2

Event models

An event is something that takes place, a happening or occurrence, and that is particularly significant, interesting or unusual. In computing systems the notion of event has a major importance since it provides a powerful abstraction making possible to model the dynamic aspects of applications. Events can represent state changes in databases, signals in message systems, changes of existing objects or the creation of new objects in object-oriented systems, or real-world events such as the departure or arrival of vehicles.

This chapter presents the concept of event. Its objective is to give the definitions and references that will enable the understanding of this document and to explore the way in which an event can be modeled and used. The chapter is organized as follows: section 2.1 presents the concept of event type and occurrence. It also presents the concepts related to these concepts: environment of production and event occurrence time. The concepts of primitive and composite event are described in sections 2.2 and 2.3 respectively. Section 2.4 presents the event composition based on event algebra and event mining approaches. It also discusses the aspects that influence the event composition within distributed systems. Finally, section 2.5 concludes the chapter.

2.1 Event types and occurrences

From the point of view of computing, an *event* (i) represents significant facts produced instantaneously in a system and (ii) it is an instantaneous effect of the termination of an invocation of an operation on an object [RW97]. The first definition supposes that events exist because there is interest in observing them. The second supposes that events represent operations and, in consequence, their existence depends on that of a source.

In our work we define an event *e* in terms of a source named *producer* in which the event occurs, and of a *consumer* for which the event is significant. Events describe facts, situations, observable in producers and significant for consumers.

Events are propagated from the producers to the consumers through an *event-based communication model*. The event-based communication asynchronously interconnects producers and consumers in a potentially distributed and heterogeneous environment. It has recently become widely used for large-scale Internet services and mobile programming environments. The event-based communication model allows event producers to anonymously communicate events to a group of event consumers, ideally without knowledge of the number and location of the event consumers.

Consumers may be interested in a subset of the potentially large number of events propagated in a system. To receive events, event consumers have to subscribe to the events they are interested in. Events are compared against the filters and are only delivered to consumers that are interested in them.

Therefore, events have to be specified beforehand in order to be able to detect them at runtime. The *semantics of events* determines what exactly events are, i.e. when and where they occur, whereas the *syntax of events* determines how events are specified. The syntax of events defines an *event specification language*, and the specification of an event determines its structure.

The structure of the propagated events varies considerably depending on the model and the application domain. Events can be seen like signals (read/write of a value) without any general characterization, but also like situations described on an abstract way. Thus, events are said to be either *generic* or *typed* [Mei00]. The information that describes a generic event is a data blob without an expressive structure. Typed events provide a well-defined and expressive data structure into which a wide variety of *event types* can be mapped.

2.1.1 Event type

An *event type* is an expression that characterizes a class of significant facts (events) and the conditions in which these events occur. Through its type, the event identifies the nature of a change occurred within a system. Changes of same nature are thus indicated by events with a similar structure.

According to the complexity of the event model, the event types are represented as sequences of strings [YBMM94], regular expressions – patterns – [BGS+94] or as expressions of an event algebra. In certain models, the type itself contains implicitly the contents of the message. Other models associate types of parameters through structures from which the attributes are accessible by the consumers.

2.1.2 Event occurrence

In general, an event is an *event type occurrence*. Events are produced within time and they are thus associated to a time point $t(e)$ named *event occurrence time*. Hence, each event is associated to a *timestamp* which indicates, at least, the event occurrence time.

The granularity of time representation (day, hour, minute, second, etc.) is determined by the system or application. Most event models and systems suppose that time line representation corresponds to the Gregorian calendar time, and that it is possible to transform this representation as an element of the discrete time domain having 0 (zero) as origin and ∞ as maximum limit. Then, a time point (event occurrence time) belongs to this domain and it is represented by a positive integer.

Besides the event occurrence time, event types can have other *event parameters* describing the circumstances in which the event occurred. In active systems, the parameters of an event are used to evaluate the condition and to execute the action of an ECA rule. Hence, the event type establishes the information that an event transmits (e.g., the event occurrence time). This information constitutes the *event production environment*.

In some event models [BGS+94] the type name is the only information and there is not other associated information. In other models an event type characterizes a production environment, represented by a set of tuples of the form (*variable*, *domain*). In both cases, an event type characterizes knowledge about the context in which it is produced. Then, an event type determines not only one, but numerous occurrences of events which are distinguished by their parameter values.

