EVA: An EVent Algebra Supporting Adaptivity and Collaboration in Event-Based Systems

Annika Hinze* and Agnès Voisard †

TR-09-006

July 2009

Abstract

In applications such as digital libraries, stock tickers, traffic control, or supply chain management, composite events have been introduced to enable to capture rich situations. Composite events seem to follow a common semantics. However, on closer inspection we observed that the evaluation semantics of events differs substantially from system to system.

In this paper, we propose a parameterized event that defines the detailed semantics of composite event operators. We introduce composite event operators that support explicit parameters for event selection and event consumption. These parameters define how to handle duplicates in both primitive and composite events.

The event algebra EVA forms the foundation for a unifying reference language that allows for translation between arbitrary event composition languages using transformation rules. This translation, in turn, allows for a meta-service that can federate heterogeneous event-based systems. Our approach supports the seamless integration of event-based applications and changing event sources without the need to redefine user profiles. The event algebra is exemplified in the domain of transportation logistics.

* Humboldt University Berlin, Germany and University of Waikato, New Zealand
†Fraunhofer Institute for Software and Systems Engineering (ISST), Berlin, Germany and Freie Universität, Berlin, Germany

I. Introduction

Event-based Systems (*EBS*) serve as a support to such diverse applications as business process monitoring, digital libraries, traffic control, facility management, remote monitoring, or supply chain management [34]. *Complex Event Processing* has gained increasing attention in the past few years with recent focussed attention by industry (see for instance [22]). A number of books has also recently been published on the issue of complex event processing (e.g., [42], [46], [13]).

Event-based systems trigger actions based on observed events. Events could be, for example, the value change of shares, a new temperature sensor value, the occurrence of a traffic jam, a new Radio Frequency IDentifier (RFID) read, or more generally a state transition. The most basic triggered action is the sending of a notification to an interested party. Business processes often call for the detection of complex events, e.g., for identifying out-of-the-ordinary usage patterns in fraud detection or in the support of business transactions.

Users (end users or application designers) define their interest in events by means of *profiles* or subscriptions, using an *event language*. Profiles define conditions on events; matching events may then trigger further actions. Profiles are similar to search queries but filter a stream of incoming events instead of static data. Users may be interested in primitive (atomic) events, their times and order of occurrence. Many applications often require a complex event processing that relies on the detection of composite events, which are formed by logical and temporal combinations of events coming from either a single source or many sources. Examples of event conditions in a logistic application may be:

P1: *a traffic jam alert occurred and one of company's trucks is affected*
P2: *a customer cancelled three orders within a month.*

A. Problem Statement

Various languages have been designed and implemented for event processing. Some of them support only primitive events while more advanced systems also provide concepts for composite events. Languages for primitive events use Boolean predicates [48], an SQL-like syntax [39] or XML-QL [17].

The description of composite events requires the definition of operators for event composition. A number of languages has been proposed with different features, e.g., the early language Snoop for active databases [14], CEDL which introduces parameters and aggregates [55], the Amit approach which considers situations [1], and SpaTeC [52] which introduces spatio-temporal reasoning. A recent overview of event languages has been presented in a tutorial at DEBS 2009 [50] We identify four issues in the handling of composite events[1]:

[1]We also introduced and discussed some of these issues at the Dagstuhl seminar on Complex Event Processing held in 2007 [15].

1) Unspecified temporal semantics: Some composition operators allow for temporal parameters, others hard-code certain semantics.
2) Unclear semantics: The evaluation of language expressions that seem to follow similar semantics (e.g., sequence of two events) does not always lead to similar results. One example is the handling of duplicate events (e.g., events that describe similar state transitions but occur at different times): depending on the application field and on the implementation, duplicates may either be skipped or retained in the event-filtering process.
3) Lack of collaboration: No common framework exist that allows for comparison of event composition languages. The collaboration between event-based systems is *a fortiori* extremely limited.
4) Lack of adaptativity: Event conditions must be redefined for different systems and for variations in event sources (e.g., different sensors). For example, truck location events may be sent on changing location or may be retrieved following a schedule The profiles and services cannot adapt to changing event sources or new event sensors automatically.

B. Contribution of the Paper

The contributions of this paper are threefold. We introduce:

- an *event algebra* to define the semantics of complex events in a way that allows to easily adapt to different application needs.
- a *(meta) language* that supports a comparison and translation between different languages.
- a *meta-service* for event-based systems as proof of concept that implements a collaboration between heterogeneous event systems.

The focus of the paper lies on the detailed presentation of our event algebra EVA – an EVent Algebra that supports adaptable composition. Each of the complex elements of the algebra is discussed in detail using simple yet illustrative examples borrowed from a logistics application.

To incorporate temporal restrictions, we use the notion of *relative time*. Our formalism is an extension of the semantics of *Event-Condition-Actions (ECA)* rules of active databases. Since EBS are typically used in the context of a distributed environment and cannot typically rely on a transactional context (unlike active database systems), the simultaneous occurrences of events have to be identified according to a distributed time reference. Therefore, all composite event operators need to be handled as temporal operators and extended by a relative time frame (similar to time handling in distributed systems). The issues of timing and ordering in a distributed environment is not addressed here. We refer to existing literature on the topic[2]. For the sake of simplicity, we assume that all occurring events can be ordered sequentially in a global system of reference.

[2]A global system of reference may be used (e.g., by applying the 2g-precedence model of [51], or the NTP protocol [44]) with a mapping for real time references or the use of calibrated GPS location sensors with access to global time.

The semantics of temporal operators as introduced, for instance, in [4] is not defined in a uniform manner across the numerous application areas. Our approach as described in this paper allows for semantic variations. This is achieved through the introduction of an event algebra which is controlled by a set of parameters. The *parameterized event algebra* handles temporally-composed events. It allows for simple changes of the filter semantics to support changing applications, to adapt to new event sources, and to support the integration of applications that combine events from different sources.

We introduce a framework for the comparison of composite event languages based on our algebra. We briefly summarize the results of a language survey that we performed using the framework.

Adaptability to changing applications and event sources as well as the collaboration between a number of EBS is a complex issue: It requires rules for transformation of profiles according to the sources, applications, and collaboration partners. As a proof of concept we implemented an adaptive system A-mediAS that uses the event algebra introduced in this paper to support the required profile changes. We also introduce our Meta-service that allows collaboration between EBS based on predefined transformation rules.

C. Organization of the Paper

The remainder of the paper is organized as follows. Section II discusses related work. Section III gives necessary background information. After introducing an application scenario that will serve as a reference throughout the paper, we briefly describe basic event-related concepts. We describe temporal event operators and semantic variations. In Section IV, we introduce our parameterized algebra and apply it to our application scenario to illustrate the influence of various parameter settings. Section V describes our proofs of concept: an implementation of the algebra in an ENS, a language survey using the algebra, and the design and implementation of a Meta-service for EBS. Section VI gives concluding remarks and directions for future work.

II. 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. Event-based systems can be distinguished according to their abstraction levels: application level, implementation / communication level, system level. For example, the Cobea System [43] (application level) is built upon a CORBA infrastructure (implementation / communication). Note that event-based systems on a higher level do not necessarily rely on event-based components at a lower level. For instance, active database systems (implementation level) are not based on event-based communication.

In the area of *active database systems*, the problem of event rule specification has been studied for several years (see, e.g., [14], [28], [57], also with special focus on composite events [24], [26], [36], [61], [58] and temporal conditions as in [20], [21]). As opposed to EBS, 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 by Allen [4] can be used (see the formal semantics in [45]). Ordering based on events, as opposed to states, has been implemented in the SAMOS system [23]. The work of Zhang and Unger [59] on semantic variation of operators is foundational and provided great insight into this matter. However, they do not consider flexible parameters nor time frames. Zimmer and Unland provide early work on comparing the semantics of different event languages [60]. Active database systems do not support adaptive system behavior and most of them are centralized systems that deal only with database-internal events. Furthermore, the approaches concentrate on the implementation level, whereas our work focuses on the application level.

In the context of active databases, temporal logic has been used to describe the semantics of triggers, e.g., in [45], [49]. Using temporal logic [5] is an alternative approach to describe composition operators. A promising approach is the Enhanced Past Temporal Logic (PTL) introduced by Sistla and Wolfson [49], because it supports relative temporal conditions and composite actions. However, the desired flexibility would be lost[3] and not all operators and parameters can be expressed. A more recent attempt towards a formal foundation of event queries and rules was proposed by Bry and Eckert [7].

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 to SQL [53], [54] and a temporal relational model and query language [47]. In contrast to EBS, however, temporal and deductive databases focus on ad-hoc querying.

The areas of *temporal data mining and time series analysis* rely on temporal association rules [2], [3], [18]. From a set of data, rules verified by the data have to be discovered. While similar event operations are evaluated, the approaches differ greatly from event filter semantics discussed in this paper. In event-based systems, 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 [29], [9], [58]. Finally, time systems in distributed environments have been addressed for instance in [38], [51].

Several other event-based systems have been implemented, however, to the best of our knowledge, none of them supports a flexible filter evaluation similar to our approach. The CQ system [41] is the only one with support for sophisticated composite events; Siena [12], and Rebecca [8] support simple composite events. Most services are message centered and do

[3]Small parameter changes for our algebra may require completely new expressions in PTL.

not rely on an event model.

Services on the communication level are called event-based infrastructures. They do not support composite events. Event-based communication on the Internet is supported by a number of protocols (e.g., Gena [19], DRP [56], and RVP [10]). None of these implement composite events.

III. Background

This section is devoted to our context of study. It first describes our running example in the field of transportation logistics. Then the basic event-related concepts are introduced.

