intention of introducing triggers in database management systems for monitoring application state and to automate applications by reducing/eliminating user intervention, currently it has become a force to reckon with as it finds use in many diverse domains. This is primarily due to the fact that a large number of real-world applications are indeed event-driven and hence the paradigm is apposite.

The other purpose of this paper is to identify and establish the synergy between stream processing and event processing. The resurgence of complex event processing (or CEP) has demonstrated the resiliency of this paradigm and its relevance to a large class of applications. In addition, the advent of stream processing based on sensor and other data generated on a continuous basis has, in our opinion, enhanced the role of events in critical ways. Instead of implicitly assuming event generation, stream processing indeed provides a powerful mechanism for generating *interesting* events.

In this paper, we briefly overview the development of the ECA (or event-condition-action) paradigm. We briefly discuss the evolution of the ECA paradigm (or active capability) in Relational and Object-oriented systems. We then describe several diverse applications where the ECA paradigm has been used effectively. The applications range from customized monitoring of web pages to specification and enforcement of role-based access control policies (RBAC). The multitude of applications clearly demonstrate the ubiquitous nature of event-based approaches to problems that were not envisioned as the ones where the active capability would be applicable.

Finally, this paper analyzes the differences between stream and event processing and proposes an integration architecture to meet the requirements of applications such as linear road bench mark and network fault management applications. These applications have a strong stream computation component to generate *interesting* events and a complex event processing component to detect *situations* for timely notification.

[*]This work was supported, in part, by NSF grants IIS-0326505, EIA-0216500, and IIS 0534611.

[†]**The author would like to dedicate this paper to the fond memory of Prof. Klaus R. Dittrich.**

Categories and Subject Descriptors

H.2 [**Database Management**]: H.2.4 Systems—*Rule-based databases*

Keywords

Complex Event Processing, Data Stream Processing, Monitoring Applications, Information Security, Expressive Pattern Search

1. INTRODUCTION

In the mid and late eighties, the inability of database management systems (DBMSs) to monitor and trigger alerts and notifications even for conventional applications such as inventory control (e.g., a part needs to be reordered when *quantity-on-hand* of a widget decreases below a certain number) prompted the work on automating the above to reduce or eliminate user intervention. At that time, process control applications had the capability to monitor system state and take appropriate actions. However, most of these systems were custom-developed and optimized for real-time operation and did not use a general framework for the purpose of monitoring and automation. A number of efforts at that time examined a large number of applications from diverse domains such as process control, power distribution, stock trading and portfolio management, network management, and the use of database in monitoring applications with the goal of developing a framework that could be used for all monitoring applications. Timeliness, near real-time response, well-defined semantics, and the ease of management of the system were the goals of this research and development.

The early explorers of the rule/trigger concept for monitoring were: HiPAC [1], Postgres [2], and ETM [3]. Although all of them had the same broader goal, their approaches and how they addressed the problem were very different. HiPAC, perhaps, took the most general approach of incorporating monitoring into a DBMS by assessing the impact on all components of a DBMS: knowledge model (or specification), query optimization, transaction management and, recovery. The separation of an event (an interest of happening) from condition (conditions correspond to queries) and action (actions correspond to transactions) was promoted based on the role of their semantics as compared to other extant approaches of that time. The event component was separated for the first time and event operators (disjunction, sequence, and closure) were proposed. Coupling modes between event, condition, and action were proposed and their effects were analyzed on transaction management. ECA (Event-Condition-Action) paradigm was seen as a flexible mechanism for automating a number of diverse functionality in a DBMS, such as view materialization, constraint enforcement, and situation monitoring. The Postgres work, on the other hand, concentrated on incorporating triggers into the Postgres framework. ETM (or event/trigger mechanism) explored the notion of active capability in the context of enforcing constraints in design databases using triggers and events. There was some activity among the DBMS vendors as well and the first commercial DBMS to incorporate triggers was surprisingly Interbase [4] which developed a DBMS from scratch in the late eighties.

The nineties saw a large amount of academic research activity in the application of active capability in the form of the ECA paradigm to Object-oriented, Object-Relational, and Relational DBMSs. A large number of prototype systems were developed during that period. Since event component was the least understood part as compared to conditions and actions, there were a number of event specification languages that were proposed along with their semantics and algorithms for their detection. Seamless integration of the ECA paradigm into Object-oriented and other systems were examined in detail to facilitate its incorporation into a number of systems. In addition, various implementation alternatives, such as integrated [5], agent-based [6], and wrapper-based [7] systems for supporting the ECA paradigm were explored. Without differentiating between the event specification languages and the systems that included events and triggers, efforts in the literature to support active capability included (in alphabetical order): A-RDL [8], ACOOD [9], ADAM [10], Alert [11], Ariel [12, 13], COMPOSE [14], HiPAC [15], ODE [16, 17], REACH [18], Rock & Roll [19], RuleCore [20], SAMOS [21, 22], SEQ [23], Snoop/Sentinel [5, 24, 25], STARBURST [26], UBILAB [27], and [28, 29]. A comprehensive introduction and description about most of these systems can be found in [30, 31] and an annotated bibliography on active databases up to 1994 can be found in [32].

The impact of all of the above was that all of the commercial DBMS vendors incorporated the notion of triggers into their products. In addition, SQL1999 [33] further refined the specification of triggers and now it is part of the SQL standard. A detailed study [34] indicated that banks and other targeted users who could really benefit from this feature were unfortunately not using them because: i) there is not enough support from the vendors on the use of triggers, ii) methodology and guidance for the usage of triggers were not available, iii) some of the databases have limitation on the number of triggers, and iv) the performance aspect of the DBMS with the usage of large number of triggers has been largely ignored by the vendors. If the performance disadvantage continues, it is unlikely that the trigger mechanism will see a wide use in real-world applications.

At the same time, the power and utility of the ECA paradigm on non-database applications were being recognized. Even within the database realm, it was shown that the ECA paradigm can not only be used for monitoring the state of user-defined objects, but can also be used for monitoring the system state. This lead to the support of multiple transaction models in a flexible manner [35, 36] using rule sets that could be changed at run time. These rule sets were defined on interesting system events such as acquire lock, release lock, etc.

Beyond the above, work continued on distributed event specification, semantics, and detection [37, 38, 39, 40, 41]. Sentinel [5] developed a complete global event detector (or GED) that had well-defined semantics and used it for a number of real-world applications, such as monitoring multiple DBMSs to check on the viability of war- and peace-time plans that could change dynamically based upon changes to independent databases such as weather, intelligence information, maintenance of vehicles required for operation of plans, etc. A number of tools for the ease of specification of events and rules as well as their analysis were developed by the Sentinel group (event/rule visualization [42], dynamic rule editors [43, 44], and rule analysis concepts and systems were developed [45]) and others [46]. Beyond this, the local event detector was decoupled from its DBMS bondage and re-implemented in Java so that it can be used with any stand-alone application written in Java [47].

In addition to the above, there were a number of other projects that used the concept of events – CORBA [48] being one of the earliest. The notion of events at a low level were being used in network management systems and TIBCO had an event-based system that was used for notification. In addition to the above, graphics user interfaces (GUIs) used event-based callbacks to carry out actions based on the movement of the cursor on the screen. Although the notion of events were used in many of the above systems, their semantics, composition, and execution aspects were not precisely defined. Other systems that have some notion of events include Weblogic [49], ILOG Jrules [50], and Vitria BusinessWare Automator [51].

Today, the notion of active capability does not connote anymore the usage in the context of databases but has been accepted and recognized as a functionality that can be used for any event-driven real-world application. Not surprisingly, it is finding usage in many applications (e.g., work flow, access control, information interchange) as a large number of applications are indeed event-driven. There is also consensus in the database and other communities on the ECA rules as being one of the most general formats for expressing rules in an event-driven application. The semantics of event specification has undergone some extensions from point-based (or detection) semantics to interval-based (or occurrence) semantics. More than that, the number of diverse applications for which it is being used in some form is the real testimony for ubiquitous nature of the abstraction and is the focus of this paper.

The rest of the paper is organized as follows. Definition of events, point-based and interval-based semantics, and event detection are briefly explained in Section 2. A brief exposure to novel applications that have adapted the ECA paradigm in unforeseen ways are provided in Section 3. Stream processing is explained briefly in Section 4 and a detailed analysis of event and stream processing is presented in Section 5. Integration of event and data stream processing is explained in Section 6. Section 7 contains where event processing is headed and Section 8 has conclusions.