For many applications, supporting only atomic events is not adequate. In many real-life applications there is a need for specifying and detecting more complex patterns of events. Therefore, event types can be classified as *primitive events* that describe elementary facts, and *composite events* that describe combinations of primitive and composite events.

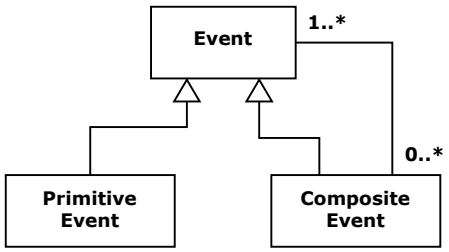

Figure 2.1 Event classification

2.2 Primitive event type

A *primitive event type* represents an atomic and instantaneous predefined elementary occurrence in a system, for example, the update of a structure, the creation of a process. In the context of databases, primitive events are related to the data modification (e.g. the insertion, deletion or modification of tuples), to the data retrieval, to the processing of transactions (e.g. begin, commit or abort transactions) and to time (e.g. absolute points in time, such as 11 November 2005, 08.00 pm, or relative points in time, such as 30 minutes after event *E* occurred). Data modification, data retrieval and processing of transactions are actions which have duration. Hence, primitive events can be signaled either *before* the action starts or *after* the action ends. In an object-oriented context, a method execution is a primitive event.

Many event models classify the primitive event types according to the operations they represent (databases, transactional, applicative). In our study, we classify the types with respect to:

Operations executed on data: an operation executed on a structure, for example, a relational table, an object. In relational systems this operation can correspond to an insert/update/delete operation applied to one or more n-tuples. In object-based systems, it can be a read/write operation of an object attribute.

Operations within processes: events can represent specific points of an execution. In SGBD, the types can represent execution points of a transaction (before or after the transaction delete/commit). In a workflow application, an event can represent the beginning (end) of a task. The production of errors inside a process can be represented by events.

User operations: an operation on a widget in an interactive interface, the connection of the user to the network, correspond to events produced by a user.

Situations coming from the execution context: events can represent situations produced out of a process: (i) specific points in time (clock), for example, it is 19:00 or 4 hours after the production of an event; (ii) events concerning to the operating system, the network, etc.

Therefore, a *primitive event expression* is a structure representing either operations executed on data, operations within processes, user operations or situations coming from the execution context. Thus, a specified primitive event expression determines a primitive event type.

In this work we consider an *interval-based semantics*, therefore primitive events occur over a *time interval* and are denoted by $E[t_1, t_2]$ (where E is the event, t_1 is the start interval of the event and t_2 is the end interval of the event). The start and the end interval of primitive events are assumed to be the same ($t_1 = t_2$). Hence, the semantic of primitive events is straightforward: a primitive event occurs and its timestamp is allocated when the occurrence is detected.

Primitive events have at least two parameters associated: an event identifier and the event timestamp, nevertheless other parameters can be associated with them, e.g. the name of the user that started the operation and, in the case of a method event, the parameters of the method.

2.3 Composite event type

A *composite event type* represents patterns of events. The constituents of a composite event may be primitive events, or composite events. From a structural point of view, a composite event is a succession of events, named *event components*, among which are: an *initiator*, a *detector* and a *terminator* event. The *initiator* of a composite event is the first event whose occurrence starts the composite event. The *detector* is the event whose occurrence implies the detection of the composite event. The *terminator* is the event that is responsible for terminating the composite event and causes its production.

A composite event E occurs over a *time interval* and it is detected at the point when its last component event is detected. In some event models the occurrence and detection semantics are not differentiated, which leads to some unintended semantics in terms of event composition.

Therefore, considering an interval-based semantics, a composite event is defined by $E[t_1,t_2]$ where E is the composite event, t_1 is the start time of the composite event occurrence and t_2 is the end time of composite event occurrence (t_1 is the starting time of the initiator component event and t_2 is the end time of the detector or terminator component event.

Occurrences of the component events of a composite event can be either *overlapping* or *disjoint*. When the events are allowed to overlap, there are thirteen possible relationships for their combination and they are shown in figure 2.2.