A. Application Scenario

Let us consider a company in Hamilton that sells various goods to its customers. The goods are stored in a warehouse and are delivered to the customers by trucks T_1, T_2, and T_3. 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 load balancing or convenient delivery time for customers. 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. The following events are of interest in this application:

- truck departure from a location
- truck arrival at a location
- goods pickup
- goods delivery
- delivery cancellations
- traffic jam alert
- accident involving truck

Figure 1 depicts the following scenario. On a given day, customers A, B, C, D, E, F, G, H, and I expect deliveries. The customers' respective locations designated by the same letters are also shown in the figure. Let us assume that each delivery takes 30 minutes on average. The delivery list for that day that must be followed by trucks T1, T2, and T3 is:

T1: load goods, deliver A,B, deliver I, return
T2: load goods, deliver D,E, deliver C, return
T3: load goods, deliver H, deliver F,G, return

All of the trucks need to cross the city center and the bridges. Potential traffic jams would cause delays that clients may want to be informed about.
The system clients considered in our example are the customers, controllers, analysts, and drivers. The following profiles are defined[4] :

P1: (Notify the controller if) a traffic jam alert occurred and (if) 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) Truck T1 leaves from Location A or B for I after 2pm.
P4: (Notify the controller when) all trucks are back in the garage.
P5: (Notify the controller if) goods have not been picked up 2 hours after the start of the shift.
P6: (Notify the controller and) give every 30 min the location of each truck.
P7: (Notify the truck driver if) a customer from the delivery lists cancels the delivery.

[4]For clarity reasons, we write down the notification between parenthesis although it is not part of the filtering process *per se*.

The scenario illustrates in particular that a wide range of composite events may be of interest within a single application.

B. Definitions

We consider a number of *objects of interest* in a system, e.g., real world objects such as trucks in a warehouse, an item with a RFID tag, a web-page on the Internet, or abstract objects such as clocks. The state of an object at a certain time is defined by its *properties* (attributes), e.g., the *location* of a truck or the *content* of a web page.

Definition 3.1 (Event): An event is a state transition of an object of interest at a certain point in time, i.e., a significant change of attribute value.

A "significant change" is obviously application dependent. For instance, while in some applications a change of location of three meters or a raises of 3 degrees Farenheit may be significant, in other ones it may be negligible.

As it is often the case, we consider that events have no duration and are "happenings of interest" at a certain time [46]. Events may be state changes in databases, real-world events such as the departures and arrivals of vehicles, a new RFID read, or even the (value of the) current time if the current time refers to the state of an abstract clock object.

Note that also sensor readings without a change of the sensor value constitute events [34]. In this case the changed value refers to the time of the sensor reading. The fact that a value did not change over time may also be of interest.

Events themselves can be represented using a collection of simple (attribute, value) pairs. Each event representation has a timestamp as a mandatory attribute that refers to the time of the event's occurrence. The time system of reference for timestamps is discrete. In our scenario, an example for a (primitive) event at a truck location is:

$$\begin{aligned} e_{truck} = event(&name =' truck\_location', \\ &truck\_number = T2, \\ &location = (x1, y1), \\ &timestamp = 14:00:00) \end{aligned}$$

We do not elaborate on attribute types here, e.g., string for attribute name, as event representation is not the focus of this work. Various approaches are conceivable and may be implemented based, for instance, on the definition of keywords (Cf. in digital libraries) or on the specification of (attribute,value) pairs to describe events. Furthermore, we consider *primitive*

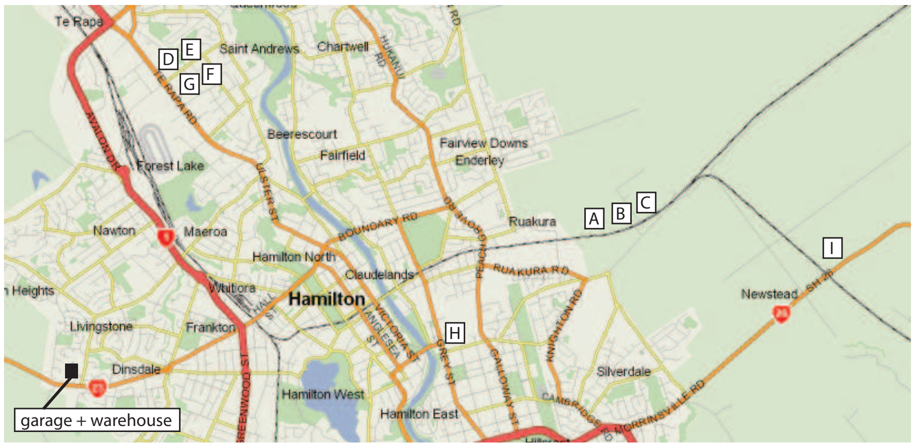


Fig. 1. Logistics Scenario: Warehouse and customers A to I in Hamilton (Map with courtesy of Wises.co.nz)

events and *composite events*, which are formed by combining primitive and possibly composite events.

The set of all events to be observed within an EBS is called *event space*:

Definition 3.2 (Event Space): The set of all possible events known to a certain system is called the event space $\mathbb{E}$. The event space is formed by the set of primitive events $\mathbb{E}_P$ and the set of composite events $\mathbb{E}_C$: $\mathbb{E} = \mathbb{E}_P \cup \mathbb{E}_C$. The set of time events is denoted $\mathbb{E}_t \subset \mathbb{E}_P$.

The set of composite events $\mathbb{E}_C$ detectable by a certain system is specified by its event algebra that defines its filter and profile semantics. Composite events are created based on an event algebra. The algebra defines temporal event composition operators. Event composition defines new events. Note that we do *not* define a composite event as a *set* of primitive events, as done, e.g., by Gehani et al. [25], [27]. This difference greatly influences the event composition. The new (composite) events inherit the characteristics of all contributing events; the event occurrence time is defined by the composition operator.

One of the central concepts of an event-based system is the (user) *profile*:

Definition 3.3 (Profile): A profile p_c is a condition c that is continually evaluated by the system against the trace of incoming events.

In an EBS, profiles are evaluated against the "dynamic" history of all observed events, also called the *trace*:

Definition 3.4 (Trace): A trace tr_{t_1,t_2} is a semi-ordered sequence of events $e \in \mathbb{E}$ with start- and end-points t_1, t_2, respectively.

Note that the start and end points might be infinite. The history of events that a service processes is then $tr_{t_0,\infty}$ with t_0 being the point in time the service started observing events[5]. We need a mechanism to refer to each event in a trace. As a trace behaves essentially as a list, we can use the operations commonly defined for lists: We apply an arbitrary local order that assigns an index-number $i \in \mathbb{N}$ to each event. The elements of a trace can then be accessed by their index-number, and $tr[i]$, $i \in \mathbb{N}$ refers to the i^{th} event of the trace tr.

The fact that a profile evaluates true for a certain event is called event-profile matching. Based upon the notation used, e.g., in [11], the matching operator is defined as follows:

Definition 3.5 (Event-Profile Matching $\sqsubset$): Consider the event e and a given profile p. Then e matches p, denoted $p \sqsubset e$, if e the condition defined by profile p is true for event e.

Profiles can also contain wildcards and other operators such that not all attributes of the event have to be exactly defined. Our example event e_{truck} matches the profile $p_{location}$: $p_{location} \sqsubset e_{truck}$. All profile evaluations start after the profile has been defined; for each positively evaluated event e_1, it therefore holds implicitly $t(e_1) > t(profile)$. The set of all matching events for a profile is an *event class*.

Definition 3.6 (Event Class): An event class $E(c) \subseteq \mathbb{E}$ is the collection of all events for which the condition c holds: $E(c) = \{e \,|\, e \in \mathbb{E} \wedge c(\{e\})\}$.

Events in an event class share some properties. For instance, the class may refer to all events regarding temperature, however, the event representations may differ in other attributes (e.g., location). An example is the class of all all events that describe the delivery of goods to Customer A. An event in the event class is the actual delivery of a package to Customer A at a certain time.

Events are denoted by lower Latin e with indices, i.e., $e_1, e_2, \ldots$, while event classes are denoted by upper Latin E

[5]At a given point in time the history of events that are already observed may be referred to as $tr_{t_0,now}$.

with indices, i.e., $E_1, E_2, \ldots$. The membership of an event e_i in an event class E_j is expressed as $e_i \in E_j$. Classes may have non-empty intersections, i.e., they do not have to be mutually disjoint. Thus, membership 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 $t(e)$.

The notion of event classes for profile queries mirrors the concept of result set for database queries. However, an important difference between the two concepts is noteworthy: A result set for a query is determined for a given database state and applying the query on a different database state would lead to a different result set. Profiles are evaluated against every incoming event over a period of time. The event class contains all events that may ever match the profile.

Definition 3.7 (Duplicate): Events that are members of the same event class are called duplicates.

Duplicates could be, for instance, all truck location events of truck T1 referring to identical coordinates, but also all events referring to truck T1. Note that duplicates need not necessarily have neither identical event representation nor identical time stamps.

C. Event Composition

This section informally describes the base operators for composite events. These operators are needed to describe the range of a profile language as well as the event filtering of a system. The set of operators is derived from the profile language considerations of [33]. We extend the event operators from active database systems introduced in [24] in order to consider the temporal demands on event composition. In the following, we look at the simplest combinations of events, namely pairs of events. The combination of more than two events can be created by nesting the operators.

We denote by the $\succ$ operator the fact that a set of events contributes to a composite event:

Definition 3.8 (Composition Contribution $\succ$): Let $e_1, ..., e_n \in \mathbb{E}$ be events that contribute to the composite event $e \in \mathbb{E}_C$. This relation is expressed as $\{e_1, ..., e_n\} \succ e$. $e_1, ..., e_n$ can be primitive or composite events.

Event composition defines new events that inherit the characteristics of all contributing events. The occurrence time of the composite event is defined by the composition operator. The time associated with a composite event is determined by the times of the contributing events and the event composition operator as defined below. The events e_1 and e_2 used in the definitions can be any primitive or composite event; E_1 and E_2 refer to event classes with $E_1 \neq E_2$. $t(.)$ refers to occurrence times based on a reference time system; T denotes time spans in reference time units. We use the contribution operator $\succ$ (Cf. Definition 3.8) to identify the events that contribute to a composite event. Temporal operators are defined on both events and event classes, resulting in further events and event classes, respectively.

