A flexible parameter-dependent Algebra for Event Notification Services

Annika Hinze, Agnès Voisard
Institute of Computer Science
Freie Universität Berlin, Germany
{hinze,voisard}@inf.fu-berlin.de

Abstract

Event notification services, or alerting services, are used in various applications such as digital libraries, stock tickers, traffic control, or facility management. However, to our knowledge, a common semantics of events in event notification services has not been defined so far. In this paper, we propose a parameterized event algebra which describes the semantics of composite events for event notification systems. We define the event operators that form composite events and we introduce parameters for event instance selection and event instance consumption. These parameters serve as a support for handling duplicates in both primitive and composite events. The event algebra is exemplified in the domain of transportation logistics.

1 Introduction

Event Notification Services (*ENS*) are used in various applications such as digital libraries, stock tickers, logistics, traffic control, collaborative work, facility management, remote monitoring, security, and project control. They have gained increasing attention in the past few years. Several systems have been implemented, such as Le Subscribe [25] or NiagaraCQ [9]. An event notification service informs its users about new events that occurred on providers' sites. User interests are defined by means of *profiles*, which may consist of queries regarding primitive (atomic) events, their time and order of occurrence, and of composite events, which are formed by temporal combinations of events. Profiles are defined using a *Profile Definition Language for Alerting* (*PDLA*).

A profile consists of two parts, a query part and a parameter part. The query part, also called query profile, defines the periodically evaluated query (similar to a search query); the parameter part holds additional information about the user. In this paper we only consider query profiles: The parameter part is left out as it contains no valuable information for the event semantics.

The query part of a profile is matched against the event descriptions. Users can be interested in primitive (atomic) events or in event combinations. Primitive events can be described, e.g., by either attributes (e.g., temperature) or time of event or duration of a certain state of an object, or any combination of those. An object can be for instance a sensor or a web page. Examples of query profiles borrowed from a logistic application are:

QP1: *Notify if a traffic jam occurred and if one of our trucks is affected*

QP2: *Notify if a customer cancelled an order a given number of times within a month.*

These two examples show that, in contrast to other fields such as information retrieval or database query languages, a PDLA has to consider time restrictions and resulting time-dependent parameters. In this paper, we use an event algebra to describe the events that are filtered by an ENS and that can be subscribed to via profiles. The events entering an event notification system are filtered according to user profiles. This event matching is based on the filter semantics defined for the service. Profiles defined by users are restricted by the filter capabilities. Therefore, the semantics of the event filtering and the one of the profiles are closely related.

The usability and power of ENS heavily depend on the expressiveness of the profile definition language. Various languages have been implemented in this context; some are rather simple and are based on the Boolean model, e.g. [25]. Others are more sophisticated and use, for instance, an SQL-like syntax [21], or XML-QL [9].

In addition to the fact that only few languages are defined for event notification services, the evaluations of languages that seem to follow similar semantics do not always lead to similar results. One example is the handling of duplicate events (similar events occurring at different times): depending on the application field and on the implementation, in the filtering process, duplicates may either be skipped or kept. The definition of composite events often relies on Boolean operators, e.g., in the Cambridge model [5]. This is not sufficient as the temporal semantics of, for instance, an AND-expression in a distributed environment remains unclear. We extend the semantics of well-known ECA rules of active databases by introducing the notion of *relative time*. Since ENS systems are used in the context of a distributed environment and cannot rely on a transactional context as in the case of active database systems, the simultaneous occurrences of events cannot be determined accurately. Therefore, all composite event operators should be handled as temporal operators and extended by a relative time frame (similar to time handling in distributed systems).

Because the semantics of temporal operators as introduced, for instance, in [3] is not defined in a uniform manner in the numerous application areas, the approach described in this paper supports various perceptions. This is achieved through the introduction of a flexible semantics which is controlled by a set of parameters. The parameterized event algebra handles temporally composite events. The algebra is presented in two steps. In a first step, we informally describe the event operations while in a second step we formally propose the parameterized definitions.

This paper is organized as follows. Section 2 gives an overview of necessary background information. After introducing an application scenario which will serve as a reference throughout the paper, we shortly describe basic concepts and related work. In Section 3 we informally introduce our event algebra which allows one to describe the event operators. We then show (Section 4) the needs to define additional parameters and we introduce the parameterized algebra. In Section 5, the algebra is applied in the logistics example, which illustrates the influence of various parameter settings. Finally, Section 6 addresses concluding remarks and gives some directions for future work in this domain.

2 Background

This section is devoted to our context of study. It first describes our running example, borrowed from the field of transportation logistics. The basic event-based concepts are then introduced. A panorama of related work ends this section.

2.1 Application Scenario

Let us consider a company that sells various goods to its customers. The goods are stored in a warehouse and are delivered to the customers by trucks T1, T2, and T3 (see Figure 1). During the night the trucks are kept in a garage. Each morning, a truck driver has a delivery list for the day, which is based on several scheduling and loading restrictions, such as convenient delivery time for customers or load balancing. He or she picks up the goods from the warehouse and follows the delivery list. All trucks have to be back at the garage at the end of the day.

Figure 1 depicts the following scenario. On a given day, customers A - I expect deliveries. The customers' respective locations are also shown in the figure. Let us assume that each delivery takes in the average 30 min. The delivery lists that must be followed by the various trucks are also given in the figure.

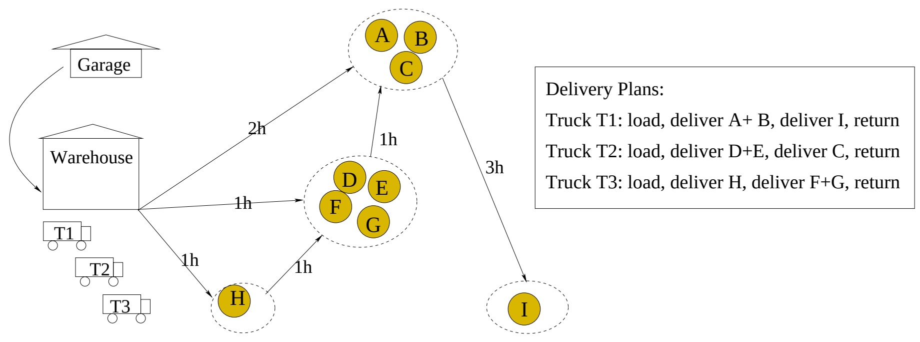

Garage
Warehouse
T1
T2
T3
2h
1h
1h
1h
1h
3h
A
B
C
D
E
F
G
H
I
Delivery Plans:
Truck T1: load, deliver A+ B, deliver I, return
Truck T2: load, deliver D+E, deliver C, return
Truck T3: load, deliver H, deliver F+G, return

Figure 1: Logistics Scenario

The actors we consider in our example are customers, controllers, analysts, and drivers. Possible events include the start and arrival of each truck, the deliveries themselves, accidents of the trucks, traffic information, and delivery cancellations. The following query profiles are defined:

P1: Notify the controller if a traffic jam occurred and one of the trucks is at that time ($\pm 5min$) in that area.