2. EVENT SEMANTICS, AND DETECTION

An event is defined [24] to be an instantaneous, atomic (happens completely or not at all) occurrence of interest. The time of occurrence of the last event in an event expression (composition of events using event operators) was used as the time of occurrence for the entire event expression.

An event was classified into primitive (e.g., depositing cash in a bank) or composite (e.g., depositing cash, followed by withdrawal of cash). Primitive events occur at a point in time (i.e., time of depositing). Composite events occur over an interval (i.e., the interval starts at the time cash is deposited and ends when cash is withdrawn) and were detected at the end of the interval. For the above example, the composite event is detected when cash is withdrawn. The event that starts a composite event was termed *initiator* and the event that completes a composite event was termed the *terminator*. Since the event expression (or a composite event) was detected as of the time of occurrence of the terminator, this was called **detection** or **point-based semantics**. For a primitive event, initiator is also the terminator. All event specification assumed the above and detected composite events using different approaches. The operators of Snoop [24, 25] subsumed most of the event specification proposed in the literature.

Snoop identified, proposed, and developed the semantics and detection for a large number of event operators based on the applications analyzed in HiPAC and additional applications analyzed later. The operators supported in Snoop are: disjunction (OR), conjunction (AND), Sequence, Periodic (with a cumulative variant), Aperiodic (with a cumulative variant), non-occurrence (NOT), and temporal (both absolute and relative). The primitive events were domain specific (finite and pre-defined for a domain) and the event expressions were domain independent. For example, relational domain consisted of insert, delete, and update events whereas in an object-oriented domain invocation of any method of any class acted as an event. To add expressive power, begin and end events were defined so that start and end of a function (or any interval) could be broken up into separate events.

The most general semantics (termed *unrestricted* or *general*) detected a large number of composite events in the presence of multiple occurrences of the same event. This seemed unnecessary for a large class of applications. Hence the notion of parameter contexts (aka event consumption modes) were proposed in Snoop [24, 25, 52] to constrain the number of events detected without affecting the application semantics. These were identified after considering classes of applications with certain common properties. For example, sensor applications generate events (e.g., temperature, pressure values) where each occurrence refines the previous value and hence the latest value is of interest when multiple occurrences are present. This observation can be effectively used to prune the events that will not be needed and retain only those that are relevant. This has an impact both on the space requirements as well as the complexity of algorithms used for event detection. The following event consumption modes were defined and supported in the local event detector (or LED) that implemented the detection of Snoop expressions.

- Recent: only the most recent occurrence of the initiator for any event (primitive or composite) that has started the detection of that event is used (useful for sensor class of applications)
- Chronicle: the initiator-terminator pair is unique for an event occurrence (applications where there is a need to match events, such as bug_report and bug_fix)
- Continuous: each initiator of an event starts the detection of that event (for applications requiring moving window concept, such as stock monitoring)
- Cumulative: all occurrences of an event type are accumulated as instances of that event until the event is detected (applications such as a bank where accumulated events are applied at the end of the day)

Rules in the form of condition and action were associated with primitive as well as composite events. Multiple rules could be associated with an event and a priority could be specified. Concurrent and cascaded execution of rules were supported. Models were developed to support rule execution semantics [1, 53, 54]

With point-based semantics, LED [5] uses an event detection graph (or EDG) as shown in Figure 1 for representing an event expression specified using Snoop. This representation is in contrast to other approaches such as Petri nets used by SAMOS [21, 22] or an extended finite state automata used by COMPOSE [14]. By combining event trees on common sub expressions, an event graph is obtained. Data flow architecture is used for the propagation of primitive events to detect composite events. All leaf nodes in an event tree are primitive events and internal nodes represent composite events. By merging common subgraphs, the same event is not detected multiple times. In addition to reducing the number of detections, this approach saves

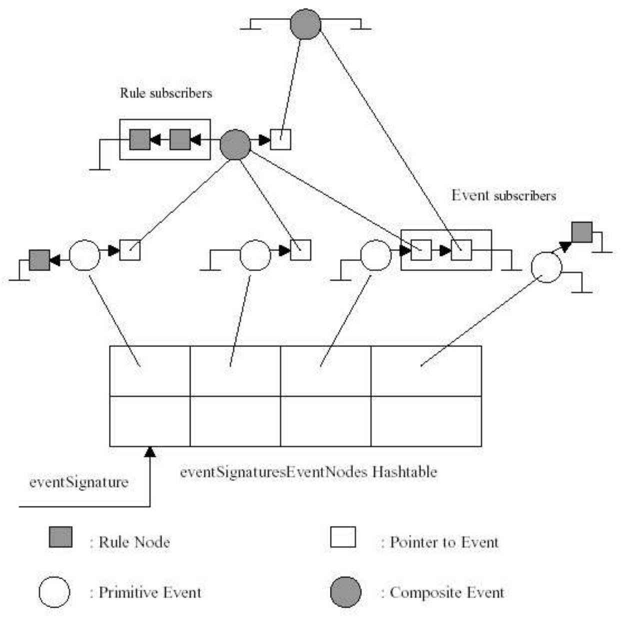

Figure 1: Event Detection Graph.

a substantial amount of storage space (for storing partial event occurrences and their parameters), thus leading to an efficient approach for detecting events. Event occurrences flow in a bottom-up fashion. When a primitive event occurs and is detected, it is sent to its leaf node, which forwards it to one or more parent nodes (as needed) for detecting one or more composite events. When composite events are detected, associated rules are triggered. If there are rules associated with the primitive event, they are also triggered. This representation is similar to the query graph used for query evaluation in a DBMS and allows set-based computation (or multiple events detected at a node). Optimizations can be performed by rewriting event expressions before generating the event graph.

2.1 Interval-Based Semantics

Point-based semantics worked well for most applications. However, when certain operators were composed in a particular way, Galton [55] pointed out that the point-based semantics failed and detected composite events which were non-intuitive. This, for example, happens when the sequence operator is composed twice. This brought out the limitation of the point-based semantics and the need for a more complex (meaningful) semantics that used intervals instead of a point for composite events. As a result SnoopIB [52, 56, 57, 58] was developed for all the operators of Snoop and for all the event consumption modes.

Briefly, interval-based semantics associated two time points with each event: start time and end time, and detect events over an interval. For primitive events, both are same. For composite events, the start time of the initiator and the end time of the terminator are used as the interval in which the composite event occurs. Allen's [59, 60] temporal combinations were used to determine the relevance of the intervals for a particular operator. Neither the event graph nor the detection approach changes with the introduction of the interval-based semantics. Only the algorithms used at each node is different and in fact both point- and interval-based semantics can be supported in the same system.

3. NOVEL APPLICATIONS OF THE ECA PARADIGM

As discussed in Section 1, ECA paradigm has been used in databases as well as stand-alone applications. A number of products support the paradigm in various ways at different levels of abstraction in distributed information exchange, topic-based event notification, as part of information bus, etc. In this section, we will discuss three novel applications where we have effectively used the ECA paradigm with minor modifications and adaptation. In fact, we have been able to reuse the entire LED code base (in Java) for these applications.

3.1 Expressive Pattern Search

Information filtering [61, 62, 63, 64, 65] is the process of extracting relevant or useful portions of documents from continuous streams of textual data based on relatively static user-provided patterns. On the other hand, *Information Retrieval* [66, 67, 68, 69] is the process of extracting relevant or useful portions of documents from a relatively static collection of documents based on a stream of incoming user patterns (or queries). Extant information filtering systems and search engines support only keyword searches and Boolean operator queries. Monitoring text streams for complex patterns have far reaching implications, such as tracking information flow among communications, web parental control, and business intelligence.

InfoFilter [65, 70, 71] allows users to specify complex patterns and detects those over text streams. Consider a real world example where an analyst is tracking information streaming from various resources. He/she is interested in the occurrence of the word "bombŤ followed by the word "ground zeroŤ occurring twice, along with the word "automotiveŤ or its synonyms (i.e., (("bombŤ FOLLOWED BY "ground zeroŤ) occurring twice) AND "automotiveŤ (or its synonyms)). This pattern contains keywords, sequence (FOLLOWED BY), phrase, frequency, synonyms, and a Boolean operator. This pattern cannot be expressed using current informational query retrieval languages (IRQLs) [67] as they do not support the following: i) quantification of multiple occurrences (or frequency) of patterns and complex compositions, ii) arbitrary composition of pattern operators, and iii) a user cannot include synonyms in the pattern, and is required to explicitly list all the synonyms as separate patterns.