Temporal disjunction:
: The *disjunction* $(E_1|E_2)$ of events occurs if either $e_1 \in E_1$ or $e_2 \in E_2$ occurs[6]. The occurrence time of the composite event $e_3 \in (E_1|E_2)$ is defined as the time of the occurrence of either e_1 or e_2 respectively: $t(e_3) := t(e_1)$ with $\{e_1\} \succ e_3$ or $t(e_3) := t(e_2)$ with $\{e_2\} \succ e_3$.

Temporal conjunction:
: The *conjunction* $(E_1, E_2)_T$ occurs if both $e_1 \in E_1$ and $e_2 \in E_2$ occur ($e_1 \neq e_2$), 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.[7] The time of the composite event $e_3 \in (E_1, E_2)_T$ with $\{e_1, e_2\} \succ e_3$ is the time of the last event: $t(e_3) := max\{t(e_1), t(e_2)\}$.

Temporal sequence:
: The *sequence* $(E_1; E_2)_T$ occurs when first $e_1 \in E_1$ and afterwards $e_2 \in E_2$ occurs. T defines the maximal temporal distance of the events. The time of the event $e_3 \in (E_1; E_2)_T$ with $\{e_1, e_2\} \succ e_3$ is equal to the time of e_2: $t(e_3) := t(e_2)$.

Temporal negation:
: The *negation* $\overline{E}_T$ defines a "passive" event; it means that no $e \in E$ occurs for an interval $[t_{start}, t_{end}], t_{end} = t_{start} + T$ of time. The occurrence time of $\overline{e}_T \in \overline{E}_T$ is the point of time at the end of the period, $t(\overline{e}_T) := t_{end}$ When clear from the context, we write $\overline{e}_T$ when referring to a passive event.

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

If several operators are to be applied, we have to distinguish whether identical events or distinct events that belong to the same event class are addressed. For that purpose, we additionally permit the Boolean operators of logical conjunction ($\wedge$) and logical disjunction ($\vee$) to be used in event composition. These operators address identical events, i.e., references to the same event class combined by logical operators refer to identical instances within that set. Logical operators are defined on event classes only. A logical combination of two sets describes the usual logical combination of the defining conditions.

Logical Conjunction
: In a *logical conjunction* $E_1 \wedge E_2$ of event classes E_1 and E_2 both conditions are true for the instances $e \in (E_1 \wedge E_2)$.

Logical Disjunction
: The *logical disjunction* $E_1 \vee E_2$ requires that at least one of the conditions is true for the instances $e \in (E_1 \vee E_2)$.

Note that *logical* combinations of event classes form a name

[6]If both events occur, the result of the disjunction is two distinct events.

[7]$(E_1, E_2)_\infty$ refers to an event composition without temporal restrictions. It is equivalent to the original conjunction constructor as defined, e.g., in [24].

space for the events involved, i.e., equal classes names such as E_1 refer to identical events in that class. Equal names combined by *temporal* event operators only define identical event descriptions and therefore a class of events. This characteristic is illustrated in the following example:

Example 3.1 (Temporal vs. Logical Conjunction): Let E_1, E_2, and E_3 be event classes. Then, the events of the temporal conjunction $E_T = ((E_1; E_2)_{T1}, (E_1; E_3)_{T2})$ are defined as

$$\begin{aligned} E_T = \{e \,|\, \exists e_{11}, e_{12} \in E_1 \; \exists e_2 \in E_2 \; \exists e_3 \in E_3 : \\ \{e_{11}, e_{12}, e_2, e_3\} \succ e \wedge \\ t(e_{11}) \leq t(e_2) \leq \big(t(e_{11}) + T1\big) \wedge \\ t(e_{12}) \leq t(e_3) \leq \big(t(e_{12}) + T_2\big)\}. \end{aligned}$$

It is not required but allowed that $e_{11} == e_{12}$.

The events of the logical conjunction $E_L = ((E_1; E_2)_{T1} \wedge (E_1; E_3)_{T2})$ are defined as

$$\begin{aligned} E_L = \{e \,|\, \exists e_1 \in E_1 \; \exists e_2 \in E_2 \; \exists e_3 \in E_3 : \{e_1, e_2, e_3\} \succ e \wedge \\ t(e_1) \leq t(e_2) \leq \big(t(e_1) + T1\big) \wedge \\ t(e_1) \leq t(e_3) \leq \big(t(e_1) + T2\big)\}. \end{aligned}$$

We now show the application of the newly introduced composition operators to our example profiles:

Example 3.2 (Scenario profiles):

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

P2: (Notify the analyst if) a customer cancelled an order three times within a month: Let E_3 be the set of cancelled orders of a particular customer. A simplified definition for the composite event could be expressed as $e = (e_{31}, e_{32}^{[2]})_{4weeks}$ with $e_{31}, e_{32} \in E_3$.

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

P4: (Notify the controller when) all trucks are back (after an 8-hour shift): Let E_5, E_6 be two sets of events describing the departure 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 set of events of loading goods, E_{10am} the set time-events defining the start of the shift. Then the composite event is defined as $e = (e_t; (\overline{e}_7)_{2h})_{2h}$, $e_t \in E_{10am}, e_7 \in E_7$.

As we shall illustrate shortly, the event operators presented informally in this section do not define the complete semantics of a profile definition language.

For the event operators described above, different (application-dependent) semantics are conceivable. We therefore introduce a *parameterized* event algebra (in the next section). An event algebra is an abstract description of the event concept of a service independently 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 the basis for an abstract implementation of a matching algorithm.

In order to describe our parameterized algebra, we first motivate the different parameters for event selection and consumption. These two concepts have been proposed in the context of active database systems and are extended to the context of event-based system.

D. Composition Parameters: Illustration

The following examples illustrate the limitations of the composite operators described in the previous section.

Example 3.3 (Semantic variations in sequences): Let us assume that we are interested in the sequence of two events $(E_1; E_2)_T$ as defined, for instance, in profiles P1 and P3. We consider the following history (trace) of events: $tr = \langle e_1, e_2, e_3, e_4 \rangle$, with $e_1, e_2 \in E_1$, $e_3, e_4 \in E_2$ as shown in Figure 2. It is not clear from the event definition which pair[8] of events fulfills our profile. Candidate pairs are, for instance, 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 (e_2, e_3) and (e_1, e_4), or by (e_1, e_3) and (e_2, e_4).

Various event combinations may be possible. Besides, for different applications, different event-history evaluations could be applied as illustrated in the following example.

Example 3.4 (Semantic variations in selections): Let us assume that we are interested in every fourth occurrence of an event $e \in E_1$, for example, every fourth cancelled order for a certain good. We could define the profile: (E_1, E_1, E_1, E_1). Let us consider the (synthetic) trace $tr = \langle e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8 \rangle$ with $e_i \in E_1$, $1 \leq i \leq 8$. The profile could be matched by event e_4, or by e_4 and e_8,

[8]Composite events may be formed by more than two events when nesting the compositions. Because the minimal contribution of events consists of two instances for a composition using binary operators, we often refer to the contributing events as pairs.

or by the events e_4, e_5, ..., since all of them are preceded by three instances.

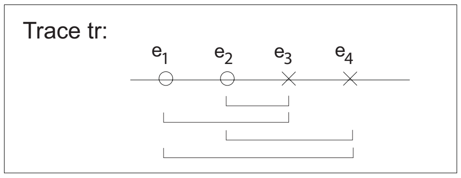


Fig. 2. Possible composite events in Example 3.4, $\circ \in E_1$ and $\times \in E_2$

For an event algebra in the EBS context, we need to identify the following *modes*[9]:

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

Since the event selection principle is not our focus, we consider only Boolean predicates on (attribute, value) pairs. Event composition patterns have been introduced in Section III-C. In the following sections we introduce modes for event selection and consumption. In [61], Zimmer and Unland regarded these modes as being independent for active databases. We show that this is not the case for EBS: event selection and consumption in EBS have to be handled together.

1) Event Selection: Duplicate instances of an event class have to be handled differently depending on the application and even on the context within the application. For the event selection, we distinguish three approaches as depicted in Figure 3. An example might be the reading of a given sensor at different times, such as the location of a truck. For the truck locations the latest sensor reading is the valid one and earlier readings are replaced, i.e., duplicates are overwritten. The delivery events also belong to a single set of events. Delivery events are duplicates, however, if each of these events needs be considered for further planning – duplicates should not be ignored. If for some reasons a customer cancels an order twice, the duplicate event can be ignored.

2) Event Consumption: We distinguish three variations in the identification of composite events, as shown in Figure 4. Matched events can be consumed by the composite event or they can contribute several times to composite events of the same set. 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 return to our delivery example: If a user is interested in the fact that all trucks are back in the evening, the profile can be defined as the sequence of the events *truck X departs* and *truck X arrives*. In this case they are only interested in unique pairs of depart/arrive events, but not, for instance, in all combinations of all depart and arrive events of the month. 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 selection and consumption.

For brevity reasons, we cannot display all six temporal operators with their possible parameter settings (for the sequence operator, this would result in 48 variations).

Extending the terminology introduced in Section IV, we refer to the event selection parameter within a composite event using the following operators for duplicate selection on an event class E: first duplicate $E^{(1)}$, last $E^{(last)}$, all $E^{(all)}$, and i-th $E^{(i)}$. We refer to the event consumption parameter for each composite event pair $(.,.)$ by an additional index: all pairs $(.,.)^{(all)}$, unique $(.,.)^{(unique)}$, and repeated $(.,.)^{(repeat)}$. In Section 5, we introduce an event language implementing the parameterized algebra.

The following Example 3.5 shows profile definitions for our example scenario when using the parameterized operators.

