Efficient Filtering of Composite Events

Annika Hinze

Institute of Computer Science
Freie Universität Berlin,Germany
`hinze@inf.fu-berlin.de`

Abstract. Event Notification Services (ENS) are used in various applications such as remote monitoring and control, stock tickers, traffic control, or facility management. The performance issues of the filtering of primitive events has been widely studied. However, for a growing number of applications, the rapid notification about the occurrence of composite events is an important issue. Currently, the detection of composite events requires a second filtering step after the identification of the primitive components. In this paper, we propose a single-step method for the filtering of composite events. The method has been implemented and tested within our ENS prototype CompAS. Using our method, the filter response time for composite events is significantly reduced. Additionally, the overall performance of the event filtering has been improved.

1 Introduction

Event Notification Services (*ENS*) are used in various applications such as traffic control and remote monitoring project control. They have gained increasing attention in the past few years. Several systems have been implemented, such as Siena [4], Le Subscribe [17], or Ready [11]. An event notification service informs its users about new events that occurred on providers' sites. Events could be, for example, the change of a web-page, a new temperature value, or the occurrence of a traffic jam. Users define their interest in events by means of *profiles*. A user profile defines a periodically-evaluated query (similar to a search query). Users may be interested in primitive (atomic) events, their time and order of occurrence, and in composite events, which are formed by temporal combinations of events. An example for a composite event is the crossing of a certain temperature threshold after an experiment has been started in a laboratory.

After having studied efficient filtering of primitive events [12], this paper concentrates on the filtering of composite events. Composite events are formed by temporally combined primitive events. The filtering of composite events is based on the detection and filtering of their primitive components. The response time for a composite event is the time between the occurrence of the last event contributing to the composition and the user notification. To the best of our knowledge, all existing methods for composite event detection consist of two steps: In a first step, the primitive profiles are evaluated and in a second step, the composite profiles are identified. Thus, the composite event is detected in a separate step after the filtering of primitive events. In this paper, we propose a new method for filtering of composite events that integrates the detection of composite events into the detection of primitive events: After the filtering of a primitive event, its contribution to a composite event is tested. In that way, the composite event is detected successively. No additional step is required for the identification of the composite event after the last contributing primitive event has been detected. The identification of the composite event after the primitive event is accelerated and the overall filtering time is reduced.

This paper is organized as follows: Section 2 gives an overview of necessary background information. Then, we analyze typical methods for the filtering of composite events in Section 3. In Section 4, we introduce our approach of combined filtering of primitive and composite events. We present and discuss the results of a performance analysis of our prototype implementation in Section 5.

2 Background

This section is devoted to our context of study. It first describes our running example from the field of computer-aided facility management (CAFM). Then, the basic event-related concepts are introduced.

2.1 Application Scenario: Remote Monitoring

Remote monitoring and control applications in commercial buildings are an important domain for the employment of event notifications. Examples are monitoring of single factories, powerplants, and facilities. Let us consider a surveillance system for several buildings that monitors lighting, heating, air conditioning, and sun protection (e.g., the OWL system [3]). Such a service aims at the improvement of security, energy saving, and flexibility, and at the reduction of service costs.

A number of sensors are located within each of the monitored buildings. The surveillance system monitors the status of these sensor readings. In case of emergency, the systems reacts, e.g., by sending notification to a technician or by triggering an automatic emergency program. The system controls the building by monitoring the sensors: Some sensors send status information on a regular basis to the system. Other sensors send only critical events, i.e., if the status values cross a predefined threshold. A third group of sensors passively collects data and is to be observed by the system.

Often, the building's technicians and service personnel are interested in certain combinations of events. The following examples demonstrate user profiles in a facility management system:

P1 The air-condition technician is to be notified if the temperature in the offices crosses a given threshold during the night.
P2 The service personnel is to be notified if a sensor does not send data for more than half an hour.
P3 The chemical technician is to be notified if in a laboratory, concurrently, the temperature is too low and the humidity too high.
P4 The security personnel is to be notified if at an office first a window is broken and then the presence detector sends a signal.

These profile examples describe profiles with human-oriented time requirements, but other examples with stronger demands towards the response time are conceivable, e.g., for safety in a chemical laboratory. Typically, surveillance or facility management systems may have of the order of 10^4 profiles and an event frequency of 10^3 events per second for each controlled building.

2.2 Concepts: Events, Profiles, and Notifications

A monitored system consists of a number of objects of interest, e.g. real world objects such as offices or laboratories, or logical objects such as time. Objects have certain states, defined by their properties, e.g., the temperature of a room.

Definition 1 (Event). *An event (or event instance) is the occurrence of a state transition of an object of interest at a certain point in time.*

In contrast to states, events have no duration. Events may be state changes in databases, signals in message systems, or real-world events such as the departures and arrivals of vehicles. Within an event notification system, events are reported as event messages describing the event. Within this paper, we do not explicitly distinguish events from their reporting messages.

We consider *primitive events* and *composite events*, which are formed by combining primitive events. We further distinguish two kinds of primitive events: *time events* and *content events*. Time events describe the occurrence of a certain point in time (e.g., 5 o'clock). Content events involve changes of object states in general (e.g., the temperature of a room). In this paper, we use the (attribute,value) pair notation for primitive event examples, for instance: $e_{laboratory} = event(room\_number = 273, temperature = 35°C, timestamp = \text{14:22:03.456})$.

Definition 2 (Profile). *A profile is a query that is periodically evaluated by the event notification system against incoming events.*

A simple example profile is $p_{temp} = profile(temperature\ < 10\ °C)$. The stream of incoming events at an event notification service is called history, or *trace* of events.

