Information Systems 48 (2015) 1–25

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

EVA: An event algebra supporting complex event specification

Annika Hinze [a,*], Agnès Voisard [b]

[a] University of Waikato, New Zealand
[b] Freie Universität, Berlin, Germany

ARTICLE INFO

Article history:
Received 6 May 2012
Received in revised form
23 June 2014
Accepted 16 July 2014
Recommended by: B. Kemme
Available online 1 August 2014

Keywords:
Event-based systems
Algebra
Complex event processing
Composite events

ABSTRACT

In applications such as digital libraries, stock tickers, traffic control, or supply chain management, composite events have been introduced to enable the capturing of rich situations. Composite events seem to follow 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 algebra 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, enables a mediator 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.

1. Introduction

Event-based systems (*EBS*) serve as a support for applications as diverse as business process monitoring, digital libraries, traffic control, or supply chain management [63]. Central to these systems is *complex event processing* (CEP), which has gained increasing attention in the past few years with recent focussed attention by industry (see for instance [44]). In addition, a number of books have recently been published on the issue of complex event processing (e.g., [79,89,27,46]).

Event-based systems trigger actions based on observed events. The most basic triggered action is the sending of a notification to an interested party. Events could be, for example, the change in the value 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. 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 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 off event conditions in a logistic application may be

P1: *a traffic jam alert occurred and a company truck is affected.*
P2: *a customer cancelled three orders within a month.*

* Corresponding authors.
E-mail addresses: hinze@waikato.ac.nz (A. Hinze), agnes.voisard@fu-berlin.de (A. Voisard).

http://dx.doi.org/10.1016/j.is.2014.07.003
0306-4379/

Various languages were designed and implemented for event processing. Some of them support only primitive events while more advanced systems also provide concepts for composite events. The description of composite events requires the definition of operators for event composition. A number of languages have been proposed with different features, e.g., the early language Snoop for active databases [29], CEDL which introduces parameters and aggregates [108], the Amit approach which considers complex events describing situations [5], and SpaTeC which introduces spatio-temporal reasoning [101]. Overviews of event languages have been presented in a tutorial at DEBS 2009 [99] and in a number of publications [68,116,38]. Despite the variety of available systems and languages, we still identify four issues in the handling of composite events[1]:

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 be either skipped or retained in the event-filtering process.
3. Lack of collaboration: No common framework exists that allows for comparison of event composition languages. The collaboration between event-based systems is a fortiori extremely limited.
4. Lack of adaptivity: Event conditions must be re-defined 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.

This paper introduces a parameterized event algebra EVA that supports adaptable event composition. Each of the complex elements of the algebra is discussed in detail using simple yet illustrative examples borrowed from a logistics application. The paper addresses the identified problems as described below.

To incorporate temporal restrictions, we use the notion of *relative time*. Our formalism is an extension of the semantics of *Event-Condition-Action* (*ECA*) rules of active databases [28]. Since EBS are typically used in the context of a distributed environment and cannot usually 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 are not addressed here. For the sake of simplicity, we assume that all occurring events can be ordered in a global system of reference.

The semantics of temporal operators as introduced, for instance, by Allen [7] 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 created 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 mediator service that allows collaboration between EBS based on predefined transformation rules.

The following list summarises the key contributions of this paper:

- Our *event algebra* EVA to define the semantics of complex events in a way that allows easy adaptation to different application needs; EVA has been used to define adaptable event composition in our prototype implementation.
- A set of *language groups* and *transformation rules* based on our algebra, which have been used to compare and translate between different composite event languages.
- A *mediator service* for event-based systems as proof of concept that implements a collaboration between heterogeneous event systems using our transformation rules for both event processing and notification post-filtering.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 gives necessary background information. After introducing an application scenario that will serve as a reference throughout the paper, we briefly define basic event-related concepts. We describe temporal event operators and semantic variations. In Section 4, we introduce our parameterized event algebra and apply it to our application scenario to illustrate the influence of various parameter settings. Section 5 describes our proofs of concept: an implementation of the algebra in an EBS, a language survey using the algebra and transformation rules to translate between different EBS, and the design and implementation of a mediator service for EBS. Section 6 compares the design of our

[1] We also introduced and discussed some of these issues at the Dagstuhl seminar on complex event processing held in 2007 [32].

algebra to related approaches. Finally, Section 7 holds concluding remarks and directions for future work.

2. 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. We also acknowledge a close relationship to stream processing and context-awareness.

We distinguish event-based systems according to their abstraction levels: application level, implementation/communication level, and system level. 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. Vice versa, event-based communication is not only used in event-based applications. Services on the communication level are called event-based infrastructures. They do not support composite events. Event-based communication on the web is supported by a number of protocols, none of which implement composite events. Finally, the problem of determining the event order based on incorrect time-stamps as well as time systems for distributed environments has been studied elsewhere and is out of scope here.

2.1. Active databases

In the area of *active database systems*, the problem of event rule specification has been studied for several years (also with special focus on composite events [51,53,114] and temporal conditions [42]). 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 [7] can be used (see the formal semantics in [88]). Ordering based on events, as opposed to states, has been implemented in the SAMOS system [50]. The work of Zhang and Unger [118] on semantic variation of operators is foundational and provided great insight into this matter. However, they do not consider the semantics of flexible parameters or time frames. Zimmer and Unland provide early work on comparing the semantics of different event languages but did not define a formal semantics [122]. Active database systems do not support adaptive system behavior and most of them are centralized systems that deal only with database-internal events. Furthermore, active database systems are event-based systems at the implementation level, whereas our work focuses on the application level.

In the context of active databases, temporal logic [8] is an alternative approach to describe composition operators (e.g., in [88,98]). A promising approach is the Enhanced Past Temporal Logic (PTL) introduced by Sistla and Wolfson [98], because it supports relative temporal conditions and composite actions. However, the desired flexibility would be lost[2] and not all operators and parameters can be expressed. A similar observation holds for Bry and Eckert's approach towards a formal foundation of event queries and rules [20].

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 [104,107] and a temporal relational model and query language [90]. In contrast to EBS, however, temporal and deductive databases focus on ad hoc querying.

2.2. Data mining

The areas of *temporal data mining* and *time series analysis* rely on temporal association rules [6,35]. 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.

2.3. Data stream processing and complex event processing

Stream processing analyses large amounts of data in real-time to detect quality-of-service patterns [27,73]. Applications generating data streams include sensor networks, news feeds, online auctions, web click-stream, network-traffic monitors, and supply chain systems with RFID tracking. Continuous queries perform operations on these streams using time-windows as filters on the continuously arriving data. Continuous queries are implementations of *transforming languages* [38]: operations that process the input streams by filtering, joining and aggregating to produce one or more output streams. Data stream elements may be relational tuples [83] or XML elements [57], and the stream operators are thus taken from relational databases or XML query languages, respectively. A large number of systems explore stream processing, such as Aurora [23], Fjord [81], NiagaraCQ [34], OpenCQ [78], MavStream [66], STREAM [12], and TelegraphCQ [31]. Stream processing has now become an established part of database management systems (e.g., Oracle 11g [36]) and dedicated commercial systems (e.g., TIBCO [106]).

Complex event processing (CEP) combines several sources of events to identify high-level patterns of events [46,79]. The systems use *pattern-based languages* to detect elements that match given conditions in the input stream. Conditions may be expressed by logical operators, content and timing conditions [38]. CEP also typically specifies actions to be taken once a pattern is detected, e.g., further processing of data or execution of business rules.

In recent years, a number of attempts have been made to combine event processing and stream processing

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

[26,4,111,41,38], sometimes referred to as event stream processing (ESP) [16,17].

2.4. Knowledge representation

The aim of knowledge representation (KR) is to facilitate reasoning and inference, with rules being one of the approaches to describe a knowledge base [109]. In KR business applications, interlinked distributed rules need to be managed and adapted at runtime.

Formalized KR rules have increasingly been used in semantic rule-based CEP [105,79,93]. These approaches aim to improve the quality of event processing by using event meta-data in combination with ontologies and rules. A number of formal languages have been proposed to express patterns and inference rules: Event Calculus [72], Situation Calculus [84] and interval-based Event Calculus [92] are formalizations of complex event patterns based on logical knowledge representation. Many languages support both event processing and reasoning [10].

Galton and Augosto provide a comparison of approaches to event definition from both the KR and active database communities [49]. The languages combining KR and ECA focus predominantly on richness of expression, including semantic reasoning and business rules. Furthermore, several works in event-based systems have used KR calculi to specify formal semantics of event algebras (e.g., [71]). None of these were developed to provide the desired flexibility, and thus do not support all parameters. Furthermore, similar to logic-based approaches to active databases (discussed in Section 2.1), they use logical event calculi in a manner that may require completely new or substantially changed expressions for small changes in the desired event semantics.

2.5. Semantics of event composition languages

Most event-based systems implement their own event composition languages [40,54,50,29,80,82,37,75,96,11] and, additionally, event pattern languages have been described in the literature [46,88,86,118].

Many of the early languages developed key language concepts that have now become well established. The HiPac project contributed the definition of coupling modes to specify when an action is executed with regard to the transaction in which the triggering event is detected (*immediate*, *deferred* or *separate*) [40]. The Ode language of the Compose system formally introduced the handling of composite events [53,54]. The SAMOS system included filtering of events external to the database [50–52]. The Snoop language of the Sentinel introduced parameter contexts to control the semantics of each operator [43,29]. SnoopIB extends Snoop to events with durations [1,3]. Rapide's language covers complex situation such as causality and event equivalences [80].

In our work, we are not concerned with database-specific constraints such as transactional context, or with the specification of triggered actions (as in HiPac). Moreover, awareness of event causality (as in Rapide) is beyond the scope of our work. Our focus is on semantics of composite event operators. Previous analyses of languages have shown that most languages provide merely syntax often with only hard-coded or informally described semantics. Moreover, many event algebras provide only event operators to describe the semantics of complex events, which leads to irregularities and confusion of concepts [120–122]. For example, both Ode/Compose and HiPac use a fixed consumption policy in each operator, which defines whether a matched event is to be excluded from further processing. Composition operators in SAMOS have fixed policies but can be partially influenced by combinations of operators.

Carson [22] compared a number of methods that have been used to formally specify event patterns: regular expressions [54], finite state automata [14,119,95], modal or temporal logic [98,71], and operator-based event algebras [29,88,122,58,64,115,22,5]. Operator-based event algebras build complex event patterns by (recursively) combining primitive events and composition operators. They have the advantage of most closely resembling the implemented event specification languages and are commonly used in active databases and general high-level event-based systems.

Most of these event algebras define similar composition operators to describe temporal relationships such as conjunction, sequence, and negation (e.g., [29,58,64,122,5,22]). It has been shown that the detection mechanisms in Snoop/Sentinel, SAMOS and Ode/Compose are based on different strategies, which introduce some subtle differences in the way patterns are defined although their respective event algebras look quite similar [22]. The work of Zimmer/Unland provided an initial framework for comparing these event algebras but did not define a formal semantics [70]. Their work was adapted by Baily and Mikulás into a formal framework using temporal logic that allows identification of decidability of event queries [15]. Etzion implements and expands on the concepts of Zimmer/Unland, which he applies in two rich language definitions [5,46], the semantics of which is not formally defined.