Example 3.5 (Scenario with Parameterized Operators): The profiles P1 and P2 from our example application 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 (see P1 in Example 3.2). The controller wants to be notified about all trucks in traffic jam areas (all pairs). For the truck location events, only the last events in the duplicate groups are considered. All traffic jam events are considered, since several traffic problems can occur within the same area, and all of them have to be taken into account. The following events have to be observed $e \in (E_1^{(all)}, E_2^{A(last)})_{5min}^{(all)}$

P2: (Notify the analyst if) a customer cancelled an order three times within a month (see P2 in Example 3.2). Only unique composite events have to be considered (unique pairs), to prevent alerts being sent after every other event. Every primitive event regarding a new order is to be considered. The following events have to be observed $e \in (E_3^{(all)}, E_3^{[2](all)})_{4weeks}^{(unique)}$

We argue here that the composite operators do need the specification of the parameters described above to define the operators' full semantics. If the parameters are missing, operators that seem to follow similar semantics (e.g., a sequence of events) may be treated differently by different systems (cf. above examples).

E. Profile-Event Situations

This section illustrates the implications of the parameters introduced above.

a) Binary operators: Figure 5 shows a matrix of selected profile-event situations with identical parameter settings for each event in a pair. Other situations can be easily constructed.

[9]We use a terminology developed in active databases [61], [60].

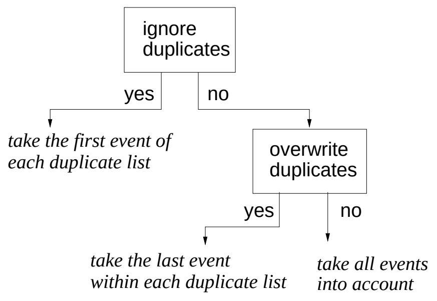

ignore duplicates
yes
no
take the first event of each duplicate list
overwrite duplicates
yes
no
take the last event within each duplicate list
take all events into account

Fig. 3. Event selection

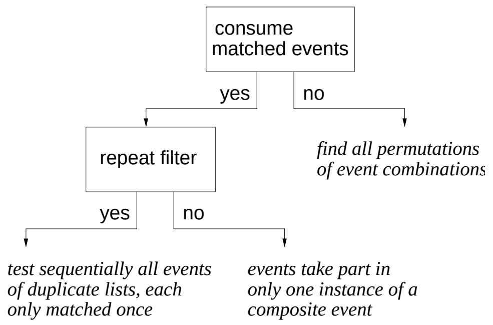

consume matched events
yes
no
repeat filter
find all permutations of event combinations
yes
no
test sequentially all events of duplicate lists, each only matched once
events take part in only one instance of a composite event

Fig. 4. Event consumption

Note that the names for the rows and columns are simplified descriptions of the different approaches. The corresponding formal algebra definitions follow.

The examples shown in the matrix refer to the composite events of a sequence $(E_1; E_2)$ in a given example trace of events. The events are referred to in the figure as $\circ \in E_1$ and $\times \in E_2$. Each composite event is marked with an arc. The position of arcs above or below the trace is only to improve legibility. The dashed arcs denote special cases that we discuss in Section IV-D. Here, we use fixed time frames as evaluation intervals that are denoted with brackets [.], in order to make the different implications easier to compare.

The horizontal dimension of the matrix shows variants of event selection. The selection either takes only the first or the last duplicate in a trace into account, or all events are considered. It is important to note that the word "duplicate" is used here with respect to a profile and not for the event itself (cf. Definition 3.7). Two events can be different, even though they may match the same profile. With regards to that particular profile they are seen as duplicates.

Using the matrix, we can demonstrate example applications for the different approaches:

I: take first event
: 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.

II: take last event
: 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.

III: take all events
: Taking all events is only useful in an application where each duplicate must be considered, 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.

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

The rows of the matrix show different versions of profile filter evaluation: apply the filter to all events, apply it only to the unmatched events, a repeat filtering after a 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. Note that events in the duplicate lists are pre-filtered, i.e., the duplicate lists contain only events that are relevant to the profile (see details on trace view in Section IV-A).

A: keep matched events
: 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 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.

B: consume matched events
: 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 when entering or leaving the building.

C: consume and repeat
: The repetition of the filtering process after discarding matched events is used, e.g., for text analysis. An example application is an event-based XML-validator, as proposed in [6]. With this method interleaving

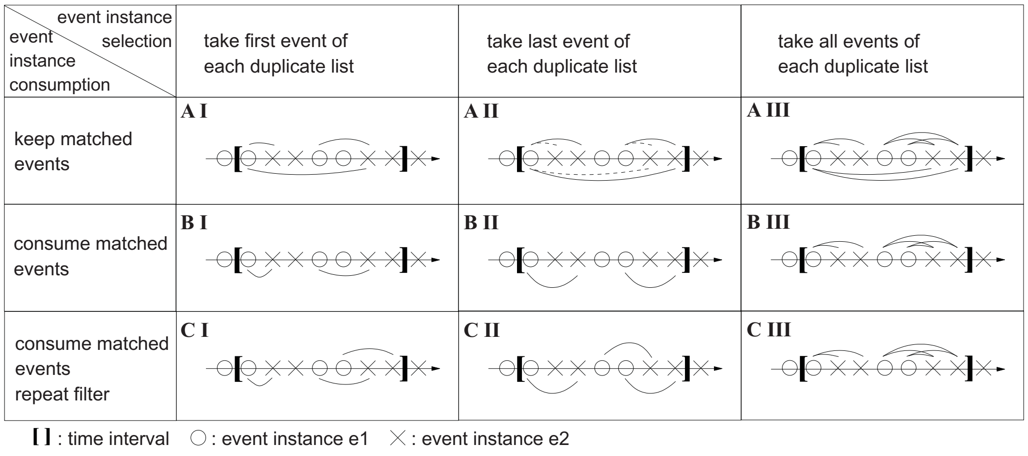

event instance selection
event instance consumption
take first event of each duplicate list
take last event of each duplicate list
take all events of each duplicate list
keep matched events
consume matched events
consume matched events repeat filter
A I
A II
A III
B I
B II
B III
C I
C II
C III
[ ] : time interval
○ : event instance e1
× : event instance e2

Fig. 5. Profile and event trace example under different evaluation conditions

event pairs can be identified.

In publish-subscribe systems, the Siena system [12] implements evaluation style as in situation C I, Rebecca implements B I and B II, CQ [40] implements C I. In active database systems, SAMOS [23] implements C I and C III depending on the event operators, Sentinel [26] uses B III, Ode [14] distinguishes evaluation styles similar to C I, C II, and A III depending on parameters.

A flexible implementation of the different styles would allow for simple adaptation of the filter semantics to changing requirements. Moreover, it would allow for expressive semantics that could be tailored to suit various applications. Our parameterized approach to enable such a flexible implementation is presented in the next section.

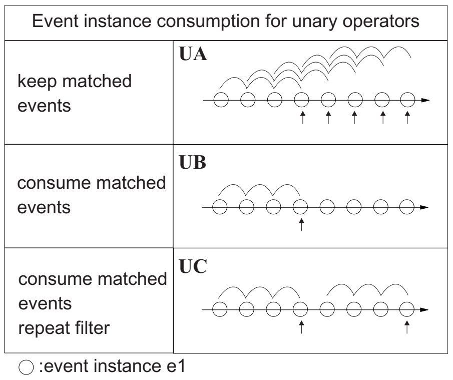

Event instance consumption for unary operators
keep matched events
consume matched events
consume matched events repeat filter
UA
UB
UC
○ :event instance e1

Fig. 6. Unary event profile and event trace example under different evaluation conditions

b) unary operators: For these operators, only the event consumption parameter applies. The semantic variations are shown in Figure 6. The examples refer to the composite event of a selection $E_1^{[4]}$ in a given exemple trace of events. The events are referenced as $\circ \in E_1$, each composite event is marked with an arrow, the arcs denote the event contributing to the composition.

IV. Parameterized Composite Event Algebra

This section presents the formal definition of our parameterized event algebra. We first concentrate on the binary operators and their semantic variations. Then, we briefly consider unary operators.

A. Definitions

To support our formal definition of composite operators, we introduce the concept of trace views. A trace view is a projection of a trace on events of certain event class[10].

Definition 4.1 (Trace View): Let E_1 be an event class. The sub-trace $tr(E_1)$ of a given trace tr is defined as the semi-ordered list of events that contains all events $e \in E_1$. We call this subsequence a trace view.

The trace view $tr(E_1, E_2)$ contains all $e_1 \in tr$ and $e_2 \in tr$ where $e_1 \in E_1, e_2 \in E_2$. 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:

Definition 4.2 (Trace Renumbering): Renumbering trace tr is equivalent to subdividing tr into disjoint sublists $tr[1], \ldots, tr[n]$ such that each sublist contains all successive events from the same event class. 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. Disjoint sublists containing only events of one set are referred to as duplicate lists:

Definition 4.3 (Duplicate List): Let E_1, E_2 be two event classes with $E_1 \neq E_2$. We then define a duplicate list $D_{E_1 \setminus E_2}$ as the ordered list of events of set E_1 that occur in a trace tr

[10]The notion of a view is inspired by database views that hide unnecessary information from the user, giving access only to a certain portion of the data.

without any events of set E_2 inbetween: $D_{E_1 \setminus E_2}(n) = tr[n]$ such that for $e_1 \in E_1$, $e_2 \in E_2$ holds

$$e_1 \in tr, \neg\exists e_2 \in tr : t(e_2) \in (t(tr[n,1]), t(tr[n, length(tr[n])]))$$

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, i.e., the first event $e_2 \in E_2$, did not occur.

Example 4.1 (Trace Renumbering): Let us consider the following trace of events: $tr = \langle e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\rangle$ with $e_1, e_3, e_4, e_8, e_9 \in E_1$, $e_6, e_7, e_{10}, e_{11}, e_{12} \in E_2$, and $e_2, e_5 \in E_3$ as shown top left in Figure 7. The (E_1, E_2)-trace view is then defined as $tr(E_1, E_2) = \langle e_1, e_3, e_4, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\rangle$.