We distinguish *event instances* from *event classes*: an event class is specified by a query while an event instance relates to the actual occurrence of an event. We will simply use the term *events* whenever the distinction is clear from the context.

Definition 3 (Event Class). *An event class is a set of event instances that share certain properties, e.g., attribute values. These properties are described by a query. Then an event class is the set of all event instances for which the query evaluates to true.*

Note that user profiles define event classes, e.g., all event instances regarding the temperature. Even though instances of the same event class share some properties (e.g., temperature of a sensor), they may differ in other event attributes (e.g., location). Events (instances) are denoted by lower Latin e with indices, i.e., $e_1, e_2, \ldots,$ while event classes are denoted by upper Latin E with indices, i.e., $E_1, E_2, \ldots$. The fact that an event e_i is an instance of an event class E_j is denoted *membership*, i.e., $e_i \in E_j$.

Definition 4 (Composite Event). *Composite event instances are formed by temporal combinations of event instances.*

Note that our notion of composite events differs from Gehani [8], where a composite event is a set of the events it is formed of.

The definition of composite events requires temporal operators and additional parameters for the handling of duplicate events, for details see [13]. Composite event operators are, e.g., sequence $(E_1; E_2)_t$, conjunction $((E_1, E_2)_t$, disjunction $(E_1|E_2)$, and negation $(\overline{E_1})_t$. For example, a *sequence* $(E_1; E_2)_t$ occurs when first $e_1 \in E_1$ and then $e_2 \in E_2$ occurs. The parameter t defines in the profile the maximal temporal distance between the events. The time of the sequence event instance $e_3 := (e_1; e_2)$ is equal to the time of e_2: $t(e_3) := t(e_2)$. Additionally to the description of the component events and the temporal restriction, profiles referring to composite events can contain predicates regarding the composite event attributes. For example, the predicate $E_1.temperature == E_2.temperature$ could be defined for two temperature events. This additional conditions are called binding predicates.

Definition 5 (Duplicate). *Duplicates of events are event instances that belong to the same event class.*

Duplicates could be, e.g., all temperature events regarding laboratory 273, but also all temperature events in the building referring to the same temperature value.

For profiles regarding composite events, the handling of duplicate events has to be defined. For example, the air-condition technician is to be notified only once per night about the high temperature, at the first occurrence. On the other hand, the service personnel is to be informed about every occurrence of a failing sensor.

Two parameters carry the information about the user's preferences for the duplicate handling:

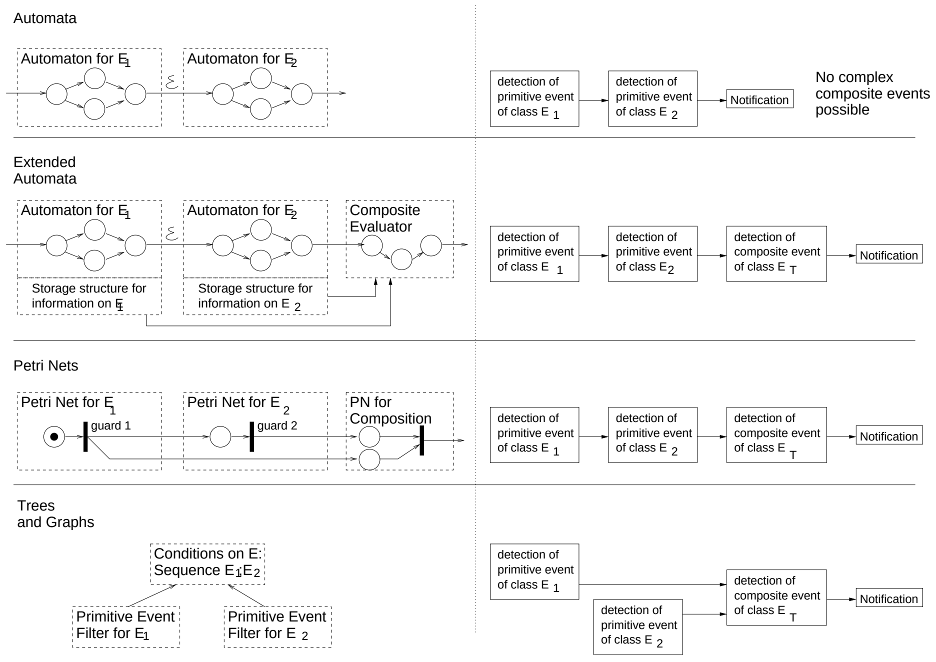
1. *Event instance selection* defines for each event, whether the user is interested in all, the first, last, or n^{th} event instance in a sequence of duplicate events.
2. *Event instance consumption* defines for each composite event, whether the event instances participate in only one pair of event instances or in several pairs.

For the event instance consumption, consider the profile defined by the chemical technician: within 30 minutes the temperature reaches a peak for three times while the humidity crosses the threshold but once. Only one notification needs to be sent. Similarly, the security personnel is not interested in all people entering a room after a broken window was signaled, but only the first one.

3 Methods for Composite Event Filtering

Several proposals have been made for the filtering of composite events. We distinguish approaches using automata (e.g., Ode [7]), petri nets (e.g., Samos [6], Eve [9]), or trees (e.g., Ready [11], GEM [16]). Here, we do not consider the specific conditions of distributed event filtering.

Example 1. Let us consider a composite event profile for the sequence of two events within a time span T: $E_T = (E_1; E_2)_T$. The steps for the detection of the composite event depend on the method used. Figure 1 displays simplified structures for detecting the composite event instance of E_T using the different methods described in this section.