P2: Notify the analyst if a customer cancelled an order three times within a month.

P3: Notify customer I if T1 leaves from A or B for I after 2pm.

P4: Notify the controller when all trucks are back.

P5: Notify the controller if goods have not been picked up 2 hours after the start of the shift.

P6: Notify the controller every 30 min about the location of each truck.

P7: Notify the driver of T1 if customer A, B, or I cancels the delivery.

P8: Notify the driver of T2 if customer C, D, or E cancels the delivery.

2.2 Concepts

An *event* is the occurrence of a state transition at a certain point in time. In contrast to states, events have no duration. Events may be state changes in databases, signals in message systems, or real-world events such as the departures and arrivals of vehicles. Events can be described as collections of (attribute,value) pairs, such as the 3 pairs in the following event: `Event (name = truck-location, truck-number = T2, location = (x1,y1))`. To keep it simple, attribute types (for attributes `name, truck-number`, and `location`) are not considered here.

We are aware of the problem of temporal order in distributed systems. Time is a rich concept that encompasses many notions. Each event has a timestamp reflecting its occurrence time. Timestamps are defined within a time system based on an internal clock. For the sake of simplicity, we assume here that all occurring events can be ordered sequentially in a global system of reference (e.g., by applying the 2g-precedence model of [27] or using the NTP protocol [22]) and that a mapping for real time references has been defined. The system of reference for the time is discrete.

We consider *primitive events* and *composite events*, which are formed by combining primitive and composite events. We further distinguish two types of primitive events: *time events* and *content events*. Time events describe the occurrence of a certain point in time (e.g., 5 o'clock). Content events involve changes of object states in general.
We do not discuss the definition of primitive events in greater detail. Various approaches are conceivable and implemented based, for instance, on the definition of keywords (cf. digital libraries) or on the definition of (attribute,value) pairs. In this paper, we use (attribute,value) pairs for the examples. An example for a primitive event at a simple temperature sensor is then, e.g.,

$$e_1 = event(sensor = xyz, \tag{1}$$

$$temperature = 40°C) \tag{2}$$

We distinguish *events instances* from *event classes*. The former are denoted event in this paper. An event class is defined by a set of event properties. Even though events of the same event class share some properties (e.g., temperature of a sensor), they may differ in other event attributes (e.g., location). An event class could be, for instance, all events that describe the delivery of goods to a customer. Then an event is the actual delivery of a package to customer A at a certain time.

User profiles specify event classes, e.g.,

$$p_1 = profile(temperature > 35°C) \tag{3}$$

Events (instances) are denoted by lower Latin e with indices, i.e., $e_1, e_2, \ldots$, while event classes are denoted by upper Latin E with indices, i.e., $E_1, E_2, \ldots$. The fact that an event e_i is an instance of an event class E_j is denoted *membership*, i.e., $e_i \in E_j$. This relationship is non-exclusive, i.e., $e_i \in E_j$ and $e_i \in E_k$ is possible even with $E_j \neq E_k$. Event classes may also have subclasses, so that $e_i \in E_j \subset E_k$. The timestamp of an event $e \in E_1$ is denoted $t(e)$.[1]
Based upon a notation used, e.g., in [7], the matching operator is defined as follows:

Definition 2.1 (Profile Matching $\sqsubset$) *Consider the event e and a given profile p. It is said that e matches p, denoted $p \sqsubset e$, if all properties of the profile and the event match.*

[1]The time of events that do not occur is set to ∞.

Profiles can also contain wildcards and other operators such that not all attributes of the event have to be defined exactly. The exemplary event e_1 matches the profile p_1.

2.3 Related Work

Event specification semantics have been developed in various research areas, such as active databases, temporal or deductive databases, temporal data mining, time series analysis, and distributed systems.

In the area of active database systems, the problem of event rule specification has been evaluated for several years (see, e.g., [8]), also with special focus on composite events [15, 16, 19, 14, 31, 30] and temporal conditions (e.g.,[14, 11, 12]

Active database systems can rely on the transactional context for the composition of events. Trigger conditions can be defined based on the old and new state of the database, thus using the concept of states rather than describing the event itself. For the ordering of database states, the temporal interval operators as defined in [3] can be used (see [23]). Ordering based on events, as opposed to states, has been implemented in the SAMOS system [14]. The work of Zhang and Unger [26] is closely related to our approach, however, parameters and time frames are missing.

The problem of temporal combination of events is also addressed in temporal and deductive databases. In these areas, various approaches have been introduced, such as temporal extensions of SQL [28, 29] and a temporal relational model and query language [24]. In contrast to ENS, however, temporal and deductive databases focus on ad-hoc querying. That is, there is no periodical evaluation of a query.

The areas of temporal data mining and time series analysis rely on temporal association rules [1, 2, 10]. From a set of data, rules verified by the data themselves have to be discovered. While similar event operations are evaluated, the approaches differ greatly from event filter semantics discussed in this paper. In event notification services, event combinations are given and the matching set of data is to be found, while in temporal analysis the data are given and the rules have to be derived. The problem of determining the event order based on incorrect timestamps has been studied in [17, 6, 30]. Finally, time systems in distributed environments have been addressed for instance in [20, 27].

