# The Formal Foundation of the Semantics of Complex Events in Active Database Management Systems

Detlef Zimmer

C–LAB*
Fürstenallee 11
D-33102 Paderborn
Phone: +49 5251 606131
Fax: +49 5251 606065
det@c-lab.de

Rainer Unland

Universität -GH- Essen
Schützenbahn 70
D-45117 Essen
unlandr@informatik.uni-essen.de

## Abstract

Active database systems have been developed for applications needing an automatic reaction in response to certain conditions or events. Events can be simple in nature or complex. Complex events are built from simpler ones with the help of operators of an event algebra.
While numerous papers propose extensions of a given set of event operators only very few address the foundations of the semantics of complex events. As a consequence most proposals mix different concepts (aspects) of complex events and offer event operators as the only means to control their semantics. This leads to peculiarities as aspects are not always handled uniformly by operators. Sometimes operators have other semantics than expected. Moreover operators of different algebras which, at first glance, look the same may have different semantics.
We developed a formal meta-model for complex events. It divides the semantics of complex events into elementary, essentially independent dimensions. The resulting elementary building blocks can be used

- to gain a solid understanding of the basic properties of complex events,
- to detect peculiarities in existing event algebras such as those mentioned above,
- to compare existing event algebras and
- to design a consistent and clear event algebra.

*Cooperative Computing & Communication Laboratory (Siemens Nixdorf Informationssysteme AG, Universität Paderborn)

## 1 Introduction

**Rules** are used in **Active Database Systems** to monitor situations of interest and to trigger a timely response when these situations occur. Rules are useful for a number of database tasks: They can enforce integrity constraints, compute derived data, control data access, gather statistics, and more. In this paper **ECA-Rules** (**E**vent-**C**ondition-**A**ction-Rules) are considered. The condition of a rule is evaluated when its triggering event occurs, and if the condition is satisfied, the rule's action will be executed.

Examples of triggering events, denoted by $e_i$, are the execution of update or retrieval operations provided by the Database Manipulation Language (DML) of the underlying database system. Such events are pre-defined and are called **primitive events**. To react to more sophisticated situations, **complex events** have been introduced. They are defined from simpler ones by using operators of an **event algebra**.
For instance, events based on a sequence operator, denoted by $e_1;e_3$, are triggered whenever $e_3$ occurs, provided that $e_1$ has already occurred. Another example is a negation operator that can be used to define events, denoted by $e_1;\neg e_2;e_3$, which are triggered whenever $e_1;e_3$ occurs, provided that $e_2$ did not occur between the occurrences of $e_1$ and $e_3$.

In general, complex events are triggered by a

set of primitive and/or complex 'component' events which, in addition, must often occur in a predefined order. The events that cause a complex event to occur are bound to it and are used for the definition of its parameters. The parameters of an event are used to transfer information about the event to other parts of the rule's implementation. Sometimes several primitive and/or complex component events may be available for binding to a complex event and a selection must be made.

We have developed a formal meta-model based on Evolving Algebras [Gur94], which defines the semantics of complex events. In this paper we informally present the basic concepts of this meta-model (for a complete and formal definition see [Zim97]). The meta-model is based on the three essentially independent dimensions **event instance pattern**, **event instance selection** and **event instance consumption**[1], which can be further split into (sub)dimensions. The **event instance pattern** is responsible for the specification of the point in time at which events occur, the **event instance selection** defines which events are bound to a complex event, and the **event instance consumption** determines when events become invalid, i.e., they cannot be considered for the detection of further complex events.

In most event algebras, such as those defined in HiPAC [DBM88], SAMOS [Gat94], Ode [GJS92], Chimera [MPC96] and NAOS [CC96], event operators are the only way to specify these different dimensions. This leads to a confusion of concepts that makes the understanding of an inherently complex area even more difficult. We will show that currently there are a number of peculiarities and irregularities in existing event algebras that can be attributed mainly to these confusions.

For instance, consider the case where event $e_1$, to be triggered before $e_3$, is triggered twice. In Snoop [Mis91, CKAK94], as we will show later, this sequence causes $e_1;e_3$ to be triggered twice while $e_1;\neg e_2;e_3$ is triggered only once.

[1]The terms *event consumption* or *consumption mode* have been used previously in the literature (see, e.g., [BBKZ93, Day95, DGG95, Gat94]). However, in these papers the authors' essential meaning corresponds to aspects of our dimension *event instance selection* and thus seems to be misleading.

The literature provides relatively few studies of the subdivision of the semantics of complex events. In Snoop [Mis91, CKAK94], parameter contexts were introduced[2]. They are responsible for the determination of the set of events that are bound to complex events. For example, SAMOS [Gat94] copes with the semantics of this by introducing additional operators called '*', 'last' and TIMES, which select the oldest ('*'), the most recent event ('last') or several events ('TIMES') out of a set of events.

As will be shown in this paper, our meta-model contributes to current research from several directions:

- It helps to gain a solid understanding of the basic properties of complex events.
- It helps to detect peculiarities in existing event algebras such as those mentioned above.
- It can be used to compare existing event algebras, and
- it provides a suitable basis on which to design a consistent and clear event algebra.

In the following section we introduce some basic definitions that are used to define our meta-model described in section 3. In the next two sections we will use our meta-model for the specification and comparison of the semantics of existing event algebras. The semantics of the event algebras defined in Snoop [CKAK94], SAMOS [Gat94] and Ode [GJS92] are specified in section 4 and compared to each other in section 5. Section 6 considers related work, and section 7 concludes the paper.

## 2 Basic Definitions

This section introduces the basic definitions needed for further discussion of our concepts. The formal and complete description of our meta-model can be found in [Zim97].

In principal, the data and transaction models of the underlying database system are orthogo-

[2]Parameter contexts were later used in ACOOD [Eri93] and in an extended version in ADL [Beh95].

nal to the functionality of ECA-rules and thus will not be considered in our model.

**Definition 1**:
An **event** is an indicator for the occurrence of a situation which may require an (automatic) reaction from the system. It is defined to be an instantaneous, atomic occurrence at one point in time (i.e., it happens completely or not at all).

A call to a database operation like the manipulation or retrieval of some data is an example of an event.

**Definition 2**:
For this paper we assume an equidistant discrete **time domain** having 0 as the **origin** and consisting of points in time represented by non-negative integers.

**Definition 3**:
In database systems a concrete event is represented by an **event instance** (EI), which contains the necessary information about the specific event.
An **event type** (ET) describes the common essentials of a sufficiently similar set of event instances on a more abstract level. It specifies the occasions on which events of the type occur, defines the parameters of the event instances, and lays down the impact of their occurrences on the occurrences of other events.

**Example 1**:
An update operation on some data item $x$ may be invoked several times. In this case the event type specifies that each call of the update operation triggers an event of its type. The actual invocation of the update operation is an event and this event is represented internally within the system by an event instance.

**Definition 4**:
An **event instance** contains **parameters** such as the identification of its event type, the event occurrence time, the actual set of event instances that caused this event instance to occur and that are bound to it and other type-specific parameters.

In general, events should be permitted to occur simultaneously. Some models (see [Gat94, CKAK94]) exclude such a behavior. However, we believe that such restrictions are not adequate as one single event may trigger a number of other (complex) events, which may even be of the same type[3]. Thus, different event instances may get the same event occurrence time.

**Definition 5**:
There are a number of elementary event types, called **primitive event types**. Primitive event types are commonly classified into either *database* or *temporal* or *external event types*. *Database event types* correspond to such database operations as data manipulation or transaction operations. On the other hand, *temporal event types* specify points in time either *absolutely* (e.g., "at 5 o'clock on the 5th of November 1997"), *relatively* (e.g., "5 minutes after calling the update operation on the data item $x$") or *periodically* (e.g., "every 5 seconds"). *External event types* represent events that occur outside the database, or even outside the computer system, and are communicated (signaled) to the database system by special database system operations.

Applications sometimes need to react to more complex situations than those expressible by primitive events. For this reason, complex events, which consist of a combination of primitive events, are introduced.

**Definition 6**:
A **complex event type** can be constructed by combining simpler event types with the help of the operators of an **event algebra**. The definition of a complex event type (recursively) consists of an operator ($op$) that combines a number of primitive and/or complex event types[4]. These types are called **component event types** while the constructed complex event type is called the **parent event type**. Complex component event types can be used to construct even more complex event types. The recursive construction of (complex) event types leads to an event type hierarchy where the primitive event types constitute its leaves and complex component event types its inner nodes.

The question of whether the component events

[3] In [CKAK94] a so-called parameter context *continuous* was introduced by which a single event can trigger multiple complex events of one type.

