Chapter 5

Adaptable Event Composition

It is not the strongest of the species that survives,
nor the most intelligent;
it is the one that is most adaptable to change
Charles Robert Darwin

In this chapter, we start addressing the first aspect of our central topic of adaptability in event notification services: *qualitative adaptation*. Integration of events provided by different sources and event handling within integrating event-based applications requires an event notification system to support several adaptation tasks. For example, the clients should not be forced to redefine their profiles in order to adapt to a new or changing source. Consequently, the profile evaluation should be adaptable[1] to changing application requirements.

Events entering an ENS are filtered according to client profiles; the event matching is based on the filter semantics defined for the service. Profiles defined by the service's clients are restricted by the filter's capabilities. Therefore, the semantics of the event filtering and the one of the profile definition language are closely related. The theoretic basis for this chapter was introduced in Chapter 3.

Our analysis of event-based approaches in Chapter 4 has shown that only few sophisticated profile languages have been proposed, e.g, in CQ, Siena, and some active database systems. The existing languages are too rigid and do not allow for semantical adaptation as demanded by Requirement R4 (cf. Chapter 2, Page 19). In addition, the evaluation of languages that seem to follow similar semantics does not always lead to similar results. One example is the handling of duplicate events: Depending on the implementation, in the filtering process, duplicates are either skipped or kept.

[1]We use the term *adaptable* to emphasize the fact that the algebra may be used to adapt the ENS filter. The terms *adaptive* and *auto-adaptive* focus on the mechanism of (automatically) adapting a system or component.

In this chapter, we extend the semantics of well-known ECA rules of active database systems by introducing the notion of *relative time*. Because ENS systems are used in the context of a distributed environment and do not rely on a transactional context as in active database systems, the simultaneous occurrences of events cannot be determined accurately. Therefore, all composite event operators should be handled as temporal operators and extended by a relative time frame (similar to time handling in distributed systems).

In addition, in different application areas different semantics for the same operators are introduced. For example, the semantics of the temporal operators introduced by Allen [All91] are not defined uniformly in the numerous application areas. Consequently, our approach supports various semantical perceptions of the same operator. Thus, different systems or applications may use a different semantics for the same operator. This is achieved through the introduction of an event algebra that is controlled by a set of parameters. The parameterized algebra defines the semantics of event operators for composite events; different parameter settings result in different composite event semantics.

Instead of supporting only one semantical variation, we propose a flexible approach that allows for adaptation of profile and filter semantics. Our parameterized event algebra for event notification services permits simple changes of the filter semantics to support changing applications, to adapt to new event sources, and to support integrating applications that combine events from different sources. Because the usability and power of a service heavily depends on the expressiveness of the profile and filter semantics, our algebra fundamentally influences our design of an integrative event notification service.

The algebra is presented in two steps. First, we describe the temporal event operators informally, in a second step the event algebra is defined formally. As summary, we analyze the event semantics of different event-based systems and compare them to the features of our algebra.

5.1 Composite Event Operators

This section describes the operators of our event algebra in an informal manner. The set of operators included in the algebra is derived from the profile language considerations of [HF99a]. In the next section, the operators are introduced formally.

An event algebra is an abstract description of the event concept of a service independent of the actual profile definition language. It enables the evaluation of the complexity of different services, supports the detection of inconsistent profile definition, and can serve as a basis for an implementation of an event filter algorithm. In order to model composite events, we employ event constructors (also called operators). We extend an event algebra for active database systems [GD93] to consider also the temporal demands on event compositions. For illustration, we use an example from facility management that is extended throughout the chapter:

Example 5.1 (Facility Management Profiles)
Let us consider the situation introduced in Scenario 1 (see Page 12) with focus on applications for monitoring of heating and security. In Table 5.1, we rephrase the set of example profiles introduced before. Note that the profiles are simplified versions of real-live profiles. For example, the proximity requirement for the two events (e.g., same room) is omitted.

Profile	*Description*
P1:	*Notify if the temperature rises above $35°C$ within 7 days after a failure in the air conditioning system.*
P2:	*Notify if the air conditioning system fails for the fourth time.*
P3:	*Notify if during the night a window is broken and within $5min$ after this a presence detector sends a signal.*
P4:	*Notify if a sensor did not send data for more than half an hour.*

Table 5.1: Refined profiles for a facility management scenario (Example 5.1)