3 Event Algebra

This section describes our event algebra in an informal manner. Its use is illustrated in our toy application from Section 2. The set of operators included in the algebra is derived from the profile language considerations of [18]. An event algebra is an abstract description of the event concept of a service independent of the actual profile definition language. It enables the evaluation of the complexity of different services, supports the detection of inconsistent profile definition, and can serve as basis for an abstract implementation of a matching algorithm. In order to model composite events, we employ event constructors (also called operators). We extend the event algebra for active database systems introduced in [13] in order to consider the temporal demands on event compositions as well. Without loss of generality we look for the simplest combinations of events, namely pairs. The events e_1 and e_2 used in the definitions below can be any primitive or composite event, $t()$ refers to occurrence times, t in $(\ldots)_t$ denotes time spans.

Temporal Disjunction: The *disjunction* $(e_1|e_2)$ of events occurs if e_1 or e_2 occurs. As mentioned before, the composite event $e_3 := (e_1|e_2)$ also has a time of occurrence, which is the time of the occurrence of the first one of either e_1 or e_2: $t(e_3) := min\{t(e_1), t(e_2)\}$.

Temporal Conjunction: The *conjunction* $(e_1, e_2)_t$ occurs when both e_1 and e_2 have occurred, regardless of the order. The conjunction constructor has a temporal parameter that describes the maximal length of the interval between e_1 and e_2.[2] The time of the composite event: $e_3 := (e_1, e_2)$ is the time of the later event: $t(e_3) := max\{t(e_1), t(e_2)\}$.

Temporal Sequence: The *sequence* $(e_1; e_2)_t$ occurs when first e_1 and then e_2 occurs. Again, t defines the temporal distance of the events. The time of the event $e_3 := (e_1; e_2)$ is equal to the time of e_2: $t(e_3) := t(e_2)$.

Temporal Negation: The *negation* $\overline{e}_t$ defines a negative event over an interval; it means that e does not occur for a given interval $[t_{start}, t_{end}], t_{end} = t_{start} + t$ of time. The occurrence time of $\overline{e}_t$ is the point of time at the end of the period, $t(\overline{e}_t) := t_{end}(\overline{e}_t)$.

Temporal Selection: The *selection* $e^{[i]}$ defines the occurrence of the i^{th} event of a list of events, $i \in \mathbb{N}$.

The model of composite events consists of (primitive or composite) events combined through event constructors. We additionally permit the Boolean operators of logical conjunction ($\wedge$) and logical disjunction ($\vee$) in order to refer the same event instance more than once:

Logical Disjunction The *logical disjunction* $e_{c1} \vee e_{c2}$ describes that one of the given alternatives has to apply. The event time of $e = (e_1; e_2)_{t_1} \vee (e_2; e_3)_{t_2}$ is $t(e) := min\{t((e_1; e_2)_{t_1}), t((e_2; e_3)_{t_2})\}$

Logical Conjunction The *logical conjunction* $e_{c1} \wedge e_{c2}$ of event compositions e_{c1}, e_{c2} requires that all subclauses need to be fulfilled. The event time of $e = (e_1; e_2)_{t_1} \wedge (e_3; e_4)_{t_2} \wedge (e_1; e_3)_{t_3}$ is $t(e) = max(t((e_1; e_2)_{t_1}), t((e_3; e_4)_{t_2}), t((e_1; e_3)_{t_3}))$ and is determined by the temporal event constructors.

Note that logical combinations of event compositions describe relationships among the terms. The logically-combined terms form a name-space (i.e., equal names such as e_1 identify the same event), whereas equal names combined by event constructors only define identical event descriptions and therefore a class of events.

Example 3.1 *The profile examples given above can be modelled using the event algebra as follows:*

P1: *Notify the controller if a traffic jam occurred and if one of the trucks is at that time ($\pm 5min$) in that area: Let E_1 be the class of traffic-jam events in city area A. Let E_2 be the class of all events regarding the location sensor of our trucks, and $E_2^A \subset E_2$ the subclass of truck location events in A. We then have to observe the composed event $e = (e_1, e_2)_{5min}, e_1 \in E_1, e_2 \in E_2^A$.*

[2] $(e_1, e_2)_\infty$ refers to an event composition no matter the time of the composing events. It is equivalent to the original conjunction constructor as defined, e.g., in [13].

P2: *Notify the analyst if a customer cancelled an order three times within a month: Let E_3 be the class of cancelled orders of a particular customer. A simplified definition for the composite event could be expressed as $e = ((e_{31}, e_{32})_t, e_{33})_{4weeks-t}$ with $e_{31}, e_{32}, e_{33} \in E_3$.*

P3: *Notify customer I if T1 leaves A and B for I after 2pm: Let E_{2pm} be the class of time-events occurring at 2pm. Let E_4 be the class of leaving-events for truck T1, $E_4^A \subset E_4$ and $E_4^B \subset E_4$ subclasses with leaving-events regarding customer A and B, respectively. The composite event is then defined as $e = (e_t; (e_{41}, e_{42})_\infty)_\infty, e_t \in E_{2pm}, e_{41} \in E_4^A, e_{42} \in E_4^B$.*

P4: *Notify the controller when all trucks are back: Let E_5, E_6 be two classes of events describing the start and return of trucks, respectively. Then for each truck we could define $e = (e_5; e_6)_{8h}$ with $e_5 \in E_5, e_6 \in E_6$.*

P5: *Notify the controller if goods have not been picked up 2 hours after the start of the shift. Let E_7 be the class of events of loading goods, E_{10am} the class time-events defining the start of the shift. Then the composite event is defined as $e = (e_t; (\overline{e}_7)_{2h})_{2h}$, $e_t \in E_{10am}, e_7 \in E_7$.*

4 The Event Algebra Revisited

The event algebra presented informally in the previous section does not define the complete semantics of a profile definition language. The semantics of, for instance, the sequence operation does need further refinement as illustrated below.

This section sets the basis for our profile definition language for alerting, the kernel of our contribution. We believe that for the temporal event operators different (application-dependent) semantics are conceivable. We therefore introduce a *parameterized* event algebra. This section first motivates the different parameters for event instance selection and consumption. It then illustrates examples of different parameter settings.

4.1 Parameters to consider

The following examples illustrates the weakness of the approach given in Section 3:

Example 4.1 *Let us assume that one is interested in the sequence of two events $(e_1; e_2)_t$ as defined, for instance, in profile P4. If we consider the following history (trace) of events: $tr = \langle e_1, e_1, e_2, e_2 \rangle$, it is not automatically clear from the event definition which pair of events fulfills our profile. Candidate pairs are the inner two events, or the first and the third. It is also not clear whether the profile can be matched twice, e.g., by pairs (2,3) and (1,4), or by (1,3) and (2,4).*

Various event combinations may be possible. Besides, for different applications, different event-history evaluations could be applied.