[4] Note, that this complex event type can be used in the definition of an even more complex event type.

of a complex event are triggered by different transactions or the same transaction does not have any impact on the design of the basic concepts of our meta-model[5]. It is, in principal, orthogonal to the event semantics considered in this paper and will therefore not be discussed here in further detail.

We assume that event types are independent of each other, i.e., events or event instances of a component event type, which is used by several (parent) event types, are available for all these types.

For the detection of complex events it is important to know which events have occurred and in what order. To describe this information we introduce event instance sequences.

**Definition 7**:
An **event instance sequence** ($EIS$) is a partially ordered set of event instances that have occurred in the system. The order of the event instances corresponds to the order of their event occurrence times. It can be differentiated between instance-oriented and type-oriented *event instance sequences*. **Instance-oriented event instance sequences** are used in complex event instances to represent the component events that caused this complex event to occur. **Type-oriented event instance sequences** are maintained for each event type and contain only those component event instances of the system that are relevant to the detection of complex events of this type.

**Notations:**
For the following discussions we use capitals to denote event types and small letters to denote events or event instances. We will use $E_i$ to denote an event type, $E_{ij}$ to denote its component event types and $EIS^{E_i}$ to denote its event instance sequence. We will use $e_i^s$ to denote the events of $E_i$, $ei_i^s$ to denote the event instance representing $e_i^s$ and $EIS^{ei_i^s}$ to denote the event instance sequence of $ei_i^s$. The superscript index $s$ reflects the order in which the events or the event instances of $E_i$ occur. The event instances of an event instance sequence are denoted in the order of their timestamps. The event instances belonging to an *instance-oriented event instance sequence* will sometimes simply be listed in brackets.

**Example 2**:
Consider a complex event that is to be signaled whenever an event of type $E_1$ occurs before an event of another type $E_2$. Such complex events are represented by an event type $E_3$, which is based on the event types $E_1$ and $E_2$ combined by a sequence operator. The sequence operator is denoted by ';' and the event type $E_3$ by $E_3$ := ; ($E_1$, $E_2$).

**Definition 8** :
The event occurrence times of the event instances belonging to $EIS^{ei_i^s}$ of a complex event instance $ei_i^s$ define the time interval during which the detection of $e_i^s$ takes place. The event instances whose event occurrence times are equal to the start of the time interval represent the **initiator events** (**initiators**) of $e_i^s$. The event instances whose event occurrence times are equal to the end of this time interval represent the **terminator events** (**terminators**) of $e_i^s$. The events which are neither initiators nor terminators are the **intervenors** of $e_i^s$. While initiators mark the beginning of the detection process of complex events, terminators mark the occurrence of complex events. The occurrence time of the terminators of a complex event instance defines its occurrence time. For simplicity we assume that an every event has only one initiator and one terminator.

We assume that primitive events are detected by the system, that the system generates the corresponding instances, and that the order of the event occurrence times of these instances reflects the order in which the events they represent have occurred. Whenever a primitive or complex event instance is detected it is inserted into the type-oriented *event instance sequences* of its parent event types.

# 3 The Meta-model

For reasons of readability this section presents the basic dimensions of the semantics of complex events on a more informal level. A detailed and formal definition can be found in [Zim97]. We focus on single event types $E_i$ and their event instance sequences $EIS^{E_i}$.

[5] This question is, however, important for the design of the rule execution model.

## 3.1 Division of the Semantics of Complex Events into Dimensions

As can be deduced from existing event algebras, there are in essence three questions that must be answered for the semantics of complex events to be defined in sufficient detail. The examples of the following questions rely on the event instance sequence $EIS^1 := ei_1^1\ ei_1^2\ ei_3^1\ ei_2^1\ ei_2^2\ ei_2^3\ ei_3^2$.

**Question 1** (*event instance pattern*):
What concrete sequences of 'component' event instances (patterns) trigger a complex event?

The answer to this question depends on a number of aspects: the types of instances belonging to a sequence, the number and order of their occurrence, and the non-existence of instances of a type, etc. For example, the event instance sequence $EIS^1$ triggers an event $e_4^1$ of event type $E_4 := ;(E_1,\ E_2,\ E_3)$. However, it does not trigger an event of a type which requires that no instances of $E_3$ are allowed to occur between the instances of $E_1$ and $E_2$.

The fact that a complex event is triggered does not mean that it is automatically clear which component event instances are associated with this complex event. That is, it may not be clear which component event instances are to be used in the condition and action parts of the rule that are triggered by the complex event. There may be a choice as to which event is to be chosen as input for the execution of a rule out of a number of instances of the same event type that occurred while the pattern of the complex event was being satisfied. So the question is:

**Question 2** (*event instance selection*):
What concrete event instances should be chosen?

The event instances that can be accessed by the triggered rule are defined by the event instance sequence of the complex event instance which triggered the rule. In general, there are several possibilities for its definition. Consider the event $e_4^1$ of $E_4$ triggered by $EIS^1$. The event instance sequence of $ei_4^1$ can be defined as $(ei_1^1\ ei_2^1\ ei_3^2)$ or $(ei_1^1\ ei_2^3\ ei_3^2)$ or $(ei_1^1\ ei_1^2\ ei_2^1\ ei_2^2\ ei_2^3\ ei_3^2)$ or any other valid combination.

**Question 3** (*event instance consumption*):
What instances are exclusively consumed by a complex event instance, i.e., what instances become unavailable for further use after they were used by this complex event instance?

For example, event instances of type $E_4$ may consume the necessary instances of type $E_1$ and $E_2$ while they preserve instances of $E_3$. Thus, two events $e_4^1$ and $e_4^2$ of $E_4$ may be triggered by $EIS^1$, where $ei_4^1$ contains the event instance sequence $(ei_1^1\ ei_2^1\ ei_3^2)$ and $ei_4^2$ $(ei_1^2\ ei_2^2\ ei_3^2)$.

Each question addresses a different dimension of an event specification. We will call these dimensions *event instance pattern*, *event instance selection*, and *event instance consumption*. In the following we will concentrate on the semantics of each dimension; the complete syntax is listed in the appendix.

## 3.2 Semantics of the Dimensions

### 3.2.1 Event Instance Pattern

The event instance pattern of an event type $E_i$ describes at an abstract level the event instance sequences $EIS^{E_i}$ that will trigger event instances of $E_i$. It must consider four aspects.

**Example 3**:
Consider the event instance sequence $EIS^2 := ei_1^1\ ei_1^2\ ei_3^1\ ei_2^1\ ei_3^2$ and the event types $E_4 := ;(E_1,\ E_2,\ E_3)$, $E_5 := ;(E_1,\ E_2)$ and $E_6 := ;(E_3,\ E_5)$.
The event type $E_4$ specifies that event instances of the types $E_1$, $E_2$ and $E_3$ must occur in the order implied by the sequence operator in the definition of $E_4$ (*type and order*).
An event $e_4^1$ of type $E_4$ will be triggered as soon as $ei_3^2$ occurs. However, the initial time interval that belongs to $ei_4^1$ starts with $ei_1^1$ and terminates with $ei_3^2$. Within this time interval there are two instances of event type $E_1$ ($ei_1^1$ and $ei_1^2$) and two instances of event type $E_3$ ($ei_3^1$ and $ei_3^2$). It has to be clarified as to how many of these instances are necessary at least and at most to trigger an event of type $E_4$ (*repetition*).
In the event type $E_5$ it is specified that first an instance of type $E_1$ has to occur and then an instance of type $E_2$. However, it

remains unclear whether these event instances must not be interrupted by any other event instance (tightly coupled), e.g., $ei_3^1$ must not occur between $ei_1^1$ (or $ei_1^2$) and $ei_2^1$ or arbitrary other event instances can occur between the instances of $E_1$ and $E_2$ (loosely coupled), e.g., $ei_3^1$ can occur between $ei_1^1$ (or $ei_1^2$) and $ei_2^1$ (*coupling*).
During the detection of an event $e_5^1$ of $E_5$ (initiated by $e_1^1$ or $e_1^2$ and terminated by $e_2^1$) an event $e_3^1$ of $E_3$ occurs. Now, the event occurrence time of the instance $ei_5^1$ corresponds to that of $ei_2^1$. Thus, the event occurrence time of $ei_3^1$ is older than that of $ei_5^1$, but younger than that of the initiator of $ei_5^1$ ($e_1^1$ or $e_1^2$). The type $E_6$ must specify whether it allows such concurrency (*concurrency*).

### Type and Order

The event types whose instances must (or must not) occur in an event instance sequence, and the restrictions concerning their order (if there are any), are defined by an event operator and its component event types.