Table 1: Summary of PSL operators

Operators	*Purpose*	*Examples*
OR	Provide optional criteria in specifying patterns	***"bomb" OR "explosive"*** ***("bomb" NEAR/3 "automotive") OR ("bomb" NEAR "building")***
NOT	Exclude patterns that are not to be detected	***NOT ("retrieval") ("information", " filtering")***
	Exclude patterns with number of pattern occurrences exceed a user specified number	***NOT/2 ("retrieval") ("information", "query language")***
	Exclude patterns within predefined simple patterns	***NOT(" information" FOLLOWED BY/4 "retrieval") (BeginPara, "EndPara")***
NEAR	Detect patterns that occur within n words of each other	***"information" NEAR/2 "filtering"***
	Detect patterns that occur within the same document	***"information" NEAR/2 "filtering"***
FOLLOWED BY	Detect patterns that occur in certain order within n words of each other	***"data" FOLLOWED BY/2 "structures"***
	Detect patterns that occur together in certain order within the same document	***"data structures" FOLLOWED BY "algorithm"***
WITHIN	Define the scope of detection within a document, a paragraph or a sentence	***("information" NEAR "retrieval") WITHIN (BeginPara, EndPara)***
	Define a range for detecting patterns within a document.	***("information" NEAR/2 "retrieval") WITHIN ("InfoFilter", "Psnoop")***
FREQUENCY	Define minimum number of occurrences of a pattern	***Frequency/5("Iraq")***
SYN	Keyword synonyms	***"bomb" [SYN]***

Pattern Specification: PSL, an expressive pattern specification language based on Snoop operators and interval-based semantics, allows the specification of complex patterns. Patterns are classified into simple and composite types. A *simple pattern* is either a word such as *filtering*, a phrase such as *information filtering systems* or a simple regular expression (regular expression on a single word) such as *info**. The occurrence of a simple pattern is denoted by P[Os, Oe], where Os = Oe (i.e., the starting and ending offset of a simple pattern is the same). PSL supports two types of simple patterns, system-defined (e.g., BeginDoc and BeginPara) and user-defined (single word, phrase and regular expression). A *composite pattern* is an expression constructed using simple patterns, previously constructed composite patterns, PSL operators and options. PSL provides a comprehensive set of operators, OR, non-occurrence (NOT/N), sequential (FOLLOWED BY/N), structural (WITHIN/N), frequency (FREQUENCY/N), proximity (NEAR/N) and the option SYN that allow users to compose *complex patterns*. Table 1 shows various operators and their functionalities with examples. For detailed explanation of PSL, please refer [65, 70]. The semantics of the operators are exactly the same as that of Snoop except that the offset from the beginning of the document is used instead of time. In addition, the distance between initiator pattern and terminator pattern can be specified. From the table, it is straightforward to infer the similarity between Snoop operators and PSL operators. Frequency is a new operator introduced for information filtering. When we analyzed the requirements of this domain with respect to the multiple occurrence of the same pattern, it was clear that the recent context was not appropriate. In information filtering, you want to not only use the last closest occurrence, it does not make sense to reuse the same pattern for another composite pattern detection. As a result, the recent context was modified to the *proximal-unique*. A new context was added to the algorithms of the interval-based semantics. For details refer to [65, 70].

Pattern Detection: A user *pattern* is represented as a pattern detection graph (same as the event detection graph). Simple patterns form leaf nodes. Composite patterns form intermediate nodes. Grouping of subpatterns is also done for efficiency. Data flow is used to notify the occurrence of simple patterns to leaf nodes, which in turn propagate them to intermediate nodes. To handle *synonyms*, *WordNet* [72] is used to determine the synonyms of the patterns, if the synonym option is specified. The incoming stream is processed to generate simple patterns to be fed to the pattern detection graph. Words are also sent to WordNet for extracting their synonyms. Once the synonyms are extracted, they are stored and compared with incoming words using a suffix trie. Given a stream, tokens (corresponding to patterns) are generated along with their offset and are notified to the leaf nodes. Detection of composite patterns is the same as the detection of a composite event.

Pattern search over stored data: The above system designed earlier for streams has been extended to handle stored data as well. The stored data (e.g., web directory) is indexed where the index keeps the offset information of each word along with the word. With this, it is possible to search for patterns using a stored index instead of streaming. Most of the system is common to these two except for a couple of modules. For details, refer to [69, 71].

3.2 Change Monitoring of Web Pages

A number of situations require monitoring changes that are made to one or more documents in a large distributed repository (e.g., select web pages). This is especially useful in the context of web where, currently, a significant amount of time and resources are spent by individuals for monitoring changes to web pages manually. By automating this approach, relevant, useful, and timely notifications can be sent to users.

WebVigiL [73, 74, 75, 76, 77] is a profile based change detection and notification system that monitors changes to structured and unstructured documents. Currently it handles HTML and XML documents. WebVigiL architecture is modular and extensible. It has been designed to handle expressive change specification, provide truly asynchronous approach to manage user's requests, detect actual changes,

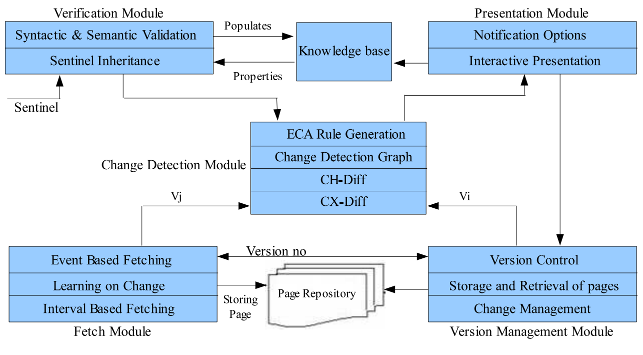

Figure 2: WebVigiL Architecture.

and notify changes – all using active capability.Figure 2 illustrates the architecture of WebVigiL.

WebVigiL accepts profiles from users and monitors for user-specified changes in an intelligent manner (using a combination of intelligent pull and push) and notifies the user in a timely manner of the changes. The profile (given by the user) can include monitoring over a period of time, specifies one or more URL's to be monitored, and the types of changes to be monitored (any change, keyword, phrase, link, image), and how the changes should be notified. Even composite changes such as conjunction, disjunction, and non-occurrence can be specified. The versions of the pages on which change needs to be computed (pair-wise, every n, moving n) can also be specified.

WebVigiL has adapted and extended the event detection graph approach used in Snoop for detecting primitive as well as composite changes. Primitive change detection involves detecting changes to links, images, phrases, keywords, etc., in a page.

Of all the modules shown in Figure 2, the change detection module is central to the system and heavily uses the active capability (actually LED and modified LED for change detection) in a novel way. Change detection module contains ECA rule generation, Change Detection Graph, CH-Diff [73, 74] and CX-Diff [75, 76, 77] as its sub-modules. ECA rule generation module is responsible for activation and deactivation of profiles, generating fetch rules for retrieving pages, detecting changes of interest and to generate time-based notification of changes. Change Detection Graphs reduce the amount of information stored, if more than one user requests for different types of changes on the same page. CH-Diff supports change detection in HTML pages where as CX-Diff is for XML pages. The change detection is done by extracting appropriate objects from the pages based on the change type specified by the user. The objects are then compared for changes and if there are any, they are reported to the notification module.

In order to monitor the page specified by the user, it has to be fetched using the specified periodicity. In WebVigiL, we use the *periodic* event to achieve this fetch in an asynchronous manner. A periodic event is an event that repeats itself within a constant and finite amount of time. The initiator and terminator are the start and end events of a user request and `t` is the interval with which the page should be monitored. The actual fetch of the page is performed by the rule associated with the periodic event. Profiles in WebVigiL can be classified into two categories: Fixed fetch-

Interval and On-Change. The rules associated with all the events (absolute, relative, plus and periodic) generated are executed in the immediate coupling mode. In this manner, ECA rules are used to asynchronously activate (enable) and deactivate (disable) profiles at run time. Once the appropriate events and rules are created, the LED handles the execution at run time. Fetching of a page is considered an event that starts the process of change detection. Each type of detected change is considered an event that is propagated to detect composite events. The WebVigiL system detects these events for each document (page) on which a profile is set. The system detects composite events which is an event expression comprising a set of events related through one or more event operators such as NOT, AND, OR.