Motakis and Zaniolo have developed an event pattern language (EPL) with semantics based on datalog rules [88]. They provided translations of the patterns used in Ode/Compose, Samos and Snoop/Sentinel into similar expressions in EPL, thereby allowing for a comparison of these pattern languages.

The existing language comparisons imply a uniformity of languages that is deceptive [22,122,68]. A detailed analysis makes apparent that there is no simple way of comparing or automatically translating from one language into the other. None of the existing algebras have the flexibility that we propose here. This paper therefore does not aim to suggest yet another language, but proposes a parameterized algebra to describe, compare and easily translate between complex event languages. A more detailed comparison between the formal event algebra approaches will be provided later in Section 6.3.

3. 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.

3.1. Application scenario

Let us consider a company in Hamilton, New Zealand, 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 by 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

Fig. 1 depicts the following scenario. On a given day, customers A–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 min on average. The delivery list for that day that must be followed by trucks T1–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. Traffic jams could potentially cause delays that users may want to be informed about. Fig. 2 shows example profiles that may be defined in our scenario application. Our scenario illustrates, in particular, that a wide range of composite events may be of interest within a single application.

3.2. 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 an RFID tag, a web-page on the Internet, or abstract objects. 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 temperature rise of three degrees Farenheit may be significant, in other ones it may be negligible.

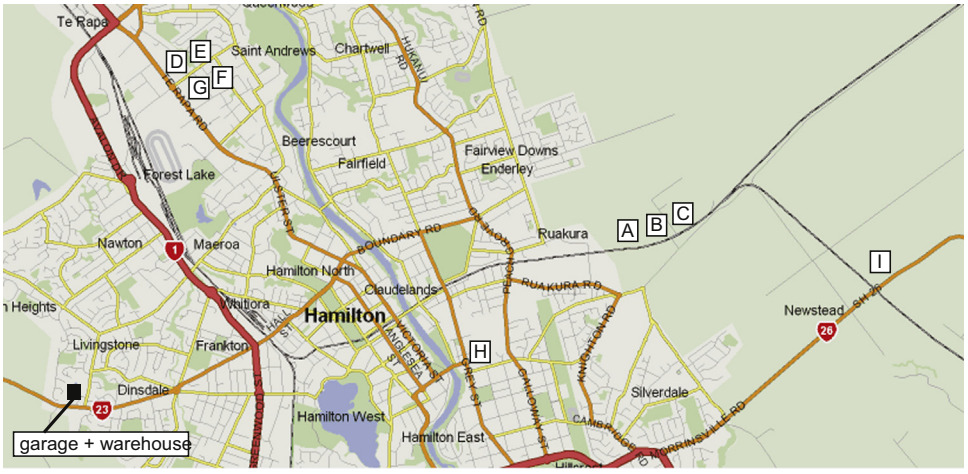

Fig. 1. Logistics scenario: warehouse and customers A–I in Hamilton (map courtesy of Wises.co.nz).

#	User	Condition
P1	Controller	a traffic jam alert occurred and (if) one of the trucks is at that time (±5 min) in that area.
P2	Analyst	a customer cancelled an order two times within a month
P3	Customer I	Truck T1 leaves from Location A or B for I after 2pm
P4	Controller	all trucks are back in the garage
P5	Controller	goods have not been picked up 2 hours after the start of the shift
P6	Controller	location of each truck every 30 min
P7	Truck driver	a customer from the delivery lists cancels the delivery.

Fig. 2. Example profiles for scenario.

As it is often the case, we assume that events have no duration and are "happenings of interest" at a certain time [89]. 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. Note that also sensor readings without a change of the sensor value constitute events [63]. 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 time-stamp as a mandatory attribute that refers to the time of the event's occurrence. The time system of reference for time-stamps is discrete. In our scenario, an example for a (primitive) event at a truck location is

e_{truck}: *event*(*truck_number*, T2;
location, (−37.783333, 175.283333);
time_stamp, 14:00:00 NZT)

We do not elaborate on attribute types here, e.g., string for attribute name, as event representation is not the focus of this work. We consider *primitive 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 the *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 and 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. [53]. 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 stream 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 $\mathrm{tr}_{t_0,\infty}$ with t_0 being the point in time the service started observing events.[3] We need a mechanism to refer to each event in a trace. Because 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 ith 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 [24], the matching operator is defined as follows:

Definition 3.5 (*Profile-event matching* $\sqsubset$). Consider the profile p and event e. If the condition defined by profile p is true for event e, this is denoted by $p \sqsubset e$. Then e is called a matching event for p.

Profiles can also contain wildcards and other operators such that not all attributes of the event have to be exactly defined. Each profile evaluation starts 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 a 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 events regarding temperature, however, the event representations may differ in other attributes (e.g., location). An example is the class of 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 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 time-stamp 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 a profile.

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

Duplicates could be, for instance, not only all truck location events of truck T1 referring to identical coordinates, but also all events referring to truck T1. Note that

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

duplicates need not necessarily have neither identical event representation nor identical time stamps.

3.3. 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. We extend the event operators from active database systems introduced in [51,28] 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 $e_1, \ldots, e_n \in \mathbb{E}$ be events that contribute to the composite event $e \in \mathbb{E}_C$. This relation is expressed as $\{e_1, \ldots, e_n\} \succ e$. $e_1, \ldots, e_n$ 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 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 $E_1 \neq E_2$. $t(e)$ refers to the occurrence time of e based on a reference time system; T denotes time spans in reference time units. 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.[4] 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.[5] 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 ith 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 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

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

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

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

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 (± 5 min) 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 all composite events $e = (e_1, e_2)_{5\ \mathrm{min}}, e_1 \in E_1, e_2 \in E_2^A$.

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

[5] $(E_1, E_2)_\infty$ refers to an event composition without temporal restrictions. It is equivalent to the original conjunction constructor as defined, e.g., in [51].

P2: (Notify the analyst if) a customer cancelled an order twice 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 $e = (e_{31}, e_{32})_{4\ \text{weeks}}$ with $e_{31}, e_{32} \in E_3$.

P3: (Notify customer I if) T1 leaves A and B for I after 2 pm: Let $E_{2\text{pm}}$ be the set of time-events occurring at 2 pm. 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 $e_t \in E_{2\text{pm}}, e_{41} \in E_4^A, e_{42} \in E_4^B$.

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 h after the start of the shift. Let E_7 be the set of events of loading goods, $E_{10\text{am}}$ the set time-events defining the start of the shift. Then the composite event is defined as $e = (e_t; (\overline{e}_7)_{2h})_{2h}$, $e_t \in E_{10\text{am}}, e_7 \in E_7$.

As we shall illustrate shortly, the event operators presented informally in this section do not describe the complete semantics of a profile definition language. For the event operators described above, different (application-dependent) semantics are conceivable. Applications may wish or need to switch between those semantics. We therefore introduce a *parameterized* event algebra (in the next section). For an event algebra in the EBS context, we need to consider the following *modes*[6]:

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 3.3. In the following sections we introduce modes 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 systems. In [122], 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.

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, three approaches can be distinguished: (1) Duplicates are ignored, (2) duplicates are overwritten by new ones, and (3) all duplicate events are taken into account. 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 the same class of events; they are duplicates. However, if each of these events needs to 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.

Event consumption: We distinguish three variations in the identification of composite events: (1) Matched events are not consumed and can contribute several times, (2) they are consumed by the composite event, or (3) after they are consumed the filter is re-applied. 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. 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 *re-applied* (repeated) after unique composite events have been identified. This means that after events are consumed, the remaining stream of events is reconsidered under the same profile condition. This approach may also be seen as a combination of the two parameters, event selection and event 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 4, 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 ith $E^{(i)}$. 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.

Example 3.3 (*Scenario with parameterized operators*). Profiles P1 and P2 from our example application are defined as follows when using the parameterized operators:

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

[6] We use a terminology developed in active databases [122].

P2: (Notify the analyst if) a customer cancelled an order two 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 $e \in (E_3^{(all)}, E_3^{(all)})_{4\ weeks}^{(unique)}$

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).

3.4. Profile-event situations

This section illustrates the implications of the parameters introduced above.

Binary operators. Fig. 3 shows a matrix of selected profile-event situations with identical parameter settings for each event in a pair. Other situations can be easily constructed. 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 4.4. Here, we use fixed time frames as evaluation intervals that are denoted with brackets [.] to make the different implications easier to compare.

The vertical dimension of the matrix (columns in Fig. 3) 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 match the same profile. With regard to that particular profile they are seen as duplicates. Using the matrix, we demonstrate example applications for the different approaches. One can easily see that using different sensors and varying application situations may require a dynamic adaptation of parameter settings.

I: *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: *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 most current value.

III: *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 rises 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 horizontal dimension of the matrix (rows in Fig. 3) show different versions of profile filter evaluation: apply the filter to all events, apply it only to the unmatched events, and 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

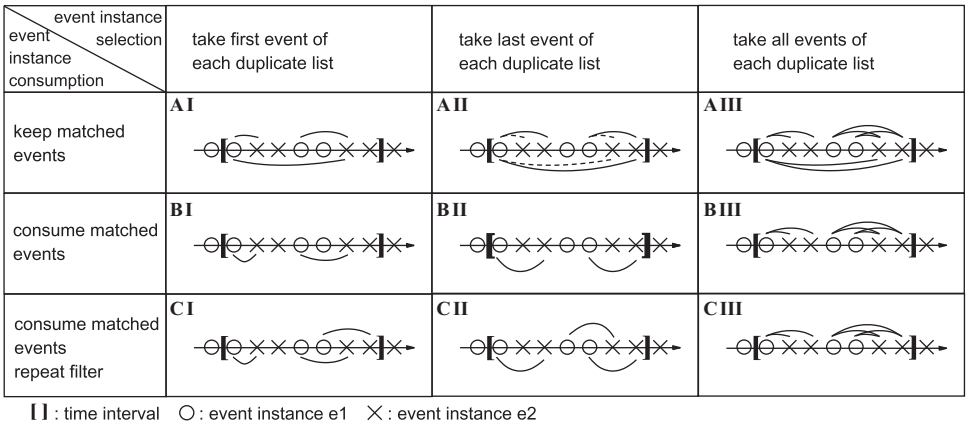

Fig. 3. Profile and event trace example under different evaluation conditions.

duplicate lists contain only events that are relevant to the profile (see details on trace view in Section 4.1).

A: *Keeping matched events* means considering all possible event combinations in a given series, which 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: *Consuming 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* 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 [9]. With this method interleaving event pairs can be identified.

In distributed event-based systems, the Siena system [25] implements evaluation style as in situation C I, Rebeca implements B I and B II, CQ [76] implements C I. In active database systems, SAMOS [50] implements C I and C III depending on the event operators, Snoop in Sentinel [29] uses B III, Compose in Ode [53] 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 that enables such a flexible implementation is presented in the next section.

Unary operators: For these operators, only the event consumption parameter applies. The semantic variations are shown in Fig. 4. The examples refer to the composite event of a selection $E_1^{[4]}$ in a given example 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. As for binary operators, changing the consumption parameter subtly changes the interpretation of the unary operator.

UA: keeping matched events means that in this case $E_1^{[4]}$ is interpreted as selecting every event $\circ \in E_1$ that is proceeded by three other E_1 events.