Recall from Chapter 3 that an event instance relates to the actual occurrence of an event while an event class is a set of events specified by a query. Event composition defines new event instances that inherit the characteristics of all contributing events. The occurrence time of the composite event is defined by the composition operator. The events e_1 and e_2 used in the definitions below can be any primitive or composite event; E_1 and E_2 refer to event classes with $E_1 \neq E_2$. $t(.)$ refers to occurrence times defined based on a reference time system, T denotes time spans in reference time units. We use the contribution operator $\succ$ (cf. Definition 3.7 on Page 32) to identify the events that contribute to a composite event. Note that temporal operators are defined on event instances as well as on event classes, resulting in event instances 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. The occurrence time of the composite $e_3 \in (E_1|E_2)$ 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, regardless of the order. The conjunction constructor has a temporal parameter that describes the maximal length of the interval between e_1 and e_2.[2] The time of the composite event $e_3 \in (E_1, E_2)_T$ 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 temporal distance of the events. The time of the event $e_3 \in (E_1; E_2)_T$ with $\{e_1, e_2\} \succ e_3$ is equal to the time of e_2: $t(e_3) := t(e_2)$.

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

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

If several operators are to be applied on an event class, we have to distinguish whether identical event instances or distinct event instances 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 event instances, i.e., references to the

[2] $(E_1, E_2)_\infty$ refers to an event composition without temporal restrictions. It is equivalent to the original conjunction constructor as defined, e.g., in [GD93].

same event class combined by logical operators reference to identical instances in that class. Logical operators are defined on event classes only. A logical combination of two classes describes the usual logical combination of the defining queries (and their expressions).

Logical Conjunction The *logical conjunction* $E_1 \wedge E_2$ of event classes E_1 and E_2 requires that the expressions of both event class queries 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 expressions of the event class queries 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 involved event instances, i.e., equal class names such as E_1 refer to identical event instances 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 5.2 (Temporal vs. Logical Conjunction)
Let E_1, E_2, and E_3 be event classes. Then, the event instances of the temporal conjunction $E_T = ((E_1; E_2)_{T1}, (E_1; E_3)_{T2})$ are defined as

$$E_T = \{e \,|\, \exists e_{11}, e_{12} \in E_1 \; \exists e_2 \in E_2 \; \exists e_3 \in E_3 : \{e_{11}, e_{12}, e_2, e_3\} \succ e \wedge \\ t(e_{11}) \leq t(e_2) \leq \big(t(e_{11}) + T1\big) \wedge t(e_{12}) \leq t(e_3) \leq \big(t(e_{12}) + T2\big)\}.$$

It is not required but allowed that $e_{11} == e_{12}$. The event instances of the logical conjunction $E_L = ((E_1; E_2)_{T1} \wedge (E_1; E_3)_{T2})$ are defined as

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

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

Example 5.3 (Facility Management Profiles using Temporal Operators)
The profile examples from our logistic application (see Table 5.1, Page 59) can be modelled using the event operators as follows:

P1: Let E_1 be the class of events regarding air conditioning failures. Let E_2 be the class of events regarding a threshold crossing (above $35°C$). The composite events are then defined as $e \in (E_1; E_2)_{7days}$.

P2: We reuse the event class E_1 for air conditioning failures. A simplified definition for the composite events may be expressed as $e \in E_1^{[4]}$.

P3: Let E_{5PM} be the class of time events occurring at 5P.M. that refer to the start of the night-shift. Let E_3 be the class of breaking-window events. Let E_4 be the class of events regarding presence detector signals. The composite events are then $e \in (E_{5PM}; (E_3; E_4)_{5min})_{14hours}$.

P4: Let E_5 be the class of all sensor events of a certain sensor. Then, the composite events are defined as $e_T \in (\overline{E}_5)_{30min}$.

5.2 Parameterized Event Algebra

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

This section sets the basis for our profile definition language for alerting. We believe that for the temporal event operators various (application-dependent) semantics have to be provided. We therefore introduce a *parameterized* event algebra. This section first motivates the parameters for event instance selection and consumption, two concepts that have been proposed in the active database area. It then illustrates examples of various parameter settings. Finally, the formal definition of the event algebra is presented.

5.2.1 Parameters to Consider

The following examples illustrate the limitations of the simple approach in Section 5.1.

Example 5.4 (Sequence)
Let us assume we are interested in the sequence of two events $(E_1; E_2)_T$ as defined, for instance, in profiles P1 and P3. We consider the following history (trace) of events: $tr = \langle e_1, e_2, e_3, e_4 \rangle$, with $e_1, e_2 \in E_1$, $e_3, e_4 \in E_2$ as shown in Figure 5.1. It is not clear from the event definition, which pair[3] of events fulfills our profile. Candidate pairs are, e.g., the inner two events, or the first and the third. It is also not clear whether the profile can be matched twice, e.g., by pairs (e_2, e_3) and (e_1, e_4), or by (e_1, e_3) and (e_2, e_4).