The renumbered trace view is then $tr(E_1, E_2) = \langle tr[1,1], tr[1,2], tr[1,3], tr[2,1], tr[2,2], tr[3,1], tr[3,2], tr[4,1], tr[4,2], tr[4,3]\rangle$ with $tr[1] = \langle e_1, e_3, e_4\rangle$, $tr[2] = \langle e_6, e_7\rangle$, $tr[3] = \langle e_8, e_9\rangle$, and $tr[4] = \langle e_{10}, e_{11}, e_{12}\rangle$. Obviously, the list length of the first sublist follows with $length(tr[1]) = 3$. The example is depicted in Figure 7.

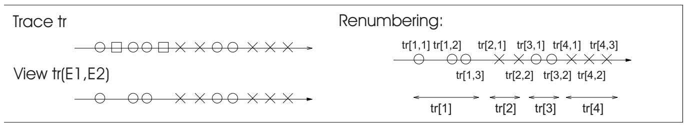

Trace tr
View tr(E1,E2)
Renumbering:
tr[1,1] tr[1,2] tr[2,1] tr[3,1] tr[4,1] tr[4,3]
tr[1,3] tr[2,2] tr[3,2] tr[4,2]
tr[1] tr[2] tr[3] tr[4]

Fig. 7. Trace and Renumbering in Example 4.1

Note that we denote (unordered) sets of events by E[11] while $tr[.]$ denotes lists (ordered sets with possible duplicates) of events.

As discuss ed previously, for each of the basic operators (e.g., sequence), several semantic variations exist. Defining each of the variations separately would require a number of definitions for each basic operator. Instead, we use a set of parameters to control the variations: In our formal definition, the values of the parameters v_{min}, v_{max}, w_{min}, w_{max}, and P_{EIC} influence the operator semantics. The values of v_{min}, v_{max}, w_{min}, $w_{max} \in \mathbb{N}$ control the event selection parameter; they refer to the lower and upper index-number of the selected events within each duplicate list. The definition of the set P_{EIC} controls the event consumption parameter; elements of this set determine the number of duplicate lists required to form the composition pairs. We discuss the different parameter values subsequent to the basic definitions. To easily distinguish the profiles for composite events, we denote the profiles with the operators that have been introduced for the event classes. For instance, $p = (p_1|p_2)$ denotes a profile containing a query regarding the disjunction of events, i.e., the defining query for $(E_1|E_2)$.

B. Binary Operators.

The disjunction implements a selection based on occurrence time (or) not exclusion (xor), i.e., the matching set of the disjunction includes all events that may match either profile. We use the matching operator $\sqsubset$ (introduced in Definition 3.5) to refer to the set of events that match a certain composite profile.

Definition 4.4 (Disjunction of Events): Let us consider two profiles p_1 and p_2, then the following holds

$$(p_1|p_2) \sqsubset e \Leftrightarrow \quad \exists e_1 \in \mathbb{E}\,\big((p_1 \sqsubset e_1 \,\vee\, p_2 \sqsubset e_1) \,\wedge\, e = e_1\big).$$

Let us consider the event classes E_1, E_2 with $E_1 = \{e\,|\,e \in \mathbb{E}, p_1 \sqsubset e\}$ and $E_2 = \{e\,|e \in \mathbb{E}, p_2 \sqsubset e\}$. The set of matching events of a given trace tr is then defined as

$$\begin{aligned}(p_1|p_2)(tr) = \big\{e \quad | \quad & e \in \mathbb{E} \,\wedge\, (p_1|p_2) \sqsubset e \,\wedge \\ & \exists v \in [v_{min}, v_{max}] \subseteq \mathbb{N}^+ \\ & \exists x \in \mathbb{N}^+ \; \exists tr[x,v] \in tr(E_1, E_2) \\ & \text{such that } \{tr[x,v]\} \succ e\big\}.\end{aligned}$$

Different values for the open parameters v_{min}, and v_{max} are discussed subsequently (see Figures 8 and 9).

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 4.5 (Conjunction of Events): Let us consider two profiles p_1 and p_2, $e \in \mathbb{E}$ and a given time span T. Then the following holds

$$\begin{aligned}(p_1, p_2)_T \sqsubset e \Leftrightarrow & \exists e_1, e_2 \in \mathbb{E}\,\big(p_1 \sqsubset e_1 \,\wedge\, p_2 \sqsubset e_2 \wedge \\ & |t(e_2) - t(e_1)| \leq T \,\wedge\, \{e_1, e_2\} \succ e\big).\end{aligned}$$

Let us consider the event classes E_1, E_2 with $E_1 = \{e\,|\,e \in \mathbb{E}, p_1 \sqsubset e\}$ and $E_2 = \{e\,|e \in \mathbb{E}, p_2 \sqsubset e\}$. The set of matching events of a given trace tr is then defined as

$$\begin{aligned}(p_1, p_2)_T(tr) = \big\{e \quad | \quad & e \in \mathbb{E} \,\wedge\, (p_1, p_2)_T \sqsubset e \,\wedge \\ & \exists (x,y) \in P_{EIC} \; \exists v \in [v_{min}, v_{max}] \subseteq \mathbb{N}^+ \\ & \exists w \in [w_{min}, w_{max}] \subseteq \mathbb{N}^+ \\ & \exists \{tr[2x-1, v], tr[2y, w]\} \in tr(E_1, E_2) \\ & \text{such that } \{tr[2x-1, v], tr[2y, w]\} \succ e\big\}.\end{aligned}$$

Again, different values for the open parameters P_{EIC}, v_{min}, v_{max}, w_{min}, and w_{max} are discussed subsequently.

Definition 4.6 (Sequence of events): Let us consider two profiles p_1 and p_2, $e \in \mathbb{E}$ and a given time span T. Then the following holds

$$\begin{aligned}(p_1; p_2)_T \sqsubset e \Leftrightarrow & \exists e_1, e_2 \in \mathbb{E}\,\big(p_1 \sqsubset e_1 \,\wedge\, p_2 \sqsubset e_2 \,\wedge \\ & t(e_2) \in \big(t(e_1), t(e_1) + T\big] \,\wedge\, \{e_1, e_2\} \succ e\big).\end{aligned}$$

Let us consider the event classes E_1, E_2 with $E_1 = \{e\,|\,e \in \mathbb{E}, p_1 \sqsubset e\}$ and $E_2 = \{e\,|e \in \mathbb{E}, p_2 \sqsubset e\}$. The set of matching

[11] $\mathbb{E}$ being the set of all events.

events of a given trace tr is then defined as

$$\begin{aligned}(p_1;p_2)_T(tr) = \{e \quad | \quad & e \in \mathbb{E} \wedge (p_1;p_2)_T \sqsubset e \wedge \\ & \exists (x,y) \in P_{EIC}\ \exists v \in [v_{min}, v_{max}] \\ & \exists w \in [w_{min}, w_{max}] \\ & \exists \{tr[x,v], tr[y+1,w]\} \in tr(E_1,E_2) \\ & \text{such that } \{tr[x,v], tr[y+1,w]\} \succ e\}.\end{aligned}$$

As stated for Definition 4.4 and 4.5, different values for the open parameters P_{EIC}, v_{min}, v_{max}, w_{min}, and w_{max} are discussed in the next paragraph.

c) Semantic Variations of Binary Operators.: We now evaluate different parameter values; we distinguish two dimensions: the selection of events from duplicate lists (EIS) and the composition of matching pairs (EIC). We use the notation *anterior* and *posterior* to refer to the two operands of the binary operators; tr_{ant} and tr_{post} denote the respective duplicate lists. For the event selection, we distinguish several options of how to select events from duplicate lists (as defined in Figure 8). Each operand has to be evaluated differently, depending on the position of the operand relative to the binary operator. The selection of the i^{th} event is a (somewhat artificial) generalization of the preceding modes.

For the composition modes for pair matching (event consumption), we distinguish two variations as shown in Figure 9: the selection of unique pairs (each event in the matching set participates in one pair only, as in Row B in Figure 5) and the selection of all pairs (as in Row A in Figure 5). The third approach, as depicted in Row C in Figure 5, is a combination of the two dimensions that have already been introduced. Here, we discuss this approach briefly: Only unique pairs are considered ($x = y$), matching pairs are removed and the filtering is repeated 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. The parameter option of all duplicates for both contributing events has the same result as without repeated filtering; the combination of different parameter values opens new result variations.

C. Unary operators

The definition of unary operators (selection and negation) is discussed only briefly. While the sequential variations support the selection of the i^{th} event within duplicate lists, the *selection* operates on the full matching list.

Definition 4.7 (Selection): Consider $e \in \mathbb{E}$ and $i \in \mathbb{N}$, then

$$\begin{aligned}p^{[i]} \sqsubset e \Leftrightarrow & \exists e_i \in \mathbb{E}\ \forall j \in \mathbb{N} \text{ with } 1 \leq j \leq i \big(\ p \sqsubset e_j \wedge \\ & t(e_j) \leq t(e)\ \wedge\ \{e_i\} \succ e\big).\end{aligned}$$

Let us consider the event class $E^{[i]} = \{e \,|\, e \in \mathbb{E}, p^{[i]} \sqsubset e\}$. The set of matching events of a given trace is then defined as

$$\begin{aligned}p^{[i]}(tr) = \{e \quad | \quad & e \in E^{[i]}\ \wedge\ p^{[i]} \sqsubset e \wedge \\ & x = 1,\ \exists v \in [v_{min}, v_{max}] \subseteq \mathbb{N}^+ \\ & \exists tr[x,v] \in tr(E^{[i]}) \\ & \text{such that } \{tr[x,v]\} \succ e\}.\end{aligned}$$

Definition 4.8 (Negation): Consider a time event $e_t \in \mathbb{E}_t$ and a given time span t_1, then

$$\begin{aligned}(\overline{p})_{T_1} \sqsubset e_t \Leftrightarrow \quad & \nexists e_1 \in \mathbb{E}\ \ \exists e_2 \in \mathbb{E}_t \big(p \sqsubset e_1 \wedge \\ & t(e_1) \in [t(e_2) - T_1, t(e_2)] \wedge \{e_2\} \succ e_t\big).\end{aligned}$$

Let us consider the event class $\overline{E}_{T_1}$ with $\overline{E}_{T_1} = \{e \,|\, e \in \mathbb{E}, (\overline{p_1})_{T_1} \sqsubset e_t\}$. Then the set of passive events for a given trace is defined as

$$\begin{aligned}(\overline{p})_{T_1}(tr) = \{e \quad | \quad & e \in \overline{E}_{T_1} \wedge (\overline{p})_{T_1} \sqsubset e \wedge \\ & x = 1,\ \exists v \in [v_{min}, v_{max}] \subseteq \mathbb{N}^+ \\ & \exists tr[x,v] \in tr(\overline{E}_{T_1}) \\ & \text{such that } \{tr[x,v]\} \succ e\}.\end{aligned}$$

Unique events are detected with $v_{min} = v_{max} = 1$, all events are detected with $v_{min} = 1$, $v_{max} = length(tr[x])$. As for binary operators, repeated filtering after event matching is more complex.