UB: refers to consuming matched events leads to $E_1^{[4]}$ identifying *the fourth event* $\circ \in E_1$. Note the difference to the binary operator example $(E1; E2)$ in Fig. 3, row B: event duplicate selection in binary operators applies within each duplicate list, which do not exist for unary operators.

UC: consuming matched events and re-applying the filter leads to *every fourth event* $\circ \in E_1$ being selected as the filter is repeatedly applied.

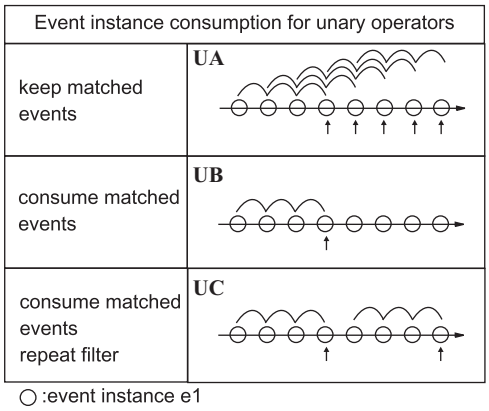

Fig. 4. Unary event profile and event trace example under different evaluation conditions.

4. 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.

4.1. Auxiliary 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.[7]

Definition 4.1 (*Trace view*). Let E_1 be an event class. The sub-trace $\mathrm{tr}(E_1)$ 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 $\mathrm{tr}(E_1, E_2)$ contains all $e_1 \in \mathrm{tr}$ and $e_2 \in \mathrm{tr}$ where $e_1 \in E_1, e_2 \in E_2$. We also use the shorthand notation $\mathrm{tr}(e_1, e_2)$. Note that the events in $\mathrm{tr}(E_1)$ 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 $\mathrm{tr}[1, \cdot], \ldots, \mathrm{tr}[n, \cdot]$ such that each sublist contains all successive events from the same event class. Every element of such a sublist is denoted with $\mathrm{tr}[x, y]$, where $x \in \mathbb{N}$ is the number of the sublist and $y \in [1, \mathit{length}(\mathrm{tr}[x, \cdot]))$ is the index-number of the element within the sublist.

[7] 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.

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

$$e_1 \in \text{tr}, \neg \exists e_2 \in \text{tr}: t(e_2) \in (t(\text{tr}[n,1]), t(\text{tr}[n, \mathit{length}(\text{tr}[n,\cdot])]))$$

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: $\text{tr} = \langle e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12} \rangle$ 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 Fig. 5. The (E_1, E_2)-trace view is then defined as $\text{tr}(E_1, E_2) = \langle e_1, e_3, e_4, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12} \rangle$.

The renumbered trace view is then $\text{tr}(E_1, E_2) = \langle \text{tr}[1,1], \text{tr}[1,2], \text{tr}[1,3], \text{tr}[2,1], \text{tr}[2,2], \text{tr}[3,1], \text{tr}[3,2], \text{tr}[4,1], \text{tr}[4,2], \text{tr}[4,3] \rangle$ with $\text{tr}[1,\cdot] = \langle e_1, e_3, e_4 \rangle$, $\text{tr}[2,\cdot] = \langle e_6, e_7 \rangle$, $\text{tr}[3,\cdot] = \langle e_8, e_9 \rangle$, and $\text{tr}[4,\cdot] = \langle e_{10}, e_{11}, e_{12} \rangle$. Obviously, the list length of the first sublist follows with $\mathit{length}(\text{tr}[1,\cdot]) = 3$. The example is depicted in Fig. 5.

Note that we denote (unordered) sets of events by E ($\mathbb{E}$ being the set of all events) while $\text{tr}[.,\cdot]$ denotes temporally ordered lists of events (allowing for duplicates). As discussed 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 $\nu_{\min}$, $\nu_{\max}$, $w_{\min}$, $w_{\max}$, and P_{EIC} influence the operator semantics. The values of $\nu_{\min}$, $\nu_{\max}$, $w_{\min}$, $w_{\max} \in \mathbb{N}$ control the event selection parameter; they refer to the lower and the 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 $(E_1 | E_2)$.

4.2. Binary operators

The disjunction implements an inclusive alternative (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

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

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

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

Different values for the open parameters $\nu_{\min}$, and $\nu_{\max}$ are discussed subsequently (see Figs. 6 and 7).

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

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

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

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

Again, different values for the open parameters P_{EIC}, $\nu_{\min}$, $\nu_{\max}$, $w_{\min}$, 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

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

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

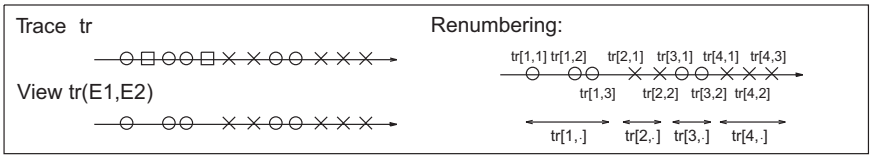

Fig. 5. Trace and renumbering in Example 4.1.

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')	$v_{min} = 1$,	$w_{min} = 1$, $w_{max} = m$
	$v_{max} = length(tr_{ant})$,	

Fig. 6. Event selection: parameters for first, ith, and last event within each duplicate list (columns I–III in Fig. 3) where $m \in \mathbb{N}$ with $\forall j > m$: $t(\mathrm{tr}_{post}[j]) > t(\mathrm{tr}_{post}[.,.]) + T$, 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. 7. Event consumption: parameter for unique and all pairs (rows A and B in Fig. 3)

events of a given trace tr is then defined as

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

As stated for Definitions 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.

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 for selecting events from duplicate lists (as defined in Fig. 6). Each operand has to be evaluated differently, depending on the position of the operand relative to the binary operator.

The selection of the ith 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 Fig. 7: the selection of unique pairs (each event in the matching set participates in one pair only, as in Row B in Fig. 3) and the selection of all pairs (as in Row A in Fig. 3).

The third approach, as depicted in Row C in Fig. 3, 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 matches 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.

4.3. Unary operators

The definition of unary operators (selection and negation) is discussed only briefly. While the sequential variations support the selection of the ith 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

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

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

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

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

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

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

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

Unique events are detected with $v_{\min} = v_{\max} = 1$, all events are detected with $v_{\min} = 1$, $v_{\max} = length(\mathrm{tr}[x,\cdot])$. Both binary and unary operators holds that the repeated filtering after event matching is more complex.

4.4. 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 at the *closing* of the evaluation interval. A *continuous* evaluation of all incoming events offers the advantage of early notification. However, the information delivered may not be as accurate as the information available at the conclusion of the time frame because new events may 'overwrite' the previous last event.

An example is shown in situation A II in Fig. 3 (dashed lines). This approach is appropriate for several applications such as catastrophe warning systems for environmental surroundings or other systems for urgent information delivery. A business example is a user who might want to know that an initial flight and hotel booking is being made for them even though this information might be overwritten later by a better offer. Evaluation at the closing of the evaluation interval would result in a single notification with the best offer, continuous evaluation would result in an earlier event being overwritten by the later, better booking.[8]

4.5. 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 4:

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,

The parameters for profiles P1 and 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 (± 5 min) in that area.
- Composite event description: $e \in (E_1^{(all)}, E_2^{A(last)})_{5\min}^{(all)}$
- Instance consumption: All pairs have to be considered: $P_{EIC} = \{(x,y) | x \in \mathbb{N}^+ \wedge y \in \mathbb{N}^+\}$.
- Instance selection: For the truck location events we consider the last event in the duplicate groups ($v_{\min} = v_{\max} = length(\mathrm{tr}_{ant})$ or $w_{\min} = w_{\max} = m$ as defined in Section 4). The events are evaluated continuously. Traffic jam events are all considered, since several traffic problems can occur within the same area, and all of them have to be taken into account ($v_{\min} = 1, v_{\max} = length(\mathrm{tr}_{ant})$ or $w_{\min} = 1$, $w_{\max} = m$).

The parameters in this example are similar to type A II in the matrix in Fig. 3. An example trace and the associated profile evaluation is given in Fig. 8.

P2: Notify if a customer cancelled an order two times within a month.
- Composite event description: $e \in (E_3^{(all)}, E_3^{(all)})_{4weeks}^{(unique)}$
- Instance consumption: Only unique composite events have to be considered: $P_{EIC} = \{(x,y) | x \in \mathbb{N}^+ \wedge y \in \mathbb{N}^+\}$ 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 ($v_{\min} = 1, v_{\max} = length(\mathrm{tr}_{post})$ and $w_{\min} = 1$, $w_{\max} = m$).

The parameters in this example are similar to type B III in the matrix in Fig. 3. An example trace and the associated profile evaluation is given in Fig. 9. The first event pair only qualifies for a partial event, no notification is sent. The next three events (i.e., 2–4) qualify, the fifth event does not qualify since only unique composite events are allowed.

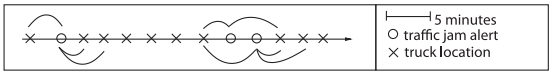

Fig. 8. Trace and evaluation example for Profile P1.

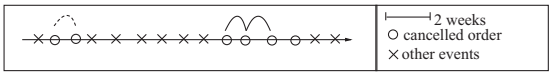

Fig. 9. Trace and evaluation example for Profile P2.

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.

4.6. Algebraic characteristics

It has already been noted that most event algebras have not been defined formally. Paschke et al. observed that procedural semantics without precise formalism are a serious omission for many real-world applications that depend on validation and traceability of the effects of events [92]. For languages with formally defined semantics, algebraic characteristics have rarely been made explicit but can be inferred from the underlying formalism (e.g., regular expressions [54], finite state automata [14,119,95], modal or temporal logic [98,71]). Carlson defines an algebra for which a number of intuitive algebraic properties have been proven [22].

As a complete and detailed proof of semantic characteristics is beyond the scope of this paper, we here sketch the necessary steps to reach both *correctness* and *completeness*.