3.3 Active Security

Controlling access decisions for resources is critical in any system or environment. Dynamically monitoring the state changes of an underlying system or environment, detecting and reacting to changes without delay are crucial for the success of any active security and access control enforcement mechanism. With their inherent nature, ECA rules are prospective candidates to carry out change detection and to provide both access control and active security. Role-Based Access Control (RBAC) has been considered as a viable alternative to both discretionary and mandatory access control and is shown as cost effective and is being employed in various domains on account of its characteristics: rich specification, policy neutrality, separation of duty relations, principle of least privilege, and ease of management.

Enterprises can model access control policies using either the RBAC standard [78] or any of its extensions. Both the *specification* and *enforcement* are critical in employing these policies in real-world systems. Most of the research has *explored* and *extended* RBAC. On the other hand, most of the current systems have concentrated mostly on policy specifications and very little on the ease of their enforcement. ECA or Active rules not only have a well-defined semantics, they can be added to existing systems and executed to *enforce access control policies* if the policies can be mapped to active rules. We have shown how active authorization rules or extended ECA rules can be used to enforce RBAC and its extensions such as temporal, and control flow dependency constraints in a uniform way [79].

Constraints such as time-based, context-aware and others play a vital role in providing fine grained access control and realizing RBAC over diverse domains. We have extended the standard RBAC with constraints based on event patterns (generalized event expressions) that are not supported by current systems. Event patterns with complex events and simple events as constituent events were used to model constraints such as temporal, context, precedence, dependency, non-occurrence, and their combinations. We have extended event detection graphs as event registrar graphs to incorporate all the event generalizations and for capturing event occurrences and keeping track of event ordering. We have shown how enhanced ECA rules and LED are used for enforcing RBAC standard together with the proposed extensions in a uniform and transparent manner.

In this application, we have extended and generalized [79, 80, 81] the ECA paradigm. The need for attribute-based semantics, masks, and other constraints were needed to specify complex access control policies. The detection of the extended ECA paradigm has also resulted in some extensions to the event detection graph. The main advantage of our approach (over other extant approaches) is that ours provides a *uniform framework* for specifying RBAC policies as well as enforcing them directly in the underlying system. A number of advantages including separation of policy from system code, ability to change rule sets dynamically, accrue from our approach.

3.4 Summary

Above, we have described the adaptation of the ECA paradigm for several newer applications. The diversity of applications indeed indicate the ubiquitous nature of this paradigm and how it can be adapted meaningfully for event-driven as well as other applications. As a matter of fact, in addition to the above applications, we have also shown how the ECA paradigm can be used as a mechanism for executing work flow instances by activating and synchronizing task executions using Snoop operators and ECA rules [82]. We have also shown the sufficiency of Snoop operators for supporting work flow specification recommended by WMC (Workflow Management Coalition) [83]. Other applications of the ECA paradigm we have experimented with include dynamic situation-based access control that goes beyond the active security discussed above.

Lately, A number of new prototype and systems have been developed for CEP and stream processing. The difference between these two systems is blurred somewhat as most of the systems claim to do both stream and event processing. This paper does not go into their details due to space constraints. The relevant systems are: StreamBase [84], Coral8 (`http://www.coral8.com`), Esper (`http://esper.codehaus.com`), Amit [85], Corona Suite (`http://redrabbitsoftware.com`), Aleri (`http://www.aleri.com`), RuleCore (`http://rulecore.com`), and MavEStream [86].

4. STREAM PROCESSING

Event processing and lately stream data processing [87, 88, 89, 90, 91, 92] have evolved independently based on situation monitoring application needs. As we have discussed, several event specification languages for specifying composite events have been proposed and triggers have been successfully incorporated into relational databases and applications. Recently, stream data processing has received a lot of attention, and a number of issues – from architecture [87, 89, 90, 91, 93] to scheduling [94, 95, 96, 97] to Quality-Of-Service (QoS) management [98, 99, 100] – have been explored. Although both of the above topics seem to be different on the face of it, we believe, based on the applications we have analyzed, that they augment/complement each other very well in terms of computational needs of real-world applications. As it turns out, the computation model used for stream data processing (data flow model) is not very dissimilar from some of the event processing models (e.g., event graph), but developed with a different emphasis.

As many of the stream applications are based on sensor data, they invariably give rise to events that need to be composed to detect composite (or complex) events on which some actions need to be taken. In other words, many stream applications not only need computations on streams, but these computations also generate interesting events (e.g., car accident detection and notification, network congestion control, smart homes, network fault management) and several such events may have to be composed, detected and monitored for taking appropriate actions. Currently, to the best of our knowledge, none of the work addresses the specification and computation of the above two threads of work and the issues that need to be addressed. A number of sensor database projects, Cougar [101, 102], TinyDB [103, 104] have tried to integrate the event processing with query processing under a sensor database environment. However, the event-driven queries proposed in TinyDB, for example, is used to activate queries based on events from underlying operating systems. Our focus in this work is to process large number of high volume and highly dynamic event streams from continuous query (CQ) processing stage for the applications that need complex event processing and CPU-intensive computation (i.e., CQs) for generating events. Our premise is that although each one is useful in its own right, their *combined* expressiveness and synergistic integration are critical for a large class of current and future applications of stream data processing.

5. ANALYSIS OF EVENT VS. STREAM PROCESSING

Processing of events using event detection graphs (analogous to a query tree) and a data flow architecture, is similar to the processing of data streams. In this section, we analyze some of the characteristics of the event and stream processing models. For more details, refer to [105]. This will form the basis of our integrated model that combines the strengths of both.

5.1 Consumption Modes Vs. Windows

Event consumption modes or contexts were introduced primarily to determine how many events from the same event sequence should be kept for the purpose of detecting composite events. The number of events to be kept depended solely on the context of the operator and the semantics of the operator. For example, for the sequence operator, one could not drop any event from the past if no context was associated (i.e., use of general context). On the other hand, for the OR operator, that is not the case. Event contexts indirectly resulted in keeping a small portion of (the head of) an event sequence based solely on the value of timestamp and the context. Also, the set of event instances retained at a binary operator depended upon the dynamics of both event streams. For instance, if the completing/terminating event did not occur, event instances for the other operator would accumulate for some contexts such as chronicle and continuous.

In contrast, the notion of a window in stream processing is defined on each stream and does not depend upon the operator semantics. Also, the window need not be defined only in terms of either time or physical number of tuples, although that is typical in most of the applications. The objective of defining a window in stream processing was to convert blocking operators into a non-blocking computation and to produce output in a continuous manner. Hence, the window generated by a context is not the same as the window concept in streams.

5.2 Event Operators Vs. CQ Operators

Event operators are quite different from the operators supported by current stream processing model. Event operators are mainly used to express and define the computation on event (tuple) level and to reduce the number of output events through consumption modes or profiles, and they solely use the timestamp of an event for detecting composite events. For example, `AND` operator in the event processing model is used to express and compute the occurrence (appearance) of two events (tuples) from its two input event streams. Thus, event operators do not perform any computation over the attributes of these events (tuples). On the other hand, current stream processing operators are mostly modified relational operators [1], which focus on how to express and define the computation at the attribute level, rather than the tuple level. Additionally, stream processing operators have input queues and internal windows in order to deal with highly bursty inputs and to convert blocking operators to non-blocking operators.

Both relational or event operators do not have input queues and internal windows, as relational operators perform computations over static relations and event operators perform computations over low input-rate event streams. This is also because of the underlying assumption that query or event processing systems have sufficient processing capacity and resources to handle their data sources.

5.3 Computation Model

Computations in event processing models are decomposed into three main components, which correspond to each component of an Event-Condition-Action rule : 1) computations performed at the tuple level, which are carried out by the event operators, 2) computations performed at the attribute level, which are carried out by the condition checks, and 3) computations for processing rules (triggering actions). In some event processing systems, a smaller part of the computations that are carried out at the attribute level are moved to the event detection component (i.e., the mask proposed in [106, 107]), improving the performance. Computations in stream processing models are not clearly partitioned into different components as in the case of event processing. However, considering the functionalities, computations in stream processing models can be viewed as two components: operator computation and window computation. The former involves complicated condition checking and attribute level manipulations, and the latter is required to maintain a snapshot of tuples or status information for blocking operator computations.