Our model provides the following basic set of operators:
The ==-operator (**simultaneous operator**) requires that instances occur simultaneously, i.e., their occurrence time is the same; the ;-operator (**sequence operator**) requires that instances occur in a specified order; the $\wedge$-operator (**conjunction operator**) requires that a number of instances occur in any order; the $\vee$-operator (**disjunction operator**) requires that at least one of the specified instances occurs; and the $\neg$-operator (**negation operator**) requires that the given instance(s) should not occur in a given period or interval.

Periods are specified in the form of event instances which mark the beginning and end of the time interval being considered for evaluation. The negation operator only makes sense in conjunction with a period during which the non-occurrence of event instances is monitored. Thus, the first operand of the negation operator specifies the beginning and the last one specifies the end of the period during which the non-occurrence of the instances of the 'inner' types is to be monitored.

### Repetition

For every component event type $E_{ij}$ a **delimiter** may be specified. It restricts the number of event instances of $E_{ij}$ which must occur to satisfy the event instance pattern. If no delimiter is given, one or more instances of a type must occur. An upper and/or a lower bound may be given as a delimiter. Thus, the number of event instances required for an event instance pattern may be restricted to a range of numbers or even to a particular number.

### Coupling and Concurrency

So-called **operator modes** can be used to define the coupling mode and the concurrency feature. The mode (*coup mode*) defines whether event instance patterns may be interrupted by event instances not relevant to the event detection (**non-continuous**), or whether they may not (**continuous**). The mode (*cc mode*) is used to define whether the time intervals associated with the event instances which cause a complex event to occur may or may not overlap (**overlapping** versus **non-overlapping**).

## 3.2.2 Event Instance Selection

Event instance selection is responsible for the construction of instance-oriented event instance sequences, i.e., it determines what event instances are taken from the type-oriented event instance sequence $EIS^{E_i}$ to form the event instance sequence $EIS^{ei_i^s}$. This selection is performed individually for each component event type $E_{ij}$ and it is quite natural to select at least all those instances that caused the complex event to occur.

**Example 4**:
Consider the event instance sequence $EIS^3 := ei_1^1\ ei_2^1\ ei_1^2\ ei_2^2\ ei_2^3\ ei_1^3\ ei_3^1$, the event type $E_4 := ;(E_1, E_2, E_3)$ and the event $e_4^1$ of $E_4$ which is triggered by $EIS^3$.

Let us assume that the instances $ei_1^2$ and $ei_3^1$ have already been selected for the types $E_1$ and $E_3$. Thus, we will focus on the selection of instances of type $E_2$. Consider the four event instance sets: (1) ($ei_1^2\ ei_2^2\ ei_3^1$), (2) ($ei_1^2\ ei_2^3\ ei_3^1$),

(3) ($ei_1^2$ $ei_2^2$ $ei_2^3$ $ei_3^1$) and (4) ($ei_2^1$ $ei_1^2$ $ei_2^2$ $ei_2^3$ $ei_3^1$). While sets (1) and (2) contain only one instance of $E_2$, sets (3) and (4) contain several instances. Set (1) contains the oldest instance of $E_2$ which, together with $ei_1^2$ and $ei_3^1$, fulfills the event instance pattern of $E_4$, while set (2) contains its most recent instance. Note that the selection of event instances depends on instances that were already selected: if we had chosen $ei_1^1$ instead of $ei_1^2$, the instance $ei_2^1$ would be the oldest instance of $E_2$ that could be selected. Set (3) contains only instances which occurred between the selected instances of $E_1$ and $E_3$, and thus takes the event instance pattern into account, while set (4) contains every instance of $E_2$.

The selection strategies can be classified according to whether only the minimum number of event instances required by the delimiter of $E_{ij}$ is selected (minimum instance set-oriented strategy) or a greater number is selected (cumulation-oriented strategy). In the most extreme case, the cumulation strategy would select the whole set of event instances of $E_{ij}$ belonging to $EIS^{E_i}$.

**First** (**last**) are minimum set instance-oriented selection strategies. These always select the set of instances with the oldest (youngest) timestamps. One cumulation-oriented selection strategy is **cumulative** which selects the complete instance set of $E_{ij}$. The other cumulation-oriented selection strategy **restricted cumulative** chooses only those instances of $E_{ij}$ that are considered by the event instance pattern of $E_i$.

To obtain a unique solution one has to define the order in which event instances must be collected from the given event instance sequence:

**Example 5**:
Let us consider the event type $E_4 :=$ ; (*last* :$E_1$, *first* :$E_2$, $E_3$) and the event sequence $EIS^3$. Since $ei_3^1$ had triggered the recognition of event $e_4^1$ of $E_4$, it is the terminator instance of the time interval assigned to $ei_4^1$. However, the initiator instance of $ei_4^1$ is not yet clear. Of course, it must be an instance of $E_1$. However, which one should be chosen: $ei_1^1$ or $ei_1^2$ or $ei_1^3$?
If the event type sequence were traversed from right to left, i.e., first the correct instance of event type $E_3$ is identified (which is, of course, trivial) then the instance of $E_2$ and finally the instance of $E_1$, we would have to choose $ei_3^1$, $ei_2^1$, and $ei_1^1$.
If we traversed the event type sequence in the opposite direction, we would first have to choose $ei_1^3$ since it represents the last occurrence of an event of $E_1$ in $EIS^3$. However, this would not lead to a legal solution because the time interval specified by the initiator instance $ei_1^3$ and the terminator instance $ei_3^1$ does not include an instance of $E_2$. Therefore, we have to backtrack. This means that first or last have to be interpreted as first or last legal instance of the given type. The next possibility would be $ei_1^2$. The interval spanned by $ei_1^2$ and $ei_3^1$ contains $ei_2^2$ which is not the first instance of $E_2$, but is the first in the interval spanned by $ei_1^2$ and $ei_3^1$. Therefore, $ei_1^2$, $ei_2^2$, and $ei_3^1$ would be the correct solution if left to right traversal were chosen.
Thus, the result of the event instance selection depends on the order of the traversal.

To summarize, we must first specify the order of traversal of the time interval that belongs to a concrete event instance of an event type, either from left-to-right or from right-to-left. The corresponding event instance sequence is then traversed in this order and the first legal solution that is identified is the correct one.
Here the semantics can be controlled by the *operator mode trav mode* that can be either **left-to-right** (default mode) or **right-to-left**.

### 3.2.3 Event Instance Consumption

Event instance consumption defines the impact of the occurrence of events of a complex event type $E_i$ on the availability of event instances of its component event types for the subsequent detection of further events of type $E_i$. The set of event instances that is considered by the detection of events of the type $E_i$ is defined by the event instance sequence $EIS^{E_i}$. Thus, event instance consumption defines the set of event instances that are deleted from $EIS^{E_i}$ whenever an event $e_i^s$ occurs.

**Example 6**:
Consider event type $E_3 :=$ ; (*last* :$E_1$, *first* :$E_2$) and its event instance sequence $EIS^{E_3} := ei_1^1$ $ei_1^2$ $ei_2^1$ $ei_2^2$. With the occurrence of $ei_2^1$, an in-

stance $ei_3^1$ is generated with the event instance sequence $EIS^{ei_3^1}$ which contains the instances $ei_1^2$ and $ei_2^1$. Now let us assume that an event instance $ei_2^s$ of type $E_2$ is consumed if it is used in an event instance sequence of event $ei_3^s$. The occurrence of $ei_3^1$ may cause the deletion of (1) $ei_1^2$ or of (2) $ei_1^1$ and $ei_1^2$ or (3) no deletion may occur. Dependent on this result, $ei_2^2$ will (1+3) or will not (2) trigger another event $e_3^2$ of $E_3$. If it is triggered, $ei_3^2$ may be considered to be caused by the event instance sequence ($ei_1^2$ $ei_2^2$) (3) or ($ei_1^1$ $ei_2^2$) (1).

We distinguish three different consumption modes that can be specified individually for each component event type. The **shared** mode does not delete any instance of $E_{ij}$ (3). The **exclusive parameter** mode removes all instances of $E_{ij}$ from $EIS^{E_i}$ that belong to the event instance sequence of an instance $ei_i^s$ (1). The **exclusive** mode deletes all instances of $E_{ij}$ from $EIS^{E_i}$ that occur before the terminator instance of $ei_i^s$ (2).

If the same component event type $E_{ij}$ is used several times in the definition of $E_i$, the strongest consumption mode will dominate the weaker ones.

## 3.3 Event Groups

A terminator instance can trigger the recognition of several events of its parent type if it is used in the *shared* mode. All such events (event instances) of the same event type $E_i$ which are triggered by the same terminator form an **event group**.

### 3.3.1 Event Instance Selection

To avoid infinite looping (e.g., by infinitely constructing event instances from the same basic event instances) different event instances of the same type must differ in that they are not allowed to exclusively use the same (basic) event instances.