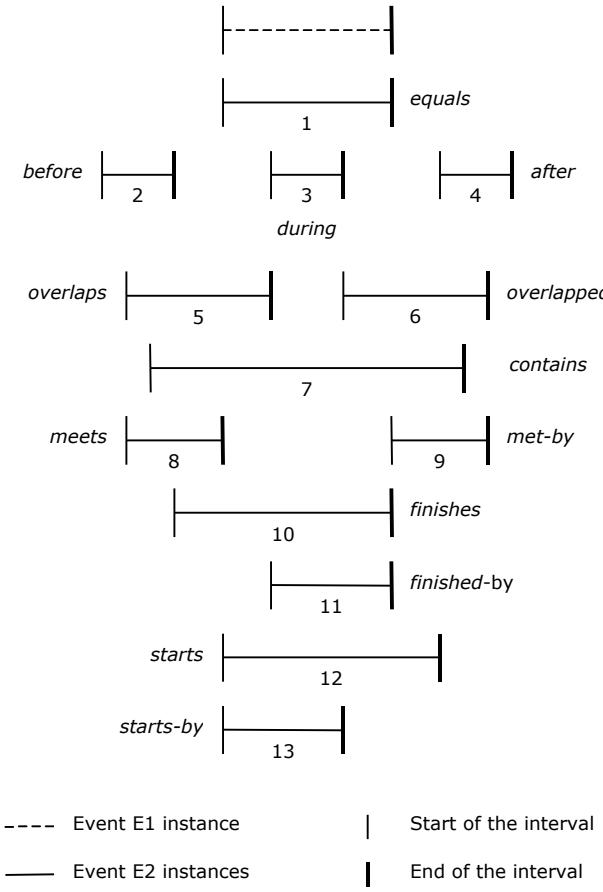

Figure 2.2 Overlapping event combinations

When events are not allowed to overlap, there are fewer combinations. This may be meaningful for many applications where the same event should not participate in the composition of more than one composite event, or only one of the overlapping events is of interest. The possible combinations are shown in figure 2.3.

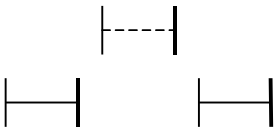

Figure 2.3 Disjoint event combinations

2.4 Event composition

Event composition is the process of creating composite events from the detected events. Detected events are stored in an *event history*. The event history consists of all occurrences of the defined events, including components of composite events. The event history begins when the first event is detected and is ordered by the timestamps of the event instances.

Because the event history may last over many sessions and transactions, a persistent event history is required if the signaling of a composite event, based on events that have occurred during different application sessions or transactions, should be possible. Resources such as memory and processor time are not unlimited and therefore can event history not be maintained indefinitely, and hence must be pruned. Minimally, the event history contains only occurrences that can still be used for event composition.

Composite events can be composed based on (i) an *event composition algebra*, where a certain pattern of event occurrences matches the composite event description, i.e., its component events have been produced, in a certain order and respecting the given constraints described by the semantic of its *event operators*, (ii) *event mining*, where event history is analyzed in search for trends and patterns from which composite events can be derived.

2.4.1 Event composition algebra

Composite events can be composed based on a defined event composition algebra. Hence, a composite event expression is defined recursively, as an event expression formed by using a set of primitive event expressions, event operators and composite event expressions.

The composition of a composite event depends on the detection of its component events. The composition process is supported by a *composition mechanism*. This maintains the structure of a composite event and the desired order of component event occurrences within it. Each time an event occurs, it is inserted into the mechanism. If the event contributes towards the composition of some composite event, the new *composition state* is derived and the old one is deleted. A certain *final state* indicates the completed composition of a composite event. If an event contributes towards the composition of multiple composite events, *rule priorities* determine the order of their evaluation. It is important that all events contributing towards a composite event occur, and that they occur in the correct order. This order is determined by the composite event expression and specifically by the event operators.

In current research projects, the composition mechanism is based on the evaluation of abstractions such as *finite state automata*, *petri nets*, *matching trees* or *graphs* (section 2.4.1.4).

2.4.1.1 Operators

The algebraic operators allow to express event composition for denote new event types. Many event models that characterize composite events consider event operators such as disjunction, conjunction and sequence. Others add the selection and negation operators. Nevertheless, no classification of the operators is established, which would make it possible to generalize the characteristics of the operators associated with the event type. In the following paragraphs, we classify the event operators in: *binary*, *selection* and *temporal operators*.

The events e_1 and e_2 used in the following definitions can be any primitive or composite event; E_1 and E_2 refer to event types with $E_1 \neq E_2$.