D. Evaluation Time

The issue of order and time in a distributed environment is crucial and has to be considered for an implementation of these operators. In this paper 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, information delivery may not be accurate as opposed to the correct information available at the conclusion of the time frame. An example is shown in situation A II in Figure 5 (dashed lines). This approach is appropriate for several applications, e.g., catastrophe warning systems for environmental surroundings or other systems for urgent information delivery (for an analysis of information correctness see [29]).

E. Application Examples

In this section we briefly discuss the implementation parameters for profiles P1 and P2 defined in the context of our logistics application. We use the event classes as defined in Section IV:

E_1	– set of traffic-jam events in city area A,
E_2	– set of all events regarding the location sensor of the trucks, with $E_2^A \subset E_2$ the subset of truck location events in A,

EIS	anterior	posterior
First event ($'first\_dup'$)	$v_{min} = v_{max} = 1$,	$w_{min} = w_{max} = 1$
i^{th} event (i'_dup')	$v_{min} = v_{max} = i$,	$w_{min} = w_{max} = i$
Last event ($'last\_dup'$)	$v_{min} = v_{max} = length(tr_{ant})$,	$w_{min} = w_{max} = m$
All events ($'all\_dup'$)	$v_{min} = 1$, $v_{max} = length(tr_{ant})$,	$w_{min} = 1$, $w_{max} = m$

Fig. 8. Event selection: parameters for first, i^{th}, and last event within each duplicate list (columns I–III in Figure 5) where $m \in \mathbb{N}$ with $\forall j > m : t(tr_{post}[.,j]) > t(tr_{post}[.,.]) + T$, where the dots are placeholders for the respective values, T as defined for the operator.

EIC	anterior & posterior
Unique pairs $(.,.)^{(unique)}$	$P_{EIC} = \{(x,y) \mid x \in \mathbb{N}^+ \ \wedge \ y \in \mathbb{N}^+ \ \wedge x = y\}$
All pairs $(.,.)^{(all)}$	$P_{EIC} = \{(x,y) \mid x \in \mathbb{N}^+ \ \wedge \ y \in \mathbb{N}^+\}$

Fig. 9. Event consumption: parameter for unique and all pairs (rows A and B in Figure 5)

The parameters for profile P1 - P2 are defined as follows:

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

- Composite event description: $e \in (E_1^{(all)}, E_2^{A(last)})_{5min}^{(all)}$
- Instance Consumption: All pairs have to be considered: $P_{EIC} = \{(x,y) \mid x \in \mathbb{N}^+ \ \wedge \ y \in \mathbb{N}^+\}$.
- Instance Selection: For the truck location events we consider the last event in the duplicate groups ($v_{min} = v_{max} = length(tr_{ant})$ or $w_{min} = w_{max} = m$ as defined in Section IV). 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 ($v_{min} = 1, v_{max} = length(tr_{ant})$ or $w_{min} = 1, w_{max} = m$).

The parameters in this example are similar to type A II in the matrix in Figure 5. An example trace and the associated profile evaluation is given in Figure 10.

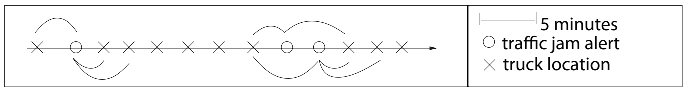


Fig. 10. Trace and evaluation example for Profile P1

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

- Composite event description: $e \in (E_3^{(all)}, E_3^{[2](all)})_{4weeks}^{(unique)}$
- Instance Consumption: Only unique composite events have to be considered: $P_{EIC} = \{(x,y) \mid x \in \mathbb{N}^+ \ \wedge \ y \in \mathbb{N}^+\}$ Otherwise notifications would be sent at every event occurrence after the third one, which is not appropriate in this case.
- Instance Selection: Every primitive event is to be considered, the event specification would have to be refined ($v_{min} = 1, v_{max} = length(tr_{post})$ and $w_{min} = 1, w_{max} = m$).

The parameters in this example are similar to type B III in the matrix in Figure 5. An example trace and the associated profile evaluation is given in Figure 11. The first event pair only qualifies for a partial event, no notification is sent. The next three events (i.e., 2, 3, 4) qualify, the fifth event does not qualify since only unique composite events are allowed.

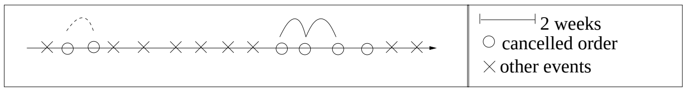


Fig. 11. Trace and evaluation example for Profile P2

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.

V. Proof of Concept

The proof of concept for our event algebra is three-fold: (1) a reference language was implemented in the A-mediAS system, (2) a set of transformation rules between language classes was developed, (3) a Meta-service was implemented.

A. Reference Language Implementation

We implemented a distributed event-based system A-mediAS ([30], [31]), for which the profile definition language is based on the algebra proposed here. In fact, the grammar of the language is a direct translation from our parameterized algebra. In Figure 12, the symbol `priEvent` refers to a primitive event class. The structure of an `Integer` follows the common rules for integers, `TimeSpan` refers to a timespan.

As introduced in this paper, the exact semantics of each composite operator is controlled by parameters. We included the additional parameter of event evaluation time in our profile definition language. This parameter has been discussed in Section IV-D.

Example 5.1 (Profiles A-mediAS language): Our previous example profiles expressed in the reference language are encoded as:

```
       E  ::=  priEvent | CoEvent
 CoEvent  ::=  BinOp | UnOp
   BinOp  ::=  '(' Dupl '(' E ')' '|' Dupl '(' E '))'Pairs |
               '((' Dupl '(' E ')' ',' Dupl '(' E '))'TimeSpan')'Pairs |
               '((' Dupl '(' E ')' ';' Dupl '(' E '))'TimeSpan')'Pairs
    UnOp  ::=  '(' E[Integer] ')Pairs' | '((' Ē ')'TimeSpan')Pairs'
    Dupl  ::=  'first_dup' | 'last_dup' | 'all_dup' | Integer'_dup'
   Pairs  ::=  'all_pairs' | 'unique_pairs' | 'repeated_pairs'
```

Fig. 12. Event Language of the A-mediAS system based on the EVA algebra

.

P1: (Notify the controller if) a traffic jam alert occurred and (if) one of the trucks is at that time
$e \in ((all\_dup(E_1), last\_dup(E_2^A))_{5min})_{all\_pairs}$

P2: Notify if a customer cancelled an order three times within a month.
$e \in ((all\_dup(E_3), all\_dup(E_3^{[2]}))_{4weeks})_{unique\_pairs}$

Note that an implementation of a new operator for each different parameter setting is not sufficient: The size of the profile language would increase significantly while providing the same semantic magnitude. Additionally, adapting the profiles to changing applications would be a complex task due to the fixed semantics of these operators. Our approach follows the concept of polymorphism: The semantics of the basic operator is determined by the parameter setting, which may change during the system's runtime. Profiles on primitive events are described using attribute-operator-value triples. The structure of profiles follows predefined event representation structures that are advertised in source profiles. Selected performance results for the filtering of composite events in A-mediAS can be found in [32]. A description of the adaptation as well as integration principles of A-mediAS have been presented in [30].

B. Language Survey

We performed a survey of event languages as a basis for comparison of existing event languages supporting composite events. Based on our algebra, we introduced a classification schema for profile definition languages. We identified five categories of profile languages. Based on these categories we proposed detailed transformation rules for translating profiles defined at the Meta-service into languages of system from the five categories (and vice versa for notifications). An extended description of our findings can be found in [37].

We identify five *groups* of composite event languages based on their support for time frames and by their support for composition operators. Parameters for Consumption Mode and Duplicate Handling are very rarely explicitly described in the literature – they will be consider separately.

We define five groups as shown in the table in Figure 13. There are two groups without time frame support: Simple Composite Events (CE) and Sophisticated Composite Events

Name	Conjunction	Disjunction	Sequence	Negation	Selection	Time Frame
CE	√	√	×	√	×	×
SCE	√	√	√	√	×	×
TCE	√	√	√	×	×	√
OTCE	√	√	√	√	×	√
STCE	√	√	√	√	×	√

Fig. 13. Groups of Event Languages

(SCE). Three groups have time frame support: Simple Time-framed Composite Events (TCE), e.g., Sentinel; Ordinary Time-framed Composite Events (OTCE), e.g., Samos; and Sophisticated Time-framed Composite Events (STCE), e.g., OpenCQ. The imbalance of the group assignment of negation and sequence is due to the different effect of time-frames on the operators.