**Example 7**:
Consider the event instance sequence $EIS^4 := ei_1^1\ ei_1^2\ ei_1^3\ ei_2^1\ ei_1^4\ ei_2^2\ ei_3^1$ and the event type $E_4 :=$ ; (.... $E_1$, .... $E_2$, *shared* :$E_3$). The occurrence of $ei_3^1$ will trigger a number of events of type $E_4$, depending on the modes assigned to $E_1$ and $E_2$. Let us assume that $E_2$ has the modes *first : exclusive parameter*, $E_1$ is either *first : shared* (the resulting parent event type is denoted by $E_5$), *last : shared* ($E_6$), or *restricted cumulative : shared* ($E_7$). Figure 1 presents the semantics of these event types.

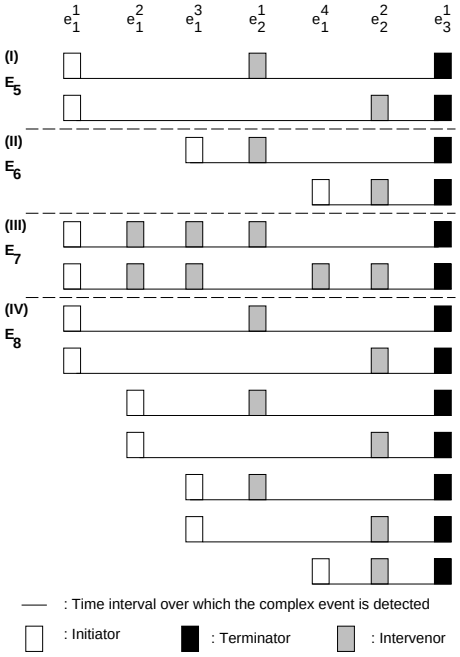


Figure 1: Event Instance Selection for Event Groups

This example shows that the event instance selection modes introduced above are not suitable for the definition of event types that behave like the event type $E_8$ (see figure 1). The instances of the event group of $E_8$ consider all those event instance sets that contain exactly one instance for each component event type.

To support the semantics here, the event instance selection modes **combinations** and **combinations minimum** are introduced. If one of these modes is used for a component type $E_{ij}$ the different instance sets of $E_{ij}$ are alternately used and combined with the event instance sets of the other component event types to form event instance sequences of the event instances belonging to a group. While the mode **combinations minimum** defines that only the minimum number of event instances required by the delimiter of $E_{ij}$ are taken into account, **combinations** does not impose this constraint, i.e., it can also consider larger sets of event instances.
Note that these modes only make sense if they

are used for event types $E_i$ whose events share their terminator events.

**Example 8**:
The event type $E_8$, the behavior of which is shown in figure 1, can be defined as $E_8$ := ; (*combinations minimum : shared :* $E_1$, *combinations minimum : shared :* $E_2$, *shared :* $E_3$).

### 3.3.2 Event Instance Consumption

**Example 9**:
Now consider the event type $E_5$ defined in example 7 and the event instance sequence $EIS^4$ extended by $ei_2^3$ $ei_3^2$. Figure 2 shows the semantics of $E_5$. Every pair of events of $E_2$ and $E_3$ will subsequently trigger an event of $E_5$. The event instance consumption modes do not offer the possibility of distinguishing between the availability of event instances inside and outside a group. Thus event instances that are shared by instances cannot be protected from being shared by other groups as well.

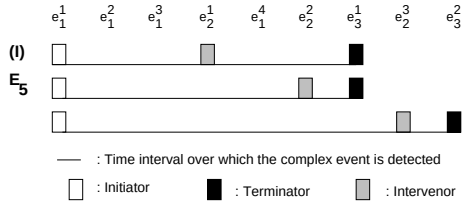


Figure 2: Event Instance Consumption for Event Groups

To cope with the semantics, we introduce the domains inside a group (**inside**) and outside a group (**outside**). They can be used in conjunction with the consumption modes and define the availability of event instances only inside or outside a group (as in figure 2).
The event instance consumption for the **outside** domain is applied to the union of the event instance sequences of the event instances belonging to the same group.

## 3.4 Dependencies between the Dimensions

The dependencies between the dimensions of the meta-model become clear when we look at the event detection processed for an event type $E_i$:

```
(1) E_i.event_instance_pattern()
(2) ei := E_i.new_instance;
(3) ei.EIS := E_i.event_instance_selection();
      :
(4) E_i.EIS :=  E_i.event_instance_consumption(ei.EIS);
```

This algorithm is executed each time an event instance is inserted into $EIS^{E_i}$.
In the first step, it is determined whether the event instance sequence of $E_i$ fulfills its event instance pattern (1). If the pattern is fulfilled a new event instance $ei$ is generated (2). Next, its event instance sequence is computed (3). Finally, the event instances that were consumed by $ei$ are deleted from the event instance sequence $EIS^{E_i}$ (4).

Thus, the dimensions of the meta-model are not independent. However they are orthogonal, i.e., modes of different dimensions can be combined without any restrictions.

# 4 Specification of Existing Event Algebras

In this section the meta-model is used to specify the semantics of some of the most sophisticated event algebras presented in the literature. In the first section of this chapter we will define the semantics of some event algebras. In the second section for each event algebra, the semantics are first described in terms of our meta-model, then some irregularities revealed by our meta-model are discussed.

## 4.1 Semantics of the Event Algebras

### 4.1.1 Snoop

Snoop has been developed at the University of Florida. The concepts of Snoop have been implemented in a prototype called Sentinel [CKAK94, Kri94]. The definition of the semantics of Snoop is only partially formal. Thus sometimes - as it is shown in [Zim96] - its semantics is ambiguous.

In addition to the standard event operators conjunction ($\triangle$), disjunction ($\nabla$), sequence (;) and negation (NOT) Snoop defines the operators: ANY, $A$, $P$, $A^*$ and $P^*$. Events based on

the ANY operator, denoted as ANY(m, $E_1$, $E_2$, .., $E_n$), where m $\leq$ n, occur whenever events from $m$ types out of the $n$ distinct event types occur[6]. Events based on the a-periodic operator $A$, denoted as A($E_1$, $E_2$, $E_3$), occur whenever the a-periodic event ($E_2$) occurs during the closed time interval specified by $E_1$ and $E_3$. The periodic operator $P$ is used to define periodically occurring temporal events. $A^*$ and $P^*$ are cumulative versions of the operators $A$ and $P$, i.e. events based on them are only triggered once at the end of the time interval ($E_3$).

Based on the initiator and terminator events Snoop defines four **parameter contexts**:

In the **recent** context only the most recent instance of the set of instances that may have started the detection of a complex event is used. When a complex event occurs the instances of the component event types which cannot be initiators of future events are deleted. An initiator of an event continues to initiate new event occurrences until a new initiator occurs.
In the **chronicle** context the oldest instance of each component event type is bound to the instances of its parent type. The instances of the component event types can only be used once.
In the **continuous** context, if a terminator event is detected, for each initiator an event instance of its parent type is generated. In this context, an initiator is used at least once for detecting complex events.
In the **cumulative** context all instances of the component event types are bound to the instance of the parent type. The instances of the component event types can only be used once for an event detection.

Every Snoop operator can be combined with one of these parameter contexts to form an event type. A complex event can be constructed by applying several operators which may have assigned different parameter contexts. This aspect has not been investigated by Snoop in further detail.

[6]The semantics of the ANY-operator can be specified by the conjunction and disjunction operator. For example, the semantics of the type ANY(2, $E_1$, $E_2$, $E_3$) is equivalent to the semantics of the type (($E_1 \triangle E_2$) $\nabla$ ($E_1 \triangle E_3$) $\nabla$ ($E_2 \triangle E_3$)). Therefore we will not consider the ANY-operator in this paper any more.

In contrast to our model the parameter modes of Snoop can be specified on the level of complex event types rather than on the component event type level. The parameter contexts define a fixed combination of the values of the dimensions *event instance selection* and *event instance consumption* introduced in our model. Thus the configuration possibilities offered by Snoop are limited to these combinations.

### 4.1.2 SAMOS

In SAMOS [Gat94] the semantics of complex events are defined informally. However, a more precise definition of their semantics can be obtained from labeled petri nets which are used for their detection. Simultaneously occurring events are explicitly excluded.

SAMOS defines the binary event operators conjunction (,), disjunction (|), and sequence (;), and the unary operators negation (NOT), *, *last* and *TIMES(n,E)*. The unary operators must be used in conjunction with a monitoring time interval[7], denoted as 'IN [*start_point*, *end_point*]'. The start and the end point of the interval can be defined explicitly by an absolute or relative point in time or implicitly by the occurrence of events of specified event types. If no interval is given the system assumes the time between the definition time of the complex event type and infinity. Events which mark the monitoring interval of complex events are not bound to them.