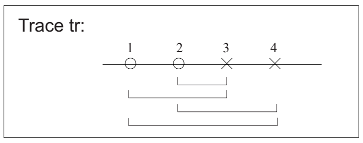


Figure 5.1: Possible composite events in Example 5.5, $\circ \hat{=} e_1 \in E_1$ and $\times \hat{=} e_2 \in E_2$

Various event combinations may be possible. Besides, for different applications, different event-history evaluation strategies could be applied. The problem similarly occurs for unary operators:

Example 5.5 (Selection)
Let us assume, we are interested in every fourth occurrence of an event $e \in E_1$, as defined, for instance, in Profile P2. We could define the profile: $E_1^{[4]}$. Let us consider the (synthetical) trace $tr = \langle e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8 \rangle$ with $e_i \in E_1$, $1 \leq i \leq 8$. The profile could be matched by event e_4, or by e_4 and e_8, or by the events $e_4, e_5, \ldots, e_8$ because all of them are preceded by three instances.

Using the terminology of active databases systems [ZU99], we need to identify the following *modes* for an event algebra for ENS:

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

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

To discuss these modes in detail, we need a formal definition of the term ’duplicate’.

Definition 5.1 (Duplicate)
Event duplicates are all event instances that belong to the same event class.

Duplicate events do not need to have the same occurrence time. Duplicates could be, for example, all temperature events regarding a certain room. Note that duplicates do not necessarily have the same event message type (see Chapter 3 for the distinction between event classes and message types).

Definition 5.2 (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 set of event instances of class E_1 that occur consecutively without intermediate event instances of class E_2.

Note that duplicate lists are subject to changes as long as the *closing event* did not occur. The closing event is the first event instance of class E_2 that follows an $e \in E_1$ with $e \in D_{E_1 \setminus E_2}$.[4] It is important to note that in contrast to [ZU99], the term ’duplicate’ is used here with respect to a profile (i.e., an event class) and not for the event instance itself. Two events can be different, nevertheless, they may match the same profile. With regards to that particular profile they are seen as duplicates.

Different *event selection principles* have been discussed in Chapter 3. *Event instance patterns* have been introduced in Section 5.1. Here, we introduce modes for *event instance selection* and *event instance consumption*. Due to the temporal event operators, event selection and consumption in ENS cannot be handled independently (as proposed for active database systems [ZU99]).

Event Instance Selection (EIS). Duplicate instances of an event class have to be handled differently depending on the application context. For the event instance selection, we distinguish the three variants as depicted in Figure 5.2. First, only the *first* event of a list of duplicate events is considered, i.e., the duplicates are ignored. Second, the duplicates are not ignored, but each new event instance overwrites the older one and the *last* duplicate is considered. In the third variant, the duplicates are neither ignored and nor overwritten: *All* event instances are taken into account. It is also conceivable to select the i^{th} duplicate. Extending the terminology introduced in Section 5.1, we refer to the event instance selection parameter within a composite event using the following operators on a event class E: $first\_dup(E)$, $last\_dup(E)$, $all\_dup(E)$, and $i\_dup(E)$ (see Figure 5.4).

[4]Note that our concept of duplicate lists is an extension of the star operator typically used in active database systems that support composite events [MZ97]. For an ENS, the notion of the start operator as a consecutive events with no interleaving (other) events is too restrictive. The typical restrictions applying in active databases, such as transactional context or specified client, table, or tuple, do not apply here.

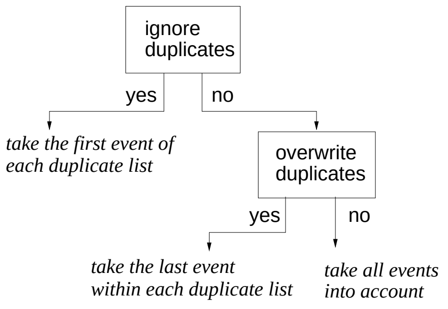

ignore duplicates
yes
no
take the first event of each duplicate list
overwrite duplicates
yes
no
take the last event within each duplicate list
take all events into account

Figure 5.2: Event instance selection: first, last, or all events

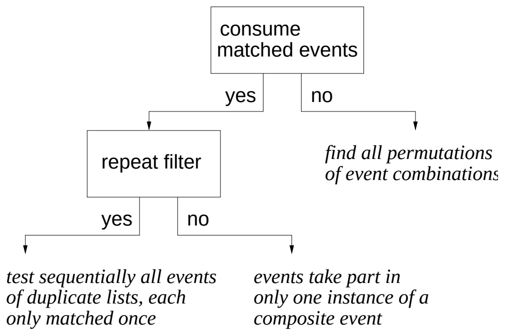

consume matched events
yes
no
repeat filter
find all permutations of event combinations
yes
no
test sequentially all events of duplicate lists, each only matched once
events take part in only one instance of a composite event