Binary operators

Binary operators derive a new composite event from two input events (primitive or composite events). The following binary operators are distinguished in mostly event models:

- **Disjunction: ($E_1 | E_2$)**
  There are two possible semantics for the disjunction operator (|): *exclusive-or* and *inclusive-or*. Exclusive-or means that the composite event ($E_1 | E_2$) is initiated and terminated by the occurrence of $e_1 \in E_1$ or $e_2 \in E_2$, whereas inclusive-or considers both events if they occur "at the same time". In centralized systems, no couple of events can occur "at the same time" and hence, the disjunction operator always corresponds to exclusive-or. In distributed systems, two events at different sites can occur "at the same time" and hence, both exclusive-or and inclusive-or are applicable.

$$(E_1 \mid E_2) \qquad e_1 \in E_1, e_2 \in E_2 \Rightarrow e_1[t_1,t_2] \vee e_2[t_1,t_2] \Rightarrow [t_1,t_2]$$

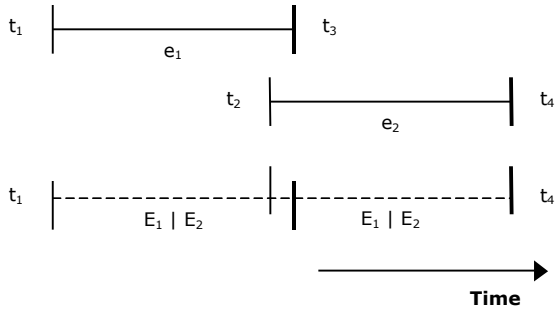

Figure 2.4 Disjunction operator

- **Conjunction: (E_1 , E_2)**
  (E_1 , E_2) occurs if both $e_1 \in E_1$ and $e_2 \in E_2$ occur, regardless of their occurrence order. Event occurrences e_1 and e_2 may be produced at the same or at different sites. The event e_1 is the event initiator of the composite event and the event e_2 is its

terminator event, or vice versa. Event occurrences e_1 and e_2 can overlap or they can be disjoint.

(E_1 , E_2) $e_1 \in E_1, e_2 \in E_2$
$\Rightarrow [t_1,t_2]$
$\exists_{t,t'} (t_1 \le t \le t_2 \wedge t_1 \le t' \le t_2 \wedge ((e_1[t_1,t] \wedge e_2[t',t_2]) \vee (e_1[t',t_2] \wedge e_2[t_1,t]))$

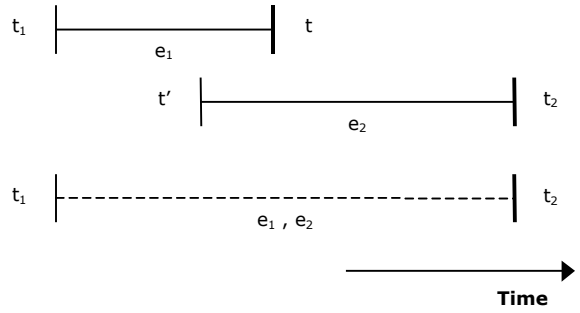

Figure 2.5 Conjunction operator

- **Sequence: (E_1 ; E_2)**
(E_1 ; E_2) occurs when first $e_1 \in E_1$ and $e_2 \in E_2$ afterwards occurs. The sequence denotes that event e_1 "happens before" event e_2. This implies that the end time t of event e_1 is guaranteed to be lees than the start time t' of event e_2. However, the semantics of "happens before" differs, depending of whether composite event is a local or a global event. Therefore, although the syntax is the same for local and for global events, the two cases have to be considered separately. The event e_1 is the event initiator of the composite event (E_1 ; E_2) and the event e_2 is its detector and terminator event.

$(E_1 ; E_2)$ $e_1 \in E_1, e_2 \in E_2$
$\Rightarrow [t_1,t_2]$
$\exists_{t,t'} (t_1 \le t \le t' \le t_2 \wedge e_1[t_1,t] \wedge e_2[t',t_2])$

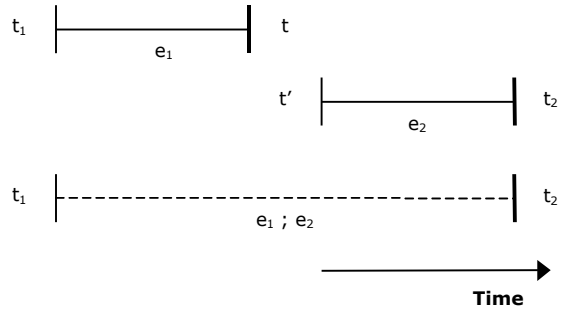

Figure 2.6 Sequence operator

- **Concurrency: (E_1 _ E_2)**
(E_1 _ E_2) occurs if both events $e_1 \in E_1$ and $e_2 \in E_2$ occur virtually "at the same time". This implies that this operator applied to two distinct events is only applicable in

global events; the events E_1 and E_2 occur at different sites and it is not possible to establish an order between them.