The operators * and *last* are used whenever complex events should only be signaled once during a given time interval, even if the pattern specified by the corresponding event types occurs several times. While the * operator selects the oldest occurrence of an event of a given type the *last* operator chooses the most recent occurrence. Events based on the * operator are signaled as soon as the complex event to be monitored occurs for the first time while events based on the *last* operator are signaled at the end of the monitoring interval.
The *TIMES* operator can be used in different variants. Events based on the variant

[7]A monitoring interval can also be defined for the binary event operators. But for these operators it is optional.

*TIMES*(n,E) IN $I$ are signaled each time $n$ events have occurred during the time interval $I$, events based on the variant *TIMES*([$n_1$-$n_2$],E) IN $I$ are signaled at the end of $I$ if events of the type $E$ occurred between $n_1$ and $n_2$ times in $I$ and events based on the variant $TIMES([> n_1], E)$ IN $I$ are signaled at the end of $I$ if events of the type $E$ have occurred more than $n_1$ times. The *TIMES* operator uses the cumulative selection mode, i.e. all events of the type E which caused the complex event to occur are bound to it.

For the operators of the event algebra of SAMOS the event instance selection and consumption policies are fixed. But the user has the possibility to partly influence these aspects by using the operators TIMES, * and last.

#### 4.1.3 Ode

Ode [GJS92, WC96] defines its semantics of complex events on a formal level. Its definition is based on event histories (*histories*) which contain a finite set of events that are totally ordered by their event times. The semantics of an event type $E$ is defined as a mapping from histories to histories, i.e. $E$: $histories \rightarrow histories$. The resulting history $E[h]$ contains those events of $h$ which trigger an event of the type $E$. An event type can be $NULL$, any primitive event type $a$, or a complex event type that consists of component event types. These can be combined by one of the operators $\wedge$, ! (not), $relative$, $relative+$, $\vee$, $any$, $prior$ and $sequence$[8]. Let $E$ and $F$ be event types. Their semantics are defined as follows:

1. $E[null] = null$ for any event type $E$, where $null$ is the empty history.

2. $NULL[h] = null$.

3. $a[h]$ is the maximal subset of $h$ composed of events of the type $a$, where $a$ is a primitive event type.

4. $(E \wedge F)[h] = E[h] \cap F[h]$

5. $(!E)[h] = (h$ - $E[h])$

6. $relative(E, F)[h]$ are the events in $h$ that trigger events of $F$. However, only that parts of $h$ are considered that at each time start immediately after an instance of $E$ is detected.
   Let $E^i[h]$ be the $i^{th}$ event in $E[h]$. Let $h_i$ be obtained from $h$ by deleting all events with an event time older than or equal to the event time of $E^i[h]$. Then
   $relative(E, F)[h] = \bigcup_i F[h_i]$,
   where $i$ ranges from 1 to the cardinality of $E[h]$.

7. $relative+(E)[h] = \bigcup_{i=1}^{\infty} relative^i(E)[h]$ where
   $relative^1(E) = E$ and
   $relative^i(E) = relative(relative^{i-1}(E), E)$.

8. $(E \vee F)[h] = !(!E \wedge !F)[h]$.

9. $any$ denotes the disjunction of all the primitive event types.

10. $prior(E, F)[h] = (relative(E, any) \wedge F)[h]$.
    The operator *prior* specifies, that an event of $F$ has to take place after an event of $E$ has taken place. These events of $E$ and $F$ may - in contrast to the operator *relative* - overlap.

11. $sequence(E, F)[h] = (relative(E, !(relative(any, any))) \wedge F)[h]$.
    The operator *sequence* specifies immediately successive triggering of events of the types $E$ and $F$.

In Ode [GJS92] the semantics of the event instance selection is discussed shortly. The selection of event instances consists of two steps: In the first step the alternative event instance sequences that fulfil the event instance pattern are computed. In the second step the event instance selection is performed through queries on the event instance sequence set computed in the first step. A detailed description of possible selection strategies as well as their realization is postponed to a future paper.

[8] Ode defines a number of further event operators, that are not considered in this paper.

## 4.2 Specification of the Event Algebras

In this section we will specify the semantics of the event algebras of Snoop, SAMOS and Ode on the basis of our meta-model (for the specification of further event algebras, such as ACOOD [Eri93], ADL [Beh95], Chimera [MPC96], NAOS [CC96] see [Zim97]). For each of these event algebras we will present a table which contains its specification. Each table is organized as follows.
For each operator of an event algebra there is an entry that may stretch over a maximum of three rows. The first column of each entry contains the event type to be specified (notation of the event algebra to be examined, upper row) and the skeleton of the event type of the meta-model on which the specification is based (notation of the meta-model, lower row). The second column lists the component event types for which the subsequent columns contain the modes that must be assigned to them to form the complete event type of the meta-model.
The table for Snoop contains a separate column that specifies the semantics for each parameter context.

We use abbreviations for the different modes: *sh* for **sh**ared, *ex* for **ex**clusive, *in* for **in**side, *out* for **out**side, *param* for **param**eter, *cum* for **cum**ulative, *comb* for **comb**inations, *min* for **min**imum and *rest* for **rest**ricted.

Whenever the semantics of an event algebra, from our point of view, is ambiguous or irregular we mark it by adding the symbol '?' and using bold letters for the mode that causes the irregularity.

**Example 10**:
Consider the Snoop operator $(E_1 \triangle E_2)$ in the parameter context *chronicle* (see table 3). The table should be read as follows:

- first column, first row: $(E_1 \triangle E_2)$ is the notation of Snoop
- first column, second row: $\wedge\ (E_1,\ E_2)$ is the principal notation in our meta-model
- second column, first (second) row: $E_1$ $(E_2)$, means that the occurrence of $E_1$ $(E_2)$ in the skeleton notation of the meta-model must be enriched by the specifications in column 4 (chronicle).

Altogether this results in the following meta-model expression, that is equivalent in its semantics to the original notation of Snoop:
$\wedge$ (*first : exclusive parameter :* $E_1$, *first : exclusive parameter :* $E_2$).

### 4.2.1 Snoop

Table 3 presents the specification of Snoop on the basis of our meta-model. Let us consider some examples describing some irregularities of Snoop that were detected with the help of our specification presented in table 3 (Further examples can be found in [ZUM97, Zim97]). In these examples the Snoop event types are denoted by a parameter context followed by an event operator and its operands.

**Example 11**:
Consider the event types: $E_4$ := *recent* $E_1$;$E_3$ and $E_5$ := *recent* NOT($E_2$)[$E_1$,$E_3$] and the event instance sequence $EIS^5 := ei_1^1\ ei_3^1\ ei_3^2$. The sequence $EIS^5$ causes the recognition of two events $e_4^1$ and $e_4^2$ of $E_4$ represented by $ei_4^1$ $(ei_1^1 ei_3^1)$ and $ei_4^2$ $(ei_1^1 ei_3^2)$ but only one event $e_5^1$ of $E_5$ represented by $ei_5^1$ $(ei_1^1 ei_3^1)$.
An event type based on the negation operator is an extension of the event type based on the sequence operator that defines the time interval during which the non-existence of events is monitored. Thus, in the case of the non-occurrence of the specified events, one would assume that the behavior of these event types is the same. Unfortunately, these event types use different event instance consumption modes for their first component event type: while the event type based on the sequence operator uses the shared mode, the event type based on the negation operator uses the exclusive mode.

**Example 12**:
The *recent* and the *chronicle* parameter contexts use different event instance consumption modes. While in the *recent* parameter context the event instances are often[9] shared, they are used exclusively in the *chronicle* parameter context.