Figure 5.3: Event instance consumption: all, unique, or repeated composites

Event Instance Consumption (EIC). We distinguish three variations in the identification of composite events, as shown in Figure 5.3. Matched events can be consumed or they can contribute several times to composite events of the same class. If matched events are consumed, only *unique* composite events are supported. An 'unique composite event' means that an event instance cannot be used twice as a result for a certain profile query, i.e., in a certain event class. But the event instance may be additionally matched by one or more different profiles. If the filter is applied more than once, a primitive event participates in *all* composite events regarding a single profile. If events are consumed by composite events, the filtering process could be *repeated* after unique composite events have been identified. This approach can be seen as a combination of the two parameters event instance selection and consumption. We refer to the event instance consumption parameter for each composite event by an additional index: all_pairs, $unique\_pairs$, and $repeated\_pairs$ (see Figure 5.4).

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). Instead, Figure 5.4 gives the EBNF for the algebra's operators. Our implementation of the algebra as profile definition language in the A-MEDIAS system is introduced in Chapter 9. The following Example 5.6 shows profile definitions for our facility management scenario when using the parameterized operators.

```
      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'
```

Figure 5.4: Constructs of our event algebra: The symbol `priEvent` refers to a primitive event class. The structure of a `Integer` follows the common rules for integers, `TimeSpan` refers to a timespan.

.

Example 5.6 (Facility Management Profiles with Parameterized Operators)
The profiles P1 and P2 from our the example application (see Table 5.1, Page 59) can be modelled using the event algebra as follows

P1: The air conditioning system signals each failure only once, therefore, all events are taken into account. For the temperature events, we assume that a threshold crossing is detected by several sensors. Then, only the first event in each list of duplicates has to be considered. All occurrences of the composite event (all pairs) are considered. $e \in ((all\_dup(E_1); first\_dup(E_2))_{7days})_{all\_pairs}$.

P2: After the first four air conditioning failures the notification is sent: $e \in (E_1^{[4]})_{unique\_pairs}$. *Alternatively, for energy management, all composite events may be of interest:* $e \in (E_1^{[4]})_{repeated\_pairs}$

We discuss the usage of the algebra's parameters by examples for the sequence operator with different combinations of the two parameters. Figure 5.5 shows a matrix of selected profile-event situations. For clarity reasons we only show situations with identical parameter values for both contributing events. Note that the names for the rows and columns are simplified descriptions of the different approaches. 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 $\circ \hat{=} e_1 \in E_1$ and $\times \hat{=} e_2 \in E_2$, each composite event instance is marked with an arc. The position of arcs above or below the trace follows only readability reasons. The dashed arcs denote special cases that we discuss on Page 70. Here, we use fixed time frames as evaluation intervals that are denoted with brackets [.], such that the different implications are easier to compare.

The horizontal dimension of the matrix shows examples for the event instance selection. The selection either takes only the first or the last duplicate in a trace into account, or all events are considered.

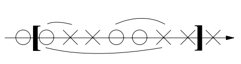
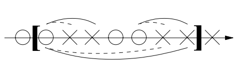
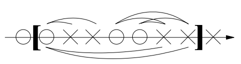
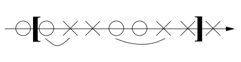
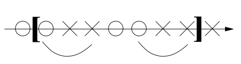
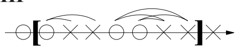
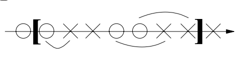
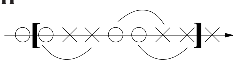
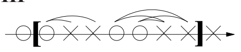
From the matrix, we can already derive example applications for the different approaches:

Column 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.

Column 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 current value.

Column III Taking all events is only useful in an application in which no duplicates should be omitted, e.g., in 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., temperature raised by $5°C$), or in a document-oriented application that delivers new documents.

The examples above clearly illustrate that the appropriate semantics and the various possible profiles are heavily application dependent. The vertical dimension of the matrix shows different versions of profile filter evaluation: apply the filter to all events (i.e., keep matched events), apply to only the unmatched events (i.e., consume matched events), and repeat filtering after a first profile match (i.e., consume matched events and repeat filter). The last approach can lead to a successive matching of possibly all

event instance consumption \ event instance selection	take first event of each duplicate list	take last event of each duplicate list	take all events
keep matched events	**A I** 	**A II** 	**A III** 
consume matched events	**B I** 	**B II** 	**B III** 
consume matched events repeat filter	**C I** 	**C II** 	**C III** 

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

Figure 5.5: Profile evaluation under different conditions, $\circ \hat{=} e_1 \in E_1$ and $\times \hat{=} e_2 \in E_2$

events in a list of duplicates (see first/last event of duplicate list). Here, sequences of matching pairs can be found.