We aim to adjust the algebra such that each composite event definition becomes expressible as an automaton. The definition of a trace as a semi-ordered list (cf. Definition 3.4) is required by the application domain and cannot be reduced into a strict order. However, we argue that to achieve sound results (e.g., in the selection of the first event of a list) an arbitrary but deterministic order can

[8] This is an adaptation of an example in [46].

```
      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. 10. Event Language of the A-mediAS system based on the EVA algebra.

be defined on events with the same time-stamp (e.g., based on event source). Under these conditions, each of the components of the event composition can then be shown to be expressed by automata (thus ensuring *soundness* and *completeness*).[9] The events created by our event algebra form a *closed set* as both binary and unary operators are functions on the event space that refer back into the event space (see Definitions 4.4–4.8). The matching events of any profile on a given trace form another trace of (composite) events.

We further present a selection of properties of the algebra and show to what extent the operators follow their intuitive behavior. We use the notion of equivalence of profile evaluation on traces:

Definition 4.9 (*Trace equivalence*). For two profiles p_1 and p_2, we define equivalence $p_1 \equiv p_2$ to hold iff $p_1(\mathrm{tr}) = p_2(\mathrm{tr})$ for any trace tr.

The following properties for profiles p_1 and p_2 with time span T follow from the definitions in Sections 4.1–4.3 for all settings of EIS and EIC:

1. $(p_1|p_1) \equiv p_1$
2. $(p_1|p_2) \equiv (p_2|p_1)$
3. $(p_1, p_2)_T \equiv (p_2, p_1)_T$
4. $(p_1, (p_2, p_3)_T)_T \equiv ((p_1, p_2)_T, p_3)_T$
5. $(p_1; (p_2, p_3))_T \equiv ((p_1; p_2)_T, (p_1; p_3)_T)$
6. $(p_1; (p_2|p_3))_T \equiv ((p_1; p_2)_T|(p_1; p_3)_T)$
7. $(p_1; (p_2; p_3)_T)_T \not\equiv ((p_1; p_2)_T; p_3)_T$
8. $(p_1; p_2)_T^{[i]} \not\equiv (p_1^{[i]}; p_2^{[i]})_T$

Property 7 is a well-known property and also discussed in Section 6.1, property 8 has been discussed in the comment on UB (consuming matched events) in Section 4.3.

[9] We did not originally define the algebra using automata as these would need to exhaustively create all possible parameter combinations leading to a loss of the flexibility desired for EVA.

5. Proof of concept

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

5.1. Reference language implementation

We implemented a distributed event-based system A-mediAS [59,60], 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 Fig. 10, 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, which has been discussed in Section 4.4.

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

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

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

An implementation of a new operator for each different parameter setting would not be sufficient: The size of the profile language would increase significantly while providing the same semantic magnitude. Furthermore, adapting the profiles to changing applications and sensors would be a complex task due to the fixed semantics of these operators.

Table 1
Language groups based on their support for operators and time frames. The asymmetrical assignment regarding negation and sequence is due to the effect of time-frames.

Group	Conjunction	Disjunction	Sequence	Negation	Selection	Time frames
CG−①	×	×		×		–
CG−②	×	×	×	×		–
CG−③	×	×	×			×
CG−④	×	×	×	×		×
CG−⑤	×	×	×	×	×	×

Our approach follows the concept of polymorphism: The semantics of each basic composition 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 system has been tested for both performance and scalability. Selected performance results for the filtering of composite events in A-mediAS can be found in [61,62]. The system uses a recursive optimization approach where later parts of a composite profile are only instantiated once the earlier parts have been matched [61]. A description of the adaptation as well as the integration principles of A-mediAS has been presented in [59,60].

5.2. Composite event language transformation

After briefly describing five language groups, the details of language transformations are introduced separately for operators and parameters. We show selected examples of transformation rules for one language group.

Language groups: Based on our algebra, a classification schema for profile definition languages was developed. Five groups of composite event languages are identified based on their support for time frames and composition operators (see Table 1). There are two groups without time frame support and three groups with time frames. Parameters for event consumption and duplicate handling have been described inconsistently in the literature and were thus considered separately. When assigning languages to these groups, the decisions were largely made based on support for sequence and negation as these are important concepts that are difficult to express using other operators.

Profile transformation: For each group and each parameter, we defined transformation rules for translating a filter expression of a mediator service (see Section 5.3) into the target languages of other systems. The mediator service is assumed to support all of the concepts introduced in EVA. Equivalent transformation of rules between different groups may not always be possible and we therefore have to identify expressions that are semantically close. In these cases, auxiliary profiles and post-filtering will be applied.

Transcriptions of profiles may be more or less expressive than the original expression. We used four types of transformations (extending concepts of Boolean transformations [33]):

- equivalent transformation $\longleftrightarrow$
- positive transformation $\xrightarrow{+}$
- negative transformation $\xrightarrow{-}$
- transferring transformation $\xrightarrow{\#}$

Equivalent transformations result in expressions that lead to identical result sets. Positive transformations lead to larger result sets (expressions are less selective), and may require post-filtering for false positives. Negative transformations would have smaller result sets (more selective) and may lead to missed events – these need to be avoided. Transferring transformations may use auxiliary profiles to construct equivalent queries from alternative operators.

For supported operators, transformations are used to insert time-frames where needed. Transformations for operators that are not supported are more complex. In the mediator service, profiles may be timed (indicated by subscript T) or without time-frame (subscript ∞). Profiles from the mediator service are marked here by a superscript *med*, profiles from one of the language groups with the respective group number.

For simplicity we do not define here the transformation rules for all groups in detail, but instead show significant examples. Selected transformation rules for languages in CG−① are as follows:

- timeless conjunction: $(E_1,E_2)^{med}_{\infty} \longleftrightarrow (E_1,E_2)^{①}$
- timed conjunction: $(E_1,E_2)^{med}_{T} \xrightarrow{+} (E_1,E_2)^{①}$
- timeless sequence: $(E_1;E_2)^{med}_{\infty} \xrightarrow{\#} (E_1,E_2)^{①}$; additional post-filtering is needed to achieve $t(e_1) < t(e_2)$ for $e_1 \in E_1$ and $e_2 \in E_2$.
- (timeless) selection: $(E_1)^{[n]med} \xrightarrow{+} (E_1,\ldots,E_1)^{①}$

We refer to [68,69] for the detailed transformation rules defined at the mediator service for translating profiles into languages of the five groups (and vice versa for event notifications). An approach that uses evaluation strategies similar to composite transformation rules is implemented in ZStream [85]: the system uses non-deterministic finite automata to achieve efficient detection of composite events (e.g., rewriting the query plan for sequence detection into a faster detection using conjunction).

Parameter transformation: Using our parameterized algebra EVA, transformation of parameters is now straightforward. Note how the transformation of EIC and EIS is not independent but expresses the interconnectedness of the two parameters. The influence of parameter transformations on operator transformations (as introduced above) is described in Table 2.

Table 2
Transformation impact for selected parameter settings of event instance selection (EIS) and event instance consumption (EIC).

EIS	EIC	First		Last		All		*i*th	
Parameter		All	Unique	All	Unique	All	Unique	All	Unique
First	All	$\longleftrightarrow$							
	Unique	$\overset{+}{\longleftarrow}$	$\longleftrightarrow$						
Last	All	–	–	$\longleftrightarrow$					
	Unique	–	–	$\overset{+}{\longleftarrow}$	$\overset{+}{\longleftarrow}$				
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	–	$\overset{+}{\longrightarrow}$	–	–	$\overset{+}{\longleftarrow}$	$\overset{+}{\longleftarrow}$	$\overset{+}{\longleftarrow}$	$\longleftrightarrow$

Finally, Table 3 shows an overview of languages with operators, time frames and their identified composition group, and support for consumption (EIC) and duplicate handling during event selection (EIS). Languages with identified groups can be handled according to the transformation rules. Rules from other languages may have to be adapted manually.

5.3. Mediator service implementation

The event algebra introduced here has also been used to implement a mediator 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 mediator service is the solution to three problems occurring when using several independent EBS (see Fig. 11(a) for illustration): (1) Users (clients) of heterogeneous systems are forced to subscribe the same profile to a number of services using different filter languages, (2) composite events combining events from different providers that are handled by different services have to be identified by client-based post-filtering, and (3) changes in applications or sensors require fine-grained manual adaptations. Implementing yet 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 mediator service to enable collaboration between services (see Fig. 11(b)). The advantages are evident: Users can have a uniform access for profile definition while addressing several event sources. Users are not repeatedly notified about the same event, i.e., duplicate recognition can be implemented on the mediator service level. In addition, security and privacy issues are easier to address. The system was developed in three stages:

(1) *Simple mediator*: This service sets up communication between the mediator and a number of event services with different languages. It syntactically translates and distributes profile definitions to the services and combines incoming results, without profile and result transformations.

(2) *Transformation mediator*: The service extends the simple mediator to incorporate semantic transformation rules. When a profile for a composite event class is defined at the mediator service, it is transformed according to the rules described in Section 5.2. The resulting profiles are then subscribed to all target services. The resulting profile matches are post-filtered at the mediator, where necessary, and final matches are signaled to the subscribed users.

(3) *Collaboration mediator*: Composite profiles defined at the mediator are not only syntactically and semantically transformed into different languages as in Stage 2. Events that are composed by contributing events from different services are also detected and signaled at the mediator service. This is achieved by decomposing the composite event profile into a set of primitive profiles, which are subscribed to the services.

Architecture and design: Fig. 12 shows the mediator service architecture. The mediator service has two data stores for capturing client information and profiles, respectively. The client storeholds addresses and descriptions of each of the client systems. The profile store holds both users' original profiles and transformed profiles. When the mediator service receives a new profile, they are entered into the profile store and then forwarded to profile transformation. During the transformation process, each profile is transformed into different formats (one for each entry in the client store) by applying the transformation rules outlined in Section 5.2. The transformed profiles are then entered into the profile store (for post-filtering and notification transformation) and forwarded to the corresponding clients.