[9] Only terminator events are used exclusively.

| Snoop−ET | | parameter contexts | | | |
|---|---|---|---|---|---|
| Meta−Model−ET | CP−ET | recent | chronicle | continous | cumulative |
| $E_1 \triangle E_2$ | $E_1$ | last : **sh** | first : ex param | comb min : sh in : ex out | cum : ex param |
| $\wedge (E_1, E_2)$ | $E_2$ | last : **sh** | first : ex param | comb min : sh in : ex out | cum : ex param |
| $E_1 \triangledown E_2$ | $E_1$ | ex param | ex param | ex param | ex param |
| $\vee (E_1, E_2)$ | $E_2$ | ex param | ex param | ex param | ex param |
| $E_1 ; E_2$ | $E_1$ | last : **sh** | first : ex param | comb min : ex param | cum : ex param |
| $; (E_1, E_2)$ | $E_2$ | ex param | ex param | sh in : ex param out | ex param |
| NOT $(E_2)$ $[E_1,E_3]$ | $E_1$ | last : **ex** | first : ex param | comb min : ex param | ?**first :** ex param |
| $\neg (E_1,E_2,E_3)$ | $E_3$ | ex param | ex param | sh in : ex param out | ex param |
| A $(E_1,E_3,E_2)$ | $E_1$ | last : sh | first : **ex param** | comb min : ex param in : **sh out** | **first :** ex param |
| $\neg (E_1,E_2,E_3)$ | $E_3$ | ex param | ex param | sh in : ex param out | ex param |
| A* $(E_1,E_2,E_3)$ | $E_1$ | last : **ex** | first : ex param | comb min : ex param | ?**first :** ex param |
| $\vee (\neg (E_1,E_2,E_3)$ , | $E_2$ | ? rest cum : ex | rest cum : ex param | rest cum : sh in : ex out : | rest cum : ex param |
| $;(E_1,E_2,E_3))$ ?! | $E_3$ | ex param | ex param | sh in : ex param out | ex param |

Figure 3: The semantics of the complex events defined in Snoop

Consider the event types $E_3 :=$ *recent* $(E_1 \triangle E_2)$ and $E_4 :=$ *chronicle* $(E_1 \triangle E_2)$ and the event instance sequence $EIS^6 := ei_1^1\ ei_2^1\ ei_2^2$. The sequence $EIS^6$ causes the recognition of two events $e_3^1$ and $e_3^2$ of $E_3$ represented by $ei_3^1$ $(ei_1^1 ei_2^1)$ and $ei_3^2$ $(ei_1^1 ei_2^2)$ but only one event $e_4^1$ of $E_4$ represented by $ei_4^1$ $(ei_1^1 ei_2^1)$.

**Example 13**:
The event instance consumption mode of the first component event type of events based on the a-periodic operator used in the *chronicle* context is *exclusive parameter*. Thus, the characteristics of the a-periodic operator get lost, as it is signaled only once (for every event instance of the first component event type).
Consider the event type $E_4 :=$ *chronicle* A($E_1$, $E_2$, $E_3$) and the event instance sequence $EIS^7 := ei_1^1\ ei_2^1\ ei_2^2\ ei_2^3\ ei_3^1$. The sequence $EIS^7$ causes the recognition of only one event $e_4^1$ of $E_4$ represented by $ei_4^1$ $(ei_1^1 ei_2^1)$ and not - as one may expect - three events.

**Example 14**:
Let us consider event types based on the a-periodic operator A used in the *continuous* parameter context. The event instance consumption mode of their first component event type is *exclusive parameter inside : shared outside*, i.e., its instances can only be used once inside a group but they are shared between different groups. Thus, instances of the second component event type occurring after an instance of the first type are not only associated with this instance but also with every older instance of the first type:
Consider the event type $E_4 :=$ *continuous* A($E_1$, $E_2$, $E_3$) and the event instance sequence $EIS^8 := ei_1^1\ ei_2^1\ ei_1^2\ ei_2^2\ ei_2^3$. The sequence $EIS^8$ causes the recognition of the events $e_4^1$, $e_4^2$, $e_4^3$, $e_4^4$ and $e_4^5$ of $E_4$ represented by $ei_4^1$ $(ei_1^1 ei_2^1)$, $ei_4^2$ $(ei_1^1 ei_2^2)$, $ei_4^3$ $(ei_1^2 ei_2^2)$, $ei_4^4$ $(ei_1^1 ei_2^3)$ and $ei_4^5$ $(ei_1^2 ei_2^3)$.

### 4.2.2 SAMOS

Table 4 presents the specification of SAMOS on the basis of our meta-model. Let us consider some examples of the irregularities of SAMOS we have noticed through the specification of its semantics.

**Example 15**:
Consider the event type $E_4 :=$ NOT $E_2$ IN [$E_1$-$E_3$] and the event instance sequence $EIS^9 := ei_1^1\ ei_1^2\ ei_2^1\ ei_3^1\ ei_3^2$. The time interval [$E_1$-$E_3$] can be instantiated several times, e.g., $I_1 = [ei_1^1, ei_3^1]$ and $I_2 = [ei_1^2, ei_3^2]$. As during both time intervals an instance of $E_2$ $(ei_2^1)$ occurs, one might

| SAMOS–ET<br>Meta–Model–ET | CP–ET | CPET–Modes |
|---|---|---|
| $E_1$ , $E_2$ | $E_1$ | first : ex param |
| ∧ ( $E_1$ , $E_2$ ) | $E_2$ | first : ex param |
| $E_1$ ❙ $E_2$ | $E_1$ | ex param |
| ∨ ( $E_1$ , $E_2$ ) | $E_2$ | ex param |
| $E_1$ ; $E_2$ | $E_1$ | first : ex param |
| ; ( $E_1$ , $E_2$ ) | $E_2$ | ex param |
| $E_1$ ; $E_2$ IN [$E_1$ $E_3$] | $E_1$ | ? first : **sh** |
| ¬ ( $E_1$,$E_3$,$E_2$) | $E_2$ | ex param |
| *$E_1$ ; $E_2$ | $E_1$ | first : ex |
| ; ( $E_1$ , $E_2$ ) | $E_3$ | ex param |
| last $E_1$ ; $E_2$ | $E_1$ | last : ex |
| ; ( $E_1$ , $E_2$ ) | $E_2$ | ex param |
| TIMES ([$n_1$ $n_2$] , $E_2$)<br>IN [$E_1$ $E_3$] | $E_1$ | ? first : ex param |
| | $E_2$ | rest cum :ex param |
| ;( $E_1$,($n_1$ $n_2$) $E_2$,$E_3$)) | $E_3$ | ex param |
| TIMES ([ >n ], $E_2$)<br>IN [$E_1$ $E_3$] | $E_1$ | ? first : ex param |
| | $E_2$ | rest cum :ex param |
| ;( $E_1$,($n_1$) $E_2$,$E_3$)) | $E_3$ | ex param |
| TIMES (n , $E_2$)<br>IN [$E_1$ $E_3$] | $E_1$ | ? first : sh |
| ¬ ( $E_1$,$E_3$,(n)$E_2$) | $E_2$ | ex param |
| NOT ($E_3$) IN [$E_1$ $E_2$] | $E_1$ | ? first : ex param |
| ¬ ( $E_1$,$E_3$,$E_2$) | $E_3$ | ex param |

Figure 4: The semantics of the event operators defined in SAMOS

expect that $EIS^9$ does not cause an event of $E_4$ to occur. But the event detection algorithm of the NOT operator specified by a labeled petri net (see [Gat94], page 110) deletes instances of $E_2$, and thus $ei_3^2$ causes the occurrence of an event of $E_4$.

**Example 16**:
Consider the event type $E_4 := {}^*E_2$ IN [$E_1$-$E_3$] and the event instance sequence $EIS^{10} := ei_1^1\ ei_2^1\ ei_1^2\ ei_2^2\ ei_3^1\ ei_3^2$. Again the time interval [$E_1$,$E_3$] can be instantiated several times, e.g.,: $I_1 = [ei_1^1, ei_3^1]$ and $I_2 = [ei_1^2, ei_3^2]$. While both event instances $ei_2^1$ and $ei_2^2$ of $E_2$ occur during $I_1$, only $ei_2^2$ occurs during $I_2$. The sequence $EIS^{10}$ causes the occurrence of only one instance of $E_4$: $ei_4^1$ ($ei_2^1$) rather than two instances $ei_4^1$ ($ei_2^1$) and $ei_4^2$ ($ei_2^2$), as one might expect.

**Example 17**:
Consider the event type $E_4 := (E_1;\ *E_2)$ IN [$E_1$-$E_3$] and the event instance sequence $EIS^{11} := ei_1^1\ ei_2^1\ ei_2^2\ ei_3^1$. The sequence $EIS^{11}$ causes the occurrence of two events of $E_4$ (see [Gat94], p. 111).

**Example 18**:
Consider the event type $E_4 := TIMES(2, E_2)$ IN [$E_1$,$E_3$] and the event instance sequence $EIS^{12} := ei_1^1\ ei_2^1\ ei_1^2\ ei_2^2\ ei_3^1\ ei_2^3\ ei_2^4\ ei_3^2$. Event instances of type $E_3$ invalidate all instances of $E_2$ which have already occurred. Thus, events of type $E_4$ are triggered twice: $ei_4^1$ ($ei_1^2 ei_2^2$) and $ei_4^2$ ($ei_2^3 ei_2^4$). Therefore, the event instance $ei_4^2$ is **not** signaled after an even number of event occurrences of type $E_2$ relative to the lower bound of the time interval, as one might expect.

