A Parameterized Algebra for Event Notification Services

Annika Hinze, Agnès Voisard
Freie Universität Berlin
{hinze, voisard@inf.fu-berlin.de}

Abstract

Event notification 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. The parameters serve as a basis for flexible handling of duplicates in both primitive and composite events.

1. Introduction

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 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*). An example profile in a logistics application is *Notify if a sensor reads temperature above* $30°C$ *three times within a given timespan*. The events entering an event notification system are filtered according to user profiles.

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. In addition to the fact that only few sophisticated profile languages are defined, the evaluation of languages that seem to follow similar semantics does not always lead to similar results. One example is the handling of duplicate events: depending on the implementation, in the filtering process, duplicates are either skipped or kept.

In the area of active database systems, the problem of event rule specification has been evaluated with focus on composite events, e.g., [2, 4]. Some ENS systems also implement composite events, e.g. [5, 6, 9]. The semantics of composite events in active databases and existing application-specific ENS are not sufficient for integrating applications: The implemented algebras do not cover all necessary event combinations, in particular combinations of subsequent duplicate groups as used in logistics support and facility management. The systems also lack the necessary flexibility in the event processing to support event providers with slightly different semantics.

Semantics of operators for composite events are not defined in a uniform manner in the numerous application areas. Our approach, therefore, supports various perceptions. This is achieved through the introduction of a flexible semantics which is controlled by a set of parameters.

This paper is organized as follows. Section 2 points out the differences of our approach to active databases and other related work. In Section 3 introduces our parameterized event algebra. Finally, Section 4 addresses concluding remarks and gives some directions for future work in this domain.

2. Concepts

An *event* is the occurrence of a state transition at a certain point in time. 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 that all occurring events can be ordered sequentially in a global system of reference. 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. Content events involve changes of non-temporal objects. We do not discuss the definition of primitive events in greater detail. One approach is the definition by (attribute,value) pairs, e.g., $e_1 = event(sensor = xyz, temperature = 35°C)$.

We distinguish *events* from *event classes*. An event class is defined by a set of event properties. Even though events of the same event class share these properties, they may differ in other event attributes. User profiles define event classes, e.g., $p_1 = profile(temperature > 30°C)$.

Events are denoted by lower Latin e, while event classes

are denoted by upper Latin E. The fact that an event e_i corresponds to an instance of an event class E_j is $e_i \in E_j$. The timestamp of an event e is denoted $t(e)$.[1] Based upon a notation used, e.g,. in [1], the matching operator is defined as follows:

Definition 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.*

The exemplary event e_1 matches the profile p_1[2].

3. Event Algebra

This section describes our event algebra. First, we informally describe the event operators. The events e_1 and e_2 in the descriptions below can be any primitive or composite event, $t()$ refers to occurrence times, and t denotes time spans.
– The *disjunction* $(e_1|e_2)$ of events occurs if either e_1 or e_2, $t(e_1|e_2) := min\{t(e_1), t(e_2)\}$.
– The *conjunction* $(e_1, e_2)_t$ occurs when both e_1 and e_2 have occurred within a time span t[3], regardless of the order, $t(e_1, e_2) := max\{t(e_1), t(e_2)\}$.
– The *sequence* $(e_1; e_2)_t$ occurs when e_1 occurs before e_2 with $t(e_2) \leq t(e_1) + t$, $t(e_1; e_2) := t(e_2)$.
– The *negation* $\overline{e}_t$ defines a negative event; e does not occur for an interval $[t_{start}, t_{end}]$, $t_{end} = t_{start} + t$ of time, $t(\overline{e}_t) := t_{end}(\overline{e}_t)$.
– The *selection* $e^{[i]}$ defines the occurrence of the i^{th} event of a list of events, $i \in \mathbb{N}$.

Let us assume a profile $p = (e_1; e_2)_t$ and the following history (trace) of events: $tr = \{e_1, e_1, e_2, e_2\}$. It is not automatically clear 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 wether the profile can be matched twice, e.g., by pairs (2,3) and (1,4), or by (1,3) and (2,4). For different applications, different event-history evaluations could be applied. As parameters we adopt the modes of event instance selection and consumption from active databases [10]. *Event instance selection* describes, which events qualify for the complex events, and how duplicated events are handled. *Event instance consumption* defines which events are consumed by complex events. In contrast to active databases, event selection and consumption in ENS cannot be handled independently.

Duplicate events are event instances that belong to the same class. We introduce the notion of a duplicate set:

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

[2]Evaluations start after the profile definition and for each positively evaluated event $p \sqsubset e$ holds implicitly $t(e) > t(p)$.

[3]$(e_1, e_2)_\infty$ 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 [3].