C. Composite Event Language Transformation

For each language group, we introduce transformation rules for translating filter expressions defined at the Meta-service into equivalent filter expressions using a language of the group. As can be derived from the group definitions, a simple translation of filter expressions between groups is not possible. Instead, for different semantic concepts in two distinct groups, we have to find expressions that are semantically close. Additionally, auxiliary profiles and post-filtering may be required.

d) Profile Transformations: If a certain operator does not exist in one language a transcription expression has to be used. These transcriptions may be more or less expressive than the source expression. We define therefore four types of transformations: equivalent ($\longleftrightarrow$), positive ($\xrightarrow{+}$), negative ($\xrightarrow{-}$), and transferring transformation ($\xleftrightarrow{\#}$). Our notation is an extension of the notation used for Boolean transformations [16]. Equivalent transformations lead to expressions that have identical result sets. Positive transformations result in expressions that are less selective than the original - potentially creating larger result sets; negative transformations result in more selective expressions compared to the original filter expression (creating smaller result sets). Larger result sets without subsequent postfiltering lead to false positives in client notifications. Smaller result sets lead to missed event notifications. Transferring transformations (when omitting event patterns) use postfiltering and auxiliary profiles.

e) Post-filtering and Auxiliary profiles: For the considered transformations between language groups, not all of the original operations can be expressed in the languages of less powerful groups. In order to use weaker systems in cooperation with stronger ones, auxiliary profiles (i.e., additional filter expressions) have to be defined at the services. The filter results are delivered to the stronger system which then needs to perform additional simple filter operations (post-filtering).

f) Notification Transformation: Differing from query transformation, the result set obtained in an event notification service is not simply a set of tuples or documents but reflects a temporal connection between the events. If for two communicating systems, the less expressive system receives a message from a more expressive one, the notification might not be comprehensible to the less expressive filter language: Consider two systems A and B, where the filter language of A supports only sequences and disjunction, the filter language of B supports only conjunction. The systems cannot cooperate directly, since their set of filter operators are disjunct. In order to cooperate, system A defines a profile p_A at the Meta-service ($p_A = ((E_1; E_2)|(E_2; E_1))$). Meta-service transforms this expression into a profile p_B that is defined at system B: $p_B = (E1, E_2)$ with $p_A \longleftrightarrow p_B$. When system B sends a notification $n_B = (e1, e_2)$ to the Meta-service, the system A is notified by the transformed message $n_A = ((e_1; e_2)|(e_2; e_1))$.

Example 5.2 (Transformation for TCE): We here show some of the transformation rules for composite operators on the example of time-framed composite events (TCE) in Table I. In the TCE group, conjunction, disjunction, and sequence are directly supported. The selection is realized using transferring transformation. Negation can only be implemented in systems with a time concept using a transferring transformation.

g) Transformation of Operator Parameters: The group definitions given above abstracted from the parameters of consumption mode and duplicate handling strategy since these parameters are rarely explicitly supported in the considered systems. In Table II, we show the influence of considering parameter transformations on operator transformations (as introduced in the previous section). We show which transformation are possible when translating the parameter set of one system (y-axis in Table II) into the parameter set of another system (x-axis in Table II). That is, for all possible combinations of the duplicate and selection parameter we state whether the transformation is not possible (indicated by a dash) or equivalent (indicated by $\longleftrightarrow$) or only possible in one of the given directions while creating a larger result set (indicated by $\xleftarrow{\pm}$ and $\xrightarrow{\pm}$, where the arrow orientation defines the direction of the possible transformation).

	TCE to Meta-service	
Operators	timeless	timed
Conjunction	$(E_1, E_2)_\infty \longleftrightarrow (E_1, E_2)_\infty$	$(E_1, E_2)_T \longleftrightarrow (E_1, E_2)_T$
Disjunction	$(E_1 \vert E_2) \longleftrightarrow (E_1 \vert E_2)$	—
Sequence	$(E_1; E_2)_\infty \longleftrightarrow (E_1; E_2)_\infty$	$(E_1; E_2)_T \longleftrightarrow (E_1; E_2)_T$
Negation	—	$(E_1)_T \xleftrightarrow{\#} \overline{(E_1)_T}$ $N = \overline{N(E_1)_T}$
Selection	$(E_1) \xleftrightarrow{\#} (E_1)^{[i]}$, $N = (N(E_1))^{[i]}$	—

TABLE I
TRANSFORMATIONS TCE - META-SERVICE: E REFERS TO EVENT CLASSES, $N(E)$ TO NOTIFICATIONS REGARDING AN EVENT IN CLASS E , $t(N(E))$ TO THE TIME OF THE EVENT NOTIFICATION

D. Meta-Service implementation

The event algebra introduced here has also been used to implement a Meta-service: In this service we take advantage of the flexibility of the algebra which can be easily adapted to different existing languages by changing the parameter setting.

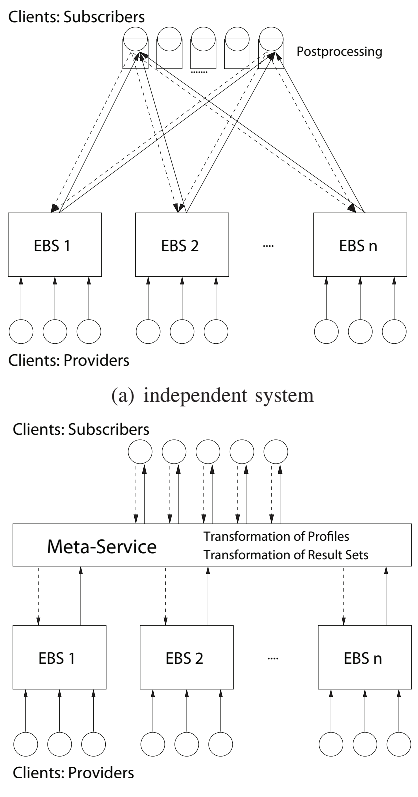
The Meta-service is the answers to three problems (see Figure 14(a) for illustration): (1) Subscribers of heterogeneous systems are forced to subscribe the same profile to a number of services using different filter languages, and (2) composite events combining events from different providers that are handled by different services have to be identified by a subscriber-based post-filtering (3) adaptation for changing applications or sensors Implementing yet another another service that hopes to combine all providers under one umbrella is futile for reasons of trust, downwards compatibility, company strategy, and required integration of legacy systems.

We introduce the meta-service as the equivalent of a Meta-Search Engine (see Figure 14(b)). The advantages are evident: Subscribers can have a uniform access for profile definition, having access to several event sources. Users are not repeatedly notified about the same event, i.e., duplicate recognition can be implemented on the meta-service level. In addition, security and privacy issues are easier to address.

Figure 15 shows the meta-service architecture. The meta-service has two data storages *profile store* & *client store*. The first one is used to store and manipulate subscribers' original profiles and transformed profiles. The second one stores the addresses and descriptions of the client systems. When the Meta-service system receives a new profile subscribed by a client, it will firstly store the profile in the profiles storage and then start Profile Transformation process for the profile. During the transformation process, the profile is transformed into different formats (one for each entry stored in the Client system Information storage) by applying transformation rules. The transformed profiles will be stored in the profiles storage (for notification transformation) and forwarded to the corresponding clients. On the other hand, when the meta-service system receives an Event Notification form one of the five client systems, it will firstly locate the corresponding transformed profile from the profiles storage and invoke post-filtering according to the transformation rule used on the

first	all	$\longleftrightarrow$							
	unique	$\overset{+}{\longleftarrow}$	$\longleftrightarrow$						
last	all	-	-	$\longleftrightarrow$					
	unique	-	-	$\overset{+}{\longleftarrow}$	$\longleftrightarrow$				
all	all	$\overset{+}{\longrightarrow}$	$\overset{+}{\longrightarrow}$	$\overset{+}{\longrightarrow}$	$\overset{+}{\longrightarrow}$	$\longleftrightarrow$			
	unique	-	-	-	-	$\overset{+}{\longleftarrow}$	$\longleftrightarrow$		
i-th	all	$\overset{+}{\longrightarrow}$	$\overset{+}{\longrightarrow}$	-	-	$\overset{+}{\longleftarrow}$	-	$\longleftrightarrow$	
	unique	-	$\overset{+}{\longrightarrow}$	-	-	$\overset{+}{\longleftarrow}$	$\overset{+}{\longleftarrow}$	$\overset{+}{\longleftarrow}$	$\longleftrightarrow$
Duplicate		first		last		all		i-th	
Parameter	Selection	all	unique	all	unique	all	unique	all	unique

TABLE II
PARAMETER TRANSFORMATIONS: DUPLICATE HANDLING AND CONSUMPTION MODE PARAMETER



(a) independent system

(b) meta-service

Fig. 14. Communication of clients with (a) several independent systems vs (b) with a Meta-service

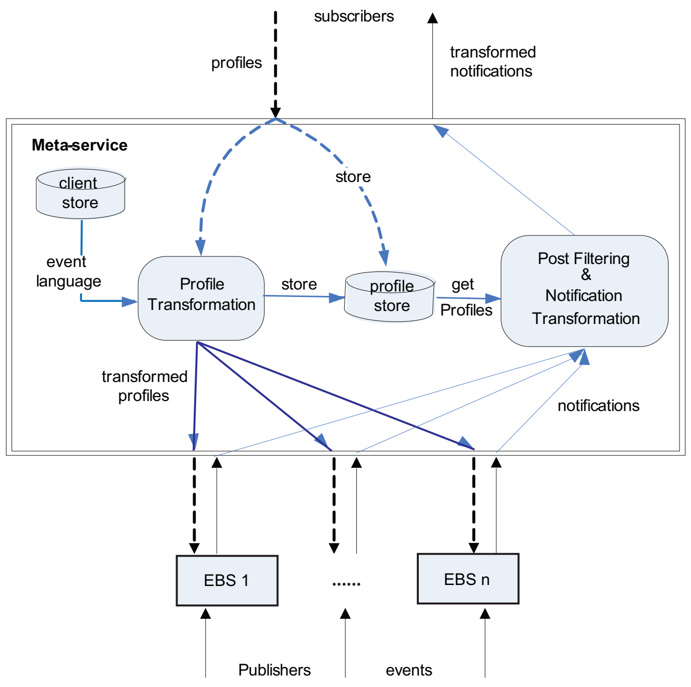


Fig. 15. Meta-service architecture

transformed profile. If the event notification was proved to be valid in the post-filtering, the system will locate the original profile from profile storage and transform the event notification to format which matches the profile language used in the original profile. Eventually, the subscriber will receive the transformed Event Notification from the Meta-service.