Automata
Automaton for E_1
Automaton for E_2
detection of primitive event of class E_1
detection of primitive event of class E_2
Notification
No complex composite events possible
Extended Automata
Composite Evaluator
Storage structure for information on E_1
Storage structure for information on E_2
detection of composite event of class E_T
Petri Nets
Petri Net for E_1
Petri Net for E_2
guard 1
guard 2
PN for Composition
Trees and Graphs
Conditions on E: Sequence E_1E_2
Primitive Event Filter for E_1
Primitive Event Filter for E_2

Fig. 1. Methods for detection of composite events

Automata Composite event expressions are similar to regular expressions if they are not parameterized. Using this observation, it is possible to implement event expressions using finite automata. The first approach for using automata has been made in the Compose system of Ode [8], an active database system. The history of events provides the sequence of input symbols to the automaton implementing the event expression. The automaton input are the primitive event components from the corresponding composite event as they occur in the history. If the automaton enters an accepting state after the input of a primitive event, then the composite event implemented by the automaton is said to occur at the time of this primitive event.

Automata are not sufficient if binding predicates have to be supported. The automata have to be extended with a data structure for storing the additional event information of the primitive events from the time of their occurrence to the time at which the composite event is detected. This extension is shown in the second row in Figure 1.

Petri Nets Petri nets are used in several event-based systems to support the detection of complex events that are composed of parameterized primitive events, for example in the active database system SAMOS [6], and the monitoring system HiFi [2]. In a Petri Net created for a profile regarding a composite event, the input places refer to primitive events, and the output places model the composite event. Each new profile describing a composite event causes the creation of the appropriate Petri Net. The incremental detection of composite events is described by the position of the markings in the petri net. The firing of the transition depends on the input tokens and the positive evaluation of the transition guards. The occurrence of the composite event is signaled as soon as the last element of a given sequence order is marked. Petri Net based composite event detection is shown in the third row in Figure 1.

Matching Trees Another approach to implement composite event filters are matching trees that are constructed from profiles describing composite event structures. This method has been used in Ready [11] and Yeast [15]. The primitive event parts are the leaves of the matching tree, composites are the parent nodes in the tree hierarchy, as shown in the fourth row in Figure 1. Parent nodes are responsible for maintaining binding information, which represents information for matched events, such as mapping of event variables and successfully matching event instances. Binding information is updated by child nodes on every match and is passed to the parent nodes. Parent nodes perform further filtering. A composite event is detected if the root node is reached and the respective event data are successfully filtered in the root node. Then context-related information and binding information (e.g. the subscriber) is passed to the notification component. Note that in tree-based composite filtering, components of a composite event may be filtered unnecessarily, e.g., the second event of a sequence is filtered even though the first event of the sequence did not occur.

Graphs An approach very similar to the tree-based one described above is the graph-based detection of composite events. Here, each composite event is represented by a directed acyclic graph (DAG), where nodes are event descriptions and edges represent event composition, see also the fourth row in Figure 1. Nodes are marked with references to respective event occurrences. After event detection, parent nodes are informed and checked for consumption recursively. References to events are stored until consumption is possible. In addition to event composition edges, nodes are accompanied with rule objects that are fired after the corresponding event occurred. Graph-based composite detection is implemented in Sentinel [5] and Eve [9].

The processing time of a composite event is the time between the occurrence of the last necessary primitive event to fulfill the composition until the notification is sent. Thus, the response time for composite events is the time to identify the (last) primitive event and the time to identify the appropriate composite events. We extend our Example 1 as follows:

Example 2. For the profile describing the set of composite events $E_T = (E_1; E_2)_T$ as before, we now consider an exemplary trace: $tr = \langle e_1, e_2, e_3, e_4, e_5, e_6, e_7 \rangle$ with $\{e_3, e_4, e_6\} \in E_1$ and $\{e_1, e_2, e_5, e_7\} \in E_2$. We are interested in all matching composite events. Figure 2 shows the event filtering using the different methods described above. We do not show the filtering of each incoming event but only depict the filter efforts contributing to the composite event. The events e_3 and e_4 do not form a composition with e_5, because the temporal distance is larger than T. Only the event instances e_6 and e_7 match our composite profile.

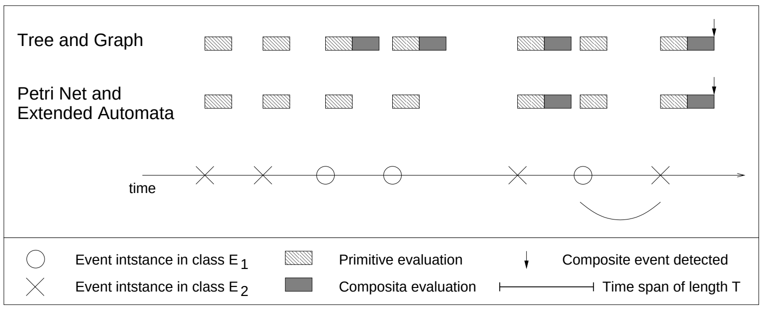


Fig. 2. Composite event detection for exemplary trace as described in Example 2

Composite event filtering based on trees or graphs considers each matching primitive event, regardless of their order or temporal distance. This results in a high number of unnecessary filter operations.

Composite event filtering based on Petri Nets or extended automata considers only matching primitive events that are in the desired order. Matching of temporal conditions is performed during the composition phase. Unnecessary filter operations are performed for non-matching composites.

All approaches for filtering of composite events applicable to parameterized composite events (i.e., all but simple automata) have in common, that two steps are necessary to identity composite events: The detection of primitive events followed by the evaluation of the composite binding predicates.

In the next section, we propose our method to identify the composite event within the single step of detecting the contributing primitive events.

