The Dynamics of Notification Requests in Moving Objects Databases

Goce Trajcevski, Peter Scheuermann
Department of ECE
Northwestern University
Evanston, Il 60208
{goce.peters}@ece.northwestern.edu

Héve Brönnimann
Department of CSI
Polytechnic University
Brooklyn, NY
hbr@poly.edu

Agnès Voisard
Institut für Informatik
Freie Universität
Berlin, Germany
voisard@inf.fu-berlin.de

Technical Report TR-25-04
Department of Electrical and Computer Engineering
Northwestern University

Abstract

This work addresses a practically relevant aspect of the problem of managing the reactive behavior of the Moving Objects Databases (MOD). Besides managing the transient (location, time) information about the mobile entities, MOD are also intended of handling users’ requests and we consider requests which are continuous (i.e., their evaluation spans over a time-interval). In such setting, a modification in the MOD (reflecting the changes in the world being modeled) may require re-evaluation of certain requests. Typically, one would think of the traditional ECA (Event-Condition-Action) rules from Active Databases as a paradigm for specifying the behavior which would properly react to the MOD modification and correctly update the requests posed by the MOD users. However, as we will demonstrate, the traditional ECA model is not suitable for dynamic spatio-temporal environments in which different context dimensions may be correlated and have impact on the satisfiability of a given requests.

In this work we propose the (ECA)² – Evolving and Context-Aware Event-Condition-Action paradigm as a tool for specifying the reactive behavior in MOD which manages continuous requests from its users.

1 Introduction and Motivation

Advances in wireless communication and the miniaturization of pervasive computing devices have spurred new classes of applications, like dynamic service discovery and mobile tour guides and on-time information delivery [11, 19, 43, 46, 49, 55], in which an important aspect is context awareness. First introduced in [63], and then refined in the past few years [1], context is basically any information pertaining to a given situation, which can be used to characterize (possibly mobile) entities. For example, the GUIDE project [18, 19], which guides city visitors equipped with a hand-held device, distinguishes between two classes of context – personal: interests, location, refreshment preferences, attractions visited, and environmental: time of day, opening time of attraction, etc.
The same technological advances motivated several important classes of applications in which the mobility of the users is an important aspect. Examples include: transportation and traffic control, emergency response, dynamic resource management, mobile electronic commerce, digital battlefield, and so on (see, for instance, [9, 16, 39, 41, 77]. An important enabling technology for these applications is location management [60, 61, 64], which is, the management of the transient location information of the objects involved and an important benefit of having a system for managing the location data for a large number of mobile users – often referred to as Moving Objects Database (MOD) [45] – is the ability to process various requests (queries and/or notifications) pertaining to the whereabouts-in-time of the entities involved. Request are, typically, of two (executional) kinds. In a pull mode, an end user explicitly asks for information stored in the data sources, such as the nearest gas station. In a push mode, the system sends relevant information to the user – according to his profile, context, and so on – without the user asking explicitly for it (note that the user may have subscribed beforehand to a type of information of interest). In this case information filtering and delivery is handled by alerting systems or Event Notification Systems (ENS).

A typical application is that of a car driving assistant which needs to adapt to dynamic user information demand. The dynamicity may be incurred either by a dynamic behavior of the user (the car driver) of by a changing environment. Consider, for example, someone driving to a given destination (which can be fixed by the user or defined by its type, e.g., a restaurant). This person needs to be notified of a restaurant that can be of interest, but also of driving conditions like traffic jam of heavy rain (see, for instance, the IN:SIGHT system [51]). This person has a particular profile that encompasses personal information, such as favorite type of food, budget, etc, which constitute the static user demand. On the other hand, associated with this person is a dynamic information demand as a function of his location and dynamic environmental condition.

A dynamic demand is an up-to-date information regarding: (i) user interests/preference (e.g., a type of restaurant), and (ii) external information coming from various data sources (e.g., current traffic conditions). It is evaluated regularly by taking the situation of the user into account. In [50] the situation of a user is a level of abstraction built on top of a user context (which, itself is built on top of sensors). The situation of a user is defined along many dimensions and allows one to derive that someone is "in a car", "on the phone", "at work", and so on. The IN:SIGHT system finds out which information may suit a current user situation and evaluates when to deliver the information, based on the notion of information value at a given time. The system needs to estimate a situation and match it against the anticipated situations of the user (e.g., "heavy rain" vs. "no rain"). This simple example first illustrates that reacting on only one change of parameter is not enough and second, that systems that combine event notification systems and situation-based services are needed.

Unlike the traditional database applications where the queries are instantaneous, in MOD settings, most of the queries of interest are continuous [66], which is, their answer-set may vary over time. Consider, for example:

**Q1:** *Retrieve the objects which will be inside the region R1 sometime between 2:00PM and 2:30PM.* Due to the changes in the *(location, time)* information, the answer-set of Q1 may have to be re-evaluated. In the settings in which the mobility of the objects is modeled as a sequence of GPS-like updates of the *(location, time)* values, where only the past can be somehow interpolated and nothing is known about the future, the system has to react to the newly arrived updates and detect their impact on Q1 [53]. Even if the motion is represented in a manner which makes its future portion known, i.e., a trajectory is constructed based on electronic maps and traffic distribution-patterns, the system will have to react to the abnormalities in the parameters used in the trajectories’ construction (e.g., a road accident) [69].

These issues have been addressed only recently and there is no unified formalism to characterize and reason about the reactive behavior in such settings. The traditional Event-Condition-Action ECA of the Active Databases field [79] has, to an extent, been applied for managing continuous range queries [69], however, it may not be quite suitable for MOD settings. As a particular example, assuming that the objects’
motions are represented as sequences of (location, time) points, it is questionable how to manage efficiently the query like:

**Q2:** *Retrieve all the objects that move continuously towards the region R2 for more than 5 minutes, between 5:00 and 5:30" In general, the ECA paradigm is not quite well suited for managing in reactive manner persistent (c.f. [66]) queries — the ones whose condition evolves over time and may involve dynamic entities with heterogeneous semantics (nature).