Thus, computations that are performed at the tuple level and the computations for rule processing in the event processing models are absent in the stream processing model. On the other hand, window computations in the stream processing model do not appear in the event processing model. Even though both of them operate at the attribute level, operator computation in the stream processing model is more powerful than the computation performed for condition checking in the event processing model. From the above it is clear that the computations in both the models have different emphasis and different purposes, and they are for different applications. Thus, integration of these two computation models is more powerful and useful and can support larger class of applications.

5.4 Best-Effort Vs. QoS

[1]The blocking operators have been converted as non-blocking in data stream model for properly computing CQs.

The notion of QoS is not present in the event processing literature. Although, there is some work on real-time events and event showers [108], event processing models do not support any specific QoS requirements. Typically, in the event processing model, whenever an event occurs it is detected or propagated to form a composite event. Thus, events are detected based on the best-effort method. On the other hand, QoS support in a data stream processing model is necessary and critical to the success of data stream management systems (DSMSs) for the following reasons: 1) The input of a data stream processing system is highly dynamic and unpredictable in contrast to its fixed computation resources. During overload periods, some queries cannot get sufficient resources to compute their results, which can cause unexpectedly long delays for the final output results. 2) Many stream-based applications require near real-time responses from underlying stream processing system. A delayed response may not be useful, and may even cause serious problems. Different applications can tolerate different response times or inaccuracy in the final query results. and 3) Queries with different QoS requirements must be treated differently with a goal to minimize the overall violation of predefined QoS specifications. A number of QoS delivery mechanisms have been explored and proposed [98, 99, 100].

5.5 Buffer Manager and Load Shedding

None of the event processing systems assume the presence of queues between event operators. Events were assumed to be processed as soon as they are detected (not necessarily occurred) and partial results are maintained in event nodes. Most of the event processing models assume that the incoming events are not bursty and hence do not provide any kind of buffer management or explicit load shedding strategies. Event consumption modes can be loosely interpreted as load shedding, used from a semantics viewpoint rather than QoS viewpoint. On the other hand, load shedding is extremely important in a stream processing environment. Even with the choice of the best scheduling strategy, it is imperative to have load shedding strategies as the input rates can vary dynamically. Several load shedding strategies, placement of load shedders, and the amount of tuples to be shed (possibly limiting the error in query results) have been proposed [98, 99, 100].

5.6 High-Level Rule Processing

ECA Rules describe how the underlying system should respond when an event occurs, making the system reactive. Rules are either used to extend the range of applications that can be supported by the underlying system or to change the way in which new applications are developed. Rules are considered important as they allow users to specify predefined actions that need to be taken when an event occurs and the corresponding conditions are satisfied. Existing event processing systems support dynamic enabling and disabling of rules. On the other hand, rule execution semantics specifies how the set of rules should behave in the systems once they have been defined. A rich set of rule execution semantics [1, 30, 31] have been proposed to accurately define and efficiently execute various rules in the literature for event processing models. Those semantics include rule processing granularity, instance/set oriented execution, iterative/recursive execution, conflict resolution, sequential/concurrent execution, coupling modes, and termination. Stream processing systems do not support high-level rule specification and processing, which are critical to many real-world applications.

5.7 Summary

Similarities and differences between event and stream processing models have been discussed above. Both models employ a similar data-flow architecture as their computation model over streaming data. However, both models have their limitations for handling applications that require stream processing followed by event processing, and most of the functionalities provided by these two models are complementary to each other. The stream processing model focuses on providing a set of functionalities similar to those provided by DBMSs to process and manage data streams. As a result, a general framework and a set of comprehensive techniques such as the notion of a window, optimization techniques, scheduling strategies, load shedding, and others, have been proposed and are being developed. On the other hand, the event processing models focus on detecting composite events and rule processing under the assumption that event sequences are generated (mostly) within the underlying systems with relatively low input rate. Consequently, scheduling strategies, load shedding techniques, QoS support, and so on were not explored for that model. However, the three component computation model (i.e., event processing, condition checking, and rule processing) developed in the event processing are specialized for event detection and rule processing. In contrast, event computation at tuple level and rule processing are absent in the data stream processing model. Although CQs consisting of modified relational operators and aggregation operators can be used in situation monitoring, its expressiveness and computation capability of complicated events are limited and it is also not well-suited for detecting composite events and applying contexts to reduce the number of meaningful events compared with the techniques provided by the event processing models.

Clearly, it is desirable and natural to combine the strengths of both models into an integrated model with a general framework and a set of comprehensive techniques of stream processing model plus the event computation model (i.e., computation at tuple level, consumption modes, and so on) and sophisticated rule processing capabilities. This integrated model will be much stronger and can serve a larger class of applications than what are currently supported by both the models individually.

6. INTEGRATION OF STREAM AND EVENT PROCESSING

Our integrated model is shown in Figure 3 consists of four stages: 1) CQ processing stage used for computing CQs over data streams, 2) event generation stage that is responsible for generating events that are only of interest to the event processing stage (using the notion of masks and other constraints), 3) event processing stage that is used for detecting composite events, and 4) rule processing stage that is used to check conditions, and to trigger predefined actions once events are detected.

The above architecture is possible as both of our stream and event processing models use the *same* data flow architec-

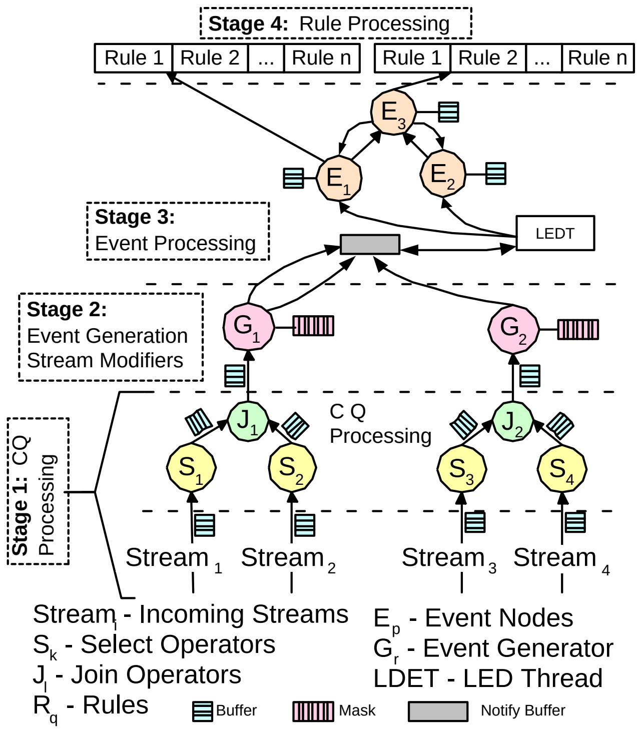

Figure 3: Four Stage Integration Model.

ture and the operators form an event graph or a query graph. Use of other event detection approaches, such as Petri nets and extended automata would have made the architecture more complex and afford less scope for seamless integration. The interface between the two has been designed to avoid interprocess communication and seamless specification of continuous queries and event expressions that can be combined in arbitrary ways.

The LED has been used almost unchanged for the above integration. The event generation stage was added to combine the two relatively easily. Of course, a number of additional issues still need to be addressed in the above integration which can be found in [86, 105, 109, 110].

Using the above architecture and additional stream operators, we have able to show how the linear road benchmark can be supported [105] elegantly using the above model. We also have experimented with other applications in network fault management [109] and others to simplify the specification of continuous queries as well as complex event processing.

7. CURRENT AND FUTURE TRENDS

In addition to the extensive development and application of the event-based rules in conventional and non-traditional applications, there has been a steady use of these concepts in several commercial products beyond CORBA and other earlier systems. JavaBeans [111], for example, incorporated attribute level events for each attribute of a class using standard interface. InfoBus is an extension of JavaBeans with advanced dynamic interfaces for exchanging data. vendors such as Weblogic, TIBCO, and others have used event-driven approaches (if not the ECA format and the use of composite events and other extensions) in their systems to provide flexibility for event-driven applications. In addition to these systems ECA rules provide active capability for *applications* in several other *domains* including XML [112, 113], RDF [114], semantic web [115], sensor databases [116], ubiquitous computing [117], P2P database systems [118], and active spatial data mining [119].

The DMTF Common Information Model (CIM) [120] is a conceptual information model for describing computing and business entities in enterprise and Internet environments. It provides a consistent definition and structure of data, using object-oriented techniques. The CIM Schema establishes a common conceptual framework that describes the managed environment. The CIM Event Model defines the Event-related abstractions. It describes the CIM Indication hierarchy and the use of Indications to model Events. The Event Model also describes the use of subscriptions to register to receive Indications. This is another area where we will see the utilization of the ECA paradigm coming to fruition.

