Context-Based Event Processing Systems

Opher Etzion, Yonit Magid, Ella Rabinovich,
Inna Skarbovsky, and Nir Zolotorevsky

Abstract. The concept of context has recently emerged as one of the major abstractions in event processing modeling with presence in event processing products. In this chapter we discuss the notion of context as a first class citizen within event processing modeling, and discuss its implementation in event processing products and models that arise from the current state-of-the practice

Keywords: Context, context-aware computing, context-driven architecture, temporal processing, spatial processing, event processing modeling, event processing products.

1 Introduction: Context in Event Processing Modeling

In real life, many activities are done within a context; we might behave differently within different parts of the day, in different locations, in different states of the weather, these are all types of context. Indeed, context [8] plays the same role in event processing that it plays in real life; a particular event can be processed differently depending on the context in which it occurs, and it may be ignored entirely in some contexts. Contexts have been formalized in some works such as [4], [5], and [11].

There are three main uses of context by event processing applications:

- An event stream is defined [3] as open-ended set of events. If you want to perform an operation on the stream you cannot wait until all these events have been received. Instead you have to divide the stream up into a sequence of context partitions, or *windows*, each of which contains a set of consecutive events. You can then define the operation in terms of its effect on the events in a window. The rule that determines which event instances are admitted into which window is something we call a *temporal context*.
- A collection of events, arriving in one or multiple streams, may contain events that are not particularly connected to one another, even though they might occur close together in a temporal sense. They might, for example,

Opher Etzion · Yonit Magid · Ella Rabinovich · Inna Skarbovsky · Nir Zolotorevsky
IBM Haifa Research Lab, Haifa, Israel
e-mail: {opher,yonit,ellak,inna,nirz}@il.ibm.com

refer to occurrences in different locations, or to occurrences involving different entities in the real world. Suppose you were to do some processing of an event stream, such as a simple *aggregate* agent that counts the number of events. By default this would count all the events in the stream, but what if you want to see separate totals for each location where the events occurred? To do this you need to have a separate agent, or at least a separate *instance* of the agent, processing the events for each location. *Spatial* contexts and *Segmentation-oriented* contexts let you assign related events to separate context partitions. You can then have each partition processed by a distinct instance of the event processing agent, so that events in one context partition are processed in isolation from the events in other partitions.

- Context also allows event processing agents to be context sensitive, so that an agent that is active in some contexts may be inactive in others. We refer to this as *state-oriented* context.

In essence, in event processing modeling, an EPA (Event Processing Agent) serves as the basic unit of event processing computation. An EPA may be of three types: filter EPA, transform EPA, and pattern detect EPA (which is the most general one). Context may apply to all of these types; Fig. 1 taken from [8] shows that each EPA is associated with a context, the association is aimed to achieve one or more of the context roles mentioned before.

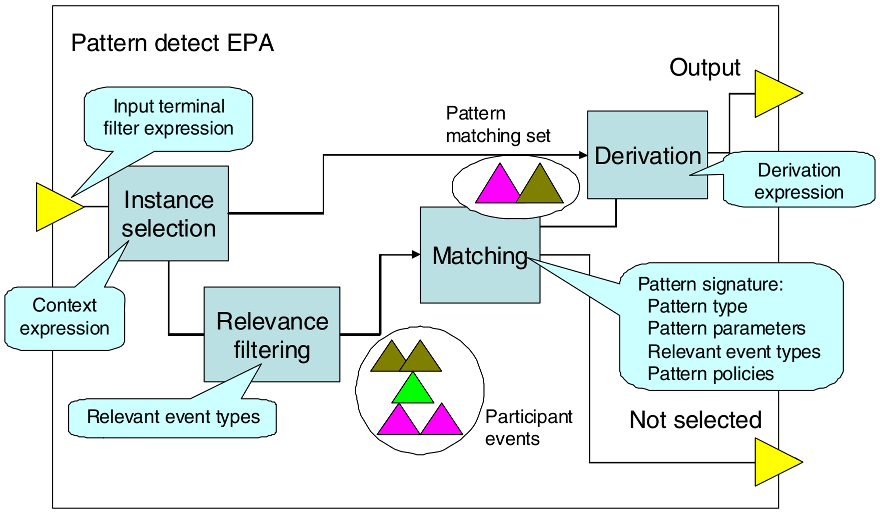


Fig 1. The Event Processing Agent functionality model**.**

Context affiliation for EPA determines:

- Whether an input event is relevant at all for a specific EPA – this makes the EPA context sensitive; depends on the context an EPA may or may not be involved

- If the EPA has several instances, to which of these instances, an event is relevant for – this serves as grouping of events together according to shared context.

In order to understand the role, let's look at a simple example of segmentation oriented context; events that relate to transactions may be partitioned by customer, since we would like to process the events related to each customer separately.

Within event processing languages, contexts may be explicit, implicit, or partially explicit. Explicit context means that context primitives are first class primitives in the language. Some languages do not support any notion of context, and some support partial notion of contexts. A survey of contexts in various languages will be presented in Section 5, dealing with contexts in practice.

Fig. 2 taken from [8] shows the various context dimensions: temporal, spatial, state oriented, and segmentation oriented that we describe in this section.