This is precisely the main goal of our work. We propose the Evolving and Context-Aware Event-Condition-Action (ECA)$^2$ paradigm as a tool which enables the users to specify how the system should monitor its evolution and properly react in desired situations. Traditionally, the concept of a Context means any information that can be used to characterize the entities of interest [1] and Context-Awareness of a system means incorporating the informations from various categories (e.g., environmental, personal, etc.). It has been exploited in many Event Notification Systems (ENS) [55] where, for example, the users information of interest has to be delivered based on his location and preferences. In this work, we take the concept of Context-Awareness to yet another level — the system itself. Besides expressing the preferences about the reactive behavior with respect to the values of the entities that are modeled by the MOD, the (ECA)$^2$ paradigm allows the user to specify what is to be monitored for the purpose of optimizing the reactive behavior of the system (e.g., the response time).

In comparison to the classical ECA paradigm, the main features of the (ECA)$^2$ paradigm can be outlined as follows:

<table>
<thead>
<tr>
<th>ECA</th>
<th>(ECA)$^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON EVENT</td>
<td>ON EVENT</td>
</tr>
<tr>
<td>IF CONDITION</td>
<td>IF CONDITION</td>
</tr>
<tr>
<td>THEN ACTION</td>
<td>THEN ACTION</td>
</tr>
<tr>
<td>ELSE</td>
<td>&lt;consume_parent&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;MODIFY EVENT/CONDITION/ACTION&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;NEW EVENT/CONDITION/ACTION&gt;</td>
</tr>
</tbody>
</table>

Basically, when an event of interest is detected, in the ECA paradigm nothing is done if the condition part evaluates to false. On the other hand, in the (ECA)$^2$ paradigm, the user is allowed to specify whether he wants to modify, say, the condition for the situations in which subsequent occurrences of the event of interest occur. Similarly, he can express the desired modifications of the event whose occurrence is to be subsequently monitored, and the action part. Moreover, the user may specify that at a given state, a new trigger is to be “forked” which can either consume its “parent” trigger (consume parent = yes) or, continue to exist with it. Such behavior is of interest for reactive management of requests like:

**Q3:** *“Notify when the IBM stock increases by 20% within a 20 minutes interval IF (its value reaches 2345 AND the COMPAQ stock increased by at least 5% within 5 minute of that interval). Subsequently, also notify me when IBM stock reaches the value of 2380 IF at that time the COPMAQ stock is larger than 2200”.*

The main contributions of this work are as follows:

1. We introduce two novel predicates which are of practical importance and we present efficient algorithms for their processing. Subsequently, we address the problem of efficient management of the predicates in the MOD settings.

2. Based on the observations in 1, we introduce a new paradigm for expressing the reactive behavior in MOD which manage various (pending) requests from the users. The MOD is aware of the correlation of values in the relevant context dimensions and their impact on the requests, which may cause a dynamic modification of the conditions and events being monitored. Hence, we call this paradigm Evolving and Context-Aware Event-Condition-Action — (ECA)$^2$. 

3
3. We introduce a language CAR for specifying the the reactive behavior (active rules) in a database settings under the (ECA)² paradigm. We describe the basic linguistic constructs of the CAR and its declarative semantics.

4. We introduce the concept of Meta-Triggers and describe their role in the implementation of the \((ECA)^2\) paradigm.

The rest of this paper is organized as follows. In Section 2 we give a preliminary background. Section 3 presents the two new predicates and the algorithms for their processing. It also investigates the details involved in their processing from the perspective of an efficient reactive behavior in MOD setting, which we use to motivate the \((ECA)^2\) paradigm. Section 4 gives another example of a reactive management of users requests, and brings to light some more details of the behavioral aspects of the \((ECA)^2\) paradigm. In Section 5 we give a formal description of the syntactic elements for specifying a system which implements the evolving and context-aware reactive behavior, and we investigate its declarative semantics. Section 6 presents the concluding remarks, positions the paper with respect to the existing literature and outlines directions for future work.

2 Preliminaries

The representation of the object’s motion in MOD is based on some model of the transient \((\text{location}, \text{time})\) information. Different works, based on their respective application domains, have adopted different models and the typical assumption is that each moving object is equipped with some minimal processing power, e.g., an on-board GPS to detect its location and the ability to transmit it, along with transmitting requests and receiving answers to/from the MOD.

![Figure 1: Modeling and representing the mobility of the objects](image-url)

Three models which are, in a sense, the “attractor-centers along the spectra” are illustrated in Figure 1. The characteristics of each model are as follows:

1. The object is assumed to (periodically) send updates of its location using, for example, its on-board GPS. The characteristic of this model is that the future of the object’s motion is unknown and, as a consequence, one cannot pose spatio-temporal queries pertaining to the future. The past motion of the object is represented as a sequence of \(3D\) (2D geography + time) points of the form \((x_i, y_i, t_i)\) with the intended meaning that the object was at the location with the coordinates \((x_i, y_i)\) at the
time-instant $t_i$. Typically, between two points the object is assumed to move along a straight line and with a constant speed which implies that its motion is represented as a polyline in the 3D space.

2. Instead of only sending its location, the object also transmits the information about its velocity, whenever it changes. This is represented as a dynamic attribute (c.f. [66]) in the MOD. In this model, a “near future” is assumed known and various queries can be posed pertaining to some future time (interval) of interest. However, upon update of the dynamic attributes representing the object’s motion plan, the answers of the pending queries may need to be re-evaluated. Similarly to the previous case, the past motion is modeled as a 3D polyline, represented as a sequence of 3D points.

3. As the “other extreme”, given the object’s initial location and a set of the points that one intends to visit, by using an electronic map augmented with some information about the variations of the traffic patterns, one could construct the entire future trajectory of the object’s motion plan (c.f. [71]). The MOD model is again a polyline except now one can pose queries pertaining to the future of the objects in the MOD. Clearly, in case there are some unforeseen (traffic) perturbation which cause the change of the traffic patterns information used in the construction of the trajectories, the trajectories that are affected by those perturbations will have to be identified and their respective future portions will have to be updated in the MOD [72]. This, in turn, will cause the re-evaluation of the pending queries affected by the trajectories’ updates [70, 69].