Autonomic computing has become extremely important. To manage interdependent usage of various resources (may they be DBMSs, web gateways, communication pipes), there is a need for managing various policies that govern individual resources, but have an impact on policies of other resources. So it is extremely important to manage policies in a distributed environment. Several groups including IBM and other players are working on the flexible management of policies in an autonomic computing environment. Here again the advantages of the ECA paradigm and rules are quite evident. Work along the lines of policy analysis, identifying conflicts, dynamic change of policies, and triggering policy changes based on distributed state of the system will become extremely important.

As XML is a widely used standard for information exchange (self-describing format), there have been attempts at incorporating ECA paradigm [112] and other event-related features. In addition, a number of companies are working together for defining web services policies (WS policies) now using XML, SOAP, and WSDL extensible models. There is ongoing work on defining declarative policy specification languages for web using Rule Markup language (or RuleML) [121].

Both stream processing and pervasive computing are poised to take off on account of the availability and use of smart sensors (e.g., RFID's). This will increase the amount of data that will be generated which will have to be aggregated and abstracted in novel ways. Both stream processing and concomitant event processing will play critical role in the successful deployment of these technologies.

8. CONCLUSIONS

The purpose of this paper was to recapitulate the event-based computation research over the past 2+ decades and demonstrate its resiliency and applicability for new applications in diverse domains. We have also summarized some of the novel application we have developed over the years using the ECA paradigm. It is amazing that the work which started out with the intention of automating a number of applications in simple ways has lasted this long and it only seems to be gaining momentum. All the applications and associated systems described in this paper have been implemented as proof-of-principle systems.

Beyond the applications briefly outlined from the viewpoint of the usage of the ECA paradigm and its synergy with stream processing, we are currently working on various capabilities that utilize ECA rules: extending and adapting the ECA rules for use in information assurance, semantics for event operators for utilizing in advance applications, synergistic integration of distributed events and their detection (or the global event detector or GED) with stream processing system, and embedding ECA capability into XML for supporting business policies, negotiations and access control/security aspects of e-commerce.

9. REFERENCES

[1] S. Chakravarthy *et al.*, "HiPAC: A Research Project in Active, Time-Constrained Database Management (Final Report)," Xerox Advanced Information Technology, Cambridge, MA, Tech. Rep. XAIT-89-02, Aug. 1989.

[2] M. Stonebraker, M. Hanson, and S. Potamianos, "The POSTGRES rule manager," *IEEE Transactions on Software Engineering*, vol. 14, no. 7, pp. 897–907, Jul. 1988.

[3] K. R. Dittrich, A. M. Kotz, and J. A. Mulle., "An Event/Trigger Mechanism to Enforce Complex Consistency Constraints in Design Databases," *SIGMOD Record*, vol. 15, no. 3, pp. 22–36, Sep. 1986.

[4] *InterBase DDL Reference Manual, InterBase Version 3.0*, InterBase Software Corporation, Bedford, MA, 1990.

[5] S. Chakravarthy, E. Anwar, L. Maugis, and D. Mishra, "Design of Sentinel: An Object-Oriented DBMS with Event-Based Rules," *IST*, vol. 36, no. 9, pp. 559–568, 1994.

[6] S. Chakravarthy, G. Gopalakrishnan, R. Liuzi, and L. Wong, "Agent-based middleware for supporting active capability in rdbmss," in *International Conference on Artificial Intelligence(IC-AI)*, 2002.

[7] S. Chakravarthy and S. Nesson, "Making an Object-Oriented DBMS Active: Design, Implementation and Evaluation of a Prototype," in *Proc. of Int'l Conf. on Extended Database Technology (EDBT), Venice, Italy*, Apr. 1990, pp. 482–490.

[8] E. Simon and J. Kiernan, "The a-rdl system," in *Active Database Systems: Triggers and Rules For Advanced Database Processing.* Morgan Kaufmann, 1996, pp. 111–149.

[9] H. Engstrom, M. Berndtsson, and B. Lings, "Acood essentials," University of Skovde, Tech. Rep., 1997.

[10] O. Diaz, N. Paton, and P. Gray, "Rule Management in Object-Oriented Databases: A Unified Approach," in *Proc. of VLDB*, Sep. 1991.

[11] U. Schreier *et al.*, "Alert: An Architecture for Transforming a Passive DBMS into an Active DBMS," in *Proc. of VLDB*, 1991.

[12] E. N. Hanson, "The Design and Implementation of the Ariel Active Database Rule System," *IEEE TKDE*, vol. 8, no. 1, 1996.

[13] E. N. Hanson, "Ariel," in *Active Rules in Database Systems*, Norman W.Paton, Ed. New York: Springer, 1999, pp. 221–232.

[14] N. H. Gehani, H. V. Jagadish, and O. Shmueli, "COMPOSE: A System For Composite Event Specification and Detection," AT&T Bell Laboratories, Tech. Rep., Dec. 1992.

[15] U. Dayal *et al.*, "The HiPAC Project: Combining Active Databases and Timing Constraints," *SIGMOD Record*, vol. 17, no. 1, pp. 51–70, Mar. 1988.

[16] N. H. Gehani and H. V. Jagadish, "Ode as an Active Database: Constraints and Triggers," in *Proc. of VLDB*, Sep. 1991, pp. 327–336.

[17] D. L. Lieuwen, N. H. Gehani, and R. Arlein, "The Ode Active Database: Trigger Semantics and Implementation," in *Proc. of ICDE*, Mar. 1996, pp. 412–420.

[18] A. P. Buchmann *et al.*, *Rules in an Open System: The REACH Rule System.* Rules in Database Systems, 1993.

[19] A. Dinn, M. H. Williams, and N. W. Paton, "ROCK & ROLL: A Deductive Object-Oriented Database with Active and Spatial Extensions," in *Proc. of ICDE*, 1997.

[20] M. Seirio and M. Berndtssons, "Design and Implementation of a ECA Rule Markup Language," in *Proceedings of the International RuleML Conference*, Nov 2005.

[21] S. Gatziu and K. R. Dittrich, "SAMOS: An Active, Object-Oriented Database System," *IEEE Quarterly Bulletin on Data Engineering*, vol. 15, no. 1-4, pp. 23–26, Dec. 1992.

[22] S. Gatziu and K. R. Dittrich, "Events in an Object-Oriented Database System," in *Proceedings of Rules in Database Systems*, Sep. 1993.

[23] P. Seshadri, M. Livny, and R. Ramakrishnan, "The Design and Implementation of a Sequence Database System," in *Proc. of VLDB*, 1996, pp. 99–110.

[24] S. Chakravarthy and D. Mishra, "Snoop: An Expressive Event Specification Language for Active Databases," *DKE*, vol. 14, no. 10, pp. 1–26, 1994.

[25] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim, "Composite Events for Active Databases: Semantics, Contexts, and Detection," in *Proc. of the VLDB.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 606–617.

[26] J. Widom, "The Starburst Rule System: Language Design, Implementation, and Applications," *in IEEE Quarterly Bulletin on Data Engineering*, vol. 15, no. 1-4, pp. 15–18, December 1992.

[27] A. Kotz-Dittrich, "Adding Active Functionality to an Object-Oriented Database System - a Layered Approach," in *Proc. of the Conference on Database Systems in Office, Technique and Science*, Mar. 1993.

[28] I. Motakis and C. Zaniolo, "Formal Semantics for Composite Temporal Events in Active Database Rules," *Journal of System Integration*, vol. 7, no. 3-4, pp. 291–325, 1997.

[29] I. Motakis and C. Zaniolo, "Temporal Aggregation in Active Database Rules," in *Proc. of SIGMOD*, 1997, pp. 440–451.

[30] J. Widom and S. Ceri, *Active Database Systems: Triggers and Rules.* Morgan Kaufmann Publishers, Inc., 1996.

[31] N. W. Paton, *Active Rules in Database Systems.* New York: Springer, 1999.

[32] U. Jaeger and J. C. Freytag, "An annotated bibliography on active databases." *SIGMOD Record*, vol. 24, no. 1, pp. 58–69, 1995.

[33] A. Eisenberg and J. Melton, "Sql: 1999, formerly known as sql 3," *SIGMOD Record*, vol. 28, no. 1, pp. 131–138, 1999.