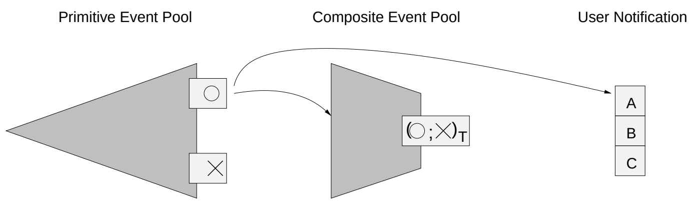
4 Composite Event Detection in a Single Step

We use the idea of partial evaluation [14]: Primitive profiles contributing to composite ones are evaluated only if they potentially contribute to a composite event. For example, for the sequence of events $(E_x; E_y)$, first only the E_x-profile is evaluated. The E_y-profile is included into the evaluation process only after an event e_x did occur.

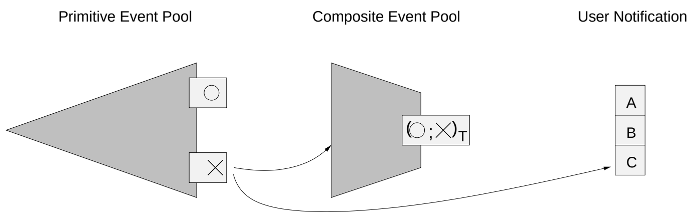
We further illustrate the idea of composite filtering in a single step by explaining the differences to the two-step filtering. For doing that, we consider the following three example profiles:

- User A: $E_A = E_1$ (profile regarding primitive events)
- User B: $E_B = (E_1; E_2)_T$ (profile regarding composite events as in P4 in Section 2.1 and Example 2)
- User C: $E_C = E_2$ (profile regarding primitive events)

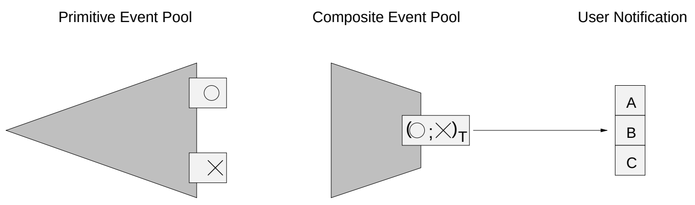
Figure 3 shows the principle of *composite event detection in two steps* for these three profiles: The triangle represents the primitive event pool. The pool contains all profiles



(a) Situation after occurrence of event $\circ \in E_1$



(b) Situation after occurrence of event $\times \in E_2$



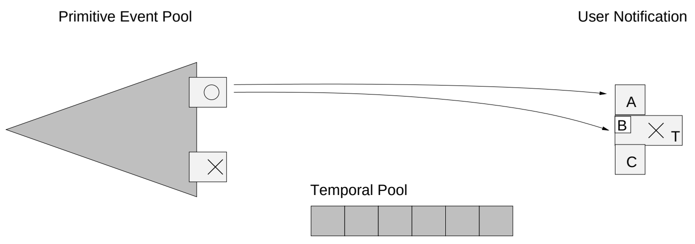
(c) Composite event filtering

Fig. 3. Composite event detection using two-step methods

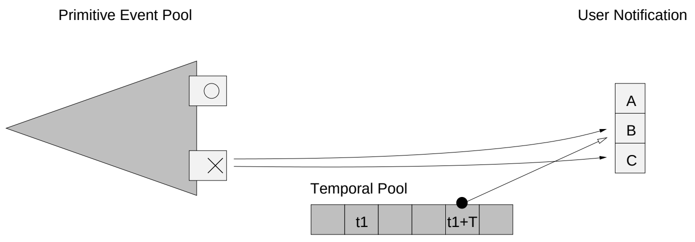
regarding primitive events. Each incoming primitive event has to be filtered against all profiles in that pool, e.g., using the efficient tree-based algorithm [1,10,12]. Users with profiles regarding primitive events (i.e., users A and C) are notified after the detection of these events. This detection of the primitive events is the first step in the event detection mechanism.

The results of the primitive filtering serve as input for the composite filtering (events $\mathrm{e}_1 = \circ$ and $e_2 = \times$ in the Figures 3(a) and 3(b)). The profiles regarding composites are stored in the composite pool, represented by the square in Figure 3. The incoming primitive events are assigned to the composite profiles. If all contributing events for a certain composite did occur, the composite event is signaled to the interested users (user B in Figure 3(c)). If the time-span between the primitive events is larger than T, the composite profile is not matched, and the detected primitive events are dismissed.

Figure 4 shows the principle of *composite event detection in a single step*: The primitive event pool and a temporal pool are required, the composite pool is not used. After the detection of a primitive event, the interested users are notified (user A in Figure 4(a)).



(a) Situation after occurrence of event $\circ \in E_1$ at time $t1$



(b) Situation after occurrence of event $\times \in E_2$

Fig. 4. Composite event detection using our single-step method

For storing the information about composite profiles, auxiliary profiles are created. Instead of notifications to interested users, an internal notification regarding the auxiliary profile is created. This notification triggers the insertion of the remaining composite part into the primitive profile pool.[1]

Consider the auxiliary profile for the composite profile of user B: an internal auxiliary user is notified about the occurrence of the partial composite event (see Figure 4(a)). The auxiliary profiles for the internal user carry the information about the composite profile as well as the information about its partial evaluation.

In Figure 4(b), after the detection of event $\circ$, the profile for event $\times$ has been inserted into the pool. For the observation of the maximal time span T, a reference is inserted into the temporal pool - an auxiliary terminator. Terminator references cause the removal of the referenced profile from the pool. The temporal terminator in Figure 4(b) causes the removal of the profile for user B at time $t1 + T$, where $t1$ is the time of the primitive event $\circ$.