Row A Considering all possible event combinations in a given series (also keeping matched ones) results in sets of composite events that have single events in common. This applies for instance in scenarios where each event itself represents a set of events. Examples include trucks delivering goods to customers, where a set of goods is loaded in the morning but the unloading is realized by several events (cf. Scenario 2 on Page 15). In this case, the starting event is a combination of several primitive events *load product on truck*, which can be seen as factorized.

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

Row C The repetition of the filtering process after discarding matched events is used, e.g., for parsers and compilers. A sensible application is an event-based XML-validator, as proposed in [AF00]. With this method, interleaving event pairs can be identified.

For unary operators, we briefly discuss the profile-event situations for a selection as shown in Figure 5.6. For unary operators, only the event consumption parameter applies. 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 $\circ \hat{=} e \in E_1$, each composite event instance is marked with an arrow, the arcs denote the event instances contributing to the composition. While the sequential variations support the selection of the i^{th} event within each duplicate list, the *selection* operates on the full matching list.

In event notification systems, the Siena implements an evaluation style as in situation C I, Rebecca implements B I and B II, OpenCQ implements C I. In active database systems, SAMOS implements C III and U B depending on the event operators, Sentinel uses B III, Ode distinguishes evaluation styles similar to C I, C II, and A III depending on parameters. Even though the systems seem to implement the same operators, their evaluation leads to different results due to different (implicit) operator semantics.

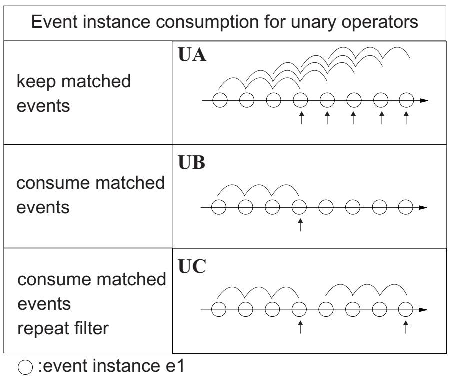

Event instance consumption for unary operators
keep matched events
UA
consume matched events
UB
consume matched events repeat filter
UC
:event instance e1

Figure 5.6: Unary event operators, profile evaluation under different evaluation conditions

5.2.2 Definition of the Event Algebra

An algebra is a set together with a collection of operations on this set; our event algebra defines a set of operators (as described informally in Section 5.1) on the event space $\mathbb{E}$. Our algebra consists of three binary operators (disjunction, conjunction, and sequence) and two unary operators (selection and negation) enhanced by temporal restrictions. In this section, we define our parameterized event algebra by describing the effect of profiles containing the event operators on a given trace. Note that the parameters and the operator's temporal restrictions are not independent.[5]

We introduce a new terminology[6] to refer to each event instance in a trace. First, we concentrate on the binary operators and their semantical variations. Then, unary operators are considered briefly. We now define the projection of a trace on events of a certain event class as a *trace view*.[7]

Definition 5.3 (Trace View)
Let E_1 be an event class. The subset $tr(E_1)$ of a given trace tr is defined as ordered list of events that contains all events $e \in E_1$. We call this subset a trace view.

The trace view $tr(E_1, E_2)$ contains all $e_1 \in E_1, e_2 \in E_2$ with $\{e_1\} \in tr$ and $\{e_2\} \in tr$. Note that the events in a trace view keep all their attributes including occurrence time, but obtain a new index-number. Trace views define a partition of a trace into duplicate lists. We define a re-numbering on the list tr:

Definition 5.4 (Trace Renumbering)
The list tr is subdivided into disjunct sublists $tr[1], \ldots, tr[n]$ each containing successive events that are elements of the same event class. Every element of such a sublist is denoted with $tr[x, y]$, where $x \in \mathbb{N}$ is the number of the sublist and $y \in \left[1, length(tr[x])\right]$ is the index-number of the element within the sublist.

[5]For illustration, consider a sequence of two events $((first\_dup(E_1); last\_dup(E_2))_T)_{all\_pairs}$: The event instance of $last\_dup(E_2)$, which is to be taken into account is identified based on the time span T. That means that the last event instance *within the time span T* is considered. Thus, the parameter EIS is not independent of the time-restriction of the operator.

[6]Notations for composite events have already been introduced. Most of the notations are not sufficient – they are only as expressive as our informal event algebra, e.g., [Car98]. The others are already too restrictive [GD93, GJS92a], they define separate event operators for (some) semantical variations. These languages operate on the implementation level while our algebra describes the conceptional level..

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

The length of the sublists is defined as the number of list elements. Note that the disjunct sublists $tr[i]$ are duplicate lists as introduced in Definition 5.2.