Regardless of the model adopted, one distinct feature of the MOD is that, unlike the traditional database applications where the queries are instantaneous, most of the queries of interest to the MOD are continuous, which is, they span over a time-interval\(^1\). As we indicated, due to the dynamics of the entities involved, the answers to continuous queries change over time.

We assume that a mobile or a static user can pose requests to the MOD and we distinguish between two basic types:

- **Request for Notification (RN)** where the user wants to be simply notified when a certain condition (predicate) of interest is satisfied in the current state of the MOD (possibly combined with the occurrence of a relevant event). An example is:

  **RN1**: “Notify me when I am within 3 miles from a motel, between 7:00PM and 9:00PM”.

  Clearly, due to the mobility of the object, the desired condition (within 3 miles from a motel) needs to be continuously monitored, and several notifications may be sent, during the time-interval of interest ([7:00PM,9:00PM]).

- **Request for a Query (RQ)** where the user is interested in the identity of the objects which satisfy a certain condition (predicate) of interest. An example is:

  **RQ1**: “Retrieve the objects which will be inside the region $R$ for longer than 10 minutes, between 8:00AM and 10:00AM”.

  Again, due to the changes of the (location, time) information in the MOD, the condition (inside the region $R$) will have to be monitored during the entire time-interval of interest ([8:00AM,10:00AM]), as the objects can move in-and-out the region $R$.

A particular request may also be a combination of the two basic types, like in a typical example of the context-aware tourist information systems [43, 55]:

- **Rqnt**: “Notify me when I am within 0.5 miles from a landmark and send me its name, location and historic data, between noon and 3:00PM”.

  Each mobile user has a Personal class of context which consists of Location, Time, Motion Plan, Mobility, Device and Preferences. We will also assume that there are some Environmental classes of context such as ,

\(^1\) Another type of queries relevant to the MOD settings are the persistent ones, formally specified in [66], which we do not treat explicitly in this work.
for example Traffic Conditions and Special Events (e.g., Opera, Sports Game). Throughout the rest of this work, we will concentrate on scenarios in which (the value of) the Motion Plan context dimension in the MOD is obtained via a sequence of GPS-like generated (location,time) updates and the object is moving along a straight line and with a constant speed between two update points. We will focus on issues which pertain to managing Requests for Notification in a push-mode, which is typically handled by event notification systems.

Typically, an event denotes an occurrence of something of interest, e.g., a particular predicate becoming satisfied or a detection of an occurrence of external stimuli (being registered in the system). We assume that there is a distinct set of primitive events, which can be internal – corresponding to the execution of an elementary database operation (update, insert, delete), or external – corresponding to detection of a particular entity by an application program, for which there exists an interface with the DBMS [3, 14] which, subsequently, manages it. Another (simpler) way of formulating the definition of primitive events is that they are the ones that are pre-defined by the system [15, 42]. The occurrence of a primitive event is assumed instantaneous (e.g., is happens in a distinct time-value) and it is registered with the system until explicitly deleted.

We also assume that there is an event algebra [15, 29, 42, 54] which enables specifications of composite events using the primitive ones like, for example:

- \( E = e_1 \land e_2 \) – (conjunction) which is detected when both \( e_1 \) and \( e_2 \) have occurred, regardless of their order of occurrence/detection.
- \( E = e_1 \lor e_2 \) – (disjunction) which is detected whenever \( e_1 \) or \( e_2 \) has occurred.
- \( E = e_1 \rightarrow e_2 \) – (sequence) which is detected when \( e_2 \) occurs, provided that \( e_1 \) has also occurred.
- \( E = \neg (e_1, e_2, e_3) \) – (non-occurrence/negation) which is detected whenever there is an interval bounded by the occurrences of \( e_1 \) and \( e_3 \), during which \( e_2 \) has not occurred.
- \( E = (e_1, e_2, e_3) \) – (occurrence) which is detected whenever there is an interval bounded by the occurrences of \( e_1 \) and \( e_3 \), provided that \( e_2 \) has occurred inside that interval. One may be interested in the cumulative versions: \( E = (e_1, e_2^+, e_3) \) (resp. \( E = (e_1, e_2^-, e_3) \)) which detects \( E \) whenever there is one or more (resp. zero or more) occurrences of \( e_2 \) within the interval bounded by \( e_1 \) and \( e_3 \).

Given a particular event algebra, we also assume that there is an underlying mechanism for detecting the occurrence of the (specified) composite event like, for example, Petri Nets [29], or Event Graphs [15]. Similarly to the EECA paradigm in [28], we assume that there is an available Event-Base (EB) which manages both the primitive and composite events: \( EB = \langle PE; CE \rangle \).

We also allow an existence of deductive (inference) rules for the standard facts (e.g., tuples). Thus, the database consists of an Extensional (EDB) part and an Intensional (IDB) part, which is, \( DB = \langle EDB, IDB \rangle \). We assume that the deductive rules are of the form Head \( \rightarrow \) Body and the standard criteria for the safety of the variables in negated predicates and the fixpoint of the computations for the intensional predicates apply [74].

Throughout the rest of the paper, we will not formally define many of the semantic dimensions of the ECA paradigm in the standard database settings (e.g., coupling modes) and we refer the reader to the existing literature [28, 57, 79].

3 Dynamics of the Topological Predicates and Reactive Behavior

In this section we address the issue of the dynamics of the reactive behavior in MOD systems which manage Requests for Notifications. First, we introduce two predicates and for each of them we identify the issues relevant for their processing from two perspectives: operational (or, algorithmic) efficiency and behavioral, in the sense of efficient and correct reactive behavior of a MOD. We use these examples to motivate the \((ECA)^2\) paradigm and introduce its basic concepts in an order of increasing generality.
3.1 The Moving-Along Predicate

First we consider a request pertaining to the moving-along predicate in spatio-temporal environments.