$(E_1 \_ E_2)$ $\quad e_1 \in E_1, e_2 \in E_2 \Rightarrow [t_1,t_2]$

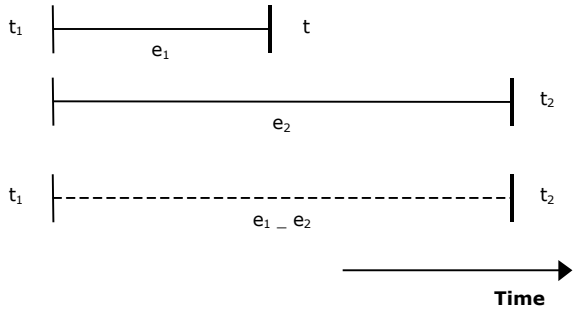

Figure 2.8 Concurrency operator

Selection operators

Selection operators allow searching production patterns of instances of an event type in the event history. The selection $E^{[i]}$ defines the occurrence of the i^{th} event $e \in E$ of a sequence of events of type E, $i \in$ N. The following selection operators are distinguished in event models such as SAMOS [GD92] and NAOS []:

- **First occurrence: (*E in I)**
  The event is produced after the first occurrence of an event $e \in E$ in an interval I. The event will not be produced by all the other occurrences of the event e in the interval.

- **History: (Times(n, E) in I)**
  An event is produced all the n occurrences of an event $e \in E$ in a time interval I.

- **Negation: (Not E in I)**
  With a limited interval, the event E is produced if any occurrence of an event is produced in the time interval I. The negation _ defines a passive event; it means that no $e \in E$ occurs within an interval of time I.

- **(occurred, holds)**
  Query an event history and verify whether an instance of an event type was produced.

- **(old, new)**
  Operators to access the content of events that represent updates. The production environment contains the data value before and/or after the execution of the update. Thus, old (new) allow accessing the value before (after) the update.

Temporal operators

Temporal event types are expressions that allow specifying a point in time. Temporal events refer to:

- **an absolute time value**: for example, 2005/03/15(08:00:00),
- **a time value relative to a reference point**: for example, <event E> + 00:15 to indicate fifteen minutes before the occurrence of an event of type E,
- **a periodical time value**: for example, 2005/*/*(23:00:00) to describe all the days of the year 2005 at 23h.

2.4.1.2 Event parameters

When specifying composite events it may be necessary to impose further restrictions on the possible combinations of primitive or composite events. Those event restrictions state conditions on the event parameters, which must be fulfilled at runtime by the component events of a composite event.

The parameters of a composite event are derived from the parameters of its component events. Table 2.1 gives an overview of other parameters of composite events depending on the binary event operators.

Composite event	Other parameters
(E_1 , E_2)	parameters of E_1 and parameters of E_2
(E_1 | E_2) exclusive-or	parameters of E_1 or parameters of E_2
(E_1 | E_2) inclusive-or	parameters of E_1 and/or parameters of E_2
(E_1 ; E_2)	parameters of E_1 and parameters of E_2
(E_1 _ E_2)	parameters of E_1 and parameters of E_2

Table 2.1 Parameters of composite events

2.4.1.3 Event consumption

When composing events, there may be several event occurrences which could satisfy a composite event, for example, consider the composite event (E_1 ; E_2) and three event occurrences for E_1: e_{11}, e_{12} and e_{13}. On the occurrence of e_{21}, the *event consumption* must be well-defined, namely, what E_1-event(s) to combine with e_{21}.