Example 5.7 (Trace Renumbering)

Let us consider the following trace of events: $tr = \langle e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\rangle$ *with* $e_1, e_3, e_4, e_8, e_9 \in E_1$*,* $e_6, e_7, e_{10}, e_{11}, e_{12} \in E_2$*, and* $e_2, e_5 \in E_3$ *as shown top left in Figure 5.7. The* (E_1, E_2)*-trace view is then defined as* $tr(E_1, E_2) = \langle e_1, e_3, e_4, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\rangle$*.*

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

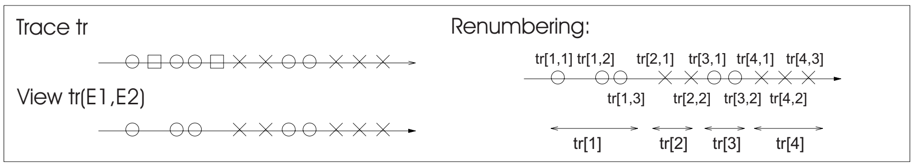

Trace tr
View tr(E1,E2)
Renumbering:
tr[1,1] tr[1,2] tr[2,1] tr[3,1] tr[4,1] tr[4,3]
tr[1,3] tr[2,2] tr[3,2] tr[4,2]
tr[1] tr[2] tr[3] tr[4]

Figure 5.7: Trace and renumbering in Example 5.7, $\circ\hat{=}e_1 \in E_1$, $\times\hat{=}e_2 \in E_2$, $\square\hat{=}e_3 \in E_3$

Note that we denote (unordered) sets of events with $\mathbb{E}$ or E while $tr[.]$ denotes lists (ordered sets with possible duplicates) of events.

As discussed before, for each of the basic operators (e.g., sequence), several semantical variations exist. Defining each of the variations separately would require a number of definitions for each basic operator[8]. Instead, we use a set of parameters to control the variations: The values of the parameters v_{min}, v_{max}, w_{min}, w_{max}, and P_{EIC} influence the operator semantics. The values of v_{min}, v_{max}, w_{min}, w_{max} $\in \mathbb{N}$ control the event instance selection parameter; they refer to the lower and upper index-number of the selected events within each duplicate list. The definition of the set P_{EIC} controls the event instance consumption parameter; elements of this set determine the number of the duplicate lists to form the composition pairs. We discuss the different parameter values subsequently 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 $(E_1|E_2)$.

Binary Operators. The disjunction implements a selection based on occurrence time (or), no exclusion (xor), i.e., the matching set of the disjunction includes all event instances that match either profile. We use the matching operator $\sqsubset$ (introduced in Definition 3.13 on Page 33) to refer to the set of events that match a certain composite profile.

Definition 5.5 (Disjunction of Events)

Let us consider two Profiles p_1 *and* p_2*, then holds*

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

[8]As pointed out, 48 variations for the sequence operator can be constructed from the described variations for EIS and EIC

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

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

Different values for the open parameters v_{min}, and v_{max} are discussed subsequently (see Page 69 and Tables 5.2 and 5.3).

We define the semantics of a conjunction of events as follows:

Definition 5.6 (Conjunction of Events)
Let us consider two Profiles p_1 and p_2, $e \in \mathbb{E}$ and a given time span T. Then holds

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

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

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

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

Definition 5.7 (Sequence of Events)
Let us consider two Profiles p_1 and p_2, $e \in \mathbb{E}$ and a given time span T. Then holds

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

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

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

As stated for Definition 5.5 and 5.6, different values for the open parameters P_{EIC}, v_{min}, v_{max}, w_{min}, and w_{max} are discussed in the next paragraph.

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

Table 5.2: Event instance selection: parameters for first, i^{th}, and last event within each duplicate list (columns I–III in Figure 5.5) where $m \in \mathbb{N}$ with $\forall j > m : t(tr_{post}[.,j]) > t(tr_{post}[.,.]) + T$, where the dots are placeholders for the respective values, T as defined for the operator.

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

Table 5.3: Event instance consumption: parameter for unique and all pairs (rows A and B in Figure 5.5)

Semantical Variations of Binary Operators. We now evaluate different approaches for the parameter values, which implement different semantics of the operators. 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 instance selection, we distinguish several variations to select events from duplicate lists (as defined in Table 5.2). Each operand has to be evaluated differently, depending on the position of the operand relative to the binary operator. The selection of the i^{th} event is a (somewhat artificial) generalization of the preceding modes.