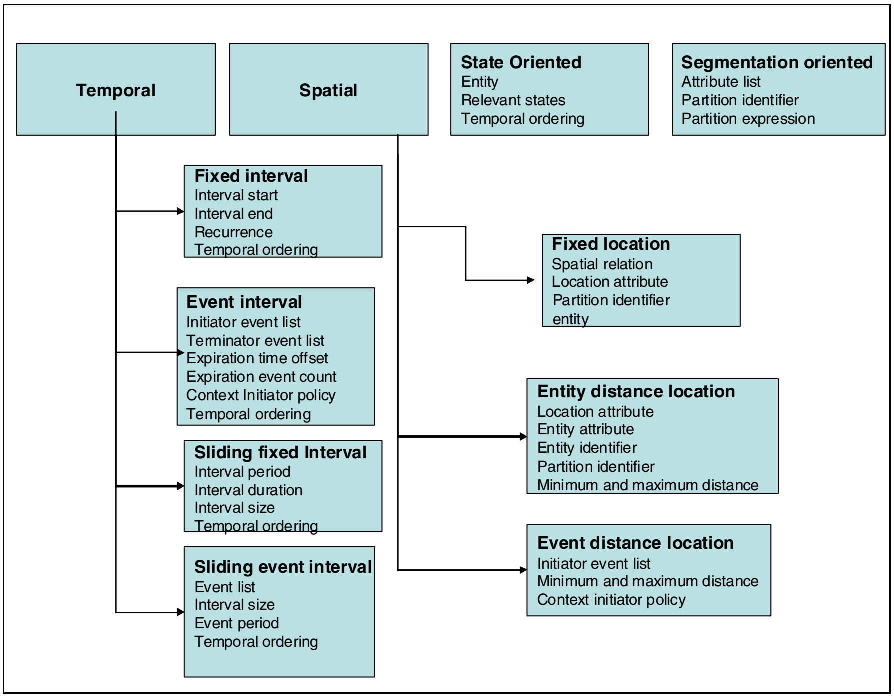


Fig 2. Context dimensions.

The rest of this chapter is structured in the following way: Section 1.2 discusses temporal contexts, Section 1.3 discusses spatial contexts, Section 1.4 discusses segmentation oriented context, Section 1.5 discusses state oriented context, Section 1.6 discusses context composition, Section 1.7 discusses the implementation of context in practice, and Section 1.8 concludes the chapter with discussion on the state of the art, its limitations, and the potential future of contexts in event processing.

2 Temporal Context

Temporal context is aimed to partition the temporal space into time intervals. It is used for two purposes:

- Events grouping: the typical use of temporal contexts is to group events to process them together based on the fact they have occurred during the same interval.
- EPA applicability: Another use of temporal contexts is to indicate that a certain event processing agent is applicable within certain intervals and is not applicable in other intervals. The temporal interval convention is having half-open interval [Ts, Te), such that Ts is the interval starting point, included in the interval, and Te is the interval end-point, not included in the interval.

An example of event grouping is: count all the bids within the auction period. In this case the auction period stands for a temporal interval, and all the events of type bid that occur within this interval are grouped together for calculating the aggregation function (count).

An example of EPA applicability is: detect if all printers at the same floor are off-line during working hours. In this case there is a pattern that is used to check whether all printers in the same floor are off-line at the same time. However monitoring for this pattern is relevant only within working hours, since it is assumed that after working hours there is no technician on-site.

One of **the** design decisions in associating events to temporal context is to determine whether a specific event falls within the interval. In some cases this is being determined by the detection time, which is the timestamp created by the system [9] when the event enters the system, or by occurrence time which is the time in which the event source specifies as the time in which the event occurred in reality. This decision is determined by using the temporal ordering parameter in the various temporal context definitions

There are four types of temporal contexts: fixed interval, event interval, sliding fixed interval and sliding event interval. We survey briefly each of these types.

2.1 Fixed Temporal Interval

A fixed temporal interval consists of one or more temporal intervals whose boundaries are pre-defined timestamp constants. This can be a one time interval: [July 12 2010 13:30, July 12 2010 17:00) which designates a specific session within a specific conference; it also can be stated as [July 12 2010 10:30, + 3.5 hours), where Te is an offset relative to Ts. A fixed temporal interval can also be recurrent, and in this case the frequency should be specified. Some example of that is [7:00, 9:00) daily, or [Monday 13:30, 14:30) weekly which determines event processing operations that are applicable periodically, for example: some traffic monitoring is done during morning rush hour only, or that during a weekly meeting some monitoring is disabled.

2.2 Event Interval

Event interval is a temporal interval that is being opened or closed when one or more events occur; the meaning of "event occurs" is determined by the temporal ordering parameter and can be interpreted either as the detection time or the occurrence time as explained earlier. The collection of events that open such an interval are called initiator events, and the collection of events that close the temporal interval is called terminator events. An interval may also expire after a certain time offset is reached.

Some examples of event interval are:

- A temporal interval is initiated when a patient is admitted to a hospital, and ends with the release of the same patient from the hospital. Note that here the temporal context is combined with segmentation context, since the temporal context refers to a single patient.
- A temporal interval that starts with a shipment of a package and ends with the delivery of the same package; the temporal interval expires after 3 days, which is the delivery designated time, even if the package has not been shipped, thus there are two ways in which this interval can terminate: the desired way (delivery) and the time-out way (3 days have passed).

2.3 Sliding Fixed Interval

In a sliding fixed interval context, new windows are opened at regular intervals. Unlike the non-sliding fixed interval context these windows are not tied to particular times, instead each window is opened at a specified time after its predecessor. Each window has a fixed size, specified either as a time interval or a count of event instances.

A sliding fixed interval is specified by the interval period which designates the frequency in which new windows are opened, the interval duration, which specifies how long this interval spans, and the interval size, which determines how many events are included in a single window. The specification must include an *interval period* parameter and either an *interval duration* or *interval size* (or both, in which case it may end earlier than the duration if the event count is exceeded). Sliding intervals may be overlapping or non-overlapping; they are overlapping if and only if the *interval period* < *interval duration*.

Some examples are as follows:

- A temporal interval starts every hour, with the duration of one hour. In this case we partition the time into time windows of one hour each.
- A temporal interval starts every day, and ends when 50 orders have been placed in the system, or at the end of the day. In this case the interval partitions the temporal space into time windows of one day; however, the daily interval can terminate earlier if there are 50 orders.
- A temporal interval starts every 10 minutes, and lasts for an hour. In this case, each event (starting from the second hour of operation) is associated with 6 different windows that are active in parallel. This type of context can be used for trend seeking windows in time series' events.

2.4 Sliding Event Interval

The *sliding event interval* context is similar to the sliding fixed interval context. The difference is that the criterion for opening a new window is specified as a count of events, rather than as a time period. A sliding event interval is specified by list of events (possibly with a predicate for each event), interval size (in event count), and event period. The event list designates event types whose instances can start the temporal window; the interval size determines the event count that closes the interval, while the event period (which defaults to the interval size) determines the event count to open an additional window.

An example is the following:

- A sensor measures the temperature; an interval consists of five consecutive measurements and starts with every single measurement. Each measurement is associated with five different intervals.