When the mediator service receives an event notification from one of the client systems, it locates the corresponding transformed profile in the profile store and invokes the post-filtering according to the transformation rules used on the original profile. If the event notification is identified to be valid in the post-filtering, the system will locate the original profile in the profile store and transform the event notification to the format matching the original profile language (notification transformation). Finally, the user receives the transformed event notification from the mediator service.

Evaluation: We tested the system in connection with five base systems each supporting a different event language (one from each language group defined in Section 5.2). In addition, each base system was configured to receive events from only

Table 3
Event composition semantics of selected languages. Some languages implement a simultaneity operator (e.g., CQ, Ready, Eve), which we consider to be a special case of the conjunction (similar to EPL [88]). Other languages have repetition or iteration operators (e.g., ECCO, SAMOS, and Eve) – some of these we identify as explicit EIS parameters, others are equivalent to our selection operator. More detailed language comparisons can be found in [5,115,38] (based on adapted Zimmer/Unland policies) and [68,69] (based on both EVA and Zimmer/Unland).

System	Operators					Time frames	Composition Group	Consumption modes			Duplicate handling				
	Conjunction	Disjunction	Sequence	Negation	Selection			Keep	Consume	Con. and reapply	First	Last	All	*i*th	n to m
A-mediAS [60]	×	×	×	×	×	×	⑤	×	×	×	×	×	×	×	×
Amit [5]	×	×	×	×	×	×	⑤	×	×		×	×	×	×	
CEA [14]	×	×	×	×			②				×				
CQ [77]			×			×									
DistCED [96,97]	×	×	×		×	×	③	×					×		
Eve [55]	×	×	×	×	×	×	⑤			×	×			×	
ECCO [115]	×	×	×	×	×	× [a]	⑤	×		×	×	×	×		
Etalis [11]	×	×	×	×		×	④	×	×	×	×	×	×		
GEM [82]	×	×	×	×		×	④		×		×				
Padres [75]	×	×	×		×	×	③	×					×		
PLAN [112]	×	×		×		×					×				
Ready [58]	×	×	×	×			②				×	×	×	×	×
Rebeca [91]	×	×	×	×		×	④			×					
Samos [50]	×	×	×	×	×	×	⑤			×	×				
Siena [25]			×						×		×				
Snoop [53]	×	×	×	×	×	×	⑤		×						
SpaTeC [87]	×	×	×			× [a]	③		×	×		×	×		
T-Rex [37]	×	×	×	×	×	×	⑤	×	×		×	×	×	×	×
Ode [53]	×	×	×		×		③	×	×		×		×		
Yeast [74]	×	×	×	×		×	④				×		×		

[a] ECCO and SpaTeC detect composites based on both temporal and spatial relationships.

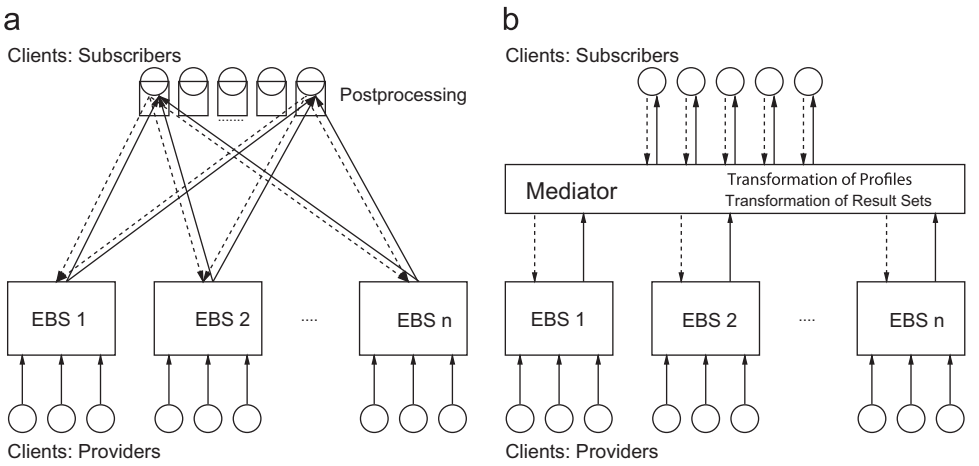

Fig. 11. Communication of clients with (a) several independent systems vs (b) with a mediator service.

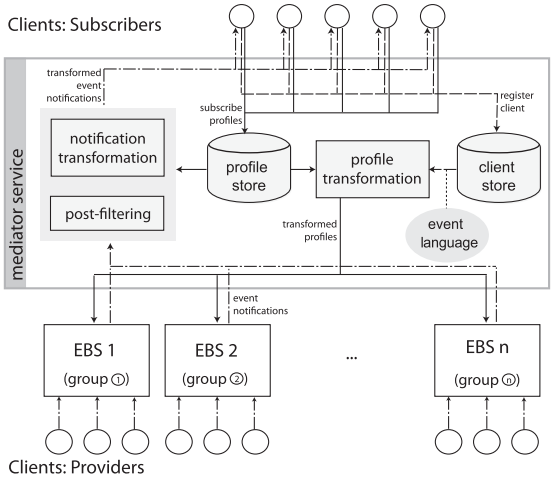

Fig. 12. Mediator service architecture.

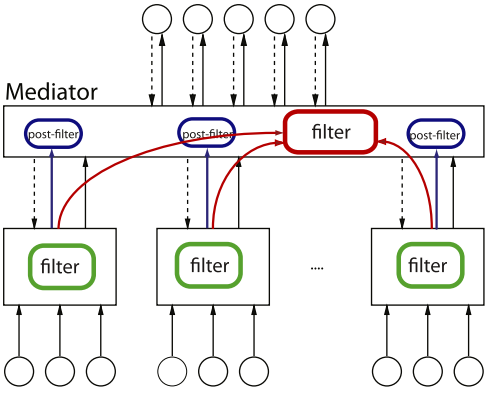

Fig. 13. Filtering and post-filtering. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this paper.)

one publisher so that no duplicate event notifications would be reported to the mediator service. We evaluated the influence of profile transformation (subscription performance) and post-filtering and notification transformation (notification performance). We summarize our findings as follows:

- #connected services: The subscription time increases with the number of connected services as each requires specialized profile transformations. The notification time is not affected.
- #profiles: The subscription time increases with the number of profiles. The notification performance at the mediator is independent of the number of profiles (as they are handled by a hash map).
- composition group: The simpler the supported profiles (lower composition group number), the greater the subscription time (due to the number of transformations required) and the notification time (influence of complexity in required post-filtering and transformations).
- collaboration support: A mediator with collaboration between services (for detection of composite events contributed by different services, stage 3) creates additional primitive profiles in comparison to a transforming mediator (stage 2). The final filtering of these profiles is done at the mediator (see Fig. 13) and its performance follows the characteristics of the filter algorithms used.

The overall filter and notification performance of the system (base services and mediator) is a composite of the independent filtering at the base services (indicated in green in Fig. 13) and the post-filtering/transformations in the mediator service (in blue). The collaboration support to detect composite events from different base services uses an additional filter in the mediator (in red), which may also use post-filtering results as input.

At current, the filter process at the mediator is multi-threaded while the subscriptions are handled by a single thread. Distributing the subscription process would alleviate its dependence on the number of connected services and profiles. Collaboration support adds a second filter process. This naturally impacts on performance but also allows detection of composite events that may otherwise

go unnoticed. Detailed evaluation of the impact of collaboration on the mediator is part of future work.

Further details about the mediator service implementation are available in [65]. We are aware of only one project that also aimed to provide uniform access to multiple event-based systems: the DENS system was developed as overlay service to give access to multiple event systems/languages in MANETs [102]. However, neither this system nor the mediated languages did support composite event detection.

6. Comparison of EVA to related approaches

Here we compare the approach taken in EVA with those of other languages and event algebras. We discuss semantic variations for event time, the use of window operators, and compare the EVA semantics to other approaches.

6.1. Interval vs point semantics

We defined events to occur at single points in time (see Definition 3.1) to represent the notion of events as instantaneous state transitions. This *point semantics* is a common approach in the literature (e.g., [19,110,46]) as well as in many system implementations (e.g., SAMOS [51], SNOOP [29], Reach [21], CEP over streams [113]).

An alternative approach uses *interval semantics*: events may be said to occur over a time interval [49,100,2]. We believe that here the distinction between events and states is blurred. For example, the arrival of a truck at the warehouse (see Scenario in Section 3.1) is a process which may start with the truck entering through the gate and is finished by the driver handling in the completed list of tasks done during the day. Point semantics describes this as the state of 'arriving', with a start and an end point. Interval semantics explicitly models arrival as an event with a duration.

Chakravarthy refers to point semantics as *detection-based semantics* as the event time is typically defined by the end of the detection interval, whereas interval semantics is referred to as *occurrence-based* [27,3].

Interval, or detection-based, semantics has some intuitive advantages in the ordering of complex events: here the oft-cited example of $E_1;(E_2;E_3)$ is matched only by the sequence $\mathrm{tr}=\langle e_1,e_2,e_3\rangle$. In point semantics, $E_1;(E_2;E_3)$ is also, somewhat counter-intuitively, matched by $\mathrm{tr}=\langle e_2,e_1,e_3\rangle$ (see, e.g., [49]). However, this is remedied in our algebra by an explicit use of bracketing (i.e., $E_1;(E_2;E_3)\neq E_1;E_2;E_3$), which is instantiated in our language implementation by the use of $seq(E_1;seq(E_2,E_3))$ for the above example.

Yoneki [116] additionally refers to *hybrid point-interval-based time-stamps* – these are point-based time-stamps expressed in interval-based format. The interval represents error margins introduced by detection delays and processing time. These types of time-stamps are used, e.g., in ECCO [115] and Rebeca [91]

Our event algebra defines temporal composites that rely on point semantics in a given time frame (i.e., interval). Its parameterized semantics could be translated into interval context, while composites based on order (e.g., sequence) would have to be adapted. Our main contribution of parameterization would also be applicable in this new context. However, interval-based semantics follows a different philosophy to point-based semantics and is beyond the scope of this paper.

6.2. Window operators

Stream processing targets detection of accumulated or aggregated events over (sliding) time windows. Time intervals (windows) of fixed lengths are applied on streams to extract data into relations. Then SQL-like queries are executed on these relations of changing database states. Sliding windows have also been used in active databases [1]. The notion of window operators is essential to allow for non-blocking evaluation of active streams. This is particularly significant for set operators such as join, sort and aggregates [27].

Different window models have been introduced: landmark window, damped window, and sliding window [103]. Landmark windows use all data between a fixed point in time (the landmark) and the current time. Damped windows give preference and weight to newer data, and sliding windows use a fixed-length interval of most recent data in the stream. The sliding window frame is widely used in the mining of data streams [56,39,48,27]. A number of sliding window models have been specified, such as tumbling windows in Aurora [23] and partitioned windows in CQL [13].

Our event algebra EVA is concerned with temporal relations between events and does not focus on aggregation. Both sliding and fixed windows (i.e., landmark windows) can be created in EVA using time events and durations. Additional windows can be freely defined based on time or events. The main difference is that such windows in EVA create filters on the event trace, resulting in trace views on the original trace (see Definition 4.1), not relations as in stream processing.

Adaikkalavan and Chakravarthy have shown that windows created by event consumption modes and sliding windows in stream processing do not fulfill the same purpose and behave independently of each other [1]. However, fine-grained and elaborate window specifications are not targeted in EVA and might often result in cumbersome specifications. They are outside the scope of this paper.

6.3. Algebras and semantic comparisons

It has been repeatedly observed that event consumption rules are mostly applied in implementations without clear semantic definition [117,86,22,122]. A first approach to compare the semantics of event detection in active database system was presented in [47], however, event composition was beyond the scope of that work. Motakis and Zaniolo have developed an event pattern language (EPL) with semantics based on datalog rules [88]. They provided translations of the patterns used in Ode, Samos and Snoop into similar expressions in EPL, thereby allowing for a comparison of these pattern languages. However, they encoded semantic variations into different operators

Table 4
Semantics comparison between policy contexts in Snoop and parameters in EVA. Note that cumulative context in Snoop additionally aggregates all event instances of E_1, whereas EVA does not contain aggregation.

Snoop	EVA		
	EIS E_1	EIS E_2	EIC
Unrestricted	all	last	keep
Recent	last	all	keep
Chronicle	first	first	consume
Repeat continuous	all	first	keep
Cumulative	agg(all)	first	consume

(e.g., immediate sequence and relaxed sequence), which is the very approach that our current work aims to avoid. The comparison of formal specifications provided in [94] only recognized transactional coupling modes and did not consider event composition policies.

The following algebras and frameworks are most closely related to our work; we provide a brief comparison of their approaches to our work and note the extent to which language comparison and mediation have been implemented.