After the match of the final primitive event within the composite profile, the notification to the user is sent. Using the single step method, the notification about the composite does not suffer additional delays due to additional filtering of binding information for the composition.

In analogy to the example for the two-step approach (see Example 2), we discuss the composite evaluation for an exemplary trace in the following example.

[1] The remaining parts of profiles can be identified, for example, by analyzing the respective Petri Net for the composite profile.

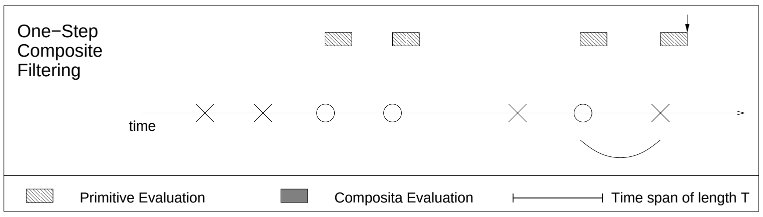


Fig. 5. Composite event detection for exemplary trace as described in Example 3

Example 3. We consider the profile describing the set of composite events $E_T = (E_1; E_2)_T$ and the exemplary trace $tr = \langle e_1, e_2, e_3, e_4, e_5, e_6, e_7 \rangle$ with $e_1, e_2, e_5, e_7 \in E_1$ and $e_3, e_4, e_6 \in E_2$. Figure 5 shows the single-step filtering for that trace. Again, we only depict the filter efforts contributing to the composite event.

Only those primitive events are evaluated that contribute to the composite event. Additionally, the composite event is detected earlier. Few unnecessary filter operations are performed for non-matching composites.

As we can clearly see in Figure 5, the filtering costs of the one-step method are less than the costs of the two-step methods (cf. Figure 2).

Temporal Operators for Composite Events In our example, we have shown the single-step filtering for a simple temporal sequence. The principle can be translated for the other temporal composition operators conjunction, disjunction, selection, and negation, see Table 1. Here, for each temporal operator, we give the filter procedure that implements the respective single-step method. The filter procedure is defined both graphically and by description. In the graphics, we show the order of insertions into the profile and temporal pool. For example, the temporal disjunction $E_1 \| E_2$ requires the insertion of both the profiles regarding E_1 and E_2; after the detection of either event instances in E_1 or E_2, a notification is sent. For the negation, terminator references are required: At starting time of the profile, a temporal profile is inserted that triggers a notification after the time span t_1. If an event instance of E_1 occurs within that time span, the temporal profile is removed and no notification is sent. Different from the sequence handling, here the auxiliary terminator is attached to the primitive profile for the event in question. The temporal restriction provides the accepting reference to the user to be notified.

Parameterized Composite Events Profiles regarding composite events can also carry additional parameters, as briefly introduced in Section 2. These parameters can also be supported in the single-step approach, see Table 2.

The parameter for event instance detection influences which of the duplicated partial events is stored. For example, for the *first* event in the duplicate list, the auxiliary profile refers to the remainder of the composite profile. For the *last* event, two auxiliary profiles have to be created: One that refers to the remainder of the composite profile (in case there is only a single event instance), and another one that refers, again, to the already matched partial profile (to detect the duplicates).

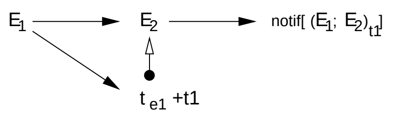
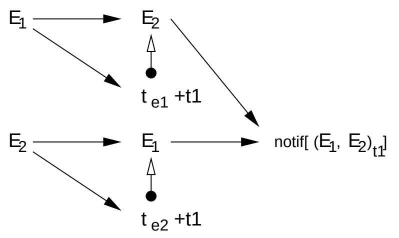
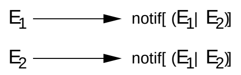
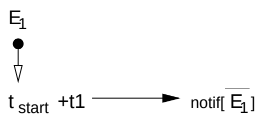
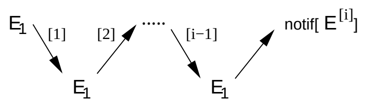
Operator	Filter Procedure
Temporal Sequence e.g. $(E_1;E_2)_{t1}$	 insert profile E_1, after $e_1 \in E_1$ insert profile E_2 and time profile $t_{e_1} + t1$ with terminator to profile E_2
Temporal Conjunction e.g. $(E_1,E_2)_{t1}$	 E1 E2 t e1 +t1 E2 E1 notif[(E1, E2)t1] t e2 +t1 insert profiles E_1 and E_2, after $e_1 \in E_1$ insert profile E_2 and time profile $t_{e_1} + t1$ with terminator to profile E_2, react on $e_2 \in E_2$ respectively
Temporal Disjunction e.g. $(E_1\|E_2)$	 insert profiles E_1 and E_2, notify after $e_1 \in E_1$ or $e_2 \in E_2$
Negation e.g. $(\overline{E_1})$	 insert profile E_1 with an terminator to the time profile $t_{start} + t1$, notify after matching time profile, each A starts the process anew
Selection e.g. $(E_1^{[i]})$	 E1 [1] [2] ..... [i−1] notif[E [i]] E1 E1 insert profile E_1, after $e_1 \in E_1$ insert profile E_1, notify after the $i^{th} e_1 \in E_1$
Legend: A ⟶ B: after event A insert profile B A ●⟶▷ B: after event A remove profile B	

Table 1. Profile handling for various composite operators using our single-step algorithm