In a purely spatial context, the “alongness” property has already been investigated. As a topological concept in GIS setting, [26, 44] present the 9-intersection model which gives the conditions for detecting the satisfiability of the along predicate for (the boundaries) of two 2D objects, invariant to transformations like translation, scaling and rotation of the objects. An approach which is more geared towards processing of such predicates in the context of Spatial Databases can be found in [37]. In Spatio-Temporal Databases [36, 47] have addressed the satisfiability of certain spatio-temporal predicates over a given time-interval, during which the objects move (or, evolve). However, as we will demonstrate shortly, the existing approaches are not well-suited for applications which require processing of certain RNs.

Observe that when it comes to “alongness” in mobile environments, in reality one cannot expect that a mobile user, say, driving a car, can move exactly along a river. Thus, we introduce a distance threshold \( d \) with its intuitive meaning that for as long as the object is within distance \( d \) from a given 2D polyline \( P \), we will assume that it is moving “along” \( P \). Moreover, one may be interested if the predicate is satisfied within a portion \( \Delta t \) of the time-interval of interest \([t_1, t_2]\). As a particular example, consider the following notification request:

**RN2**: “Notify me when the object \( o_{b1} \) is moving along the polyline \( P \) and within distance \( d \) less than 90% of the time between 5:00 and 5:30”.

![Figure 2: Notification for Spatio-temporal Moving-Along Predicate](image)

An equivalent statement of the request is:

**RN2'**: “Notify me when the object \( o_{b1} \) is not moving along the polyline \( P \) and within distance \( d \) for more than 10% of the time between 5:00 and 5:30”.

\(^2\)Observe that if one insists on the exact moving along property, then \( d = 0 \).
An illustrating scenario is shown in Figure 2. Each circle indicates a \((location, time)\) update sent to the MOD server and assume that, for this example, that they are sent every two minutes. Blank circles indicate the \((location, time)\) pairs which are of no interest for processing the RN2 because the value of their \textit{time} component is outside the time-interval of interest for RN2 \([5:00, 5:30]\).

In order to determine the spatial region of interest for RN2, one can construct the \textit{Minkowski Sum} \(P \oplus d\) of the polyline \(P\) and a disk of radius \(d\). This is essentially the 2D projection of the trajectory of \(oid_1\) with \(P \oplus d\) similarly to [71] and, subsequently, use linear interpolation to calculate the total time for which the object \(obj_1\) was (not) within distance \(d\) from \(P\). If that time is less than 27 minutes (equivalently, if the object was outside \(P \oplus d\) more than 3 minutes), the notification can be sent to the user.

Similarly, a more database-ready approach is to use (composition of the) readily available spatio-temporal operators, like the ones presented in [47, 36]. The processing of the respective operators is based on the \textit{plane sweep} technique (c.f. [47]) and, as a methodology, it can be used to verify if the object was within distance \(d\) from \(P\) the desired amount of time.

However, in practice, none of these approaches is acceptable because they both operate over the entire history of the objects motion throughout the time-interval of interest for the RN2. They may be suitable for processing a query like:

**RQ2**: “Retrieve that time that the object \(obj_1\) is (not) moving along \(P \oplus d\) between 5:00 and 5:30.”

or its \textit{yes/no} variant:

**RQ2**: “Was the object \(obj_1\) moving along \(P \oplus d\) for more than 90% of the time between 5:00 and 5:30?”

Many mission-critical applications (e.g., detecting an enemy’s activity in a battlefield) cannot tolerate such behavior. As indicated in Figure 2, the system should be able to notify the user as early as 5:18 that RN2 is satisfied – by then, the object has already spent more than three minutes (10% of the time-interval of interest) outside \(P \oplus d\). Hence, any further \((location, time)\) update, indicated with lighter-shaded circles in Figure 2, is of no interest for RN2.

2. One may want to utilize the capability of active behavior available in most existing (commercial) systems and set up a trigger:

**TRN2**, for which the basic elements (in a pseudo-syntex) are:

\begin{verbatim}
EVENT:  ON location_update(oid_1, x, y, t)
CONDITION: IF time_outside P \(\oplus d \geq 3\)
ACTION: SEND NOTIFICATION
\end{verbatim}

This will achieve the desired behavior – the user will be notified at 5:18 that RN2 is satisfied.

However, even this approach has some potential drawbacks. Namely, at every update, the system will evaluate the \textit{time_outside} for the entire past trajectory between 5:00 and the time of the current update. As illustrated in Figure 2, the object has already been outside \(P \oplus d\) for one minute by 5:12 (and for two minutes by 5:14). An intelligent system should know that there is no need to evaluate the condition of TRN2 on the entire past trajectory.

\(^3\)Formally, given two sets in \(\mathbb{R}^2\), say \(P_1\) and \(P_2\), their \textit{Minkowski Sum}, denoted by \(P_1 \oplus P_2\), is defined as \(P_1 \oplus P_2 = \{p_1 + p_2 \mid p_1 \in P_1, p_2 \in P_2\}\), where the summation is of vector \(p_1\) with vector \(p_2\) [2].
3.1.1 Algorithmic Aspects of Processing the Moving Along Predicate

Observe that the trajectory $T$ of $\text{obj}_1$ is a single polyline, and in fact each location update generates a new single line-segment. Thus, it appears that a possible approach is to handle segment queries (“how long a portion of this given segment lies in the given Minkowski sum?”) and to simply accumulate the results at every update. This can be handled quite generally for any region (not just Minkowski sums) and so this type of solutions applies to the more general setting of moving inside a given region.