Example 4.2 *If we are interested in any fourth occurrence of an event e_1 we define the profile: (e_1, e_1, e_1, e_1). Considering the (synthetical) trace $tr = \langle e_1, e_1, e_1, e_1, e_1, e_1, e_1, e_1 \rangle$ the profile could be matched by event number 4, or by 4 and 8, or event by 4,5,..., since all of them are preceded by three e_1-instances.*

Similarly to the terminology used in active databases [31], for an event algebra for ENS we need to identify the following modes:

1. *Event selection principle*: how to identify primitive events based on their properties
2. *Event instance pattern*: which event operators form composite events
3. *Event instance selection*: which events qualify for the complex events, how are duplicated events handled
4. *Event instance consumption*: which events are consumed by complex events

We do not consider different event selection principles here, and for the sake of simplicity we assume (attribute,value) pairs to describe events. Event instance patterns have been introduced in Section 3. In the following sections we introduce modes for event instance selection and consumption. Event selection and consumption in ENS cannot be handled independently, as proposed in [31].

Event Instance Selection and Duplicate Events Duplicates of events are event instances that belong to the same event class. Duplicates have to be handled differently depending on the application and even on the context within the application.

Examples of events include the reading of a sensor at different times, such as the location of a truck in our example. For the truck locations the latest sensor reading is the valid one and earlier readings are overwritten.

The delivery events also belong to a single class of events. Delivery events are duplicates, however, each of these events having to be considered for further planning, duplicates in this case should not be ignored.

If for some reasons a customer cancels an order twice, the duplicate event can be ignored. Selecting the i^{th} duplicate would be the equivalent of application of the selection-operator as introduced in Section 3. The variants for the instance selection parameter for the algebra are shown in Figure 2.

Event Instance Identification and Consumption We distinguish the variations in the identification of composite events. The possible decisions are shown in Figure 3. Matched events can be consumed by the composite event or they can contribute several times to composite events of the same class. If matched events are consumed, only unique composite events are supported. If the filter is applied more than once, a primitive event can participate in several composite events of the same type. Let us get back to our delivery example. If we are interested in the fact that all trucks are back in the evening, the profile is defined as the sequence of the events *truck X starts* and *truck X arrives*. In this case we are only interested in unique pairs of start/arrive events, but not in all combinations of all start and arrive events of the month, for instance.

If events are consumed by composite events, the filtering process could be reapplied after unique composite events have been identified. This approach can be seen as a combination of the two parameters event instance selection and consumption. The next subsection describes examples for the combinations of these two parameters.

4.2 Profile-Event Situations

Figure 4 shows a matrix of profile-event situations, which illustrates the implications of the parameters introduced above. Note that the names for the rows and columns are

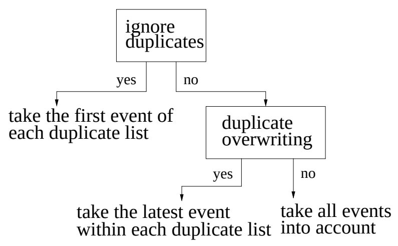

ignore duplicates
yes
no
take the first event of each duplicate list
duplicate overwriting
yes
no
take the latest event within each duplicate list
take all events into account

Figure 2: Event instance selection

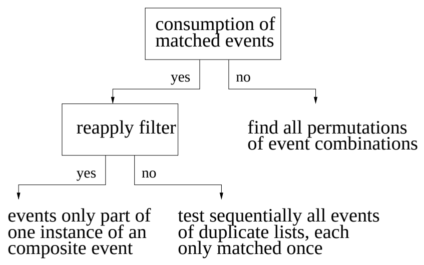
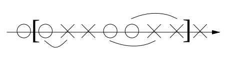
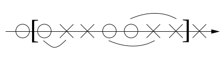
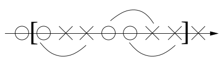
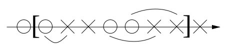
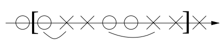
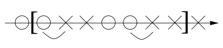
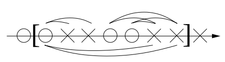
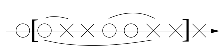
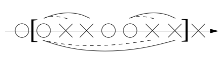

consumption of matched events
yes
no
reapply filter
find all permutations of event combinations
yes
no
events only part of one instance of an composite event
test sequentially all events of duplicate lists, each only matched once

Figure 3: Event instance identification

simplified descriptions of the different approaches. The corresponding accurate definition follows shortly after. The examples shown in the matrix refer to the composite event of a sequence $(e_1; e_2)$ in a given exemplary trace of events. The events are referred to in the figure as $e_1 = \circ$ and $e_2 = \times$, each composite event instance is marked with an arc. We use here fixed time frames as evaluation intervals so that the different implications are easier to compare.

event instance identification \ event instance selection	take all events	take first event of each duplicate list	take last event of each duplicate list
consume matched events reapply filter			
consume matched events			
keep matched events			

[ ]: time interval ○: event type e1 ×: event type e2

Figure 4: Profile and event trace example under different evaluation conditions

The horizontal dimension of the matrix shows the event selection. The selection either takes all events into account, or only the first or the last duplicate in a trace. It is important to note that the word ”duplicate” is used here with respect to a profile and not for the event instance itself. Two events can be different, nevertheless, they may match the same profile. With regards to that particular profile they are seen as duplicates.

From the matrix we can already derive exemplary applications for the different approaches. Taking all events is only useful in an application where no instance duplicates can occur, e.g., security systems where any event has to be recorded and analyzed. This is also interesting in applications where information about changes is crucial (e.g., share value raised by 5%), or in a digital library application that delivers new articles.

Taking the first event of a sequence of duplicates is used in applications where duplicates do not deliver new information, e.g., whether the value of a sensor reading goes beyond a certain threshold. In such cases, only the first event delivering the information about a change needs to be evaluated. Taking the last event of a sequence of duplicates is useful in applications that handle, for instance, status information about different sources. The temperature control of a building works on the basis of scheduled sensor readings. In this case the last reading shows the current value.