Definition 2 (duplicate set) *Let e_1, e_2 be two events with $e_1 \neq e_2$. We then define a duplicate set as the ordered set of events of type e_1 that occur in a sequence without any events of type e_2 in between, e_2 without e_1 respectively.*

For the instance selection parameter we distinguish the selection of the first/last event of each duplicate set or the consideration of all events. For the identification of composite events the following holds: Matched events can be consumed by the composite event or they can contribute several times to composite events of the same class. If events are consumed by composite events, the filtering process could be reapplied after unique composite events have been identified. We now introduce the terminology for our formal event algebra.

Definition 3 (event space) *The set of all possible events known to a system is called the event space $\mathbb{E}$. The set of all time events is denoted $\mathbb{E}_t$.*

Definition 4 (trace) *A trace $tr_{t1,t2}$ is a sequence of ordered events $e \in \mathbb{E}$ with defined start- and end-points $t1$, $t2$ respectively.*

The history of events that 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 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 $L[i]$.

Definition 5 (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 a sublist is defined as the number of list elements. Disjunctive sublists containing only similar events are referred to as duplicate lists. Note that we denote (unordered) sets of events with $\mathbb{E}$ or E_t while $tr[]$ denotes ordered sets or lists of events. Without loss of generality, we assume $tr(e_1, e_2)$ to start with $tr[1, 1] = e_1$. The following definition holds for all $w \in [w_{min}, w_{max}]$, $x \in [1, \infty)$,

$y \in [1, \infty)$, and $z \in [z_{min}, z_{max}]$. The parameters P_{xy}, w_{min}, w_{max}, z_{min}, z_{max} substantially influence the operator semantics. Due to space restrictions we only show the formal definition an event sequence, full definition od all operators can be found in [8].

Definition 6 (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 $t \in \mathbb{R}$: $(p_1;p_2)_t \sqsubset (e_1, e_2) \Rightarrow \{p1 \sqsubset e_1, p2 \sqsubset e_2, t(e_2) \in (t(e_1), t(e_1)+1]\}$ The set of matching events of a given trace tr is then defined as*

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

Semantical Variations We now evaluate different approaches for the parameter values, which implement different semantics of the operators. For consumption modes we distinguish the selection of unique pairs $P_{xy} : x = y$ and the selection of all pairs: $P_{xy} : x \leq y$.

For the event instance selection, we distinguish several variations to select events from duplicate lists. Each has to be evaluated 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. We make the distinction between the selection of the first/last event, or of all of them.

	anterior	posterior
first	$z_{min} = z_{max} = 1$	$w_{min} = w_{max} = 1$
last	$z_{min} = length(tr_{ant})$	$w_{min} = w_{max} = m$
	$z_{max} = length(tr_{ant})$	
all	$z_{min} = 1$	$w_{min} = 1$
	$z_{max} = length(tr_{ant})$	$w_{max} = m$

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 different cases can be combined, taking as posterior event every first in a duplicate set and as anterior event the respective last duplicate match.

The third approach 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 sets, the second match are second/next-to-last events, and so forth. Other combinations are plausible. We only show two intuitive examples.

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

The issue of order and time in a distributed environment is crucial and has to be considered for an implementation of this operators. We distinguish event evaluation at the end of the defined time span vs continuous evaluation of events. The latter one offers the advantage of early notification. Here, fast but probably incorrect information is delivered in opposition to correct but later information after the ending of the time frame.

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 [7]).

4. Conclusion & Outlook

We proposed a parameterized event algebra for integrating event notification services that support differently structured event sources. 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 are currently implementing a 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.

References

[1] A. Carzaniga, D. Rosenblum, and A. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *PODS*, 2000.

[2] S. Chakravarthy and D. Mishra. Snoop: An Expressive Event Specification Language for Active Databases. *Knowledge & Data Engineering Jour.*, 14:1–26, 1994.

[3] S. Gatziu and K. Dittrich. Events in an active object-oriented database system. In *Int. Workshop on Rules in Database Systems.*, 1993.

[4] S. Gatziu and K. Dittrich. Detecting Composite Events in Active DB Systems Using Petri Nets. In *RIDE-ADS*, 1994.

[5] A. Geppert and D. Tombros. Event-based distributed workflow execution with EVE. Technical Report ifi-96.05, University of Zurich, 1996.

[6] R. Gruber, B. Krishnamurthy, and E. Panagos. The architecture of the READY event notification service. In *ICDCS workshop, Austin, Texas*, 1999.

[7] A. Hinze. How does the observation strategy influence the correctness of alerting services? In *Proc. of the BTW (German National Conf. on Databases)*, 2001.

[8] A. Hinze and A. Voisard. A flexible parameter-dependent algebra for event notification services. Technical Report Number tr-b-02-10, Freie Universität Berlin, 2002.

[9] C. Liebig, M. Cilia, and A. Buchmann. Event Composition in Time-dependent Distributed Systems. In *CoopIS*, 1999.

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