Event consumption is used to decide which component events from the event history are considered for a composite event, and how event parameters of the composite event are computed from its components. The *event consumption modes* are classified in the following event contexts:

Recent: only the newest instance of the initiator event E is used in the composite event. In the above example, the instance e_{11} of event E_1 is the initiator of the composite event (E_1 ; E_2). If a new instance of event E_1 is detected (e.g. e_{12}), the older instance is

overwritten by the newer instance. Then, the instance e_{21} of event E_2 is combined with the newest event occurrence available: (e_{13}, e_{21}). When the composite event has been detected, all components of that event, that cannot be future initiators, are flushed from the event history. This consumption mode is useful, e.g. in systems where there is a high rate of sensor readings and it does not matter if some readings are lost.

Chronicle: for an event occurrence, the initiator, terminator pair is unique. The oldest initiator and the oldest terminator for each event are coupled to form the composite event. In the example, the instance e_{21} is combined with the oldest event E_1 occurrence available: (e_{11}, e_{21}). When the composite event has been detected, all constituents of the composite event are flushed from the event history. This consumption mode is used when there is a connection between events that has to be maintained.

Continuous: each initiator of a composite event starts the detection of that composite event. The terminator event occurrence may then detect one or more occurrences of the same composite event, i.e. the terminator terminates those composite events where all the components have been detected (except for the terminator). In the example, e_{21} is combined with all event E_1 initiators: ((e_{11}, e_{21})(e_{12}, e_{21})(e_{13}, e_{21})) and does not delete the consumed events. The continuous consumption mode is different from the recent and chronicle consumption modes in the way that with recent and chronicle one initiator is coupled with one terminator, the continuous consumption mode couples one terminator with one or many initiators. The major problem with this consumption mode is that it produces combinations of events where some or all events are of interest. This adds more overhead to the system and requires more storage capacity.

Cumulative: all occurrences of an event type are accumulated as instances of that event until the composite event is detected. In the example, e_{21} is combined with all event E_1 occurrences available (e_{11}, e_{12}, e_{13}, e_{21}) and deletes the consumed events. When the terminator has been detected, i.e. the composite event is detected; all the event instances that make up the composite event are flushed from the event history.

2.4.1.4 Event composition approaches

The following approaches have been used for event composition:

Finite state automata: the automaton input is the set of 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.

Petri nets: are used in several event-based systems to support the detection of complex composite events. In a petri net created for a subscription regarding a composite event, the input places refer to primitive events, and the output places model the composite event. Each new subscription 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 tokens 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.

Matching trees: are constructed from subscriptions describing composite event structures. The primitive event parts are the leaves of the matching tree, composites are the parent nodes in the tree hierarchy. Parent nodes are responsible for maintaining information for matched events, such as mapping of event variables and successfully matching event instances. This information is updated by child nodes on every match and is passed to the parent nodes. Parent nodes perform further detection. 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 additional information is passed to the notification component.

Graphs: each composite event is represented by a directed acyclic graph (DAG), where nodes are event descriptions and edges represent event composition. 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 by rule objects that are fired after the corresponding event occurred.

In order to visualize composite events and consumption modes ***time graphs*** can be used. Time graphs have the following notation and semantics:

- A timeline, which represents the event history. Each instance of an event is marked on the time line in order of occurrence.
- One or more time intervals, over which the composite events are detected. Each interval represents the detection of a composite event for a given consumption mode. The interval includes one initiator, one terminator, and zero or more events participating in the composite event.

The instance of an event E_x will be denoted as e_{xy}, where x is the event type and y is the relative occurrence of event x. In other words, e_{12} is the second occurrence of event E_1.

To show what time graphs look like, and how the different consumption modes work, the following example will be used. The example includes three primitive events, E_1, E_2, and E_3, and event history H. The event history has the following event instances:

$$H = \{\{e_{11}\}, \{e_{12}\}, \{e_{21}\}, \{e_{31}\}, \{e_{22}\}, \{e_{41}\}, \{e_{32}\}, \{e_{42}\}\}$$

There are two composite events used: E_5 and E_6. Event E_5 is defined as a conjunction of E_1 and E_2 ($E_5 = (E_1 , E_2)$). Event E_6 is defined as a sequence of E_5 and E_3, ($E_6 = (E_5 ; E_3) = ((E_1 , E_2) ; E_3)$).