The examples above clearly illustrate that the appropriate semantics and the various possible profiles are heavily application dependent.

The vertical dimension of the matrix shows different versions of profile applications: apply to all events, apply to only the unmatched events, and reapply after a first profile match. The last approach can lead to a successive matching of possibly all events in a duplicate list (see first/last event of duplicate list). Here, sequences of matching pairs can be found.

Considering all possible event combinations in a given series (also keeping matched ones) results in sets of composite events that have single events in common. This applies for instance in scenarios where each event itself represents a set of events. Examples include trucks delivering goods to customers, where a set of goods is loaded in the morning but the unloading is realized by several events. In this case, the starting event is a combination of several simple events *load product on truck*, which can be seen as factorized.

Discarding matched events ensures that each event only takes part in one composite event of a certain type. This approach is sufficient for applications where single event pairs have to be found and where no implicit event combinations occur, such as personal ID systems for security purposes, with personalized cards that have to be checked in and out if entering or leaving the building.

The combination of reapplying the filter after discarding matched events is used, e.g., for parsers and compilers. A sensible application is an event-based XML-validator, as proposed in [4]. With this method interleaving event pairs can be identified.

4.3 Parameterized Semantics

In this section we formally define our parameterized event algebra. We first introduce the terminology.

Definition 4.1 (event space) *The set of all possible events known to a certain system is called the event space $\mathbb{E}$. The set of all time events is denoted $\mathbb{E}_t$.*

A trace, or history of events, is defined as follows:

Definition 4.2 (trace) *A trace tr_{t_1,t_2} is a sequence of ordered events $e \in \mathbb{E}$ with defined start- and end-points t_1, t_2 respectively.*

The history of events a service processes is then $tr_{t0,\infty}$ with $t0$ being the point in time the service started observing events.

As a trace behaves essentially as a list, we can use the operations commonly defined for lists. For each list we apply an arbitrary local order as defined above, that assigns an index-number $i \in \mathbb{N}$ to each event. The elements of a list L can then be accessed by their index-number, and $L[i]$, $i \in \mathbb{N}$ retrieves the i^{th} event of the list.

All profile evaluations start after the profile has been defined. For each positively evaluated event e_1 therefore holds implicitly $t(e_1) > t(profile)$.

The semantics of negative events is defined as follows:

Definition 4.3 (negative events) *Let e_1 be an event $e_1 \in \mathbb{E}$, e_t a time event $e_t \in \mathbb{E}_t$, t_1 a time span given as $t_1 \in \mathbb{R}$ based on a certain time system then*

$$(\overline{p_1})_{t_1} \sqsubset e_t \rightarrow \{\nexists e_1 \in \mathbb{E} : t(e_1) \in [t(e_t) - t_1, t(e_t)]\}$$

Then the set of negative events for a given trace is defined as

$$(\overline{p_1})_{t_1}(tr) \sqsubset \{e_t | e_t \in tr, (\overline{p_1})_{t_1} \sqsubset e_t\}$$

Definition 4.4 (trace view) *Let E_1 be a class of events. The subset $tr(E_1)$ of a given trace tr is defined as the list of continuously ordered events that contains only events $e \in E_1$. We call this subset a trace view.*

The trace view $tr(E_1, E_2)$ contains all $e_1 \in E_1, e_2 \in E_2$ with $e_1 \in tr$ and $e_2 \in tr$. We also use the shorthand notation $tr(e_1, e_2)$. Note that the events in $tr(E_1)$ keep all their attributes including occurrence time, but obtain a new index-number.

We now define a re-numbering on the list tr:

Trace Renumbering The list is subdivided into disjunct sublists $tr[1], \ldots, tr[n]$ each containing successive events of identical types. Every element of such a sublist is denoted with $tr[x, y]$, where $x \in \mathbb{N}$ is the number of the sublist and $y \in [1, length(tr[x]))$ is the index-number of the element within the sublist.

The length of the sublists is defined as the number of list elements. Disjunct sublists containing only similar events are referred to as duplicate lists. Informally duplicate lists are often called ordered duplicate lists:

Duplicate List Let E_1, E_2 be two event classes with $E_1 \neq E_2$. We then define a duplicate list $D_{E_1 \backslash E_2}$ as the ordered list of events of class E_1 that occur in a trace tr without any events of class E_2 in between.

$$D_{E_1 \backslash E_2}(n) = tr[n] \tag{4}$$

such that for $e_1 \in E_1$, $e_2 \in E_2$ holds

$$e_1 \in tr, \nexists e_2 \in tr : t(e_2) \in (t(tr[n, 1]), t(tr[n, length(tr[n])])) \tag{5}$$

The $n \in \mathbb{N}$ defines an ordering on similar duplicate lists.

Note that duplicate lists are subject to changes as long as the *closing* event did not occur.

Example 4.3 *Let us consider the following trace of events: $tr = \langle e_1, e_3, e_1, e_1, e_3, e_2, e_2, e_1, e_1, e_2, e_2, e_2\rangle$. The (e_1, e_2)-trace view is then defined as $tr(e_1, e_2) = \langle e_1, e_1, e_1, e_2, e_2, e_1, e_1, e_2, e_2, e_2\rangle$.*

The renumbering results in $tr(e_1, e_2) = \langle tr[1,1], tr[1,2], tr[1,3], tr[2,1], tr[2,2], tr[3,1], tr[3,2], tr[4,1], tr[4,2], tr[4,3]\rangle$ with $tr[1] = \langle e_1, e_1, e_1\rangle$, $tr[2] = \langle e_2, e_2\rangle$, $tr[3] = \langle e_1, e_1\rangle$, and $tr[4] = \langle e_2, e_2, e_2\rangle$. Obviously, the list-length of the first sublist follows with $length(tr[1]) = 3$. The example is depicted in Figure 5.

Note that we denote (unordered) sets of events with $\mathbb{E}$ or $\mathbb{E}_t$ while $tr[]$ denotes lists (ordered sets with possible duplicates) of events. Without loss of generality, we assume $tr(e_1, e_2)$ to start with $tr[1, 1] = e_1$.

For the following operator definitions, the values of the parameters P_{xy}, w_{min}, w_{max}, z_{min}, z_{max} substantially influence the operator semantics. We therefore introduce different semantical approaches.