3 Spatial Context

A spatial context groups event instances according to their geospatial characteristics. This type of context assumes that an event has a spatial attribute designated its location. Location can be represented in three ways [8]: point in space, line or polyline, and area (polygon). As shown in Fig.3, there are three types of spatial contexts: fixed location, entity distance location and event distance location.

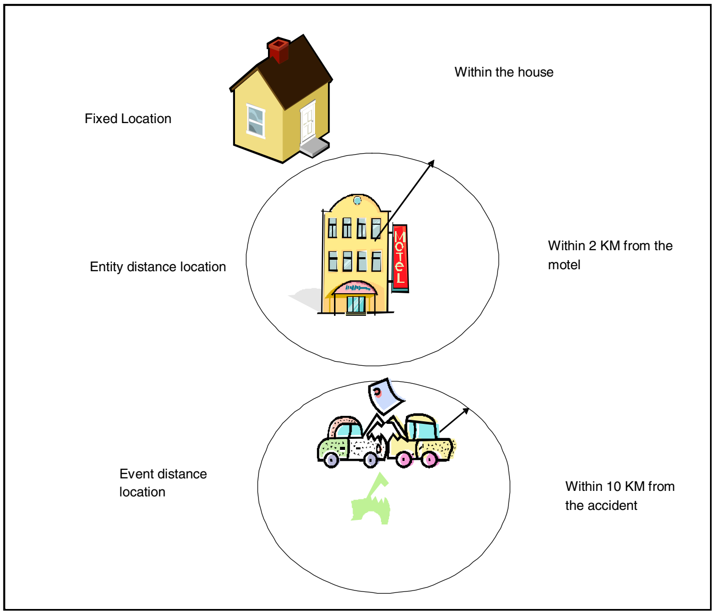


Fig 3. Spatial contexts.

3.1 Fixed Location

A *fixed location* context has one or more context partitions, each of which are associated with the location of a reference entity, sometimes referred to as a *geofence*. An event instance is classified into a context partition if its location's attribute correlates with the spatial entity in some way. Like the event location, the location of the reference entity can be of any of the three types: point, line, area, thus there can be several relations between the event's location and the reference location as shown in Fig.4.

The *contained in* relation is true if the entity location is completely enclosed within the event location.

The *contains* relation is true if the event location is completely enclosed within the entity location.

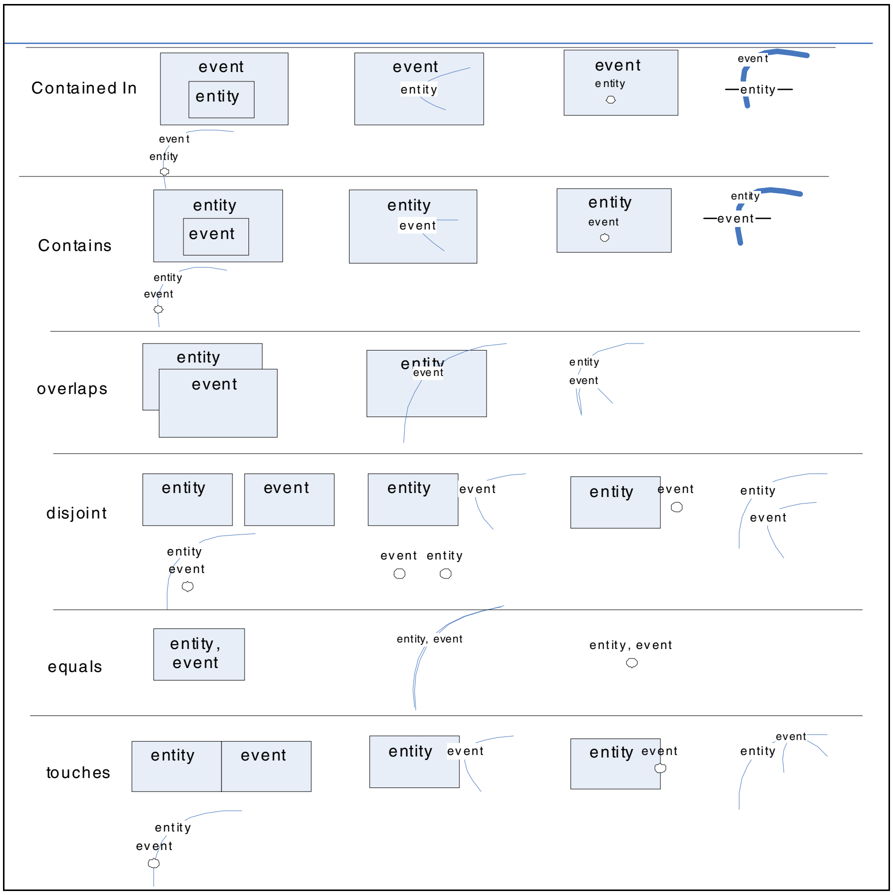


Fig 4. Fixed location relations - taken from [8].

The *overlaps* relation is true if there is some overlap between the entity location and event location, but neither of them is contained within the other.

The *disjoint* relation is true if there is no overlap between the entity location and event location.

The *equals* relation is true if the entity location and event location are identical.

The *touches* relation is true if the event location borders the entity location. Depending on the implementation, some of these relations may have some tolerance, e.g. two points can be considered equal if they are no more than some distance apart.

Not every combination of entity location type and event location type is valid, as seen in Fig. 4, for example: a point cannot be contained in another point, an area cannot be equal to a line, and a point cannot overlap an area.

Here are some examples:

- A car fleet management application follows cars that are driving on a certain highway. The car location, obtained by the GPS, is represented as a point, the highway is represented by a polyline, and the relationship is "contained in".
- A plague is detected within a geographical area that overlaps the borders of a certain city, both the event's location and the entity location are areas, and the relationship is "overlaps."

3.2 Entity Distance Location

An *entity distance location* context gives rise to one or more context partitions, based on the distance between the event's location attribute and some other entity. The word distance refers to the shortest distance between two spatial entities.

This entity may be either stationary or moving. If the entity is moving, the distance relates to the location of the entity at the time that the event occurred (its *occurrence time*). The entity may either be one that is specified by another attribute of the event, or one that is specified in the context definition.