For the composition modes for pair matching (event instance consumption), we distinguish two variations as shown in Table 5.3: the selection of unique pairs (each event in the matching set participates in one pair only, as in Row B in Figure 5.5) and the selection of all pairs (as in Row A in Figure 5.5). The third approach as depicted in Row C in Figure 5.5, is a combination of the two dimensions that are already introduced. Here, we discuss this approach briefly: Only unique pairs are considered ($x = y$), matching pairs are removed and the filtering is repeated until all matches are found. The first matches are, e.g., first/last events of duplicate lists, the second match are second/next-to-last events, and so forth. Other combinations are plausible. The parameter option of all duplicates for both contributing events has the same result as without repeated filtering; the combination of different parameter values opens new result variations.

Unary operators. The definition of unary operators (selection and negation) is shown briefly.

Definition 5.8 (Selection of Events)
Consider $e \in \mathbb{E}$ and $i \in \mathbb{N}$, then

$$p^{[i]} \sqsubset e \Leftrightarrow \exists e_i \in \mathbb{E}\, \forall j \in \mathbb{N} \text{ with } 1 \leq j \leq i \left(p \sqsubset e_j \ \wedge \ t(e_j) \leq t(e) \ \wedge \ \{e_i\} \succ e \right).$$

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

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

Definition 5.9 (Negation of Events)
Consider a time event $e_t \in \mathbb{E}_t$ and a given time span t_1, then

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

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

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

For selection and negation holds: Unique event instances are detected with $v_{min} = v_{max} = 1$, all event instances are detected with $v_{min} = 1$, $v_{max} = length(tr[x])$. Similarly to binary operators, repeated filtering after event match is more complex.

Evaluation Time. The issue of order and time in a distributed environment is crucial and has to be considered for an implementation of composite operators. We described sets of matching events without especially 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. We assumed that event matches including last duplicates are evaluated after the evaluation interval is finished (final evaluation). But the evaluation could already start during the evaluation interval (continuous evaluation). In this case, the continuous evaluation of all incoming events offers the advantage of early notification. Here, a fast but possibly incorrect information is delivered in opposition to the correct but later information after the ending of the time frame. An example is shown in situation A II in Figure 5.5 (dashed lines). This approach is appropriate in several applications, e.g., catastrophe warning systems for environmental surroundings or other systems for urgent information delivery. We call this additional parameter the *event evaluation time*: final vs. continuous. Final evaluation waits for the end of the composite profile's maximal temporal distance T. Continuous evaluation is carried out continuously until T is reached.

Adapting Event Composition. As argued in the motivation, a flexible implementation of the different operator styles allows for simple adaptation of a profile's semantics to changing requirements. For example, our facility management scenario describes a *multi-purpose* application that processes information coming from differently structured sources. Therefore, different parameter settings may have to be applied for the same initial profile: Our parameterized algebra allows for such as simple adaptation of client profiles to changing event sources and applications (see Example 5.8). The implementation of this adaptivity is discussed in Chapter 8.

Example 5.8 (Adaptable Profile)
Let us consider profiles P1 from our logistic application (see Table 5.1, page 59). We assume that the air conditioning system signals a failure only once, therefore, all events are taken into account. For the temperature events, the handling of events may depend on the current usage of the building. One option is that each temperature event is detected by several sensors because large rooms contain several sensors. Then, the sensor readings are duplicated events for that particular room (consider $first\_dup(E_2)$ in Profile P1). Another option is that each (smaller) room contains only one sensor; the sensor readings are not duplicates (consider $all\_dup(E_2)$ in Profile P1). For the different options, different profile parameters have to be applied. The options may change dynamically based on the usage of the building, but clients do not want to redefine their profiles.

5.3 Related Work

In this section, we briefly review related approaches for event algebras. Event specification semantics have been developed in various research areas, such as active databases, temporal or deductive databases, temporal data mining, time series analysis, and distributed systems.

In the area of active database systems, the problem of event rule specification has been addressed for several years, e.g., [CM94, HW93, Wid96], also with special focus on composite events [GD94, GJS92a, GJS92b, JS92, YC99, ZU99] and temporal conditions, e.g., [DBB88, DG93, GD94]. 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 state rather than describing the event itself. For the ordering of database states, the temporal interval operators as defined by Allen [All91] can be used [MZ97]. Ordering based on events (as opposed to states) has been implemented in SAMOS [GD94]. SAMOS's rule language supports a few parameters. However, these parameters mainly concern database-typical concepts such as sessions and transactions. The times-parameter is equivalent to our selection operator. The work of Zhang and Unger [ZU96] is closely related, but parameters and time frames are missing. Active database systems do not support a flexible adaptation of event evaluation as required in the context of open and distributed ENS. Most aDBS are centralized systems that deal only with database internal events. Furthermore, the approaches concentrate on the implementation level, whereas our work focuses on the application level and provides a higher abstraction level.