We tested the system in connection with five base systems of which each supports a different event language. In addition, each system is configured to receive events from only one publisher so that that no duplicate event notifications will be reported to the Meta-service.

Further details about the meta-service implementation can be found in [35].

VI. CONCLUSION

In this paper, we proposed a parameterized event algebra, EVA, for event-based systems. 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 selection and event consumption. Event selection describes which qualifying events from the trace are to be taken into account for composite events, and how duplicate events are handled. Event consumption defines whether a unique composite event or all possible combinations of events are taken into account. The combination of both parameters also offers the definition of filter patterns similar to the ones applied in parsing. The algebra and parameters are defined based on binary operators, by nesting composite events.

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

The approach presented here allows for event based applications that support changing or new event sources, as typical, e.g., for location-based services, without forcing the users to redefine their profiles for each new source. It is also suitable for integrating applications that combine events from sources with different event semantics as in logistics applications. We are currently implementing the prototype of a generic parameterized event system that is based on the parameterized event algebra introduced here. The service can be adapted to different application fields using various parameter settings. It can also be used for mixed applications such as the logistics scenarios introduced here. Additionally, our parameterized event algebra offers the possibility of easily integrating various event sources that are structured differently.

Acknowledgements: We wish to thank the participants of the Dagstuhl Seminar on Complex Event processing [15], who provided encouraging feedback on the algebra. We also wish to thank Sven Bittner, Xiaotie Huang, Doris Jung, and Steven Koenig for their work on the implementations related to the EVA event algebra.

References

[1] A. Adi and O. Etzion. Amit - the situation manager. *VLDB J.*, 13(2):177–203, 2004.

[2] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD*, Washington, D.C., 26–28 1993.

[3] J. M. Ale and G. Rossi. An approach to discovering temporal association rules. In *Proceedings of the SAC*, 2000.

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

[5] J. F. Allen and G. Ferguson. Actions and events in interval temporal logic. Technical Report TR521, University of Rochester, Computer Science Department, 1994.

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

[7] F. Bry and M. Eckert. Towards formal foundations of event queries and rules. In *Proceedings of the Second Int. Workshop on Event-Driven Architecture, Processing and Systems, Proceedings of 33rd International Conference on Very Large Data Bases, Vienna, Austria (23rd–27th September 2007)*, 2007.

[8] A. Buchmann C. Liebig, B. Boesling. A notification service for next-generation it systems in air traffic control. In *GI-Workshop "Multicast - Protokolle und Anwendungen", Braunschweig, Germany*, 1999.

[9] A. Buchmann C. Liebig, M. Cilia. Event composition in time-dependent distributed systems. In *Proceedings of the (CoopIS'99), Edinburgh, Scotland*, 1999.

[10] M. Calsyn. RendezVous Protocol (RVP). Internet draft, Microsoft Corporation, November 1997.

[11] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *Symposium on Principles of Distributed Computing*, 2000.

[12] A. Carzaniga, D. S. Rosenblum, and A. L Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.

[13] S. Chakravarthy and Q. Jiang. *Stream Data Processing: A Quality of Service Perspective*. Springer, 2009.

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

[15] M. Chandy, O. Etzion, and R. von Ammon, editors. *07191 Abstracts Collection – Event Processing*, number 07191 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[16] Chen-Chuan K. Chang, Hector Garcia-Molina, and Andreas Paepcke. Predicate rewriting for translating boolean queries in a heterogeneous information system. *ACM Transactions on Information Systems*, 17(1), January 1999. as technical report available at http://www-diglib.stanford.edu/diglib/WP/PUBLIC/DOC253.pdf.

[17] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. In *Proceedings of the ACM SIGMOD*, 2000.

[18] 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.

[19] J. Cohen and S. Aggarwal. General event notification architecture base. Internet Draft, July 1998. available at http: //www.alternic.org/drafts/drafts-c-d/draft-cohen-gena-p-base-01.txt.

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

[21] 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.

[22] EPTS. Event processing technical society. online at http://www.ep-ts.com.

[23] S. Gatziu and K. R. Dittrich. SAMOS: An Active Object-Oriented Database System. *IEEE Quarterly Bulletin on Data Engineering, Special Issue on Active Databases*, 15(1-4):23–26, December 1992.

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

[25] N. Gehani and H. Jagadish. Ode as an active database: Constraints and triggers. In *Proceedings of the Seventeenth International Conference on Very Large Databases (VLDB), pages 327–336, Barcelona, Spain, September 3-6 1991. NN-10*, 1991.

[26] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Composite event specification in active databases: Model & implementation. In *Proceedings of the VLDB*, 1992.

[27] 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.

[28] E. Hanson and J. Widom. Rule processing in active database systems. *Int. Journal of Expert Systems*, 6(1):83–119, 1993.

[29] A. Hinze. How does the observation strategy influence the correctness of alerting services? In Andreas Heuer, Frank Leymann, and Denny Priebe, editors, *Proceedings of the 9. German National Database Conference (BTW 2001), Oldenburg, 7.-9. March 2001*, Informatik Aktuell. Springer Verlag, New York, Heidelberg, Berlin, 2001.

[30] A. Hinze. A-mediAS: An adaptive event notification system. In *Proceedings of the DEBS International Workshop on Distributed Event-Based Systems at the SIGMOD/PODS*, San Diego, CA, 2003.

[31] A. Hinze. *A-MEDIAS: Concept and Design of an Adaptive Integrating Event Notification Service*. PhD thesis, Freie Universitaet Berlin, 2003.

[32] A. Hinze. Efficient filtering of composite events. In *Proceedings of the BNCOD British National Conference on Datbases*, London, UK, 2003.

[33] A. Hinze and D. Faensen. A Unified Model of Internet Scale Alerting Services. In *Proceedings of 5th International Computer Science Conference, ICSC'99 (Internet Applications.), LNCS 1749, Hongkong, China*, 1999.

[34] A. Hinze, K. Sachs, and A. Buchmann. Event-based applications and enabling technologies. In *Proceedings of the International Conference on Distributed Event-Based Systems (DEBS-2009), Nashville, TN, USA , July 6-9*, 2009.

[35] Xiaotie Huang. Design and implementation of a meta-ENS. Postgraduate project report, University of Waikato, Computer Science Department, 2005.

[36] H. V. Jagadish and O. Shmueli. Composite events in a distributed object-oriented database. In *Proceedings of the International Workshop on Distributed Object Management*, 1992.

[37] D. Jung and A. Hinze. A meta-service for event notification. In *Proceedings of CoopIS conference, LNCS*, 2004.

[38] L. Lamport. Times, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.
[39] 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*, Hong Kong, 27-30 May 1996. IEEE Press.
[40] L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. *IEEE Transactions on Knowledge and Data Engineering*, Special issue on Web Technologies, January 1999.
[41] L. Liu, C. Pu, and W. Tang. Supporting internet applicatioons beyond browsing: Trigger processing and change notification. In *Proceedings of 5th International Computer Science Conference, ICSC'99 (Internet Applications.), LNCS 1749, Hongkong, China*, 1999.
[42] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston,, 2001.
[43] Chaoying Ma and Jean Bacon. COBEA: A CORBA-based event architecture. In *Proceedings of the COOTS-98*, Berkeley, April 1998.
[44] D.L. Mills. Network time protocol (version 3) specification, implementation and analysis. Technical Report Network Working Group Report RFC-1305, University of Delaware, March 1992.
[45] I. Motakis and C. Zanilo. Formal semantics for composite temporal events in active database rules. *JOSI*, 37(1), 1997.
[46] G. Muehl, L. Fiege, and P. R. Pietzuch. *Distributed Event-based Systems*. Springer Verlag, Berlin/Heidelberg/New York, 2006.
[47] S. Navathe and R. Ahmed. A temporal relational model and a query language. *Information Sciences*, 49, 1989.
[48] 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.
[49] A. Prasad Sistla and O. Wolfson. Temporal conditions and integrity constraints in active database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Jose, CA, 1995.
[50] J. Riecke, O. Etzion, F. Bry, M. Eckert, and A. Paschke. Event processing languages. Tutorial at DEBS 2009.
[51] S. Schwiderski. *Monitoring the Behaviour of Distributed Systems*. PhD thesis, Selwyn College, University of Cambridge, April 1996.
[52] S. Schwiderski-Grosche and K. Moody. The spatec composite event language for spatio-temporal reasoning in mobile systems. In *Proceedings of the International Conference on Distributed Event-Based Systems (DEBS-2009), Nashville, TN, USA , July 6-9*, 2009.
[53] 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.
[54] D. Toman. Point-based Temporal Extensions of SQL. In *Proceedings of the DOOD, LNCS 1341*, 1997.
[55] S. Urban, I. Biswas, and S. Dietrich. Filtering features for a composite event definition language. In *SAINT '06: Proceedings of the International Symposium on Applications on Internet*, pages 86–89, Washington, DC, USA, 2006. IEEE Computer Society.
[56] A. van Hoff, J. Giannandrea, M. Hapner, S. Carter, and M. Medin. The HTTP Distribution and Replication Protocol(DRP).
[57] J. Widom. The Starburst active database rule system. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):583–595, 1996.
[58] S. Yang and S. Chakravarthy. Formal semantics of composite events for distributed environments. In *Proceedings of the International Conference on Data Engineering (ICDE 99)*, 1999.
[59] R. Zhang and E. Unger. Event specification and detection. Technical Report 96-8, Kansas State University, June 1996. available at http://www.cis.ksu.edu/~schmidt/techreport/1996.list.html.
[60] D. Zimmer, A. Meckenstock, and R.Unland. A general model for event specification in active database management systems. In *In Proceeding 5th DOOD*, pages 419–420, 1997.
[61] D. Zimmer and R. Unland. On the Semantics of Complex Events in Active Database Management Systems. In *Proceedings of the 15th International Conference on Data Engineering*, 1999.

Contents