Policy contexts in Snoop: The Snoop language of Sentinel uses the semantics of *event expressions* and four policy contexts (*recent*, *chronicle*, *continuous*, and *cumulative*).[10] Formalization of recent, chronicle and continuous context is defined in [30], and [1,2] define an interval-based semantics of the contexts. Event consumption semantics in Snoop depended on both the context policy and the operator semantics [26]. For example, the *recent* context for a sequence $(E_1; E_2)$ is implemented as all pairs of last E_1 and all E_2 (short: all, last, keep) and the recent context for a disjunction is implemented as (all, all, consume). It has further been shown that Snoop policy contexts may lead to occasionally ambiguous semantics [122]. Table 4 compares the semantics of EVA and Snoop using the predominant semantics of each policy context while ignoring the semantic ambiguities.

SAMOS [50] used the *chronicle* context as default unless specified otherwise by the operator (i.e., semantics are mixed). Ode [53] and Reach [21] offer *chronicle* and *recent*; Acood [45] used *recent*, *chronicle*, and *cumulative*. Benauer et al. implemented composite event detection for XML documents extending Snoop's policy contexts for hierarchical as well as temporal order [18]. ECCO uses a combination of *unrestricted*, *recent* and *chronicle* for consumption modes, and a subset policy is used for event instance selection based on intervals [115]. Event duplicates in ECCO are handled by specific *selection* and *aggregation* operators. Yoneki [116] provides a simple language comparison in relation to the ECCO semantics but no semantic details are given. No language transformations have been suggested.

All four policy contexts have been used in the works of Mellin [86] and Carlson [22]. Whereas in both Snoop and the work of Carlson, policy contexts are applied only once to the expression as a whole [22], Mellin redefines these contexts such that they become independent from the operators [86]. In this respect, the work by Mellin is similar to the algebra defined here.

Decorators in YALES: The work of Zhang and Unger [118] on semantic variation of operators is foundational for the research reported here. They use *rules* with semantic *decorators* for *selection* and *trimming* of events. In their YALES language, time frames are not assigned to operators but are inserted by means of separate calendar events. The result of a rule execution in YALES is the creation of another event trace. The selection operator specifies which of the matching events should be selected, and the trimming operator specifies the formation of the new event trace based on these selections. The trimming decorators have influenced those in EVA but do not provide a direct match as their function is different (see Table 5). The YALES semantics has been used neither for language comparisons nor language transformations.

Policies by Zimmer/Unland: The model from [122,120,121] documents early work on comparing the semantics of different event languages. Zimmer and Unland informally introduce a meta-model as a foundation for a structured and systematic evaluation, which uses policies of *instance selection* and *instance consumption*. In their model, the semantics of the instance selection policy is unclear as it is inconsistently used for examples (per event type) and language comparisons (per composition). The lack of formalism and the resulting problems have already been noted in [22,70].

Zimmer and Unland [122] described that the two policies of event *consumption* and *selection* are "to an extent" independent of each other and the composition operator, but gives no further details. From our semantics definition in Section 4, the points of correlation between EIC and EIS can be clearly seen.

The Zimmer/Unland model is used in a comprehensive analysis of the semantics of selected languages (restricted to EIC and EIS for initiator events); no language transformations have been explored. Our work builds on and extends the research done by Zimmer and Unland (for comparison see Table 6).

Policies by Cugola et al. [38]: This comparison of flow processing models refers to the notion of selection and consumption policies as defined in [122] but significantly differs in the semantic details (see Table 7). Their *selection policy* describes whether rules are allowed to fire *single* or *multiple* times. For *consumption policy*, they distinguish between *zero* consumption (no invalidation) and *selected* consumption (invalidation of events). Additionally they allow for strategies to be *programmable* for each rule. Programmable rules seem to refer to explicit parameter settings whereas all other settings seem to be hard-coded by each system. The comparison thus hides the possible complexity and parameter support of the advanced policy settings (indicated by question marks in Table 7).

Policies by Etzion/Niblett: Etzion and Niblett build on the Zimmer/Unland model and significantly extend its policy ranges [46]. The language semantics is very rich but described only informally. Earlier versions of some concepts stem from Amit [5]. The language described by Etzion and Niblett targets complex event processing with

[10] Snoop also provides the additional context of *unrestricted* as a generic default.

Table 5
Semantics comparison between decorators in YALES and parameters in EVA. Note that the trimming operator manipulates how YALES creates new event traces; decorator k refers to removal of unselected events from that trace. The removal of selected events (using decorator r) offers the potential for the remaining events to be included in a later match.

YALES		EVA	
Selecting decorators	Trimming decorators	EIS	EIC
1: latest		last	
e: earliest		first	
a: all		all	
	c: clear range		consume (+delete unmatched)
	r: remove selected in range		consume (potential for repeat filter)
	k: keep selected & remove rest		keep (+delete unmatched)
	u: no change to range		keep

Table 6
Semantics comparison between policies in [122] and parameters in EVA. We do not consider *ext-cumul* as it includes partial matches, i.e., event instances that do not match the order specified in the composition operator. The *ext-exclusive* policy is similar to the *c: clear range* decorator in YALES.

Zimmer and Unland		EVA	
EIS	EIC	EIS	EIC
first		first	
last		last	
cumulative		all	
	ext-cumul		n/a
	shared		keep
	exclusive		consume
	ext-exclusive		consume (+delete unmatched)

Table 7
Semantics comparison between policies in Cugola et al. and parameters in EVA. Note that no information is given about the expressiveness of programmable contexts policies.

Cugola et al.		EVA		
EIS	EIC	EIS E_1	EIS E_2	EIC
single		1st, last	all	
multiple		all	all	
programmable		(?)	(?)	
	zero			keep
	selected			consume
	programmable			(?)

aspects of both stream processing and event detection. It supports the detection of complex situations, such as trend patterns and spatial patterns, and has thus a different focus than EVA.

The language supports a number of policies referring to *temporal*, *spatial*, *state-oriented* and *segmentation-oriented* contexts. The ones most closely related to our work are the pattern policies of *consumption*, *repeated type* and *cardinality* (see Table 8). *Repeated type* refers to the handling of events of the same type. *Override* refers to a replacing of an older matching event with a new one that arrived later. The application of additional filter conditions (specifying max or min values) is possible but neglected here. The *cardinality policy* determined how many matching sets of events can be created. These policies are not independent as shown in Table 8. The semantics of repeated events and event duplicates are blurred, and the overall semantics of the language is very complex.

The predecessor language Amit has been compared with Snoop in [5,4]. Adaikkalavan and Chakravarthy [4] observe that in Amit the semantics of complex events using multiple modes is not sufficiently clear. EVA also uses modes for each complex event component but integrates the parameters into the operator semantics instead of handling them as separate policies. For the language proposed by Etzion and Nibblet in [46], neither comparison nor language transformations have been explored.

Consumption policies by Bailey and Mikulás, and Kiringa: Bailey and Mikulás provide a temporal-logic framework for analyzing the expressiveness of event algebras [15]. In particular, they are interested in the identification of decidability of event queries. Their work has been extended by Kiringa (using a situation-calculus framework) including the definition of further consumption modes [71]. Neither of these two frameworks has been used for language comparisons or language transformations. The relationship between their frameworks and EVA is shown in Table 9.

Summary of comparisons: We follow Zimmer/Unland [122] in the observation that algebras based on Snoop's policy contexts are too restrictive as the event semantics is defined by both operator and contexts. Mellin's approach [86] of clearly separating the context semantics from the operators is a necessary step. However, the contexts still combine (and therefore hide) the aspects of event selection and consumption (as can be seen in Tables 4 and 9).

Of the algebras that provide more than one parameter dimension, only YALES has been formally defined. However, YALES only formally defines the language operators; the decorator applications are introduced by means of formal syntactic rules but their effects are only described informally.

The algebra introduced here is the only approach that is parameterized throughout the complete semantics definition (as observed by Carlson [22]) instead of operator semantics with additional parameters.

None of the approaches discussed here use the notion of duplicates as introduced in EVA. Overall, a discussion of why certain event repetitions may or may not be included

Table 8
Semantics comparison between policies in [46] and parameters in EVA. The results of single/every and bounded/every are not specified in [46]. Consumption only applies to unrestricted events. The semantics of override depends on the evaluation time (end of interval vs continuous evaluation, see Section 4.4), here we show the results for the end-of-interval evaluation as this is most in keeping with the intended semantics.

Etzion and Nibblet			EVA	
Cardinality	Repeat type	Consumption	EIS	EIC
single	every	n/a	not specified	
single	override	n/a	last	consume
single	first/last	n/a	first/last	consume
bounded	every	n/a	not specified	
unrestricted	every	consume	all	consume & reapply
unrestricted	every	re-use	all	keep
unrestricted	every	bounded re-use	all	keep + cond.
unrestricted	override	consume	last	consume
unrestricted	override	re-use	last	keep
unrestricted	override	bounded re-use	last	keep + cond.
unrestricted	first/last	consume	first/last	consume & reapply
unrestricted	first/last	re-use	last/last	keep
unrestricted	first/last	bounded re-use	first/last	keep + cond.

Table 9
Semantics comparison between consumption modes in [15] and [71], and parameters in EVA. *Non-consumed last* is described in [70] as the most recent $e_1 \in E_1$ being unconsumed as long as there is no other occurrence of E_1. This is a direct application of our duplicate list concept.

Bailey and Mikulás	Kiringa	EVA		
		EIS E_1	EIS E_2	EIC
most recent	consumed last	last	first	consume
cumulative	cumulative	all	first	consume
FIFO	FIFO	first	first	consume & reapply
LIFO	LIFO	first	last	consume & reapply
	First	first	first	consume
	non-consumed last	last	all	consume

in the results is largely missing from the literature. Kiringa's *non-consumed last* is closest in concept, but no reasoning is given for its definition. Furthermore, Mellin [86] noted that some policy combinations allowed by Zimmer/Unland [122] are not meaningful (multiple pairings of (first E_1); (first E_2)). However, with the use of duplicate lists in EVA these combinations are meaningful because each consecutive duplicate list contributes their own first element. The reason for the difference is that in our approach, event selection parameter EIS refers to each duplicate list, whereas in [122] it referred to all observed events.