The disjunction implements a selection based on occurrence time (or), no exclusion (xor):

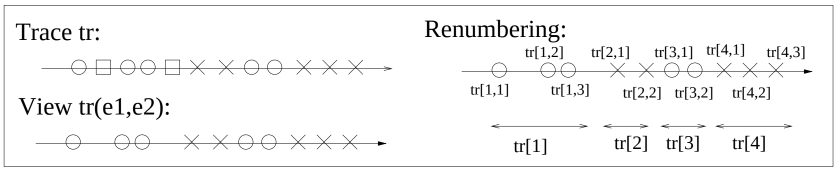


Figure 5: Trace and Renumbering in Example 4.3

Definition 4.5 (disjunction of events) *Let us consider an event $e \in \mathbb{E}$, then*

$$(p_1|p_2) \sqsubset e \rightarrow \quad \{((p_1) \sqsubset e \vee p_2 \sqsubset e\}$$

The set of matching events of a given trace tr is then defined as

$$(p_1|p_2)(tr) \sqsubset \{tr[x,z]|tr[x,z] \in tr(e_1,e_2), (p_1|p_2) \sqsubset tr[x,z], \\ \forall x \in [1,\infty), \forall z \in [z_{min}, z_{max}]\}$$

The matching set of the disjunction includes all event instances that match either profile. Different values for the open parameters z_{min}, z_{max} are discussed subsequently.

The conjunction profile $(p_1,p_2)_t$ is matched by the set of events $(\{e_1,e_2\})$. We define the semantics of a conjunction of events as follows:

Definition 4.6 (conjunction of events) *If two events $e_1, e_2 \in \mathbb{E}$ form a conjunction, the following condition holds for a given time span $t \in \mathbb{R}$:*

$$(p_1,p_2)_t \sqsubset (e_1,e_2) \Rightarrow \{p_1 \sqsubset e_1, p_2 \sqsubset e_2, |t(e_2) - t(e_1)| \leq t\}$$

The set of matching events of a given trace tr is then defined as

$$(p_1,p_2)_t(tr) \sqsubset \{(tr[x,z], tr[y+1,w])| \quad tr[x,z], tr[y+1,w] \in tr(e_1,e_2), \\ (p_1,p_2)_t \sqsubset (tr[x,z], tr[y+1,w]), \forall x \in [1,\infty), \forall y \in [1,\infty), \\ \forall w \in [w_{min}, w_{max}], \forall z \in [z_{min}, z_{max}] \wedge P_{xy}\}$$

Different values for the open parameters P_{xy}, w_{min}, w_{max}, z_{min}, z_{max} are discussed subsequently.

Definition 4.7 (sequence of events) *If two events $e_1, e_2 \in \mathbb{E}$ form a sequence the following condition holds for a given time span $t \in \mathbb{R}$:*

$$(p_1;p_2)_t \sqsubset (e_1,e_2) \Rightarrow \{p_1 \sqsubset e_1, p_2 \sqsubset e_2, t(e_2) \in (t(e_1), t(e_1)+1]\}$$

The set of matching events of a given trace tr is then defined as

$$(p_1;p_2)_t(tr) \sqsubset \{(tr[2x-1,z], tr[2y,w])| \quad tr[2x-1,z], tr[2y,w] \in tr(e_1,e_2), \\ (p_1;p_2)_t \sqsubset (tr[2x-1,z], tr[2y,w]), \forall x \in [1,\infty), \forall y \in [1,\infty), \\ \forall w \in [w_{min}, w_{max}], \forall z \in [z_{min}, z_{max}] \wedge P_{xy}\}$$

Different values for the open parameters P_{xy}, w_{min}, w_{max}, z_{min}, z_{max} are discussed subsequently.

Semantical Variations We now evaluate different approaches for the parameter values, which implement different semantics of the operators. As motivated in an earlier section, we distinguish two dimensions: the composition of matching pairs and the selection of events with duplicate lists.

For the composition rules for pair matching we distinguish the selection of unique pairs (each event in the matching set participates in one pair only) and the selection of all pairs:

Unique pairs:	$P_{xy} : x = y$
All pairs:	$P_{xy} : x \leq y$

For the second dimension, we distinguish several variations to select events from duplicate lists. Each has to be evaluated differently, depending on the position of the event relative to the binary operator. We use the notation *anterior* and *posterior* to refer to the two operators, tr_{ant} and tr_{post} denote the respective duplicate lists.

	anterior	posterior
first event	$z_{min} = z_{max} = 1,$	$w_{min} = w_{max} = 1$
i^{th} event	$z_{min} = z_{max} = i,$	$w_{min} = w_{max} = i$
last event	$z_{min} = z_{max} = length(tr_{ant})$,	$w_{min} = w_{max} = m$
take all events	$z_{min} = 1, z_{max} = length(tr_{ant})$	$w_{min} = 1, w_{max} = m$

with $m \in \mathbb{N} : \forall j > m : t(tr_{post}[., j]) > t(tr_{post}[., .]) + t$, where the dots are placeholders for the respective values, $t \in \mathbb{R}$ as defined for the operator.

The selection of the i^{th} event is a (somewhat artificial) generalization of the preceding one. The different cases can be combined, taking as posterior event every first in a duplicate list and as anterior event the respective last duplicate match.

The third approach as depicted in the upper left picture in Figure 4, is a combination of the already introduced dimensions. We consider unique pairs only, but reapply the filter until all matches are found. The first matches are, e.g., first/last events of duplicate lists, the second match are second/next-to-last events, and so forth. Other combinations are plausible. We only show below two intuitive examples.

P_{xy}:	$x = y$
first event	$z_{min} = 1,$
	$z_{max} = min(length(tr_{ant}), length(tr_{post}))$
	$w_{min} = 1,$
	$w_{max} = min(length(tr_{ant}), length(tr_{post}))$
last event	$z_{min} = length(tr_{ant}),$
	$z_{max} = length(tr_{ant}) - min(m, length(tr_{ant}))$
	$w_{min} = m,$
	$w_{max} = m - min(m, length(tr_{ant}))$

While the sequential variations support the selection of the i^{th} event within duplicate lists the *selection* operates on the matching set.