### 4.2.3 Ode

Table 5 presents the specification of Ode on the basis of our meta-model. The semantics of the conjunction and negation operators are quite different from that of other event algebras:
A conjunction of two event types occurs whenever events of both component event types occur simultaneously. A negation of an event type occurs whenever an event of a different type occurs.

| Ode–ET / Meta–Model–ET | CP–ET | CPET–Modes |
|---|---|---|
| $E_1 \wedge E_2$ | $E_1$ | ex param |
| == ( $E_1$ , $E_2$ ) | $E_2$ | ex param |
| $E_1$ ! $E_2$ | $E_1$ | ex param |
| $\vee$ ( $E_1$ , $E_2$ ) | $E_2$ | ex param |
| relative ($E_1$, $E_2$) | $E_1$ | ? first : sh |
| non–overlap ;( $E_1$ , $E_2$ ) | $E_2$ | ex param |
| prior ($E_1$, $E_2$) | $E_1$ | ? first : sh |
| ; ( $E_1$ , $E_2$ ) | $E_2$ | ex param |
| sequence ($E_1$, $E_2$) | $E_1$ | last : **ex** |
| continuous ; ( $E_1$ , $E_2$ ) | $E_2$ | ex param |
| ! $E_1$ | $E_2$ | ex param |
| $\vee$ ( $E_2$ , $E_3$ ,....,$E_n$) | $E_n$ | ex param |

Figure 5: The semantics of the event operators defined in Ode

# 5 Comparison of the Event Algebras

In the following we will compare the semantics of the conjunction, negation and sequence operators of the different event algebras with the help of our meta-model. The disjunction-operator is omitted since it has the same semantics in each event algebra. Each other operator is represented by a separate table.

There is an entry (row) for each specialization of the semantics of an operator in a given event algebra. If the same operator can be expressed in several models there will be several entries in the first two columns of a row, separated by a dashed line. If there is no entry for an event algebra in a given table this simply means that the corresponding operator is not supported by the algebra. Otherwise, all possible specializations of an operator are listed which implies that missing specializations are not supported.

The first column of an entry defines the event algebra and the second the event type to be specified. The other columns contain the specification of an event type based on the meta-model. The third column defines the skeleton of the event type of the meta-model on which the specification is based. As the skeleton of most of the operators is the same, it is listed at the head of the table. The other rows only contain information when there are additional modes or different skeletons. The remaining columns define the modes assigned to the component event types listed at the head of the column. These columns serve the same purpose as the parameter context columns in the previous tables: they refine the skeleton in that the appropriate component event type of the skeleton has to be enriched with the specification of the column.
Note: In Snoop the same semantics can be expressed by different expressions (as can be seen from the first row of table 8).

| Target–ET | | Meta–Model–ET | | |
|---|---|---|---|---|
| Name | ET | ET | CP–ET | |
| | | $\wedge$ ( $E_1$ , $E_2$ ) | $E_1$ | $E_2$ |
| SAMOS | $E_1$ , $E_2$ | | first : ex param | first : ex param |
| Snoop | chronicle $E_1 \triangle E_2$ | | first : ex param | first : ex param |
| Snoop | recent $E_1 \triangle E_2$ | | last : **sh** | last : **sh** |
| Snoop | cumulative $E_1 \triangle E_2$ | | cum : ex param | cum : ex param |
| Snoop | continous $E_1 \triangle E_2$ | | comb min : sh in : ex out | comb min : sh in : ex out |
| Ode | $E_1 \wedge E_2$ | == ( $E_1$ , $E_2$ ) | ex param | ex param |

Figure 6: The Conjunction Operator

# 6 Related Work

A systematic and profound debate about event specification has not yet started. Instead, most papers treat only specific aspects of a rule model in an isolated manner or concentrate on the narrow model of a specific prototype. For example, ACOOD [Eri93], ADL [Beh94, Beh95], Chimera [MPC96], NAOS [CC96], REACH [BBKZ93] and Reflex [NI94] are based on the event algebras discussed in more detail in the previous sections.
Other papers that do not directly rely on a specific model discuss formal aspects on a more general level (for example [HJ91, Wid92, Zan93, GHJ+93, FMT94, CFPT95, CC95]; for an overview see [PCFW95]). They do not or only insufficiently treat complex event specification. To our best knowledge our paper is the

| Target−ET | | Meta−Model−ET | | | |
|---|---|---|---|---|---|
| Name | ET | ET | CP−ET | | |
| | | ; ( $E_1$ , $E_2$ ) | $E_1$ | $E_2$ | $E_3$ |
| Ode | prior ( $E_1$,$E_2$) | | ? first :sh | ex param | |
| Ode | relative ( $E_1$,$E_2$) | non−overlap | ? first :sh | ex param | |
| SAMOS | $E_1$;$E_2$ IN [ $E_1$− $E_3$] | | ? first : **sh** | ex param | |
| SAMOS | $E_1$; $E_2$ | | first : ex param | ex param | |
| Snoop | chronicle $E_1$; $E_2$ | | first : ex param | ex param | |
| SAMOS | *$E_1$; $E_2$ | | first : **ex** | ex param | |
| SAMOS | last$E_1$; $E_2$ | | last : ex | ex param | |
| Ode | sequence ( $E_1$,$E_2$) | continuous | last : **ex** | ex param | |
| Snoop | recent $E_1$; $E_2$ | | last : **sh** | ex param | |
| Snoop | cumulative $E_1$; $E_2$ | | cum :ex param | ex param | |
| Snoop | continous $E_1$; $E_2$ | | comb min : ex param | sh in : ex param out | |
| SAMOS | TIMES ([ $n_1$− $n_2$] ,$E_2$) IN [ $E_1$− $E_3$] | ;( $E_1$ ( $n_1$− $n_2$) $E_2$,$E_3$)) | ? first : ex param | rest cum : ex param | ex param |
| SAMOS | TIMES ([ >n ], $E_2$) IN [ $E_1$− $E_3$] | ;( $E_1$ ( $n_1$−) $E_2$,$E_3$)) | ? first : ex param | rest cum : ex param | ex param |
| Snoop | recent A* ( $E_1$,$E_2$,$E_3$) | ∨ (¬ ( $E_1$,$E_2$,$E_3$) , ;( $E_1$,$E_2$,$E_3$)) | last : **ex** | ? rest cum : ex | ex param |
| Snoop | chronicle A* ( $E_1$,$E_2$,$E_3$) | ∨ (¬ ( $E_1$,$E_2$,$E_3$) , ;( $E_1$,$E_2$,$E_3$)) | first : ex param | rest cum : ex param | ex param |

Figure 7: The Sequence Operator

| Target−ET | | Meta−Model−ET | | |
|---|---|---|---|---|
| Name | ET | ET | CP−ET | |
| | | ¬ ( $E_1$,$E_2$,$E_3$) | $E_1$ | $E_3$ |
| SAMOS | NOT ($E_3$) IN [ $E_1$− $E_2$] | | ? first :ex param | ex param |
| Snoop | chronicle ¬ ($E_2$) [$E_1$,$E_3$] | | first :ex param | ex param |
| Snoop | cumulative ¬ ($E_2$) [$E_1$,$E_3$] | | ? **first :** ex param | ex param |
| Snoop | cumulative A ( $E_1$,$E_3$,$E_2$) | | **first :** ex param | ex param |
| Snoop | chronicle A ( $E_1$,$E_3$,$E_2$) | | first : **ex param** | ex param |
| Snoop | recent A ( $E_1$,$E_3$,$E_2$) | | last : sh | ex param |
| Snoop | recent ¬ ($E_2$) [$E_1$,$E_3$] | | last : **ex** | ex param |
| Snoop | continous A ( $E_1$,$E_3$,$E_2$) | | comb min : ex param in : **sh out** | sh in : ex param out |
| Snoop | continous ¬ ($E_2$) [$E_1$,$E_3$] | | comb min : ex param | sh in : ex param out |
| SAMOS | TIMES (n , $E_2$) IN [ $E_1$− $E_3$] | | ? first : sh | ex param |
| Ode | ! $E_1$ | ∨ ( $E_2$ , $E_3$ ,....,$E_n$) | ex param | ex param |

Figure 8: The Negation Operator

first one that discusses the semantics of complex events in detail.

# 7 Conclusions

In this paper we have presented the basic concepts of a formal meta-model that defines the semantics of complex events. It is based on the three dimensions *event instance pattern*, *event instance selection* and *event instance consumption*. The **event instance pattern** is responsible for the specification of the point in time at which events occur, the **event instance selection** defines which events are bound to a complex event, and the **event instance consumption** determines when events become invalid, i.e., cannot be considered for the detection of further complex events.
In principle, while an event instance pattern is associated with an event type, the dimensions event instance selection and event instance consumption are related to the component event types of complex event types.
The three dimensions are independent in respect of their usage, i.e., they can be combined without any restrictions. In particular, the event instance selection and the event instance consumption policies can be chosen separately for each component event type of a complex event type. Moreover, our model allows for simultaneously occurring events.