7. Summary and future work

In this paper, we proposed a parameterized event algebra, EVA, for event-based systems. The event algebra was introduced to describe the event operators that form composite events. In an additional step, we described 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, as the relational algebra lacks 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 implemented a generic parameterized event system A-mediAS 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.

Our parameterized event algebra offers the possibility of easily integrating various event sources that are structured differently. For this purpose, we identified five language groups and defined transformation rules for both operators and parameters. The rules take advantage of the flexibility of the algebra which can be easily adapted to different languages by changing the parameter setting. These rules were used in a prototype implementation of a *mediator service*. This service provides a unified profile language for subscribing to multiple event-based systems and can easily adapt to fine-grained changes in the semantics of available event sources.

Future work on the event algebra itself is planned in two directions: a full formal proof of the algebraic characteristics sketched in Section 4.6, and the adaptation of

our algebra to handle interval-based semantics (as discussed in Section 6.1). Furthermore, an in-depth exploration of scalability of the mediator service implementation would complement the existing performance evaluation.

Our event algebra can further serve as a foundation for the specification of a higher level, user-friendly language. Such a language could follow, for example, a graphical paradigm as initiated in [67]. It is our belief that many users of event-based systems require simple yet powerful means of expressing subscriptions without having to deal with the intricacies of the underlying system details.

Acknowledgments

We wish to thank the participants of the Dagstuhl Seminar on Complex Event Processing [32], 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] R. Adaikkalavan, S. Chakravarthy, Formalization and detection of events over a sliding window in active databases using interval-based semantics, in: Proceedings, East-European Conference on Advances in Databases and Information Systems, 2004, pp. 241–256.
[2] R. Adaikkalavan, S. Chakravarthy, Formalization and detection of events using interval-based semantics, in: Proceedings of the International Conference on Management of Data, 2005, pp. 58–69.
[3] R. Adaikkalavan, S. Chakravarthy, SnoopIB: interval-based event specification and detection for active databases, Data Knowl. Eng. 59 (October (1)) (2006) 139–165.
[4] R. Adaikkalavan, S. Chakravarthy, Seamless event and data stream processing: reconciling windows and consumption modes, in: Proceedings of the 16th International Conference on Database Systems for Advanced Applications—Volume Part I, DASFAA'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 341–356.
[5] A. Adi, O. Etzion, Amit—the situation manager, VLDB J. 13 (2) (2004) 177–203.
[6] R. Agrawal, T. Imielinski, A.N. Swami, Mining association rules between sets of items in large databases, in: Proceedings of the ACM SIGMOD, 1993, pp. 26–28.
[7] J. Allen, Time and time again: the many ways to represent time, Int. J. Intell. Syst. 6 (1991) 341–355.
[8] J.F. Allen, G. Ferguson, Actions and Events in Interval Temporal Logic, Technical Report TR521, University of Rochester, Computer Science Department, 1994.
[9] M. Altinel, M.J. Franklin, Efficient filtering of XML documents for selective dissemination of information, in: International Conference on Very Large Data Bases, 2000, pp. 53–64.
[10] D. Anicic, P. Fodor, S. Rudolph, R. Stühmer, N. Stojanovic, R. Studer, A rule-based language for complex event processing and reasoning, in: Web Reasoning and Rule Systems, Springer, 2010, pp. 42–57.
[11] D. Anicic, S. Rudolph, P. Fodor, N. Stojanovic, Stream reasoning and complex event processing in ETALIS, Semant. Web 3 (4) (2012) 397–407.
[12] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, K. Ito, R. Motwani, U. Srivastava, J. Widom, Stream: The Stanford Data Stream Management System, Springer, 2004.
[13] A. Arasu, S. Babu, J. Widom, The CQL continuous query language: semantic foundations and query execution, VLDB J. 15 (June (2)) (2006) 121–142.
[14] J. Bacon, J. Bates, R. Hayton, K. Moody, Using events to build distributed applications, in: Proceedings of the Seventh ACM SIGOPS European Workshop, Connemara, Eire, Society Press, 1996, pp. 148–155.
[15] J. Bailey, S. Mikulás, Expressiveness issues and decision problems for active database event queries, in: Proceedings of the Eighth International Conference on Database Theory, ICDT '01, Springer-Verlag, London, UK, UK, 2001, pp. 68–82.
[16] R. Barga, J. Goldstein, M. Ali, M. Hong, Consistent streaming through time: a vision for event stream processing, in: Proceedings of the Conference Innovative Data Systems Research, CIDR'07, 2007.
[17] T. Bass, Mythbusters: event stream processing versus complex event processing, in: Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems, DEBS '07, ACM, New York, NY, USA, 2007, pp. 1–1.
[18] M. Bernauer, G. Kappel, G. Kramler, Composite events for XML, in: Proceedings of the 13th International Conference on World Wide Web, WWW '04, ACM, New York, NY, USA, 2004, pp. 175–183.
[19] E. Bertino, E. Ferrari, G. Guerrini, An approach to model and query event-based temporal data, in: The Fifth Workshop on Temporal Representation and Reasoning, TIME '98, Sanibel Island, IEEE Computer Society, Florida, USA, May 16–17, 1998, pp. 122–131.
[20] F. Bry, M. Eckert, Towards formal foundations of event queries and rules, in: International Workshop on Event-Driven Architecture, Processing and Systems, VLDB, 2007.
[21] A.P. Buchmann, H. Branding, T. Kudrass, J. Zimmermann, Reach: a real-time, active and heterogeneous mediator system, IEEE Data Eng. Bull. 15 (1–4) (1992) 44–47.
[22] J. Carlson, Event pattern detection for embedded systems (Ph.D. thesis), Malardalen University, June 2007.
[23] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, S. Zdonik, Monitoring streams: a new class of data management applications, in: Proceedings of the 28th International Conference on Very Large Data Bases, VLDB '02, VLDB Endowment, 2002, pp. 215–226.
[24] A. Carzaniga, D.S. Rosenblum, A.L. Wolf, Achieving scalability and expressiveness in an internet-scale event notification service, in: Symposium on Principles of Distributed Computing, 2000.
[25] A. Carzaniga, D.S. Rosenblum, A.L. Wolf, Design and evaluation of a wide-area event notification service, ACM Trans. Comput. Syst. 19 (August (3)) (2001) 332–383.
[26] S. Chakravarthy, R. Adaikkalavan, Events and streams: harnessing and unleashing their synergy! in: Proceedings of the Second International Conference on Distributed Event-Based Systems, DEBS '08, ACM, New York, NY, USA, 2008, pp 1–12.
[27] S. Chakravarthy, Q. Jiang, Stream Data Processing: A Quality of Service Perspective, Springer Verlag, Berlin/Heidelberg/New York, 2009.
[28] S. Chakravarthy, V. Krishnaprasad, E. Anwar, S.-K. Kim, Composite events for active databases: semantics, contexts and detection, in: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994, pp. 606–617.
[29] S. Chakravarthy, D. Mishra, Snoop: an expressive event specification language for active databases, Knowl. Data Eng. J. 14 (1994) 1–26.
[30] S. Chakravathy, J.D. Yang, A Recursion-Based Framework for Detecting Composite Events in Active Databases, Technical Report 98-3, Chonbuk National University, 1998.
[31] S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S.R. Madden, F. Reiss, M.A. Shah, TelegraphCQ: continuous dataflow processing, in: Proceedings of the International Conference on Management of Data, SIGMOD '03 ACM, 2003, pp. 668–668.
[32] M. Chandy, O. Etzion, R. von Ammon (Eds.), 07191 Abstracts collection—event processing, in: number 07191 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007.
[33] C.K. Chang, H. Garcia-Molina, A. Paepcke, Predicate rewriting for translating boolean queries in a heterogeneous information system, ACM Trans. Inf. Syst. 17 (January (1)) (1999).
[34] C. Chen, Visualising semantic spaces and author co-citation networks in digital libraries, Inf. Process. Manag. 35 (3) (1999) 401–420.
[35] X. Chen, I. Petrounias, H. Heathfield, Discovering temporal association rules in temporal databases, in: Issues and Applications of Database Technology, 1998, pp. 312–319.
[36] O. Corporation, Oracle® Streams: Concepts and Administration 11 g release 2 (11.2). Technical Report, Oracle Corporation, August 2011. Available at ⟨http://www.docs.oracle.com/cd/E11882_01/server.112/e17069.pdf⟩.
[37] G. Cugola, A. Margara, Complex event processing with T-REX, J. Syst. Softw. 85 (8) (2012) 1709–1728.
[38] G. Cugola, A. Margara, Processing flows of information: from data stream to complex event processing, ACM Comput. Surv. 44 (June (3)) (2012). 15:1–15:62.
[39] M. Datar, A. Gionis, P. Indyk, R. Motwani, Maintaining stream statistics over sliding windows, SIAM J. Comput. 31 (6) (2002) 1794–1813.

[40] U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R. Ledin, D. McCarthy, A. Rosenthal, S. Sarin, M.J. Carey, M. Livny, R. Jauhari, The HiPac project: combining active databases and timing constraints, SIGMOD Rec. 17 (March (1)) (1988) 51–70.
[41] N. Dindar, P.M. Fischer, M. Soner, N. Tatbul, Efficiently correlating complex events over live and archived data streams, in: ACM International Conference on Distributed Event-Based Systems (DEBS'11), New York, NY, USA, July 2011.
[42] K.R. Dittrich, S. Gatziu, Time issues in active database systems, in: Proceedings of the International Workshop on an Infrastructure for Temporal Databases, 1993, pp. 1–6.
[43] D. Mishra, Snoop: an event specification language for active database systems (Master's thesis), University of Florida, 1991. Available at ⟨http://www.cis.ufl.edu/sharma/theses/deepakm/main.ps⟩.
[44] EPTS, Event processing technical society, online at ⟨http://www.ep-ts.com⟩, last accessed August 2013.
[45] J. Erikson, CEDE: composite event detector in an active object-oriented database (Master's thesis), University of Skövde, 1993.
[46] O. Etzion, P. Niblett, Event Processing in Action, 1st ed. Manning Publications Co., Greenwich, CT, USA, 2010.
[47] P. Fraternali, L. Tanca, A structured approach for the definition of the semantics of active databases, ACM Trans. Database Syst. 20 (December (4)) (1995) 414–471.
[48] M.M. Gaber, A. Zaslavsky, S. Krishnaswamy, Mining data streams: a review, ACM Sigmod Record 34 (2) (2005) 18–26.
[49] A. Galton, J.C. Augusto, Two approaches to event definition, in: Proceedings of the International Conference on Database and Expert Systems Applications, Springer-Verlag, 2002, pp. 547–556.
[50] S. Gatziu, K.R. Dittrich, SAMOS: an active object-oriented database system, IEEE Q. Bull. Data Eng. (Special Issue on Active Databases) 15 (December (1–4)) (1992) 23–26.
[51] S. Gatziu, K.R. Dittrich, Events in an active object-oriented database system, in: Proceedings of the First International Workshop on Rules in Database Systems, 1993.
[52] S. Gatziu, K.R. Dittrich, Detecting composite events in active database systems using petri nets, in: RIDE-ADS, 1994, pp. 2–9.
[53] N.H. Gehani, H.V. Jagadish, O. Shmueli, Composite event specification in active databases: model & implementation, in: Proceedings of the VLDB, 1992.
[54] N.H. Gehani, H.V. Jagadish, O. Shmueli, Event specification in an active object-oriented database, in: ACM SIGMOD International Conference on Management of Data, San Diego, CA, 1992, pp. 81–90.
[55] A. Geppert, D. Tombros, Event-based distributed workflow execution with eve, in: Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, Middleware '98, Springer-Verlag, London, UK, 1998, pp. 427–442.
[56] L. Golab, M.T. Özsu, Issues in data stream management, SIGMOD Record 32 (June (2)) (2003) 5–14.