[34] A. K. Dittrich and E. Simon, "Active database systems: Expectations, commercial experience, and beyond." in *Active Rules in Database Systems*, 1999, pp. 367–404.

[35] E. Anwar, "Realizing extended transaction models : An extensible approach," Ph.D. dissertation, CISE Department, University of Florida, Gainesville, FL 32611, Fall 1996. [Online]. Available: http://itlab.uta.edu/sharma/People/ThesisWeb/emsa_phd_thesis.pdf

[36] E. Anwar, S. Chakravarthy, and M. Viveros, "An extensible approach to realizing advanced transaction models," in *Proc. of the workshop on Advanced Transaction Models and Architecture (ATMA).*

[37] S. Schwiderski, A. Herbert, and K. Moody, "Composite events for detecting behavior patterns in distributed environments," in *TAPOS Distributed Object Management*, 1995.

[38] U. Jaeger and J. K. Obermaier, "Parallel event detection in active database systems: The heart of the matter." in *ARTDB*, 1997, pp. 159–175.

[39] S. Yang and S. Chakravarthy, "Formal semantics of composite events for distributed environments." in *ICDE*, 1999, pp. 400–407.

[40] S. Chakravarthy and H. Liao, "Asynchronous monitoring of events for distributed cooperative environments." in *CODAS*, 2001, pp. 25–32.

[41] W. Tanpisut, "Design and Implementation of Event Based Subscription/Notification Paradigm for Distributed Environments," Master's thesis, ITLab, CSE Dept., The University of Texas at Arlington, Arlington, TX, U.S.A, 2001. [Online]. Available: http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/Weera_thesis.pdf

[42] S. Chakravarthy and S. Yang, "Architecture and Implementation of an Interactive Tool for the Design and Visualization of Active Capability," in *VDB*, 2002, pp. 111–125.

[43] H. Chu, "A flexible dynamic eca rule editor for sentinel: Design and implementation," Master's thesis, Database Systems R&D Center, CISE, University of Florida, E470 CSE Building, Gainesville, FL 32611, December 1997.

[44] S. R. Varakala, "Design and implementation of a dynamic programming environment for active rules," Master's thesis, Information Technology Laboratory, CSE Dept., The University of Texas at Arlington, Arlington, TX, U.S.A, 2003. [Online]. Available: http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/Sridhar.pdf

[45] S.-K. Kim and S. Chakravarthy, "A practical approach to static analysis and execution of rules in active databases," in *Proceedings of the CIKM*. New York, NY, USA: ACM Press, 1997, pp. 161–168.

[46] E. Baralis and J. Widom, "An algebraic approach to static analysis of active database rules," *ACM Trans. Database Syst.*, vol. 25, no. 3, pp. 269–332, 2000.

[47] R. Dasari, "Events And Rules For JAVA: Design And Implemenation Of A Seamless Approach," Master's thesis, CIS Department, The University of Florida, Gainesville, 1999. [Online]. Available: http://itlab.uta.edu/sharma/People/ThesisWeb/rajesh_thesis.pdf

[48] O. M. Group, *CORBAServices: Common Object Services Specification v1.0*. John Wiley & Sons, Inc., NJ, 1995.

[49] "WebLogic Events Architecture," WebLogic - BEA Systems, Inc., 1999. [Online]. Available: http://www.weblogic.com/docs/techoverview/em.html

[50] "ILOG JRules," ILOG, Inc., 2002. [Online]. Available: http://www.ilog.com/products/jrules/whitepapers/index.cfm?filename=WPJR%ules4.0.pdf

[51] "Vitria BusinessWare," Vitria Technology, Inc., 1999. [Online]. Available: http://www.vitria.com

[52] R. Adaikkalavan and S. Chakravarthy, "SnoopIB: Interval-Based Event Specification and Detection for Active Databases (in press)," *DKE*, 2005. [Online]. Available: http://dx.doi.org/10.1016/j.datak.2005.07.009

[53] U. Dayal, M. Hsu, and R. Ladin, "Organizing long-running activities with triggers and transactions." in *Proc. ACM SIGMOD Int'l. Conf. on Management of Data*, Atlantic City, May 1990.

[54] J. Widom, "A denotational semantics for the Starburst production rule language." *ACM SIGMOD Record*, vol. Vol.21, no. No.3, September 1992.

[55] A. Galton and J. Augusto, "Two Approaches to Event Definition," in *Proc. of the DEXA*. Springer-Verlag, 2002, pp. 547–556.

[56] R. Adaikkalavan and S. Chakravarthy, "SnoopIB: Interval-Based Event Specification and Detection for Active Databases," in *Proc. of the ADBIS*. Germany: LNCS 2798, Sep. 2003, pp. 190–204.

[57] R. Adaikkalavan and S. Chakravarthy, "Formalization and Detection of Events Over a Sliding Window in Active Databases Using Interval-Based Semantics," in *Proc. of the ADBIS*, Budapest, Hungary, Sep. 2004, pp. 241–256.

[58] R. Adaikkalavan and S. Chakravarthy, "Formalization and Detection of Events Using Interval-Based Semantics," in *Proc. of the COMAD*, Goa, India, Jan. 2005, pp. 58–69.

[59] J. Allen, "Towards a general Theory of action and time," *Artificial Intelligence*, vol. 23, no. 1, pp. 23–54, 1984.

[60] J. Allen and G. Gerguson, "Action and Events in Interval Temporal Logic," *Journal of Logic and Computation*, vol. 4, no. 5, pp. 31–79, 1994.

[61] T. Yan and H. Garcia-Molina, "The sift information dissemination system," in *ACM Transactions on Database Systems (TODS)*, Dec. 1999.

[62] C. Stevens, "Knowledge based study for accessing large, poorly structured information spaces," Ph.D. dissertation, University of Colorado at Boulder, 1993.

[63] U. Manber and S. Wu, "Glimpse: A tool to search through entire file system," in *USENIX Winter 1994 Technical Conference*, 1994.

[64] M. Arafiujo, G. Navarro, and N. Ziviani, "Large text searching allowing errors," in *Proceedings of WSP*, 1997.

[65] L. Elkhalifa, R. Adaikkalavan, and S. Chakravarthy, "InfoFilter: A System for Expressive Pattern Specification and Detection over Text Streams," in *Proc. of the ACM SAC*, Santa Fe, NM, USA, Mar. 2005, pp. 1084–1088.

[66] G. Salton and M. McGill, *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., 1983.

[67] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[68] M. W. Berry, "Survey of text mining: clustering, classification,and retrieval," *Springer - Verlag*, 2004.

[69] N. Deshpande, "Infosearch: A system for searching and retrieving documents using complex queries," Master's thesis, The University of Texas at Arlington, 2006. [Online]. Available: http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/Des05MS.pdf

[70] L. Elkhalifa, "Infofilter: Complex pattern specification and detection over text streams," Master's thesis, The University of Texas at Arlington, 2004. [Online]. Available: http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/Laali.pdf

[71] S. Chakravarthy, L. Elkhalifa, N. Desphande, R. Adaikkalavan, and R. Liuzzi, "Pattern Search over Streaming and Stored Data," in *Proc. of the ICAI*, Nevada, USA, Jun 2006.

[72] C. Fellbaum, "Wordnet: An electronic lexical database," in *MIT Press*, 1998.

[73] N. Pandrangi, "Webvigil: Adaptive fetching and user-profile based change detection of html pages," Master's thesis, The University of Texas at Arilngton, 2003. [Online]. Available: http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/naveen.pdf

[74] N. Pandrangi *et al.*, "Webvigil: User-profile based change detection for html/xml documents," in *Proceedings 20th British National Conference on Data Bases*, Coventry, UK, 2003.

[75] J. Jacob, "Webvigil: Sentinel specificatin and user-intent based change detection for xml," Master's thesis, The University of Texas at Arilngton, 2003. [Online]. Available: http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/Jyoti.pdf

[76] J. Jacob, A. Sachde, and S. Chakravarthy, "Cx-diff: A change detection algorithm for xml content and change presentation issues for webvigil," in *Proceedings of XSDM Workshop*, Chicago, October 2003, pp. 273–284.

[77] J. Jacob, A. Sachde, and S. Chakravarthy, "Cx-diff: a change detection algorithm for xml content and change visualization for webvigil," *Data and Knowledge Engineering*, vol. 52, no. 2, pp. 209–230, Feb. 2005.

[78] *RBAC Standard, ANSI INCITS 359-2004*, ANSI INCITS 359-2004, InterNational Committee for Information Technology Standards, 2004.