The model presented in this paper contributes to the ongoing work in the area of active database systems in several ways:

1. We developed a flexible meta-model which can be used to specify the semantics of complex events defined in other rule models.

2. This model can be used to compare the semantics of complex events defined in existing rule models.

3. Our model can be used as a basis for the definition of new event algebras.

4. The semantics of our model is defined formally. Thus, different interpretations of the semantics of a given complex event are not possible.

5. The meta-model helps in gaining a solid understanding of the basic properties of complex events and the interrelationships between their components.

# 8 Appendix

The syntax of a complex event type is defined as follows:

$< CET >$ ::= $< pattern >$
$< pattern >$ ::= $[< op\ mode >] < OP >$ ( $< CPET\_list >$ )
$< CPET\_list >$ ::= $< CPET >$ | $< CPET >$ , $< CPET\_list >$
$< CPET >$ ::= $[< opd\ mode >] < ET >$
$< ET >$ ::= $< PET >$ | $< CET >$
$< OP >$ ::= == | ; | $\wedge$ | $\vee$ | $\neg$
$< op\ mode >$ ::= $[< cc\ mode >:]$ $[< coup\ mode >:]$ $[< trav\ mode >:]$
$< cc\ mode >$ ::= **non-overlapping** | **overlapping** | [default : **overlapping**]
$< coup\ mode >$ ::= **continuous** | **non-continuous** | [default : **non-continuous**]
$< trav\ mode >$ ::= **left-to-right** | **right-to-left** | [default : **left-to-right**]
$< opd\ mode >$ ::= $[< par\ select >:]$ $[< cosu >:]$ $[< delimiter >:]$
$< par\ select >$ ::= **first** | **last** | **cumulative** | **restricted cumulative** | **combinations** [**minimum**] | [default : **last**]
$< cosu >$ ::= **inside** $< cosu\ mode >$ : **outside** $< cosu\ mode >$ | $< cosu\ mode >$
$< cosu\ mode >$ ::= **exclusive** | **shared** | **exclusive parameter** | [default:**exclusive parameter**]
$< delimiter >$ ::= ($< integer >$) | ($< range >$)
$< range >$ ::= $< integer >$ - $< integer >$ | $< integer >$ - | - $< integer >$

The definition of the syntax of a complex event type (CET) reflects the three dimensions of the semantics of a complex event:
The **event instance pattern** is defined by an operator (*op*), an operator mode (*op mode*) and a list of component event types ($CPET\_list$) if necessary supplemented by a delimiter. A component event type (CPET) can be a complex or primitive event type (PET). The **event instance selection** is defined by the modes

*par select* and *trav mode* while the **event instance consumption** is defined by the mode *cosu*.

# References


[BBKZ93] H. Branding, A. Buchmann, T. Kudrass, and J. Zimmermann. Rules in an Open System: The REACH Rule System. In N.W. Paton and M.H. Williams, editors, *Rules in Database Systems (RIDS-93), First Int'l Workshop, Edinburgh*, pages 111–126, 1993.

[Beh94] H. Behrends. An Operational Semantics for the Activity Descpription Language ADL. Technical Report TR-IS-AIS-94-04, Universität Oldenburg, June 1994.

[Beh95] H. Behrends. Beschreibung ereignisgesteuerter Aktivitäten in datenbankgestützten Informationssystemen. Dissertation TR-IS-AIS-95-03, University of Oldenburg, Germany, April 1995.

[CC95] Thierry Coupaye and Christine Collet. Denotational Semantics for an Active Rule Execution Model. In Timos Sellis, editor, *Rules in Database Systems (RIDS-95), Second Int'l Workshop, Athens, Greece*, pages 36–50, September 1995.

[CC96] C. Collet and T. Coupaye. Composite Events in NAOS. In R. Wagner and H. Thoma, editors, *Proc. 7th DEXA, Zurich, Switzerland*, pages 475–481, September 1996.

[CFPT95] S. Comai, P. Fraternali, G. Psaila, and L. Tanca. A Uniform Model to Express the Behaviour of Rules with Different Semantics. In M. Berndtsson and J. Hansson, editors, *First Int'l Workshop on Active and Real-Time Database Systems (ARTDB-95)*, 1995.

[CKAK94] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite Events for Active Databases: Semantics, Contexts and Detection. In *Proc. 20th Very Large Data Bases*, pages 606–617, October 1994.

[Day95] U. Dayal. Ten years of Activity in Active Database Systems: What Have We Accomplished? In M. Berndtsson and J. Hansson, editors, *Proc. 1st Intl. Workshop on Active and Real-Time Database Systems, Skoevde, Sweden*, pages 3–22, June 1995.

[DBM88] U. Dayal, A. Buchmann, and D. McCarthy. Rules are Objects Too: A Knowledge Model For An Active, Object-Oriented Database System. In *2nd Int'l Workshop on Object-Oriented Database Systems*, pages 129–143, September 1988.

[DGG95] K. Dittrich, S. Gatziu, and A. Geppert. The Active Database System Manifesto: A Rulebase of ADBMS Features. In Timos Sellis, editor, *Rules in Database Systems (RIDS-95), Second Int'l Workshop, Athens, Greece*, pages 3–17, September 1995.

[Eri93] J. Eriksson. CEDE: Composite Event DEtector In A Active Object-Oriented Database. Master Thesis University of Skövde, 1993.

[FMT94] P. Fraternali, D. Montesi, and L. Tanca. Active Database Semantics. In *Proc. of the 5th Australasian Database Conf. (ADC), Christchurch, New Zealand*, pages 195–212, January 1994.

[Gat94] Stella Gatziu. *Events in an Active, Object-Oriented Database System.* Phd-Thesis. Dr. Kovac, November 1994.

[GHJ+93] S. Ghandeharizadeh, R. Hull, D. Jacobs, J. Castillo, M. Escobar-Molano, S. Lu, J. Luo, C. Tsang, and G. Zhou. On Implementing a Language for Specifying Active Database Execution Models. In *Proc. 19th Very Large Data Bases*, pages 441–454, October 1993.

[GJS92] N.H. Gehani, H.V. Jagadish, and O. Shmueli. Composite Event Specification in Active Databases: Model and Implementation. In *Proc. 18th Very Large Data Bases*, pages 327–338, October 1992.

[Gur94] Y. Gurevich. Evolving Algebra 1993: Lipari Guide. *Specification and Validation Methods*, 1994. E. Börger (ed.), OUP, Oxford.

[HJ91] Richard Hull and Dean Jacobs. Language Constructs for Programming Active Databases. In *Proc. 17th Very Large Data Bases*, pages 455–467, September 1991.

[Kri94] V. Krishnaprasad. Event Detection for Supporting Active Capability in an OODBMS: Semantics, Architecture and Implementation. Master's thesis, University of Florida, 1994.

[Mis91] D. Mishra. Snoop: An Event Specification Language for Active Database Systems. Master's thesis, University of Florida, 1991.

[MPC96] R. Meo, G. Psaila, and S. Ceri. Composite Events in Chimera. In *Proc. EDBT, Avignon, France*, pages 56–76, March 1996.

[NI94] W. Naqvi and M. Ibrahim. EECA: An Active Knowledge Model. In *Proc. 5th DEXA, Athen, Greece*, pages 380–389, September 1994.

[PCFW95] Norman W. Paton, Jack Campin, Alvaro A.A. Fernandes, and M. Howard Williams. Formal Specification Of Active Database Functionality: A Survey. In Timos Sellis, editor, *Rules in Database Systems (RIDS-95), Second Int'l Workshop, Athens, Greece*, pages 21–35, September 1995.

[WC96] J. Widom and S. Ceri. *Active Database Systems*. Morgan Kaufmann Publishers, 1996.

[Wid92] J. Widom. A Denotational Semantics for the Starburst Production Rule Language. *ACM record*, 21(3):4–9, September 1992.

[Zan93] C. Zaniolo. A Unified Semantics for Active and Deductive Databases. In N. W. Paton and M H. Williams, editors, *Rules in Database Systems, Edinburgh 1993*, pages 271–287, 1993.

[Zim96] D. Zimmer. A Formal Metamodel for the Definition of the Semantics of Complex Events. C-LAB Report 29, C-LAB, Fürstenallee 11, 33102 Paderborn, Germany, http://www.c-lab.de, December 1996.

[Zim97] D. Zimmer. *Ein Meta-Modell für die Definition der Semantik von komplexen Ereignissen in Aktiven Datenbanksystemen*. PhD thesis, Universität Paderborn, 1997. To be published.

[ZUM97] D. Zimmer, R. Unland, and A. Meckenstock. A General Model for Event Specification in Active Database Management Systems. In *Proc. 5th DOOD, Montreux, Switzerland*, 1997. To be published.