Some examples are as follows:

- The example shown in Fig.3, where the entity is a motel and the context is used to track fire alarms within 2 km of the motel, in order to alert the motel manager of possible danger. This context has a single partition, and both the event location and the entity location are given as points.
- Vehicle breakdown events partitioned according to their distance from a particular service center. They are grouped by distance as follows: less than 10 km, between 10 km and 30 km, between 30 km and 60 km and more than 60 km. In this case the service center is a fixed entity specified in the context definition, and the partition specifications are <10 km; ≥ 10 km and < 30 km; ≥ 30 km and < 60 km; ≥60 km.
- An alerting service at a big conference, which attendees can use to receive alerts when they are in the proximity of another person. Attendees send periodic events giving their own location and these events include the name of the person they are interested in meeting. This example shows the use of an entity distance location context where the entity is

itself moving. The context specification has an entity attribute which tells the context which attribute from the event gives the name of the person being tracked and an explicit partition with a maximum distance of 100m. This means that the alert generation event processing agent is only invoked if the two people are less that 100m apart.

3.3 Event Distance Location

This type of context specifies an event type and a matching expression predicate. A new partition is created if an event occurrence is detected that matches this predicate. Subsequent events are then included in the context partition if they occurred within a specific distance of the initiating event. Distance is defined as in the entity distance location context.

Some examples are as follows:

- Detecting the presence of an aircraft within a 10 km radius of a volcanic ash
- Detecting a case of scarlet fever within a distance of 100km from a previous outbreak of this disease (*event type* is disease report, predicate is disease="scarlet fever", *maximum distance* is 100 km).

4 Segmentation Oriented Context

A *segmentation-oriented* context is used to group event instances into context partitions based on the value of some attribute or collection of attributes in the instances themselves. As a simple example consider an event processing agent that takes a single stream of input events, in which each event contains a customer identifier attribute. The value of this attribute can be used to group events so that there's a separate context partition for each customer. Each context partition only contains events related to that customer, so that the behavior of each customer can be tracked independently of the other customers.

In this kind of segmentation-oriented context, the context specifies just the attribute (or attributes) to be used, and this implicitly defines a context partition for every possible value of that attribute. Alternatively, a segmentation-oriented context definition can list its context partitions explicitly. Each partition specification includes one or more predicate expressions involving one or more of the event attributes; an event is assigned to a context partition if one of its predicates evaluates to true.

Partitions can be specified:

- By attribute value: e.g. there is a context partition for each customer, for all events related to this customer
- By combination of attributes: e.g. all car accidents are grouped by car type and driver license type.
- By explicit predicate: e.g. there is a context partition by an employee's band. One partition for band < 5, one partition for band = 6 or 7, one partition for band 8, one partition for band 9, and one partition for band > 9.

5 State-Oriented Context

State-oriented context differs from the other dimensions in that the context is determined by the state of some entity that is external to the event processing system. This is best illustrated with some examples:

- An airport security system could have a threat level status taking values green, blue, yellow, orange, or red. Some events may need to be monitored only when the threat level is orange or above, while other events may be processed differently in different threat levels.
- Traffic in a certain highway has several status values: traffic flowing, traffic slow, traffic stationary.

Each of these states issues a context partition.

6 Context Composition

Event processing applications often use combinations of two or more of the simple context types that we have discussed so far. In particular a temporal context type is frequently used in combination with a segmentation-oriented context, and we show and example of this in Fig. 5.

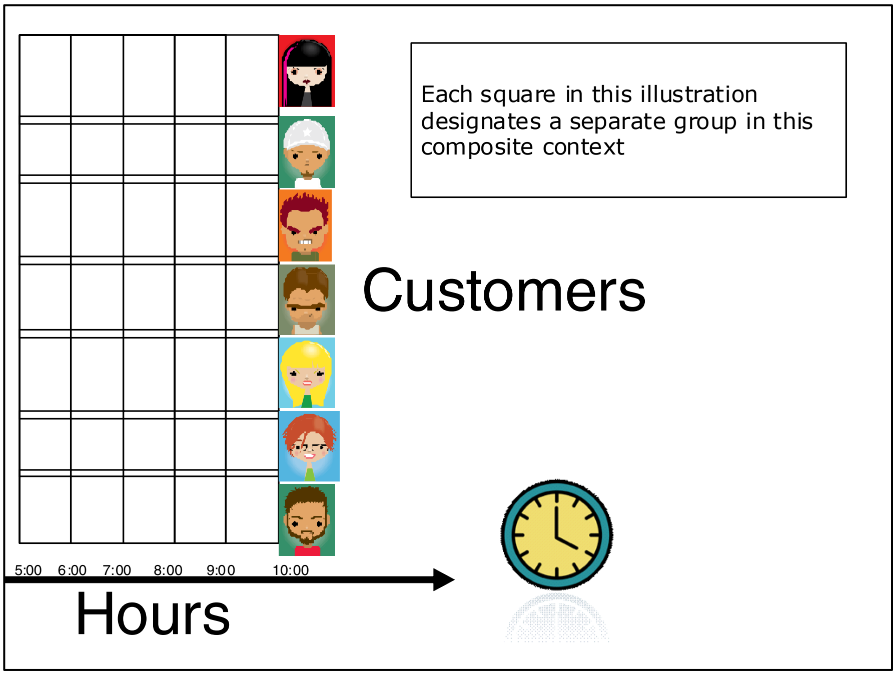

Each square in this illustration designates a separate group in this composite context
Customers
5:00
6:00
7:00
8:00
9:00
10:00
Hours

Fig 5. Composite context.

Fig. 5 is showing composition of segmentation-oriented context and temporal context. Each square is a separate context partition and designates the combination of an hour-long interval and a specific customer.

Composition of context is typically an intersection between two context dimensions, i.e. an event belongs to the composite context if it belongs to each of the member contexts; other possible set operations for composition union and set difference. In these cases typical uses are cases in which the member contexts are of the same context dimension (example: union of temporal intervals, union of spatial fixed areas). It should be noted that the definition of composite context in any case may relate both to member contexts of the same dimension or of different dimension.

These context dimensions generalize the functionality that exists within various products and models of event processing. Next, we move on to context implementation in five event processing products that samples that implementation context in various products.