Figure 2.9 shows how the composite event E_6 is visualized in different consumption modes using time graphs. At the top there is a timeline where all the event instances that are detected are shown in the order they were detected. If an event instance is used in a composite event, a new event instance symbol (a rectangle) is created. This symbol is placed directly below the event instance on the timeline. The event instance rectangle is unfilled (white) if the event instance is an initiator to the composite event, black if the event

instance is a terminator to the composite event, and gray if the event is just a participating event in the composite event. In figure 2.9, the composite event E_6 is visualized in the four different consumption modes. As can be seen in figure 2.9, two instances of the composite event E_6 are detected when the recent, chronicle, and continuous consumption modes are used, but only one instance is detected when the cumulative consumption mode is used.

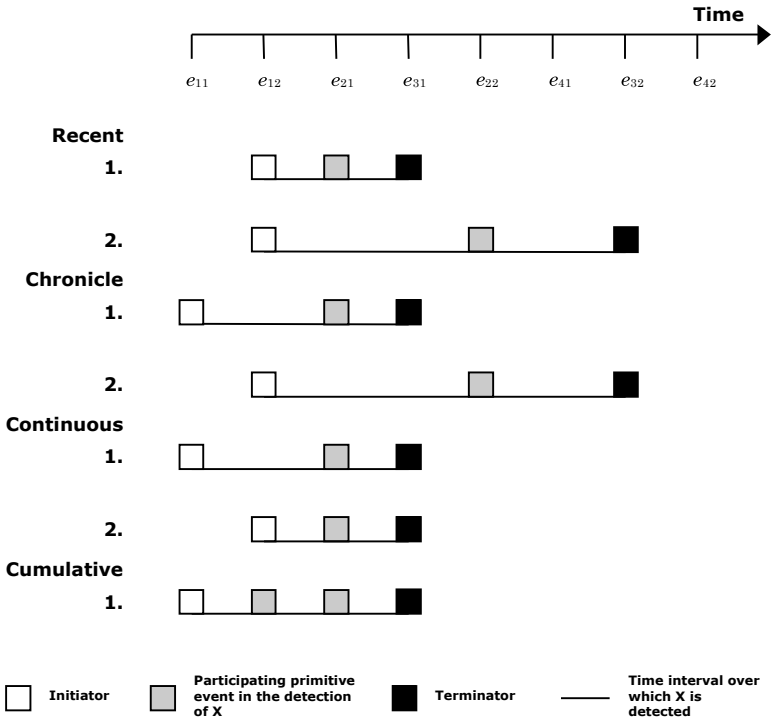

Figure 2.9 Composite event E_6 visualized using time graphs.

Time graphs explicitly show the difference between initiator, terminator and other participating events in the detection of the composite event by using different colors. On the other hand, time graphs do not contain information about the structure of the composite event, e.g. the event operators.

Using ***event graphs*** is another way of visualizing composite events and consumption modes in an event-based system. The semantics of an event graph is as follows:

- A timeline, which represents the event history. Each instance of an event is marked on the timeline in order of occurrence.
- One or more nodes. Each node represents an event operator.
- Leaves. Each leave represents a primitive event.
- Arcs. Each arc represents a connection between a node and its two children (leaves or nodes).

To show what event graphs look like, the same example as the one used to show what time graphs look like will be used. Figure 2.10 shows the detection of the composite event visualized using event graphs. There is a timeline where all the event instances that are

detected are shown in the order they were detected. When a composite event is detected, a new node is created. Each side of the node is labeled with event instances that are used to compose events, and inside the node is the operator that is used. Arcs connect the node to its two children.

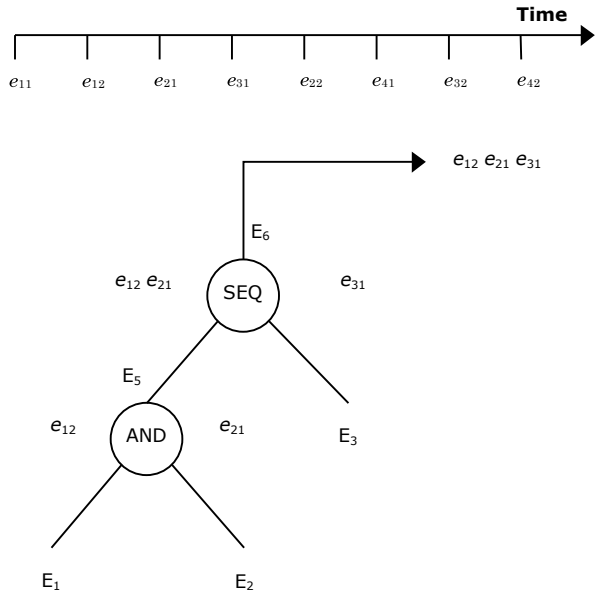

Figure 2.10 Composite event E_6 visualized using event graphs.