In our approach, we pre-compute a decomposition of the plane compatible with $R$, such that each region obtained by the decomposition is simple enough to trace $T$ through the decomposition in time $t(n)$ times the number of intersection between $T$ and the decomposition. Note that all we care is identifying the intersections $T \cap R$, but the regions of $R$ need not necessarily be simple enough. One remedy is to process some kind of a decomposition or triangulation [21], but then $T$ could intersect the extra edges of the decomposition many times without ever intersecting $R$, preventing an output-sensitive bound in terms of $t(n)$ and $|T \cap R|$. There are structures for ray shooting in the plane which take $O(\log n)$ time inside a simple polygon, and can be extended to a Minkowski sum $R = P \oplus d$ (with arcs of circle) as long as $R$ remains simply connected (no holes, which is equivalent to say that the path $P$ is intersected by any disk $d$ in a connected portion)[]. If we have several connected components, we may preprocess each one; also, one may compute the convex hull of $R$ and treat every ”pocket” as another region of $R$. Making such a structure for every region inside $R$, then one can shoot a ray in time $O(\log n)$ from each intersection until the update is visible. If $T$ goes outside the convex hull, ray shooting against the convex boundary can be done in $O(\log n)$ time as well. The whole preprocessing can be done in $O(n \log n)$ time, the total complexity of the updates becomes $O(m + |T \cap R| \log n)$, and each update incurs a cost of $O(\log n)$ times one plus the number of intersections of the current segment of $T$ with $R$. Moreover, if a triangulation of $R$ is available, the whole preprocessing can be done in $O(n)$ time.

A concise description of the above analysis is:

**Moving Along**($R$, ($x_i, y_i, t_i$), ($x_{i+1}, y_{i+1}, t_{i+1}$))

1. Decompose $R$
2. Shoot a ray from ($x_i, y_i$) towards ($x_{i+1}, y_{i+1}$) and obtain the intersections with $R$
3. Use linear interpolation to obtain the total time inside $R$

3.1.2 The Moving Along in Time Predicate

If the previous predicate where the trajectory is given in the spatio-temporal domain (as is done in []), the question RN2” becomes:

**RN2”**: “Notify me when the object $\text{obj}_1$ is not moving along the polyline $P$ within distance of where it should be at that time on $P$, for more than 10% of the time between 5:00 and 5:30,”

The approach is similar to the moving along predicate with a region $R = P \oplus d$ that is a tubular neighborhood of a polyline $P$, except that in the previous section, there was no constraint on the time so the region $R$ was essentially a cylinder $R_2 \times \text{cal} R$ (there are no changes to $R_2$ over time). In 3D, one requires in RN2” that the object, at all time, is within distance $d$ of where it should be at that time on the path $P$ (there was no such requirement in the previous section). The approach described before apply as well, but now there is an advantage that both the updates and the region $R$ are monotone in the $z$-direction (time). This enables a sweep approach by a plane orthogonal to the $z$-direction. In that case, one only has to maintain a kinetic decomposition of the sweep region. Namely, one only needs to maintain the disk $p(t) + d$ of radius $d$ centered at $p(t)$ and radius $d$, where $p(t)$ is the point of $P$ at time $t$, and the point $\text{obj}_1(t)$ which moves also on a polyline $T$; detecting whether the distance between these two points becomes less or greater than $d$ is a kinetic predicate than can be evaluated as the events (vertices of $P$ or updates) are swept by the horizontal plane, and that can only change once (when $\text{obj}_1(t)$ goes through the orthogonal projection of $p(t)$ on $T$
between two events). Thus the runtime of $RN2^*$ is linear in $O(n + m)$, and between two updates, in the number of vertices of $P$ that are encountered.

3.1.3 Behavioral Aspects of the Moving-Along Predicate

The algorithms in Section 3.1.1 gives the operational specifications of how the desired behavior can be achieved. However, it is not straightforward to "translate" it operational semantics into a declarative manner using the SQL-like statements. At present, the existing prototype implementations of MOD and spatio-temporal databases [8, 38] lack the mechanisms which would enable an elegant and declarative specification of these kind of reactive behavior. On the other hand, one may want to use the available extensibility features of the commercial ORDBMS, e.g., Oracle or DB2, and use them as a MOD [71]. However, their available triggers cannot achieve all the aspects of the reactive behavior that one would expect from a notification service. One of the reasons is the limited set of primitive events (update, insert, delete) operating on the tables. However, another reason is that the very model of a trajectory which can be done as a User-Defined Type (UDT) in an ORDBMS has its own semantic subtleties. Namely, despite the lexical similarity, the classical meaning of the update (e.g., a salary of a particular employee) in a regular (relational) settings has different impact from location update of a particular moving object in MOD. The net-effect of a sequence of location updates increases the number of points (the representation of) the trajectory of a given moving object. Meanwhile, a sequence of salary updates of a given employee has the net-effect of one single (lump) increase.

To better illustrate the peculiarities of RN2, recall the request RN1 (c.f. Section 2), which is common in MOD settings:

**RN1**: "Notify me when I am within 3 miles from a motel, between 7:00PM and 9:00PM".

Setting up a trigger TRN1, of the form:

**EVENT**: ON location_update(oid, loc, t)

**CONDITION**: IF within_distance(loc, Motel)

**ACTION**: SEND NOTIFICATION

would yield a correct behavior, without the processing overhead exhibited by the similar triggers TRN2 for RN2. However, a careful observation will reveal that although the environment is dynamic, in the sense that the object oid changes its location in time, still the CONDITION part of the trigger TR1 is, in a sense, static. Namely, the condition only checks the instantaneous value within_distance predicate, which does not depend on the previous states of the MOD in which it was evaluated. Contrary to this, in the case of RN2, the conditions evolves along with the modifications to the MOD. Thus, in order properly monitor the event of interest in an efficient manner, one would like a trigger that follows the dynamics of the evolution of the MOD. Assume that we have a function, denote it $TimeInside(R, (x_1, y_1, t_1), (x_2, y_2, t_2))$, which, given two points $(x_1, y_1, t_1)$ and $(x_2, y_2, t_2)$ returns the time-interval during which the object moving along the straight line-segment $(x_1, y_1), (x_2, y_2)$ and with a constant speed between $t_1$ and $t_2$, was inside $R$. Also, let previous_last(OID) and last(OID) denote the functions which return the $(x, y, t)$ values of the next to last and the last point, respectively, in the representation of the Motion_Plan of the object OID. We have the following dynamic version of the trigger which monitors RN2:

**Trigger TRN2dyn:**
1. ON location_update (oid1, x, y, t)
2. IF $TimeInside(P \oplus d, last(oid), previous_last(oid)) \geq (3 - t_{total}^n)$
3. THEN
4. Send_Notification
5. ELSE $t_{total}^n = t_{total}^n + TimeInside(P \oplus d, last(oid), previous_last(oid))$

The variable $t_{total}^n$ is the accumulator which denotes the total time that the object oid1 had spent moving
along \( R(= P \oplus d) \) from the begin-time value specified in the request \( \text{RN2} \).