7 Context Realization within Current Event Processing Products

In the following sections we introduce context support within a sample of existing event processing commercial platforms. We classify this context support according to the four context dimensions defined in Section 4: temporal context, segmentation context, spatial context and state-oriented context. The five languages we survey are: StreamBase StreamSQL, Oracle EPL, Rulecore Reakt, Sybase's CCL (formerly Aleri/Coral8) and IBM's Websphere Business Events.

7.1 Stream Base StreamSQL

StreamSQL [17] is a SQL based Event Processing Language (EPL) focusing on event stream processing, extending the SQL semantics and adding stream-oriented operators as well as windowing constructs for subdividing a potentially infinite stream of data into analyzable segments.

The windowing constructs correspond to the temporal context dimension, where the windows are defined based on time, or based on number of events, or based on an attribute value within an event.

Additional constructs in StreamSQL, such as GroupBy and PartitionBy clauses correspond to the segmentation context dimension.

The temporal context supported is the sliding fixed interval called in StreamSQL **sliding fixed window** and sliding event interval called **sliding event window**. Default evaluation policy for the operators is deferred upon reaching the specified window size, however this can be changed in the operator specifications to immediate or semi-immediate (performing the operation once every number of units even if window size is not reached).

7.1.1 Sliding Fixed Window

A window construct of type "time" supports the notion of sliding fixed interval. New windows are opened at regular intervals relative to each other, as specified in the ADVANCE parameter; the windows remain open for a specified period of time, as stated in the SIZE parameter. Additional parameters further influence the behavior of the window construct, such as OFFSET which indicates a value by which to offset the start of the window.

For example, Listing 1 creates sliding overlapping windows which are opened every second and remain open for 10 seconds

```
CREATE WINDOW tenSecondsInterval(
        SIZE 10 ADVANCE 1 {TIME});
```

Listing 1. StreamSQL sliding fixed window example.

7.1.2 Event Interval and Sliding Event Interval

A window construct of type "tuple" supports the notion of sliding event intervals, where the window size is defined based on the event number, and the repeating intervals are defined in terms of events as well.

```
CREATE WINDOW eventsInterval (
    SIZE 10 ADVANCE 1 {TUPLE} TIMEOUT 20);
```

Listing 2. StreamSQL sliding event window example.

The repetition is optional, a policy determines whether it is necessary to open new windows on event arrival or not, therefore this supports the notion of event interval.

For example, Listing 2 defines a context where a new window is opened upon arrival of a new tuple (event).

The windows are closed either after accumulating 5 events or when a timeout of 20 seconds from window opening expires.

Options exist to use the temporal and segmentation context dimensions in conjunction thus introducing a composite context consisting of multiple dimensions.

7.2 Oracle EPL

The Oracle CEP [27] offering is a Java-based solution targeting event stream processing applications. Oracle EPL supports both the segmentation and temporal context dimensions [18]. The temporal context supports four types of sliding windows: row-based, time-based, batched row and time-based, which correspond to sliding event and sliding fixed interval in this dimension with different initiation policies.

The sliding row-based window supports retention of last N event entries. For example, the clause shown in Listing 3 retains the last five events of the Withdrawal stream, when the oldest event is "pushed out" of the sliding window, providing a form of sliding event interval.

```
SELECT * FROM Withdrawal RETAIN 5
```

Listing 3. Oracle EPL sliding raw-based window example.

Fig. 6 demonstrates the flow of events through the event stream and the content of the window for this statement.

The time based sliding window is a moving window aggregating events for a specified time, implementing the case of sliding fixed temporal interval. For example, the clause shown in Listing 4 defines a window holding a view into an event stream of all events received in the last 4 seconds.

Fig. 7 demonstrates the time-based window behavior over a period of time

An example of a batched time windows is shown in the clause presented in Listing 5.

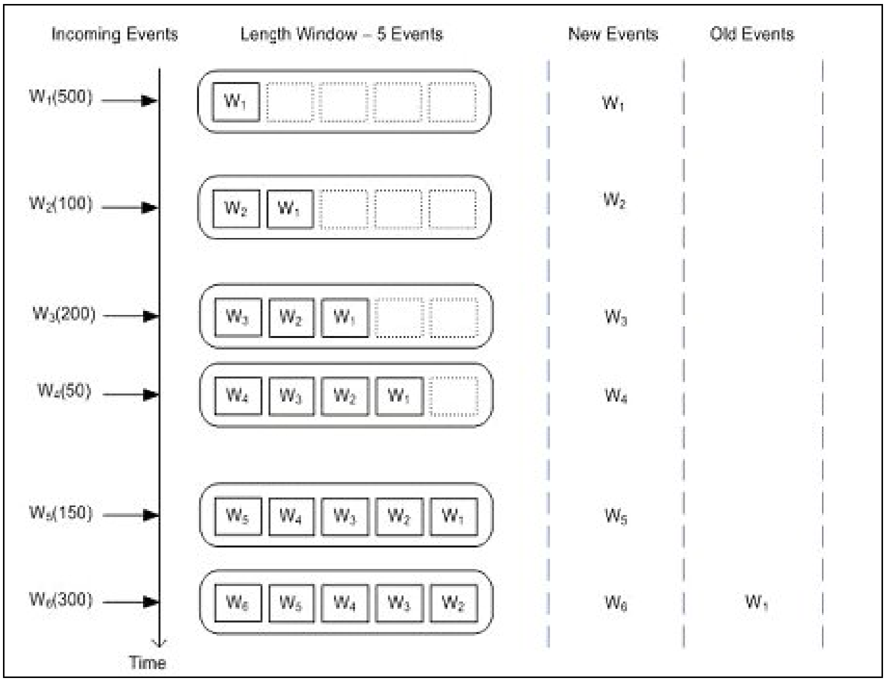


Fig 6. Oracle EPL sliding row-based window retaining 5 events.

```
SELECT * FROM Withdrawal RETAIN 4 SECONDS
```

Listing 4. Oracle EPL sliding time-based window example.