In the context of active databases, temporal logic has been used to describe the semantics of triggers, e.g., in [MZ97, PW95]. Using temporal logic [AF94] is an alternative approach to describe composition operators. The most promising approach is the Enhanced Past Temporal Logic (PTL) introduced by Sistla and Wolfson [PW95], because it supports relative temporal conditions and composite actions. However, the desired flexibility would be lost [9] and not all operators and parameters can be expressed [10].

The problem of temporal combination of events is also addressed in temporal and deductive databases. In these areas, various approaches have been introduced, such as temporal extensions of SQL [Tan93, Tom97] and a temporal relational model [NA89]. In contrast to ENS, however, temporal and deductive databases focus on ad-hoc querying. That is, there is no periodical evaluation of a query as needed in

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

[10] For example, the PTL expression `B last A` describes the same composition as our $(last(A); first(B))_{unique\_pairs}$. But for the very similar $(last(A); last(B))_{unique\_pairs}$, no representation exists in PTL.

Systems	Primitive Events	Composite Events	Operators						Supported Parameter Settings	Explicite Parameters
			time	conjunction	disjunction	sequence	selection	negation		
SIFT	X	-	-	-	-	-	-	-	-	-
Elvin	X	-	-	-	-	-	-	-	-	-
Siena	X	X	-	O	O	X	O	O	CI	No
OpenCQ	X	X	X	X	X	X	-	X	CI	No
CORBA NS	X	-	-	-	-	-	-	-	-	-
OmniNotify	X	X	-	X	X	X	-	X	BI-III	No
COBEA	X	X	X	X	X	X	-	X	AIII	No
Keryx	X	-	-	-	-	-	-	-	-	-
LeSubscribe	X	-	-	-	-	-	-	-	-	-
Gryphon	X	-	-	-	-	-	-	-	-	-
REBECA	X	X		X	X	X	-	X	BI-II 1)	(Yes) 1)
NiagaraCQ	X	-	-	-	-	-	-	-	-	-
SAMOS	X	X	X	X	X	X	X	X	CIII, UB	Yes
Sentinel	X	X	X	X	X	X	-	-	CI-II, UB	Yes

1) via event-evaluation-time parameter

Figure 5.8: Overview of event operators supported in the profile languages of different systems. The supported parameters are not implemented as such but are illustrative translations of the operators' definitions using our terminology. Further details of our analysis of these languages may may be found in [Jun03].

the context of ENS. The areas of temporal data mining and time series analysis rely on temporal association rules [AIS93, AR00, CPH98]. From a set of data, rules verified by the data themselves have to be discovered. While similar event operations are evaluated, the approaches differ greatly from event filter semantics discussed here. In event notification services, composite event descriptions are given and the matching set of data is to be found, while in temporal analysis the data is given and the rules have to be derived.

Our event algebra may be implemented as profile definition language (as done in our A-MEDIAS system). Several event notification services have been implemented, but none of these supports a flexible filter evaluation similar to our approach. Figure 5.8 shows the results of our evaluation of profile definition languages in various systems using the parameter set introduced here. We distinguish whether the operators are supported ($\times$), modelled but not implemented ($\circ$) or not supported ($-$). The column referring to *supported parameter settings* describes the semantical forms supported in the systems expressed in our notation. The overview shows that none of these systems support all parameter variations and that the semantics for the same parameters varies. *Explicit Parameters* refer to the fact that the client is aware of the parameter setting and may, eventually, change it.

Only a few systems support advanced composite events. Additionally, in most systems a single mode is implemented for event instance selection and consumption, respectively. Often, these can only be derived from each system's behavior: No operators or parameters refer to semantical variations.

5.4 Summary

In this chapter, we proposed a parameterized event algebra for event notification services. The event algebra was introduced in order to describe the event operators that form composite events. In an additional step, we introduced parameters for event instance selection and event instance consumption.

Event instance selection describes which qualifying events from the trace are taken into account for composite events, and how duplicate events are handled. Event instance consumption defines whether unique composite events or all combinations of events are taken into account. The combination of both parameters also offers the definition of filter pattern similar to the ones applied in parsing. The algebra and parameters are defined based on binary operators, by the nesting of composite events. The introduced semantics can easily be extended to sets of events. Note that the formalism used in this chapter is similar to the one of the relational algebra. The relational algebra lacks the concept of ordering, therefore, we introduced an ordering relation on event traces.

The chapter was completed by a detailed analysis of the profile languages of related event-based approaches. The results of the analysis reinforce the uniqueness of our adaptable approach. In contrast to other systems, the approach presented here supports event-based applications that cover changing or new event sources without forcing the clients to redefine their profiles for each new source. It is also suitable for integrating applications that combine events from sources with different event semantics.

The use of our event algebra for the adaptation of client profiles to changing sources and applications is discussed in Chapter 8. The implementation of our parameterized algebra as a profile definition language in our A-MEDIAS system is introduced in Chapter 9.