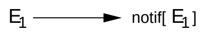
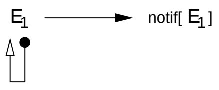
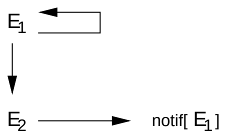
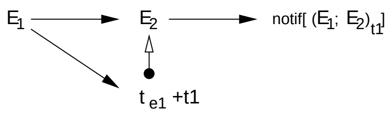
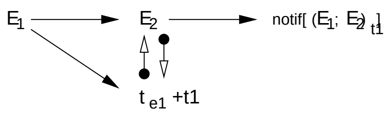
Parameter	Filter Procedure
All Duplicates e.g. regarding E_1	 do not remove profiles after match
First Duplicate e.g. first $e_1 \in E_1$	 remove profile after match
Last Duplicate e.g. first $e_1 \in E_1$ before $e_2 \in E_2$	 keep updating event information until last event instance
All Pairs e.g. of $(E_1; E_2)_{t1}$	 do not remove profiles after match
Unique Pairs e.g. of $(E_1; E_2)_{t1}$	 remove composite event information after after match
Legend: A ⟶ B: after event A insert profile B A ●—▷ B: after event A remove profile B	

Table 2. Profile handling for various composite parameters using our single-step algorithm

The parameter for event consumption influences how long partially matching events have to be stored. We distinguish three modi: remove after match, hold after match, and reapply filter after remove. For example, for remove after match mode, the partially matching events are deleted after the composite has been found (i.e., only *unique event pairs* are detected). For the hold after match mode, the auxiliary profiles for the remainder of the composite profile remain in the system (i.e., *all event instance combinations* are detected).

5 Performance Evaluation

In this section, the response times of the different filter methods for composite events are evaluated. We discuss selected results of the tests performed on our prototypical implementation. Additionally, we introduce implementation variants for the single-step method and discuss their influence on the response time.

5.1 Performance Tests and Discussion

The test implementation of the one-step method is based on our event notification system CompAS that has been developed within the project MediAS [18] at the Freie Universität Berlin. The filter component of our system is based on a modified Gough-algorithm for the filtering of primitive events [12]. For the filtering of composite events, we implemented the tree-based algorithm as used, e.g., in the ENS Ready [11].

Performance tests of composite events underly several parameters, e.g., the composite operator, the operator parameters, the time between the parts of the composite event (composition distance), and the number of events and profiles.

For the performance, we measure the overall response time per event or per composite event. The response time of composite events is the time between the occurrence of the last contributing event and the time the user notification is sent.

We discuss selected test results obtained with our prototypical implementation. Several combinations of parameters have been tested. For brevity, we restrict the presentation to profiles regarding simple event sequences of two events $(A; Z)$ without temporal restrictions. The events are posted to the filtering mechanism as continuous stream, such that the additional delay due to event frequencies is not considered here.

We used events and profiles with one integer attribute and the attribute domain $[-100.000; 100.000]$. To eliminate the influence of profile overlapping, only distinct profiles have been defined.

For both, the two-step and the one-step method, we tested the following parameter settings:

1. The first event instance within a group of duplicate events is selected, subsequent instances of the same event class are ignored. If no duplicate event instances occur, as in most of our tests, the results for selecting the first event instance equal those for all instances and for the last event.
2. For the event composition, two opposite versions have been tested: single unique pairs and all pairs. For single unique pairs, the first occurring pair triggers a notification, afterwards the profile is removed. For all pairs, every possible combination of event instances triggers a notification.

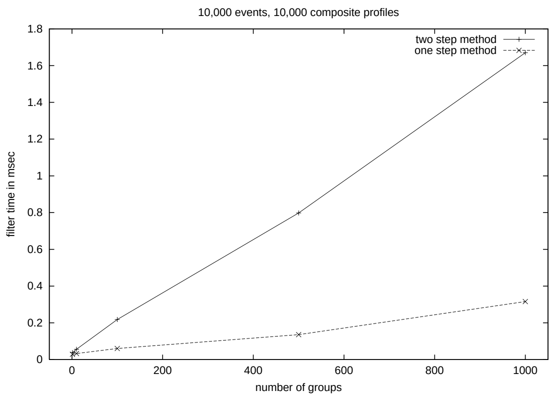


Fig. 6. Influence of composition order (event grouping

Response time depending on composition order. First, we evaluate the influence of the composition order on response time. We show the test results for 10,000 profiles regarding composite events and 10,000 starting events (distinct A-events). For our tests, we formed g groups of e_x events with an e_y event at the end of each of the groups. Within each group, first come $10,000/g$ e_x events and then one e_y event that closes the group. Figure 6 shows the influence of this event grouping on the response time.

We see that the overall response time for each event directly depends on the number of groups, i.e., on the number of interleaving e_y-events. In the two step method, each E_y profile carries a list of matching E_x-profiles. The complete list has to be checked after each e_y-event. In the one-step method the E_y is only inserted after the occurrences of e_x events. Additionally, the E_y carries only references to those E_x-profiles that match events that already occurred.

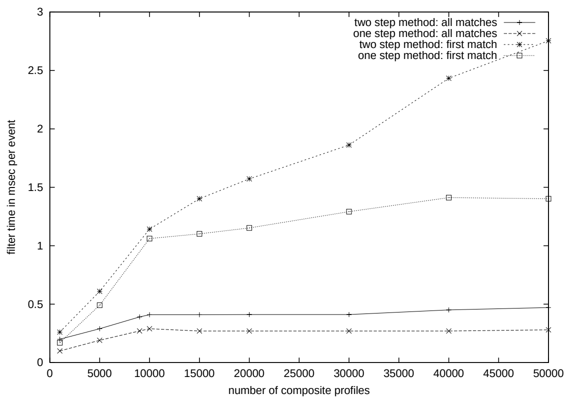
Response time depending on number of profiles. We evaluated the influence of the number of profiles on the response time. We show the results for sequences in a single group of events and for sequences in 1,000 groups of events.