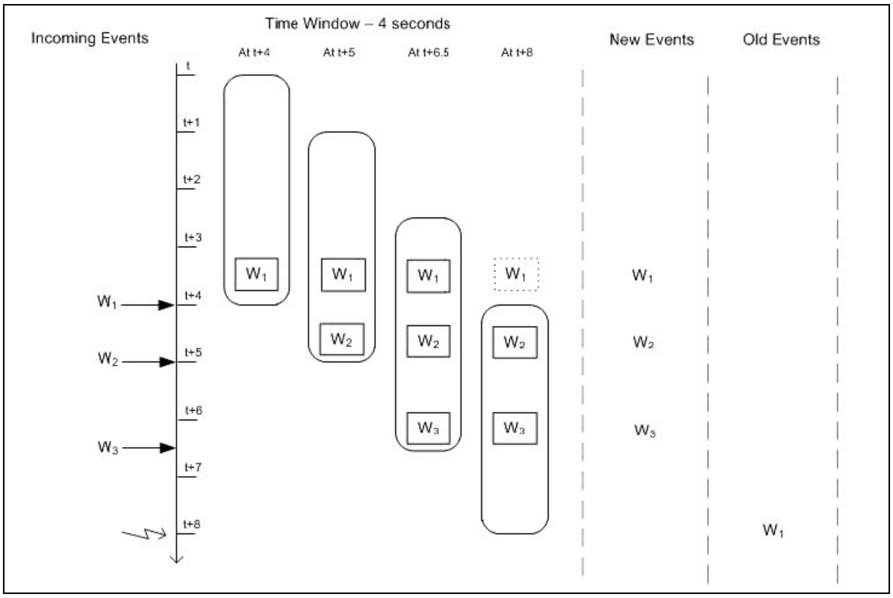


Fig 7. Oracle EPL Sliding time-based window.

```
   SELECT * FROM Withdrawal
RETAIN BATCH OF 4 SECONDS
```

Listing 5. Oracle EPL batched time window example.

The windows are non overlapping windows, and a new window is opened only when its predecessor window is closed. Additionally Oracle EPL supports the GROUP BY clause which enables further segmentation of the event stream. The EPL language supports the composition of the temporal and segmentation context dimensions together thus supporting the notion of composite context. An example of the syntax supporting the composite context is shown in Listing 6.

```
   INSERT INTO TicksPerSecond
   SELECT feed, COUNT(*) AS cnt FROM
MarketDataEvent
   RETAIN BATCH OF 1 SECOND
   GROUP BY feed
```

Listing 6. Oracle EPL composite context example.

The composite context that is shown in Listing 6 is composed of a temporal context of a sliding window of 1 second, and a segmentation context by feed.

7.3 RuleCore Reakt

RuleCore [20] executes ECA rules defined using the Reakt language. The main language element in Reakt is the rule. RuleCore uses a notion of "views" as execution context for the rule. Each rule instance maintains its own virtual view into the incoming event stream. The view provides a window into the incoming event stream providing context for rule evaluation. Reakt supports segmentation, temporal, spatial and state contexts.

The temporal context is implemented using the MaxAge view property which defines the maximum age for each event in the view.

Example: Listing 7 shows a definition that limits every rule instance to view only events from the past 10 minutes.

```
<MaxAge>00:10:00</MaxAge>
```

Listing 7. RuleCore Reakt fixed window example.

Segmentation context is implemented using the [?] match view property which is very flexible and can select values from the event bodies using the full expressive power of XPath. This allows for powerful semantic matching even of complex event structures. An example is shown in Listing 8.

```
    <Match>
      <Value>
<Event><base:XPath>EventDef[eventType="Warning]</bas
e:XPath></Even>
    <Field><base:XPath xmlns:base="base">base:
ServerIP base:XPath></Field>
      </Value>
    <Match>
```

Listing 8. RuleCore Reakt match view.

Spatial context is implemented using the Type view property by defining a new geographic zone entity type defined by its coordinates and Match view property. An event is then associated with a specific entity which may have a set of properties such as position or a Zone. Spatial Evaluation is done by applying Match property on specific Entity type with geographic location and to specific Zone that specifies the reference geo area, all events of specific entity type and which match Zone property will be classified to this specific context partition.

Example: Listing 9 illustrates the association of specific events that are contained in a location Zone. For this rule we define ZoneEntry view, which will contain events from "vehicle" entity type (Match/vehicle) and only from specific zone location (Match/Zone).

```
<ViewDef name="ZoneEntry">
<Propertie><!– All events in the view must be of types InsideZone or OutsideZone –>
  <Type><Event><XPath>EventDef[@eventType="InsideZone"]</XPath></Event>
   <Event><XPath>EventDef[@eventType="OutsideZone"]</XPath> </Event>
</Type>
  <!– All events in the view must be from the same vehicle –>
  <Match name="vehicle">
   <Value><Event><XPath>EventDef[@eventType="OutsideZone"]</XPath></Event>
    <Property name="Vehicle"/></Value>
   <Value> <Event> <XPath>EventDef[@eventType="InsideZone"]</XPath></Event>
 <Property name="Vehicle"/> </Value>
  </Match><!– All events in the view must be from the same zone –>
  <Match name="zone">
   <Value> <Event> <XPath>EventDef[@eventType="InsideZone"]</XPath> </Event>
    <Property name="Zone"/> </Value>
   <Value> <Event><XPath>EventDef[@eventType="OutsideZone"]</XPath> </Event>
    <Property name="Zone"/> </Value>
  </Match>
 </Properties>
</ViewDef>
```

Listing 9. RuleCore Reakt - spatial context example.

State contexts are implemented in a similar way to the way in which spatial contexts are implemented, using Type view property where each event entity has specific properties. State evaluation is done by applying Match property on specific Entity type property, all events of specific entity type and which match specific property will be in the context.

The sliding event-based MaxCount view supports retention of last N event entries, for example the statement guarantees that an event view never contains more than 100 events.

```
<MaxCount>100</MaxCount>
```

Listing 10. RuleCore Reakt - sliding event interval example**.**

The Reakt language allows the composition of the temporal, segmentation, spatial and state context dimensions together thus supporting the notion of composite context.

7.4 Sybase Aleri -Coral8 and CCL

The Sybase Aleri CEP offering consists of Aleri Streaming Platform and Coral8 engine. Coral8's authoring language[22] [23] CCL (Continuous Computation Language) is a SQL-based event processing language extending SQL in a different way relative to the Streambase and Oracle examples presented previously. CCL supports different combinations of window types: sliding and jumping windows, either row-based (event-based) or time-based.

Examples include:

- Time intervals ("keep one hour's worth of data").
- Row counts ("keep the last 1000 rows").
- Value groups ("keep the last row for each stock symbol").