Definition 4.8 (selection) *We assume the existence of a matching set* $E \subset \mathbb{E}$, $E = \{tr[x, z] | \forall x \in [1, \infty), \forall z \in [z_{min}, z_{max}]\}$ *for a given profile as defined above. We then define an arbitrary order* $o : E \to \mathbb{N}$ *on all events* $e \in E$ *based on the order of the occurrence times so that for* $e_1, e_2 \in E$ *follows*

$$t(e_1) \leq t(e_2) \Rightarrow o(e_1) < o(e_2)$$

Then for $i \in \mathbb{N}, i \leq |E|$

$$p^{[i]} \sqsubset e \in E, o(e) = i$$

The issue of order and time in a distributed environment is crucial and has to be considered for an implementation of this operators.

Evaluation Time. Here we defined the complete sets of matching events without considering the evaluation time. Obviously, the result of a profile evaluation over a trace heavily depends on the time of evaluation: The very last event within a duplicate list might only be known at the end of the observation interval. It is assumed that event matches including last duplicates are evaluated after the evaluation interval is closed. In this case the continuous evaluation of all incoming events offers the advantage of early notification. Here, a fast but probably incorrect information is delivered in opposition to the correct but later information after the ending of the time frame. An example is shown in the lower right case in Figure 4 (dashed lines). This approach is appropriate in several applications, e.g., catastrophe warning systems for environmental surroundings or other systems for urgent information delivery (for an analysis of information correctness see [17]).

5 Application Examples

In this section we briefly discuss the parameters for some of the profiles defined in the context of our logistics application. We first itemize the event classes as defined in Section 3:

E_1 class of traffic-jam events in city area A,

E_2 class of all events regarding the location sensor of our trucks, with $E_2^A \subset E_2$ the subclass of truck location events in A,

E_3 class of cancelled events of a particular customer

E_4 class of leaving-events for truck T1, with $E_4^A \subset E_4$ and $E_4^B \subset E_4$ subclasses of leaving-events regarding customer A and B, respectively.

E_{2pm} class of time-events occurring at 2pm.

The parameters for profile P1 - P3 are defined follows:

P1: Notify if a traffic jam occurred and if one of the trucks is at that time ($\pm 5min$) in that area.

- Composite event description: $e = (e_1, e_2)_{5min}, e_1 \in E_1, e_2 \in E_2^A$.
- Instance Consumption: We want to be notified about all occurrences of the composite event: all trucks in traffic jam areas($P_{xy} : x \leq y$).
- Instance Selection: For the truck location events we consider the last event in the duplicate groups ($z_{min} = z_{max} = length(tr_{post})$ or $w_{min} = w_{max} = m$ as defined in Section 4) The events are evaluated continuously. Traffic jam events are all considered, since several traffic problems can occur within the same area, and all of them have to be taken into account ($z_{min} = 1, z_{max} = length(tr_{post}$ or $w_{min} = 1, w_{max} = m$).

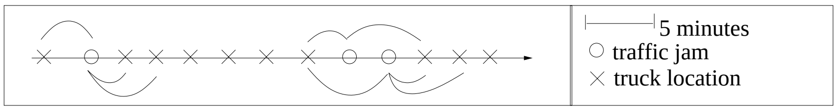


Figure 6: Trace and evaluation example for Profile P1

An exemplary trace and the associated profile evaluation is given in Figure 6.

P2: Notify if a customer cancelled an order three times within a month.

- Composite event description: $e = ((e_{31}, e_{32})_t, e_{33})_{4weeks-t}$ with $e_{31}, e_{32}, e_{33} \in E_3$.
- Instance Consumption: Only unique composite events have to be considered ($P_{xy} : x = y$). Otherwise notifications would be send at every event occurrence after the third one, which is not appropriate in this case.
- Instance Selection: Every primitive event regarding a new order is to consider, the event specification would have to be refined ($z_{min} = 1, z_{max} = length(tr_{post})$ and $w_{min} = 1, w_{max} = m$).

An exemplary trace and the associated profile evaluation is given in Figure 7. The first event pair does only qualify for a partial event, no notification is send. The next three events qualify, the fourth event does not qualify since only unique composite events are allowed.

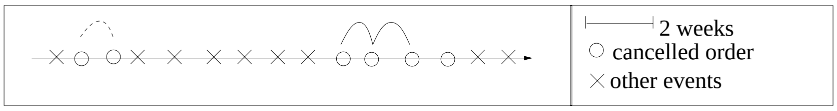


Figure 7: Trace and evaluation example for Profile P2

P3: Notify customer I, if T1 leaves for I after 2pm.

- Composite event description: $e = (e_t; (e_{41}, e_{42})_\infty)_\infty, e_t \in E_{2pm}, e_{41} \in E_4^A, e_{42} \in E_4^B$.
- Instance Consumption: Only unique composite events are to consider ($P_{xy} : x = y$), we do not want to be notified about every possible with all time events.
- Instance Selection: The last time event before the occurrence of the truck event is to consider, the time events overwrite each other ($z_{min} = z_{max} = length(tr_{post})$ as defined in Section 4). Only the first composed truck event $e_3 = (e_1, e_2)_\infty$ has to be taken into account ($w_{min} = w_{max} = 1$).

An exemplary trace and the associated profile evaluation is given in Figure 8. The two truck event pairs are identified, only the first one qualifies since its occurrence time is after the time event. The second pair does not qualify for the first time event, since only unique composite events are allowed. It does not qualify for the second event since is does not match the sequence operator.

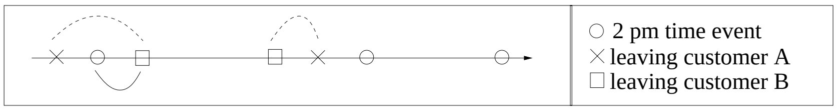


Figure 8: Trace and evaluation example for Profile P3

As shown in our examples, the logistics scenario describes a mixed application that processes information coming from differently structured sources. Therefore in this scenario we have to apply various parameter settings. Other applications use more homogenously structured sources, so that a single parameter setting could be used within the application field.

6 Conclusion & Outlook

Event Notification Services play a major role in many recent applications. In this paper we proposed a parameterized event algebra for event notification services. The event algebra was introduced in order to describe the event operators that form composite events. In an additional step we introduced parameters for event instance selection and event instance consumption. Event instance selection describes which qualifying events from the trace are taken into account for composite events, and how duplicate events are handled. Event instance consumption defines whether unique composite event or all possible combinations of events are taken into account. The combination of both parameters also offers the definition of filter pattern similar to the ones applied in parsing. The algebra and parameters are defined based on binary operators, by nesting composite events (as shown in profile P3). The introduced semantics can easily be extended to sets of events.