Figure 7(a) and 7(b) show the response time per event (measured in milliseconds) using the one-step and two-step methods for two different parameter settings.

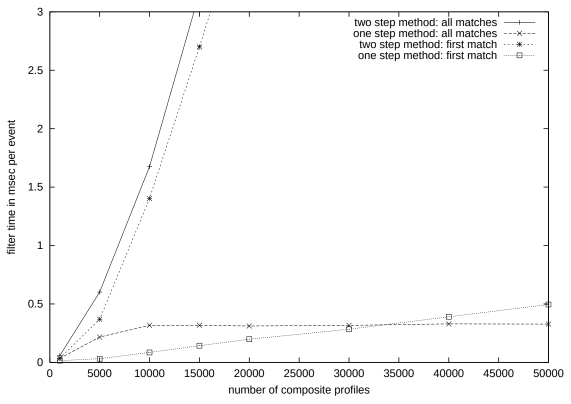
For the case of events in a single group (Figure 7(a)), both methods using all events are faster than the methods using only unique matches. For the unique matches, the profiles are removed after the matches, causing maintenance costs.

The one-step method with profile removal is faster than the two-step method with profile removal. In the first method, the profiles regarding the e_x-events are removed directly after the e_x-events have been detected. In the two-step method, all profiles are only removed at the end of the event detection, i.e., after the e_y-event. Therefore, the performance benefit due to a smaller profile pool applies to the one-step method, but does not affect the two-step method.

The one-step method without removal is faster than the two-step method without removal. For the first method, less profiles are stored in the profile pool, i.e., less profiles



(a) Events structured in 1 event-group



(b) Events structured in 1,000 event-groups

Fig. 7. Response time depending on number of profiles

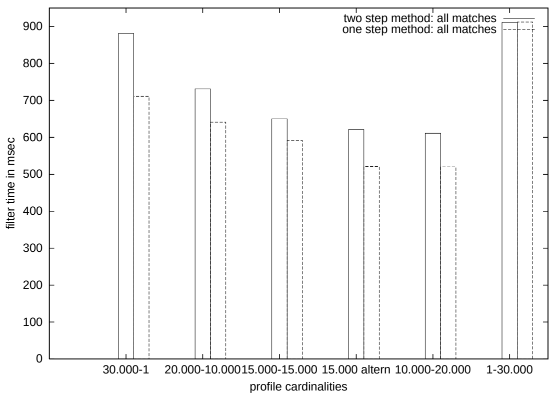
have to be evaluated. Additionally, the number composite profiles to be checked in each step (in the composite pool or as auxiliary profiles) is lower for one-step.

In all four graphs, the filter time increases linearly depending on the number of composite profiles between 0 and 10,000 profiles. In that interval, each of the profiles is matched by exactly one event pair. For more than 10,000 profiles, an increasing number of profiles remains unmatched. For the methods without profile removal, a flattened logarithmical increase can be measured – the influence of the binary search on the profile values. For the methods with removal, the removal costs have a major influence on the response time.

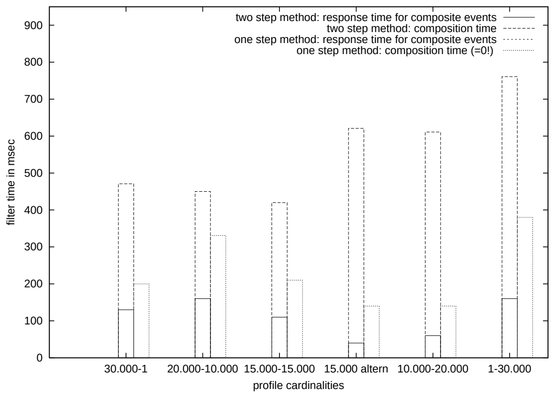
In Figure 7(b), the effects already observed in Figure 7(a) are overlayed with the influence of the event grouping as shown in Figure 6. The influence of interleaving events (grouping) is strongest in the two-step method.

Response time depending on profile cardinality In Figure 8, the response times for different profile cardinalities are shown. The profile cardinality for a set of sequence-profiles $(E_x; E_y)$ describes the number of distinct profiles for E_x and E_y. This cardinality is shown at the x-axis of the diagrams in Figures 8(a) and 8(b).

Figure 8(a) shows the overall response time for 30.000 events, for the one-step and two-step methods. The filter time of the one-step method is in most cases less than the filter tim for the two-step method. Only for the extreme case of one event starting off 30.000 sequences, the response time is equal.



(a) Overall response time



(b) Composition time

Fig. 8. Response time depending on profile cardinality

Figure 8(b) shows the composition times for the same experiments. Two values are shown for each test: the overall time needed for composing events and the additional time required after the last contributing event. As argued earlier, the latter defines, together with the primitive filter time for the last event, the response time of composite events. As composition time for the one-step approach, we measured the time for the handling of auxiliary profiles. The composition time for the one-step method is shorter than that for the two-step method. The additional response time for the composite events due to a composition step is zero.

We argued, that for the efficiency analysis of composite event filtering only the response time of the last contributing event has to be considered. We have shown in our analysis that the one-step method efficiently minimizes this response time for composite events to zero. Additionally, the overhead of unnecessary profile filtering is reduced - the overall performance increases.

5.2 Implementation Variants

For the implementation of the single-step method, several variations are conceivable, which may lead to different performance results depending on the profile and event distributions. Here, we briefly introduce the variations and discuss their consequences for the filter performance.