The variable \( t_{\text{total}}^m \) denotes the time that the object \( oid_1 \) had spent moving along \( (P \oplus d) \).

### 3.2 The Moving Towards Predicate

Now we consider another predicate in a spatio-temporal setting, which is concerned with detecting if a particular mobile object is moving towards a given static entity like a point-object, region or a (poly)line. Some algorithmic aspects for a variant of the problem, where the necessary and sufficient conditions on the object’s trajectory for the purpose of missile guidance towards a point-target, have already been addressed by the control community [13]. In this work we will use a simplified criteria for the satisfiability of the moving towards predicate. We will assume that a given mobile object \( \text{obj}_2 \) is moving towards a static target (landmark) \( LM \), if its distance from \( LM \) is non-increasing between consecutive updates. Depending on the application domain, one may be interested in different variants of continuous requests, like, for example “Notify me when the distance from the target has decreased by at least 20% when \( \text{obj}_1 \) is moving towards the target for 2 minutes”.

To illustrate the aspects of the reactive behavior that we are investigating in this paper, we will use the following example:

\( \text{RN3}: \) “Notify me when the object \( \text{obj}_2 \) is moving towards the landmark \( LM \) continuously for 5 minutes between 5:00 and 5:30”.

![Figure 3: Notification for Spatio-temporal Moving Towards Predicate](image)

A possible scenario is illustrated in Figure 3 and we have similar observations as in the case of moving along predicate. However, now the semantic implications of the continuous satisfaction of the predicate for the interval of (at least) five minutes will bring some more specific issues to light.

1. A purely query-like (MOD or spatio-temporal database) approach, in which one would wait until 5:30 and then pose the corresponding query is, again, unacceptable. As shown in Figure 3, \( RN3 \) is satisfied at 5:18 because between 5:12 and 5:18 the object was continuously moving towards \( LM \) for 6 minutes. Hence, the corresponding notification should have been sent at 5:18.

2. Again, one may be tempted to set up a trigger, say \( \text{TRN3} \), which upon every location update (EVENT) would check if the distance between \( \text{obj}_2 \) and \( LM \) was non-increasing for an interval of 5 minutes (CONDITION) and, subsequently, notify the user (ACTION). However, aside from the questions regarding the issues of how to exactly express it (e.g., one may query the past information about the object’s Motion Plan by using SQL-TS for querying over the sequence of the location update points [62]), the “classical” trigger may involve unnecessary calculations for processing the \( \text{RN3} \), by using
the entire “current history” of the object’s motion. As a particular example, at 5:14 there is absolutely no need to query any history before 5:12.

3. A peculiarity brought by this example is that within the time-interval of interest for the request, multiple notifications may be generated. The source is two-fold:

(a) Although this is not illustrated in Figure 3, one can easily think of settings in which a satisfaction of the particular moving towards is followed by moving away which, in turn, is followed by another moving towards time-interval. This can cause multiple notifications to be sent for a given request.

(b) A more intriguing reason is that at 5:20 the object has completed another interval of 6 (> 5) minutes ([5:14,5:20]) during which it was continuously moving towards the target – landmark L.M. One may object that the location updates at 5:16 and 5:18 were already used when the notification was sent at 5:18 – and the objection may be either “sustained” or “over-ruled”. This is due to the fact that it was not specified what is done to the primitive events that were already used throughout the history. To cope with such issues, the notification system must allow an explicit choice of the policy of which primitive (constituent) events are consumed upon the detection of the desired (composite) event. The Events Management System SNOOP [15] (subsequently used in the SENTINEL Object-Oriented DBMS [14]) proposed several different policies for consumption of the constituent primitive events upon a detection of a composite event. Each policy specifies what happens with the set of instances of the primitive events in between (and including) the particular pair of (initiator event, terminator event). Thus, for the scenario illustrated in Figure 3, if one does not want a notification sent at 5:20 (i.e., a brand new “observation-interval” is started at 5:20), then all the updates between 5:12 and 5:18 should be, in a sense, flushed out from consideration and a new initiator event will be sought for with the location update at 5:20. In the parlance of the SNOOP system, this corresponds to the cumulative consumption of the primitive constituent events. The other option is to flush out only the location update from 5:12 as a possible initiator, and use the one at 5:14 as the new initiator. This will combine with the next location update (5:20) as the new terminator event and generate another notification. This behavior corresponds to the chronicle consumption policy in SNOOP.

Consequently, when monitoring a particular request, one may have to choose the consumption policy beforehand.

3.2.1 Algorithmic Aspects of Processing the Moving Towards Predicate

One can use ideas similar to those presented in Section 3.1.1 for the moving along predicate, first of all, by accumulating the answers to “for how long along this segment s was obj moving towards the landmark?” over every last k updates. All that is needed to answer that query is some kind of Voronoi diagram. Indeed, the idea will be similar: construct the Voronoi diagram of the landmark R [21]. If R is described by line segments, the cells of the diagram are not necessarily convex; we separate between the cells of the segments proper and those of the endpoints, and triangulate each one of them using the geodesic triangulation of [17, 40] before preprocessing the triangulations so as to answer ray shooting queries in \(O(\log n)\) time. (In practice, one would only do a triangulation for those cells whose complexities exceed a pre-determined constant, since the cells have an average of \(O(1)\) sides.)