[57] T.J. Green, A. Gupta, G. Miklau, M. Onizuka, D. Suciu, Processing xml streams with deterministic automata and stream indexes, ACM Trans. Database Syst. 29 (December (4)) (2004) 752–788.
[58] R.E. Gruber, B. Krishnamurthy, E. Panagos, Ready: a high performance event notification service, in: International Conference on Data Engineering (ICDE), 2000, pp. 668–669.
[59] 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.
[60] A. Hinze, A-mediAS: concept and design of an adaptive integrating event notification service (Ph.D. thesis), Freie Universitaet Berlin, 2003.
[61] A. Hinze, Efficient filtering of composite events, in: Proceedings of the BNCOD British National Conference on Databases, London, UK, 2003.
[62] A. Hinze, S. Bittner, Efficient distribution-based event filtering, in: R. Wagner (Ed.), in: The 22nd International Conference on Distributed Computing Systems (ICDCS- 2002), Workshops: 1st International Workshop on Distributed Event-Based Systems (DEBS), 2002.
[63] A. Hinze, K. Sachs, A. Buchmann, Event-based applications and enabling technologies, in: International Conference on Distributed Event-Based Systems (DEBS), July 2009.
[64] A. Hinze, A. Voisard, A parameterized algebra for event notification services, in: Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME 2002), Manchester, UK, 2002.
[65] X. Huang, Design and Implementation of a Meta-ENS, Postgraduate Project Report, University of Waikato, Computer Science Department, 2005.
[66] Q. Jiang, S. Chakravarthy, Data stream management system for MavHome, in: Proceedings of the 2004 ACM Symposium on Applied computing, SAC '04, ACM, New York, NY, USA, 2004, pp. 654–655.
[67] D. Jung, Specifying single-user and collaborative profiles for alerting systems (Ph.D. thesis), Computer Science Department, University of Waikato, Hamilton, New Zealand, 2009.
[68] D. Jung, A. Hinze, A meta-service for event notification, in: R. Meersman, Z. Tari (Eds.), Proceedings of the International Conference on Cooperative Information Systems (CoopIS), Lecture Notes in Computer Science, vol. 3290, Springer, 2004, pp. 283–300.
[69] D. Jung, A. Hinze, Event Notification Services: Analysis and Transformation of Profile Definition Languages, Technical Report 12, University of Waikato, 2004.
[70] I. Kiringa, Specifying event logics for active databases, in: Proceedings of the 2002 International Workshop on Description Logics (DL2002), 2002.
[71] I. Kiringa, Logical foundations of active databases (Ph.D. thesis), Department of Computer Science, University of Toronto, 2003.
[72] R. Kowalski, M. Sergot, A logic-based calculus of events, New Gen. Comput. 4 (Januaray (1)) (1986) 67–95.
[73] J. Krämer, B. Seeger, Semantics and implementation of continuous sliding window queries over data streams, ACM Trans. Database Syst. 34 (1) (2009).
[74] B. Krishnamurthy, D.S. Rosenblum, Yeast: a general purpose event-action system, Trans. Softw. Eng. 21 (October (10)) (1995).
[75] G. Li, H.-A. Jacobsen, Composite subscriptions in content-based publish/subscribe systems, in: Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware, Middleware '05, Springer-Verlag, New York, Inc., New York, NY, USA, 2005, pp. 249–269.
[76] L. Liu, C. Pu, W. Tang, Continual queries for internet scale event-driven information delivery, IEEE Trans. Knowl. Data Eng. (Special issue on Web Technologies), January 1999, http://dx.doi.org/10.1109/69.790816.
[77] L. Liu, C. Pu, W. Tang, Supporting internet applicatioons beyond browsing: trigger processing and change notification, in: Proceedings of 5th International Computer Science Conference, ICSC'99 (Internet Applications.), Lecture Notes in Computer Science, vol. 1749, Hongkong, China, 1999.
[78] L. Liu, C. Pu, W. Tang, D. Buttler, J. Biggs, T. Zhou, P. Benninghoff, W. Han, F. Yu, CQ: A personalized update monitoring toolkit, in: Proceedings ACM SIGMOD International Conference on Management of Data, ACM, Seattle, Washington, USA, June 2–4 1997, pp. 547–549.
[79] D.C. Luckham, The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems, Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA, 2001.
[80] D.C. Luckham, J. Vera, An event-based architecture definition language, IEEE Trans. Softw. Eng. 21 (September (9)) (1995) 717–734.
[81] S. Madden, M. Franklin, Fjording the stream: an architecture for queries over streaming sensor data, in: Proceedings of International Conference on Data Engineering, ICDE '02, 2002.
[82] M. Mansouri-Samani, Monitoring of distributed systems (Ph.D. thesis), Department of Computing, Imperial College, University of London, December 1995.
[83] A. Markowetz, Y. Yang, D. Papadias, Keyword search on relational data streams, in: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, SIGMOD '07, ACM, New York, NY, USA, 2007, pp. 605–616.
[84] J. Mccarthy, Actions and other events in situation calculus, in: Principles of Knowledge Representation and Reasoning: Proceedings of the Eighth International Conference, Morgan Kaufmann Publishers, 2002, pp. 615–628.
[85] Y. Mei, S. Madden, ZStream: a cost-based query processor for adaptively detecting composite events, in: Proceedings of the International Conference on Management of data, SIGMOD '09, ACM, New York, NY, USA, 2009, pp. 193–206.
[86] J. Mellin, Resource-predictable and efficient monitoring of events (Ph.D. thesis), Department of Computer Science, University of Skövde, 2004.
[87] K. Moody, J. Bacon, D. Evans, S. Schwiderski-Grosche, Implementing a practical spatio-temporal composite event language. From Active Data Management to Event-based Systems and More, Springer-Verlag, Berlin, Heidelberg, 2010, 108–123.
[88] I. Motakis, C. Zaniolo, Formal semantics for composite temporal events in active database rules, J. Syst. Integr. 7 (3/4) (1997) 291–325.

[89] G. Muehl, L. Fiege, P.R. Pietzuch, Distributed Event-Based Systems, Springer Verlag, Berlin/Heidelberg/New York, 2006.
[90] S. Navathe, R. Ahmed, A temporal relational model and a query language, Inf. Sci. 49 (1989) 147–175.
[91] H. Parzyjegla, D. Graff, A. Schröter, J. Richling, G. Mühl, Design and implementation of the Rebeca publish/subscribe middleware. From Active Data Management to Event-based Systems and More, Springer-Verlag, Berlin, Heidelberg, 2010, 124–140.
[92] A. Paschke, ECA-RuleML: An Approach Combining ECA Rules with Temporal Interval-Based KR Event/Action Logics and Transactional Update Logics, Technical Report cs/0610167, arXiv, 2006.
[93] A. Paschke, A. Kozlenkov, Rule-based event processing and reaction rules, in: Proceedings of the 2009 International Symposium on Rule Interchange and Applications, RuleML '09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 53–66.
[94] N.W. Paton, J. Campin, A.A. Fernandes, M.H. Williams, Formal specification of active database functionality: a survey, in: Rules in Database Systems, 1995, pp. 21–37.
[95] P. Pietzuch, Hermes: a scalable event-based middleware (Ph.D. thesis), University of Cambridge, February 2004.
[96] P.R. Pietzuch, B. Shand, J. Bacon, A framework for event composition in distributed systems, in: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware, Middleware '03, Springer-Verlag New York, Inc. New York, NY, USA, 2003, pp. 62–82.
[97] P.R. Pietzuch, B. Shand, J. Bacon, Composite event detection as a generic middleware extension, IEEE Netw. 18 (1) (2004) 44–55.
[98] A. Prasad Sistla, O. Wolfson, Temporal conditions and integrity constraints in active database systems, in: ACM SIGMOD, International Conference on Management of Data, 1995.
[99] J. Riecke, O. Etzion, F. Bry, M. Eckert, A. Paschke, Event processing languages, Tutorial at the International Conference on Distributed Event-Based Systems (DEBS), 2009.
[100] C. Roncancio, Toward duration-based, constrained and dynamic event types, in: Active, Real-Time, and Temporal Database Systems, Second International Workshop, ARTDB-97, Como, Italy, September 8–9, 1997, Proceedings, Lecture Notes in Computer Science, vol. 1553, Springer, 1997, pp. 176–193.
[101] S. Schwiderski-Grosche, K. Moody, The SpaTeC composite event language for spatio-temporal reasoning in mobile systems, in: International Conference on Distributed Event-Based Systems (DEBS), July 2009.
[102] K.S. Skjelsvik, A.K. Lekova, V. Goebel, E. Munthe-Kaas, T. Plagemann, N. Sanderson, Supporting multiple subscription languages by a single event notification overlay in sparse manets, in: International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE 2006), 2006, pp. 67–74.
[103] S.K. Tanbeer, C.F. Ahmed, B.-S. Jeong, Y.-K. Lee, Sliding window-based frequent pattern mining over data streams, Inf. Sci. 179 (22) (2009) 3843–3865.
[104] A. Tansel, A temporal extension to SQL, in: Proceedings of the International Workshop on an Infrastructure for Temporal Databases, Arlington, TX, 1993.
[105] K. Teymourian, A. Paschke, Semantic rule-based complex event processing, in: Proceedings of Rule Interchange and Applications, Lecture Notes in Computer Science, vol. 5858, Springer, 2009, pp. 82–92.
[106] TIBCO, Tibco Businessevents, online at ⟨http://www.tibco.com/software/complex-event-processing/businessevents/default.jsp⟩, 2013.
[107] D. Toman, Point-based Temporal Extensions of SQL, in: Proceedings of the DOOD, Lecture Notes in Computer Science, vol. 1341, 1997.
[108] S. Urban, I. Biswas, S. Dietrich, Filtering features for a composite event definition language, in: SAINT '06: International Symposium on Applications on Internet, Washington, DC, USA, 2006, pp. 86–89.
[109] F. van Harmelen, F. van Harmelen, V. Lifschitz, B. Porter, Handbook of Knowledge Representation, Elsevier Science, San Diego, USA, 2007.
[110] K. Walzer, M. Groch, T. Breddin, Time to the rescue—supporting temporal reasoning in the Rete algorithm for complex event processing, in: Proceedings of the 19th international conference on Database and Expert Systems Applications, DEXA '08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 635–642.
[111] M. Wei, I. Ari, J. Li, M. Dekhil, Receptor: Sensing Complex Events in Data Streams for Service-Oriented Architectures, Technical Report HPL-2007-176, Hewlet Packard, 2007.
[112] B. Wu, K. Dube, Plan: a framework and specification language with an event-condition-action (ECA) mechanism for clinical test request protocols, in: Proceedings of 34th Annual Hawaii International Conference on System Sciences (HICSS-34), IEEE Press, 2001.
[113] E. Wu, Y. Diao, S. Rizvi, High-performance complex event processing over streams, in: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, SIGMOD '06, ACM, New York, NY, USA, 2006, pp. 407–418.
[114] S. Yang, S. Chakravarthy, Formal semantics of composite events for distributed environments, in: Proceedings of the International Conference on Data Engineering (ICDE 99), 1999.
[115] E. Yoneki, ECCO: data centric asynchronous communication (Ph.D. thesis), University of Cambridge, Cambridge, UK, 2006.
[116] E. Yoneki, Time/space aware event correlation, in: A. Hinze, A. Buchmann (Eds.), Principles and Applications of Distributed Event-Based Systems, IGI Global, 2010, pp. 43–74.
[117] E. Yoneki, J. Bacon, Unified semantics for event correlation over time and space in hybrid network environments, in: International Conference on Cooperative Information Systems (CoopIS), Lecture Notes in Computer Science, vol. 3760, Springer, 2005, pp. 366–384.
[118] R. Zhang, E. Unger, Event Specification and Detection, Technical Report 96-8, Kansas State University, June 1996.
[119] D. Zhu, A.S. Sethi, SEL: a new event pattern specification language for event correlation, in: The 10th International Conference on Computer Communications and Networks, 2001, pp. 586–589.
[120] D. Zimmer, A. Meckenstock, R. Unland, A general model for event specification in active database management systems, in: Proceedings of the fifth DOOD, 1997, pp. 419–420.
[121] D. Zimmer, R. Unland, The Formal Foundation of the Semantics of Complex Events in Active Database Management Systems, Technical Report MCC-INSL-131-98, University of Paderborn and Siemens Nixdorf Informations systeme AG, 1997.
[122] D. Zimmer, R. Unland, On the Semantics of Complex Events in Active Database Management Systems, in: Proceedings of the 15th International Conference on Data Engineering, 1999.