With event graphs it is possible to see the operators of the composite event. This is not the case with time graphs.

2.4.2 Event mining

Data mining, also known as *Knowledge-Discovery* in databases (KDD) is the practice of automatically analyzing large stores of data for patterns and then summarizing them as useful information. Data mining is sometimes defined as the process of navigating through the data and trying to find out patterns and finally establishing all relevant relationships.

Consequently, *event mining* delivers knowledge in real-time about a complex system based on events that denote the system's activities. Then, event mining goal is to identify all patterns of different types of recorded events which potentially indicate the production of an event. Event mining adopts data mining techniques for the recognition of event patterns:

Association: searching and identifying patterns such that one event is connected to another event.

Sequence or path analysis: searching and identifying patterns wherein one event leads to another later event.

Classification: searching and identifying new patterns.

Clustering: finding and visually documenting groups of events not previously known.

Forecasting: discovering patterns in data that can lead to reasonable predictions about the future.

Then, events can be mined in a multitude of ways: unwanted events are filtered out, patterns of logically corresponding events are aggregated into one new composite event, repetitive events are counted and aggregated into a new primitive event with a count of how often the original event occurred, etc.

Finding frequent patterns in a *long temporal event sequence* is a major task of temporal event mining with many applications. A temporal event sequence can be formed by integrating data sets from different domains according to common features such as the time order of each event. Then, types of events recorded have one common reference point – the recorded time of event occurrence. Consequently, a temporal event sequence is formed by ordering timestamps of these events.

The goal of long temporal event sequence mining can be generalized as follows. Let k types of time-related be given. A temporal event sequence is a list of events indexed by timestamp reflecting the time when the event was recorded. Let e be a selected event type called target event. Additionally, let time interval T be given. The goal is to find all frequent event patterns (from de event sequence) which potentially lead to the occurrence of the target event e within the given time interval T. Such kind of patterns is called event-oriented pattern because it has temporal relationship with the target event.

2.4.3 Distributed aspects

The composition of events from different sources requires information about occurrence times and order of events. Several aspects influence this information: the observation method, the timestamping method, the time system and the observer strategy.

The *system-wide* composite events are called *global composite events*, as opposed to *local composite events* which relate to event occurrence at a single site. Each global composite event is monitored at one specific observer site which contains a global event detector. The main implication of the special characteristics of distributed systems on global composite event detection is:

> In general, the *order* in which events are signaled at global event detector does not correspond to the order in which the events occurred.

The concept of order of events within a single process changes fundamentally when considering events in distributed systems. In a distributed environment, a set of distinct processes communicates by exchanging messages, the message delay is not negligible compared to the time between events in a single process. Furthermore, messages can outrun each other (message overtaking). Hence, it is difficult or even impossible to determine which one of two events occurred first.

The detection of global composite events must be based on the occurrence time of components events. However, the second implication of special characteristics of distributed systems on global composite event detection is:

Occurrence time parameters originating at different sites are inaccurate.

This implication represents a further complication for the detection of global composite events.

It is convenient to distinguish *local* and *global event expressions*. Since, all component events of a local event occur at the same site, the special characteristics of distributed systems do not have any influence on the semantics and the detection of local events.

A *local (composite) event expression* is a composite event expression whose component event expressions relate solely to one site. All primitive event expressions are local event expressions.

A *global (composite) event expression* is a composite event expression whose component event expressions relate to more than one site.

Hence, for primitive event, the occurrence time corresponds to the time of the local clock just after event detection. For composite events, the occurrence time is derived from the start and end times of occurrence of the component events participating in that occurrence.

Therefore, for the detection of composite events in a distributed system of event sources and event observers, the following information is needed: the local (partial) order of events on the same network site, the total (global) order of events, and the real occurrence times of events. If this information is not available or inaccurate, the consumers can be notified about false events or events may be missed.

2.5 Conclusion

In this chapter the event concepts were presented in order to understand the document and to explore the way in which events can be modeled. Primitive and composite event are defined like fundamental parts of an event-based system where event composition is required. The event composition was presented under two strategies: event composition algebra and event mining. The realized study about event models allowed us to take into account certain considerations for the event modeling and management, mainly in terms of event composition.