During an update, we trace the the trajectory inside the diagram by shooting rays, as in the previous sections. As long as the trajectory remains within the cell of a segment, it can only move towards all the time, or move away all the time. We add the length of the part of the trajectory inside the cell to the moving towards accumulator. If the trajectory is inside the cell of an endpoint, it can first move towards, then away:

\[\text{The original terminology of [15] uses the phrase consumption contexts, however, in this work the term context has a different meaning.}\]
the critical point may occur at the projection of the endpoint on the trajectory. In any case, an update \( i \) takes time proportional to the number \( k_i \) of cells it traverses (with an overhead of \( O(\log n) \) due to the ray shooting query). This is output-sensitive in the sense that \( k_i \) is the number of times the closest feature of the landmark changes during the trajectory. (Indeed, it seems hard to keep track of moving\_towards if one does not keep track of the closest feature.)

### 3.2.2 Behavioral Aspects of Moving\_Towards Predicate

The operational behavior given in Section 3.2.1 cannot be declaratively specified using the existing commercially available ORDBMS. Since we have already elaborated on the algorithmic details, for the purpose of clarity of the syntax/presentation, in the rest of this section we will use a simplified criteria for the satisfiability of the moving\_towards predicate. We will assume that a given mobile object \( obj_2 \) is moving\_towards a static target (landmark) \( LM \), if its distance from \( LM \) is non-increasing between consecutive updates. Under this assumption, the syntactic elements of the dynamics-aware trigger which monitors the request \textbf{RN3} are as follows:

**Trigger TRN3dyn:**

1. ON location\_update \((oid_2, x, y, t)\)
2. IF \((X,Y,T) = \text{previous\_Last}(oid_2) \land \) 
   \(\text{distance}(x, y, LM) < \text{distance}(X, Y, LM) \land \) 
   \((t_i - t\_reference \geq 5)\)
3. THEN
4. \textsf{Send\_Notification AND Execute the chosen consumption policy}
5. ELSE IF \(\text{distance}(x, y, LM) \geq \text{distance}(X, Y, LM)\)
6. THEN
7. \(t\_reference = t_i\)

The specification states the intended behavior for the processing of the moving\_towards predicate. The only possible ambiguity arises with the value of \( t\_reference \). Observe that this is the only non-bound variable in the IF part. Initially, it is set to the beginning time of the \textit{RN3} interval of interest – \( t\_reference = 5:00 \) – and it acts, in a sense, like a global variable for detecting if the desired condition has been valid continuously for 5 minutes. Subsequently, it is maintained up-to-date any time the condition is invalidated by a particular location update in the MOD. The second conjunct in line 4, \textit{Execute the chosen consumption policy} pertains to the discussion about multiple notifications and it affects the value of the \( t\_reference \) variable. In case a given application scenario would like another notification sent at 5:20 (chronicle policy for consumption), then \( t\_reference \) is set to the next available value after the initiator of the trace which enabled the condition in line 2., which is, \( t\_reference = 5:14 \) because its value was \( t\_reference = 5:12 \) when the sequence of updates at 5:14, 5:16 and 5:20 enabled \textbf{TRN3dyn}. In case the particular application at hand does \textit{not} want another notification sent at 5:20 (i.e. it requires a “fresh” sequence of 5 minutes during which \( oid_2 \) is continuously moving\_towards the landmark \( LM \)), then the effect of the \textit{Execute the chosen consumption policy (cumulative one)} conjunct in line 4. would set \( t\_reference = 5:20 \).

### 4 Evolution of the Triggers and Context-Awareness

In this section we present an example which demonstrates some other aspects of the \((ECA)^2\) paradigm when the data items originate from \textit{heterogeneous} sources and we use it to illustrate some subtle issues of the reactive behavior which cannot be achieved by using the traditional \(ECA\) paradigm.

Consider the following request:
RN4: “Notify me whenever 4 airplanes are moving towards the Critical Region CR continuously for at least 2 minutes and I have less than 6 airplanes available in the Air Base. Otherwise, notify me when the first one of them is within 100 miles from CR.”

![Diagram of Air Base and Critical Region]

Figure 4: Context Aware Evolution of Events

Figure 4 illustrates a situation in which there are five airplanes ($A_1, \ldots, A_5$) which have been moving towards the critical region $CR$ in the current state of the world (trajectories illustrated with solid lines). The solid boundary of the AirBase indicates the number of planes currently available. At this point, there is no need to generate any notification because there are nine airplanes available in the Air Base and the condition part of the (corresponding) trigger would fail. The (relative) future instance of the entities involved in this scenario is illustrated with dashed lines in Figure 4 and the position of each of the airplanes that were “currently” approaching the $CR$ is suffixed with $(f)$. At that point, $A_3$ is no longer moving towards the critical region $CR$, while the other 4 airplanes continued their motion towards $CR$. Moreover, at the particular “future” time-point, 4 planes have left the AirBase.

Now we proceed with identifying the corresponding syntactic elements of the trigger $TRN4_{dyn}$ which will monitor the request $RN4$.

- **EVENT:** The triggering event which will “awake” $TRN4_{dyn}$ occurs whenever there are 4 or more airplanes satisfying the moving towards predicate with respect to $CR$ and do so continuously for at least 2 minutes. To specify this, first we create a temporary table $EVENT_{RN4}$ with the attributes ($OID$, $initiator_{loc_update}$, $terminator_{loc_update}$) and we set the action part of the trigger similar $TRN3_{dyn}$ (c.f. Section 3.2.2) to insert the respective tuple$^5$ in $EVENT_{RN4}$. The detection of the desired composite event is achieved by setting another (intermediate) trigger which:
  
  ON insert TO $EVENT_{RN4}$
  IF count(distinct(OID)) $\geq$ 4
  
  Generate the event e$_{TRN4_{dyn}}$ to “awake” for $TRN4_{dyn}$.