[79] R. Adaikkalavan and S. Chakravarthy, "Active Authorization Rules for Enforcing Role-Based Access Control and its Extensions," in *Proceedings, IEEE International Conference on Data Engineering (International Workshop on Privacy Data Management)*,

Tokyo, Japan, Apr. 2005, p. 1197.

[80] R. Adaikkalavan, "Generalization and Enforcement of Role-Based Access Control Using a Novel Event-based Approach," Ph.D. dissertation, ITLab, CSE Dept., The University of Texas at Arlington, Arlington, TX, U.S.A, 2006. [Online]. Available: http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/Ada06PHD.pdf

[81] R. Adaikkalavan and S. Chakravarthy, "Event Specification and Processing For Advanced Applications: Generalization and Formalization." in *DEXA*. LNCS 4653, Sep. 2007, pp. 369–379.

[82] F. Zoufaly, "Translation and execution of workflows using the eca paradigm, thesis proposal," Ph.D. dissertation, Database Systems Research Center, CISE Dept., University of Florida, Gainesville, Gainesville, FL, U.S.A, 1999. [Online]. Available: http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/zoufaly.pdf

[83] "The Workflow Management Coalition," Web Page, Dec. 2003. [Online]. Available: http://www.wfmc.org

[84] "StreamBase," Web Page. [Online]. Available: http://www.streambase.com/

[85] A. Adi and O. Etzion, "Amit - the situation manager," *VLDB J.*, vol. 13, no. 2, pp. 177–203, 2004.

[86] V. Garg, R. Adaikkalavan, and S. Chakravarthy, "Extensions to Stream Processing Architecture for Supporting Event Processing." in *DEXA*, Sep. 2006, pp. 945–955.

[87] Q. Jiang and S. Chakravarthy, "Data Stream Management System for MavHome," in *Proc. of ACM SAC*, Mar. 2004.

[88] S. Babu and J. Widom, "Continuous Queries over Data Streams," in *ACM SIGMOD RECORD*, Sep. 2001.

[89] D. Abadi *et al.*, "Aurora: A New Model and Architecture for Data Stream Management," *VLDB Journal*, vol. 12, no. 2, Aug. 2003.

[90] J. Chen *et al.*, "NiagaraCQ: A Scalable Continuous Query System for Internet Databases," in *Proc. of SIGMOD*, 2000.

[91] S. Madden and M. J. Franklin, "Fjording the Stream: An Architecture for Queries over Streaming Sensor Data," in *Proc. of ICDE*, 2002.

[92] M. F. Mokbel *et al.*, "PLACE: A Query Processor for Handling Real-time Spatio-temporal Data Streams," in *Proc. of VLDB*, Sep. 2004.

[93] R. Motwani *et al.*, "Query Processing, Resource Management, and Approximation in a Data Stream Management System," in *Proc. of CIDR*, Jan. 2003.

[94] Q. Jiang and S. Chakravarthy, "Scheduling Strategies for Processing Continuous Queries over Streams," in *Proc. of BNCOD*, Jul. 2004.

[95] V. K. Pajjuri, "Design and implementation of scheduling strategies and their evaluation in mavstream," Master's thesis, Information Technology Laboratory, CSE Dept., The Univ. of Texas at Arlington, 2004. [Online]. Available: http://itlab.uta.edu/ITLABWEB/Students/sharma/theses/Vamshi.pdf

[96] B. Brian *et al.*, "Chain: Operator Scheduling for Memory Minimization in Stream Systems," in *Proc. of SIGMOD*, 2003.

[97] D. Carney *et al.*, "Operator Scheduling in a Data Stream Manager," in *Proc. of VLDB*, Sep. 2003.

[98] N. Tatbul *et al.*, "Load Shedding in a Data Stream Manager," in *Proc. of VLDB*, Sep. 2003.

[99] B. Babcock, M. Datar, and R. Motwani, "Load Shedding for Aggregation Queries over Data Streams," in *Proc. of ICDE*, Mar. 2004.

[100] A. Das, J. Gehrke, and M. Riedewald, "Approximate Join Processing over Data Streams," in *Proc. of SIGMOD*, 2003.

[101] P. Bonnet, J. E. Gerhke, and P. Seshadri, "Towards Sensor Database Systems," in *Proc. of MDM*, Jan. 2001.

[102] Y. Yao and J. E. Gehrke, "Query Processing in Sensor Networks," in *Proc. of CIDR*, Jan. 2003.

[103] S. R. Madden *et al.*, "The Design of an Acquisitional Query Processor for Sensor Networks," in *Proc. of SIGMOD*, 2003.

[104] S. R. Madden *et al.*, "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks," *In Proc. of OSDI*, Dec. 2002.

[105] Q. Jiang, R. Adaikkalavan, and S. Chakravarthy, "Towards an Integrated Model for Event and Stream Processing," CSE Dept., The University of Texas at Arlington, Tech. Rep. CSE-2004-10, 2004.

[106] N. H. Gehani, H. V. Jagadish, and O. Shmueli, "Composite Event Specification in Active Databases: Model & Implementation," in *Proc. of VLDB*, 1992, pp. 327 – 338.

[107] N. H. Gehani, H. V. Jagadish, and O. Shmueli, "Event Specification in an Object-Oriented Database," in *Proc. of SIGMOD*, San Diego, CA, June 1992, pp. 81–90.

[108] R. Birgisson, J. Mellin, and S. F. Andler, "Bounds on Test Effort for Event-Triggered Real-Time Systems," in *International Workshop on Real-Time Computing and Applications Symposium*, 1999.

[109] Q. Jiang, R. Adaikkalavan, and S. Chakravarthy, "NFM^i: An Inter-domain Network Fault Management System," in *Proc. of the ICDE*, Tokyo, Japan, Apr. 2005, pp. 1036–1047.

[110] Q. Jiang, R. Adaikkalavan, and S. Chakravarthy, "MavEStream: Synergistic Integration of Stream and Event Processing." in *IEEE International Workshop on Data Stream Processing, ICDT*, Jul. 2007.

[111] "JavaBeans Technology," Sun Microsystems, Inc. [Online]. Available: http://java.sun.com/products/javabeans/

[112] J. Bailey, A. Poulovassilis, and P. T. Wood, "An Event-Condition-Action Language for XML," in *Proc. of the WWW*. ACM Press, 2002, pp. 486–495.

[113] M. Bernauer, G. Kappel, and G. Kramler, "Composite Events for XML," in *Proc. of the WWW*. ACM Press, 2004, pp. 175–183.

[114] G. Papamarkos, A. Poulovassilis, and P. T. Wood, "RDFTL: An Event-Condition-Action Language for RDF," in *Proc. of The Hellenic Data Management Symposium*, 2004.

[115] G. Papamarkos, A. Poulovassilis, and P. T. Wood, "Event-Condition-Action Rule Languages for the Semantic Web," in *Proc. of the International Workshop on Semantic Web and Databases, at the VLDB*, 2003, pp. 309–327.

[116] M. Zoumboulakis, G. Roussos, and A. Poulovassilis, "Active Rules for Sensor Databases," in *Proc. of the Workshop on Data management for Sensor Networks*. ACM Press, 2004, pp. 98–103.

[117] T. Terada, M. Tsukamoto, K. Hayakawa, T. Yoshihisa, Y. Kishino, A. Kashitani, and S. Nishio, "Ubiquitous chip: A rule-based i/o control device for ubiquitous computing." in *Proc. of the PerCom*, 2004, pp. 238–253.

[118] V. Kantere and A. Tsois, "Using ECA Rules to Implement Mobile Query Agents for Fast-Evolving Pure P2P Database Systems," in *Proc. of the MDM*, Ayia Napa, Cyprus, 2005, pp. 164–172.

[119] W. Wang, J. Yang, and R. Muntz, "STING+: An approach to active spatial data mining," in *Proceedings, International Conference on Data Engineering*. Sydney, Australia: IEEE Computer Society, 1999, pp. 116–125.

[120] "Common Information Model Schema - Event Model," Distributed Management Task Force, Inc. and WBEM Solutions, Inc. [Online]. Available: http://www.wbemsolutions.com/tutorials/CIM/cim-model-event.html

[121] S. Ross-Talbot, S. tibet, S. Chakravarthy, and G. Brown, "A generalized ruleml-based declarative policy specification language for web services (position paper)," in *Constraints and Capabilities Worksop*, 2004.