Relative to the context dimension classification defined in section 4, CCL supports both the temporal and segmentation context dimensions. In the temporal dimension, the sliding fixed interval, sliding event interval and event interval are supported. The difference in CCL terminology between sliding and jumping windows is the way they treat the interval periods and the window initiation policies.

```
   INSERT INTO MaxPrice
SELECT Trades.Symbol, MAX(Trades.Price)
FROM Trades KEEP 15 MINUTES
WHERE Trades.Symbol = 'IBM';
```

Listing 11. Sybase CCL - Sliding fixed interval example.

Listing 11 illustrates a sliding fixed interval representing a 15 minute window into the Trades event stream, and the aggregation is determining the highest trade in the last 15 minutes. The interval period is once every minute, and the interval duration is 15 minutes. The initiation policy is "refresh" – the previous window is closed once the new one is created.

A similar example illustrating the sliding event interval is deducing the highest price in the last 15 trades instead of last 15 minutes. This example is shown in Listing 12.

```
   INSERT INTO MaxPrice
SELECT Trades.Symbol, MAX(Trades.Price)
FROM Trades KEEP 15 ROWS
WHERE Trades.Symbol = 'IBM';
```

Listing 12. Sybase CCL - Sliding event interval.

The window can be also defined implicitly in pattern matching queries, such as the following sliding event interval definition, in Listing 13, describing a period which starts with each event A arrival, and ends 10 seconds later; during this period event A should occur, followed by event B, and event C should not occur between the time in which B occurs and the end of the temporal window.

```
   [10 SECONDS: A, B, !C]
```

Listing 13. Sybase CCL - pattern embedded context.

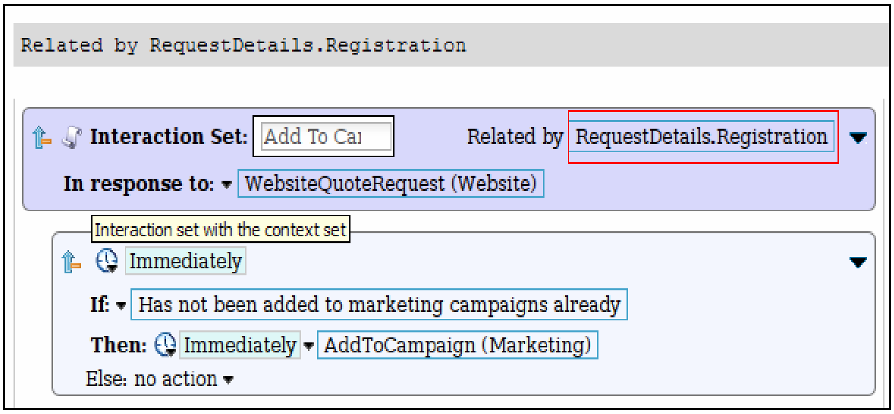
The GROUP BY clause supports the segmentation context implementation.
A combination of the temporal and segmentation context enables the expression of composite contexts. In the example shown in Listing 14, the sliding fixed interval window is further segmented according the stock's symbol. This query determines the maximum price for each stock type in the last 15 minutes.

```
   INSERT INTO MaxPrice
SELECT Trades.Symbol, MAX(Trades.Price)
FROM Trades KEEP 15 MINUTES
GROUP BY Trades.Symbol;
```

Listing 14. Sybase CCL - composite context example.

7.5 IBM Websphere Business Events

IBM WBE [21] is a business event processing system, focusing on the business user and his needs. A segmentation context is easily expressed in WBE using the "Related By" construct, shown in Fig. 8, which supports the segmentation group of events based on event field value or a composition of field values [19].


Related by RequestDetails.Registration

Interaction Set: Add To Ca Related by RequestDetails.Registration

In response to: WebsiteQuoteRequest (Website)

Interaction set with the context set

Immediately

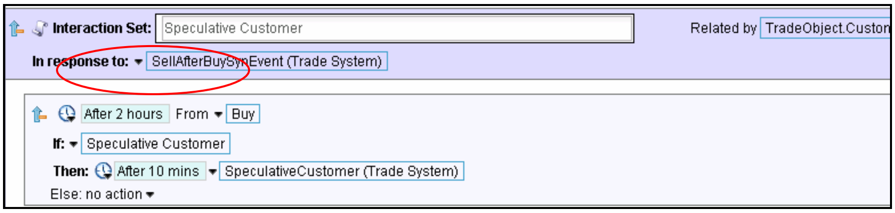
If: Has not been added to marketing campaigns already

Then: Immediately AddToCampaign (Marketing)

Else: no action

Fig 8. WBE definition of segmentation context.

Fig. 8 shows an example in which the events are related to each other by the same Registration, making it a segmentation context. The temporal context capability is supported using deferred evaluation mode (Fig. 9). Events can be evaluated using time delays from the triggering event. These time delays can be for a fixed amount of time or until a certain amount of time has elapsed after another event has occurred, or until a specific date.


Interaction Set: Speculative Customer Related by TradeObject.Custon

In response to: SellAfterBuySynEvent (Trade System)

After 2 hours From Buy

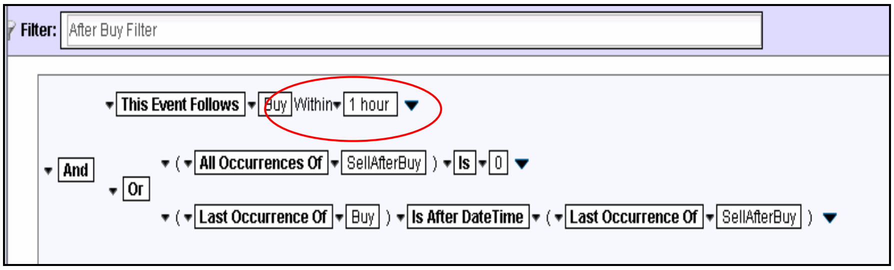
If: Speculative Customer

Then: After 10 mins SpeculativeCustomer (Trade System)

Else: no action

Fig 9. WBE definition of temporal context.