1. Dynamically handle primitive profiles: This approach corresponds directly to the algorithm introduced in the previous section. For the composite profiles, the partial and auxiliary profiles are dynamically inserted into and deleted from the primitive-profile pool. If the same profile has already been defined by a different user, then the information is already in the pool and only references to the users have to be added.
   If the profiles do not overlap to a large extent, this approach requires a filter algorithm that supports frequent changes to the profile pool for adding and deleting profiles.
2. Mark/unmark primitive profiles: Another approach is the insertion of all partial and auxiliary profiles at once, marking them for use and unmark if the profiles are not used. The links referring to the profile owners have to be removed or unmarked as well.
   Response time is expected to be slightly higher than in the first approach, since the markings have to be checked in each filtering step. If the profiles do not overlap, this approach requires minor changes to the profile pool, but the pool is larger. More disc space may be used.
3. Store primitive profiles, only reference handling: The third approach requires the fewest changes to the profile pool: All partial and auxiliary profiles are inserted at once. Active profiles have references to the respective users, inactive profiles have no users assigned. Inserting and deleting of profiles results in inserting or deleting (or marking and unmarking) of references.
   The disk usage is similar to the one in the second approach. If the profiles do not overlap, the response time is expected to be slower than in the other approaches: All profiles have to be checked, also the ones without a reference to a user.

For all three approaches, the references could be inserted/removed or only marked/unmarked. The second and the third approach are especially feasible for profile sets with a large overlap. Then, almost all profiles do reference to some users; inserting or deleting profiles results in changes to references.

6 Summary

In this paper, we presented an one-step algorithm for the filtering of composite events. Current implementations use two-step algorithms that perform unnecessary filter operations. Our algorithm takes advantage of the idea of partial evaluation: only those profiles are evaluated that may directly contribute to a composite profile. Only few unnecessary filter operations are performed.

After the general introduction of our one-step algorithm, we presented methods for the handling of profiles that use different composition operators. Additionally, we discussed profile handling under various composition parameters.

We implemented both a two-step method and our one-step algorithm within our ENS prototype CompAS. In extensive performance tests, we have shown that our one-step method significantly reduces the response times for composite events with the effect that a composite event is detected as fast as a primitive event. Additionally, the overall filter performance has been improved.

Additionally, CompAS is extended into a distributed event notification system for facility management systems to improve scalability. The system is designed for easy adaptation to changing application requirements and differently structured event sources.

Acknowledgements We wish to thank the other members of the database group at the FU Berlin for their helpful comments on our approach. We are especially grateful to Steven König and Sven Bittner for the implementation.

References

1. M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra. Matching events in a content-based subscription system. In *Proceddings of SIGMOD Principles of Distributed Computing*, 1999.
2. E. S. Al-Shaer, H. M. Abdel-Wahab, and K. Maly. Hifi: A new monitoring architecture for distributed systems management. In *International Conference on Distributed Computing Systems*, 1999.
3. Bernd Bruegge, Ralf Pfleghar, and Thomas Reicher. Owl: An object-oriented framework for intelligent home and office applications. In *Proceedings of the Second International Workshop on Cooperative Buildings (CoBuild99)*, 1999.
4. A. Carzaniga, D. S. Rosenblum, and A. L Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.
5. S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. Technical Report UF-CIS-TR-93-007, University of Florida, Gainesville, Department of Computer and Information Sciences, March 1993.
6. S. Gatziu and K. R. Dittrich. SAMOS: An Active Object-Oriented Database System. *IEEE Quarterly Bulletin on Data Engineering, Special Issue on Active Databases*, 15(1-4):23–26, December 1992.

7. N. Gehani and H. Jagadish. Ode as an active database: Constraints and triggers. In *Proceedings of the Seventeenth International Conference on Very Large Databases (VLDB)*, 1991.
8. N. H. Gehani, H. V. Jagadish, and O. Shmueli. Composite event specification in active databases: Model & implementation. In *Proceedings of the 18th International Conference on Very Large Data Bases*, 1992.
9. A. Geppert and D. Tombros. Event-based distributed workflow execution with EVE. Technical Report ifi-96.05, University Zurich, Computer Science Department, 20, 1996.
10. K. J. Gough and G. Smith. Efficient Recognition of Events in a Distributed System. In *Proceedings of the Australasian Computer Science Conference ACSC-18*, 1995.
11. R. E. Gruber, B. Krishnamurthy, and E. Panagos. The achitecture of the READY event notification service. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems Middleware Workshop*, 1999.
12. A. Hinze and S. Bittner. Efficient distribution-based event filtering. In *22nd International Conference on Distributed Computing Systems (ICDCS- 2002), Workshops: 1st International Workshop on Distributed Event-Based Systems(DEBS)*, IEEE Computer Socienty, 2002.
13. A. Hinze and A. Voisard. A parameterized algebra for event notification services. In *Proceedings of the 9th International Symposium on Temporal Representation and Reasoning (TIME 2002)*, 2002.
14. N. Jones, C. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall, 1993.
15. B. Krishnamurthy and D. S. Rosenblum. Yeast: A general purpose event-action system. *Transactions on Software Engineering*, 21(10), October 1995.
16. M. Mansouri-Samani and M. Sloman. GEM: A generalised event monitoring language for distributed systems. *IEE/IOP/BSC Distributed Engineering Journal*, 4(2), Feb 1997.
17. J. Pereira, F. Fabret, H. Jacobsen, F. Llirbat, R. Preotiuc-Prieto, K. Ross, and D. Shasha. LeSubscribe: Publish and subscribe on the web at extreme speed. In *Proceedings of the ACM SIGMOD Conference*, 2001.
18. Project MediAS: Efficient Internet-wide Notification on Composite Events. Project Homepage: http://page.inf.fu-berlin.de/∼hinze/projects/project_medias.html.