We introduced our event algebra in both an informal and a formal way. Note that the formalism used here is similar to the one of the relational algebra. The relational algebra lacks the concept of ordering, therefore we introduced an ordering relation on event traces. We applied the event algebra to the application field of transportation logistics.

We are currently implementing the prototype of a generic parameterized Event Notification System (*GENAS*) that is based on the parameterized event algebra introduced here. *GENAS* can be adapted to different application fields using various parameter settings. It can also be used for mixed application such as the logistics scenarios introduced here. Additionally, our parameterized event algebra offers the possibility to easily integrate differently structured event sources.

Acknowledgements. We wish to thank the database group at the FU Berlin as well as Klaus Dittrich for their valuable comments on our approach.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., 26–28 1993.

[2] J.M. Ale and G. Rossi. An approach to discovering temporal association rules. In *Proceedings of the SAC(1) 2000*, pages 294–300, 2000.

[3] J. Allen. Time and time again: The many ways to represent time. *International Journal of Intelligent Systems*, 6:341–355, 1991.

[4] M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *The VLDB Journal*, pages 53–64, 2000.

[5] J. Bacon, J. Bates, R. Hayton, and K. Moody. Using events to build distributed applications. In *Proceedings of the Seventh ACM SIGOPS European Workshop, pages 9-16, Connemara, Ireland, September 1996.*, 1996.

[6] A. Buchmann C. Liebig, M. Cilia. Event composition in time-dependent distributed systems. In *Fourth IFCIS Conference on Cooperative Information Systems (CoopIS'99) (In cooperation with VLDB'99), Edinburgh University, Edinburgh, Scotland*, September 2-4 1999.

[7] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Symposium on Principles of Distributed Computing*, pages 219–227, 2000.

[8] S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Knowledge and Data Engineering Journal*, 14:1–26, 1994.

[9] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. In *Proc. of the ACM SIGMOD Conf. on Management of Data, 2000*, 2000.

[10] X. Chen, I. Petrounias, and H. Heathfield. Discovering temporal association rules in temporal databases. In *Issues and Applications of Database Technology*, pages 312–319, 1998.

[11] U. Dayal. The hipac project: Combining active databases and timing constraints, 1988.

[12] K. R. Dittrich and S. Gatziu. Time issues in active database systems. In R. T. Snodgrass, editor, *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, Arlington, TX, 1993.

[13] S. Gatziu and K. Dittrich. Events in an active object-oriented database system. In *Proc. of the 1st International Workshop on Rules in Database Systems. Springer, September*, 1993.

[14] S. Gatziu and K. R. Dittrich. Detecting composite events in active database systems using petri nets. In *RIDE-ADS*, pages 2–9, 1994.

[15] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Composite event specification in active databases: Model & implementation. In L.-Y. Yuan, editor, *18th International Conference on Very Large Data Bases, August 23-27, 1992, Vancouver, Canada, Proceedings*, pages 327–338. Morgan Kaufmann, 1992.

[16] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Event specification in an active object-oriented database. In Michael Stonebraker, editor, *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, San Diego, California, June 2-5, 1992*, pages 81–90. ACM Press, 1992.

[17] A. Hinze. How does the observation strategy influence the correctness of alerting services? In *Proceedings of the 9. Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft, BTW 2001*, March 2001.

[18] A. Hinze and D. Faensen. A Unified Model of Internet Scale Alerting Services. In Lucas Chi-Kwong Hui and Dik Lun Lee, editors, *Proceedings of 5th International Computer Science Conference, ICSC'99 (Internet Applications.)*, volume 1749 of *Lecture Notes in Computer Science*. Springer, 1999.

[19] H. V. Jagadish and O. Shmueli. Composite events in a distributed object-oriented database. *IWDOM*, pages 248–268, 1992.

[20] L. Lamport. Times, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[21] L. Liu, C. Pu, R. Barga, and T. Zhou. Differential evaluation of continual queries. In *IEEE Proceedings of the 16th International Conference on Distributed Computing Systems*, pages 458–465, Hong Kong, 27-30 May 1996. IEEE Press. available at `http://web.cs.ualberta.ca/~lingliu/papers/lcdcs96.ps`.

[22] D.L. Mills. Network time protocol (version 3) specification, implementation and analysis., 1992.

[23] I. Motakis and C. Zanilo. Formal semantics for composite temporal events in active database rules. *JOSI*, 37(1), 1997.

[24] S. Navathe and R. Ahmed. A temporal relational model and a query language. *Information Sciences*, 49, 1989.

[25] J. Pereira, F. Fabret, H. Jacobesen, F. Llirbat, R. Preotiuc-Prieto, K. Ross, and D. Shasha. Le subscribe: Publish and subscribe on the web at extreme speed. In *Proceedings of the ACM SIGMOD Conference*, 2001.

[26] R.Zhang and E.Unger. Event specification and detection. Technical report, Kansas State University, jun 1996. available at `http://www.cis.ksu.edu/~schmidt/techreport/1996.list.html`.

[27] S. Schwiderski. *Monitoring the Behaviour of Distributed Systems*. PhD thesis, Selwyn College, University of Cambridge, apr 1996.

[28] A. Tansel. A temporal extension to sql. In R. T. Snodgrass, editor, *Proceedings of the International Workshop on an Infrastructure for Temporal Databases, Arlington, TX, June 1993.*, 1993.

[29] D. Toman. Point-based Temporal Extensions of SQL. In Franois Bry, Raghu Ramakrishnan, and Kotagiri Ramamohanaroa, editors, *Proceedings of the 5th International Conference on Deductive and Object-Oriented Databases (DOOD)*, volume 1341. Springer-Verlag, 1997.

[30] S. Yang and S. Chakravarthy. Formal semantics of composite events for distributed environments. In *Proceedings of the International Conference on Data Engineering (ICDE 99)*, pages 400–407, March 1999.

[31] D. Zimmer and R. Unland. On the semantics of complex events inactive database management systems. In *Proc. of the ICDE*, 1999.