WBE also supports temporal event evaluation within context applying filtering conditions to the rule. This is done using temporal functions such as: **Follows After, Follows before, and Follows Within**. These functions determine if the specified event in the associated scope will follow the event or action defined in the filter after/before/within certain amount of time. Functions such as**: Is Present** before, **Is Present After, Is Present Within**, determine if the event specified in the associated scope has previously occurred in the same context after/before/within the date/time period defined in the filter. An example of the Follows By construct is shown in Fig. 10.


Filter: After Buy Filter

This Event Follows Buy Within 1 hour

And

(All Occurrences Of SellAfterBuy) Is 0

Or

(Last Occurrence Of Buy) Is After DateTime (Last Occurrence Of SellAfterBuy)

Fig 10. WBE Follows By construct.

To conclude, most CEP offerings today support one or another variation on temporal and segmentation context dimensions. While in most cases state-oriented context can be implemented in those solutions indirectly, using event data and filtering on this data, out-of-the-box geospatial capabilities are lacking in most of the mentioned products.

8 Conclusion

This chapter has discussed various aspects of contexts in event processing. We have presented the general notion of context, a general view of context in event processing, and a sample of implementations of this notion within various products. As can be seen, most of the current languages has many of the context functions, but part of all these functions are not implemented as separate abstraction, but as part of other abstractions (like queries); the benefit of using context as a distinct abstraction may provide separation of concerns, and higher flexibility, and is consistent with current trends in software engineering; it has a trade-off of introducing a new kind of abstraction. We observe that context as an abstraction is also emerging in various applications areas: wireless networks [1], location based services [6], electronic tourist guidance [7], [15], Web 2.0 [16], Web presence [10], Business process management and design [13], [14], healthcare monitoring [12], telecommunication [25] and payment systems [26]. These works on context are expected to merge with the work existing in event processing and; Furthermore, We believe that the next generation of event processing as well as computing will view context as a separate abstraction. The experience gained in applying context in event processing in event processing will assist with the standardization of the semantics of context for the various usages of context.

References

1. Abowd, G.D., Atkeson, C.G., Hong, J., Long, S., Kooper, R., Pinkerton, M.: Cyberguide: A mobile context-aware tour guide. Wireless Networks: special issue on mobile computing and networking: selected papers from MobiCom 3(5), 421–433 (1997)

2. Adi, A., Biger, A., Botzer, D., Etzion, O., Sommer, Z.: Context Awareness in Amit. In: 5th Annual International Workshop on Active Middleware Services (AMS 2003), pp. 160–167 (2003)
3. Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Nishizawa, I., Rosenstein, J., Widom, J.: STREAM: The Stanford Stream Data Manager. In: SIGMOD Conference 2003, p. 666 (2003)
4. Buvac, S., Mason, I.A.: Propositional logic of context. In: Proc. of the 11th National Conference on Artificial Intelligence (1993)
5. Buvac, S.: Quantificational Logic of Context. In: Proc. of the 13th National Conference on Artificial Intelligence (1996)
6. Broadbent, J., Marti, P.: Location-Aware Mobile Interactive Guides: Usability Issues. In: Proc. Inter. Cultural Heritage Informatics Meeting, Paris, France (1997)
7. Heverst, K., Davies, N., Mitchell, K., Friday, A., Efstratiou, C.: Developing a Context-Aware Electronic Tourist Guide: some issues and experiences. In: Proc. CHI, The Hague, Netherlands (2000)
8. Etzion, O., Niblett, P.: Event Processing in Action. Manning Publications (2010)
9. Jensen, C.S., et al.: The Consensus Glossary of Temporal Database Concepts - February 1998 Version. In: Etzion, O., Jajodia, S., Sripada, S. (eds.) Temporal Databases Research and Practice, pp. 367–405. Springer, Heidelberg (1998)
10. Kindberg, T., et al.: People, Places, Things: Web Presence for the Real World. In: Proc. 3rd Annual Wireless and Mobile Computer Systems and Applications, Monterey, CA (2000)
11. Luciano, S., Fausto, G.: ML Systems: A Proof Theory for Contexts. Journal of Logic, Language and Information 11, 471–518 (2002)
12. Mohomed, I., Misra, A., Ebling, M., Jerome, W.: Context-aware and personalized event filtering for low-overhead continuous remote health monitoring. In: International Symposium on a World of Wireless, Mobile and Multimedia Networks (2008)
13. Rosemann, M., Recker, J., Flender, C., Ansell, P.: Context-Awareness in Business Process Design. In: 17th Australasian Conference on Information Systems, Adelaide, Australia (2006)
14. Rosemann, M., Recker, J., Flender, C.: Contextualization of Business Processes. International Journal of Business Process Integration and Management 3, 47–60 (2008)
15. Woodruff, A., Aoki, P.M., Hurst, A., Szymanski, M.H.: Electronic Guidebooks and Visitor Attention. In: Proc. 6th Int. Cultural Heritage Informatics Meeting, Milan, Italy, pp. 437–454 (2001)
16. Context in Web 2.0 realm, http://blog.roundarch.com/2010/03/25/in-the-realm-of-web-2-0-context-is-king-part-one/
17. StreamSQL-Streambase EP language, http://dev.streambase.com/developers/docs/sb37/streamsql/index.html
18. Oracle EPL-Oracle EP language, http://download.oracle.com/docs/cd/E13157_01/wlevs/docs30/epl_guide/overview.html
19. IBM Websphere Business Events Context configuration, http://publib.boulder.ibm.com/infocenter/wbevents/v7r0m0/index.jsp?topic=/com.ibm.wbe.install.doc/doc/configuringtheruntime.html
20. RuleCore CEP Server, http://rulecore.com/content/view/35/52/
21. IBM Websphere Business Events Overview, http://www01.ibm.com/software/integration/wbe/features/?S_CMP=wspace

22. Sybase-Aleri Continuous Computation Language (CCL), http://www.aleri.com/products/aleri-cep/coral8-engine/ccl
23. Sybase-Aleri CCL context definition, http://www.aleri.com/WebHelp/programming/programmers/c8pg_using_windows.html
24. Context Delivery Architecture by Gartner, http://www.gartner.com/DisplayDocument?id=535313
25. Context aware mobile communications system, http://www.geovector.com/
26. Context in mobile payment systems, http://www.nttdocomo.com/
27. Oracle CEP overview, http://www.oracle.com/technologies/soa/complex-event-processing.html