- **CONDITION:** For the condition part, we assume that there is a table $IN_{AIRBASE}$ with the attributes ($Plane_{ID}$, $arrival_{time}$, $location$). To verify whether there are (not) enough planes in the AirBase, we use:

  \[
  \text{count}(\text{distinct}(\text{Plane}_{ID})) \geq (<) 6.
  \]

$^5$Observe that in this particular case we are interested in the chronic consumption policy for the primitive events corresponding to each location update.
• **ACTION:** In case there are fewer than 6 airplanes in the AirBase (which is not the case in the “current” portion of Figure 4), the system will notify the user. However, the portion of interest is when the condition fails. In this case, the user still wants to monitor the environment as specified in the “classical” part of the trigger. However, now the user would like the system to become aware of another possible subsequent state of interest, in which one of the airplanes continuously approaching the critical region CR has come too close to it.

• This is precisely where the existing systems which provide reactive behavior based on the ECA paradigm become unsuitable. Even if one is tempted to introduce a separate brand-new trigger, e.g., by using the CREATE RULE command in a Starburst-like (c.f. [78]) manner when TRN4dyn is processed, the newly created rule will not be aware of the history of the events prior to its creation and it may not be able to properly react to the dynamic changes of the environment. Thus, assuming that the “future” position of the environment depicted in Figure 4 occurred 1.5 minutes after the “current” one, there will be no 4 (or more) airplanes moving towards the CR continuously for more than 2 minutes. Consequently, although the future location of the airplane $A_1(f)$ may be closer than 100 miles, the user will not be properly properly notified because the composite triggering event (4 (or more) airplanes moving towards the CR...), would not have been detected. This is one aspect of the context-awareness the the triggers should exhibit – possible co-existence of the parent and child triggers.

Assume that we have a function $\text{closest_distance}(Object, Region)$ which, given a set of objects and a region, returns the distance of the closest object to that region. Now we can specify the trigger $\text{TRN4dyn}$ as follows:

**Trigger TRN4dyn:**
1. ON $e_{\text{TRN4dyn}}$
2. IF $\text{count}(\text{distinct}(\text{Plane_ID})) < 6$
3. THEN
4. $\text{Send_Notification_1 AND Execute the consumption policy}$
5. ELSE IF $(\text{closest_distance}(\text{OID, CR}) \leq 100) \wedge$
   $\text{OID in SELECT OID FROM EVENT_RN4}$
6. THEN
7. $\text{Send_Notification_2 AND Execute the consumption policy}$

### 4.1 Dynamics of the Context-Awareness

Assume, for the sake of argument, that the “final” position $A_1(f)$ in Figure 4 is actually not within 100 miles from the critical region CR. Observe that, although the airplane $A_3$ is no longer moving towards the CR after the “current” time, there are still 4 airplanes continuously moving towards it. This would still satisfy the event part of the trigger $\text{TRN4dyn}$. However, since 4 airplanes have taken off the AirBase, now the condition part $\text{count}(\text{distinct}(\text{Plane_ID})) < 6$ is satisfied. The main observation is that the 4th (last) airplane might have left the AirBase at some point after the evaluation of the condition part of the trigger $\text{TRN4dyn}$ (in which case the action part did not get executed) AND before the next location update of any of the airplanes moving towards the critical region CR. Thus, there will be a time-interval during which the trigger $\text{TRN4dyn}$ will not be “awake”, although (one of) its condition is satisfied. In time-critical

---

6Note that we still have not given a formal definition of a state of an Active MOD under the (ECA)$^2$ paradigm. We introduce it in Section 7.

7One may observe the similarity with the concepts of fork in the Operating Systems setting (c.f. [65]) where a parent process creates a new child process, but they are both using the same address space so that the child has a proper access to the values of some global-like variables.
application this behavior may not be acceptable and an intelligent system should automatically awake the respective trigger.

This is the aspect of the context-awareness that, we believe, may be necessary in some applications. The changes of values in some structure, which is not explicitly listed as a triggering event for the monitored request, but may affect the evaluation of the condition, which failed during the latest consideration of the respective trigger, should be able to automatically "awake" that trigger.

This behavior is not standard in many notification systems [55, 43] which are matching the user's preferences based on his location information. Recall (c.f. Section 2) the request:

**RN1**: "Notify me when I am within 3 miles from a motel, between 7:00PM and 9:00PM".

As we already pointed out, the condition part in this request is not dynamic. Even if we consider the scenario typical for the event-based notification systems:

**RN1**: "Notify me when I am within 3 miles from a favorite restaurant of mine.

the users preferences that could be used in the condition part of the trigger that monitors RN1' are (relatively) static over time.

5 (ECA)$^2$ – Syntactic Elements of the CAR Language

Recall that the paradigm that we propose – Evolving and Context-Aware Event-Condition-Action can be intuitively stated as follows:

```
ON EVENT
   IF CONDITION
       THEN ACTION
   ELSE <consume parent>
       MODIFY EVENT/CONDITION/ACTION
       NEW EVENT/CONDITION/ACTION
```

The results in the "classical" ECA paradigm abound [79] and we will not re-hash all the possible classification of the different semantic dimensions and their respective choices in particular Active Database System – detailed analysis is provided in [28]. Below, we list the desiderata that a system implementing the (ECA)$^2$ paradigm should exhibit, assuming that they are "on top" of the features already available in (any of) the existing systems implementing the ECA paradigm.

Consider a modified (simplified) version of the scenario described in Section 4:

**R2**: "Notify me when an object is moving continuously for 5 minutes towards the landmark LM, between 5:00PM and 5:30PM, IF I have no more than 10 aiplanes in the AirBase B1 ". Subsequently, also notify me when that object is closer than 50 miles to LM."

In the context of the (ECA)$^2$ paradigm, this can be expressed by the following trigger:

**TR2dyn:**
1. EVENT: e\textit{moving\_towards}(OID,LM,5)
2. CONDITION: \textit{count\_aiplanes}(B1,X) \land X \leq 10
3. ACTION: \textit{Send Notification}_1
4. ELSE <parent consumption = no>
5. NEW EVENT: e\textit{location\_update}(OID, (x,y,t))
6. NEW CONDITION: M\textit{distance}((x,y,t),LM,D) \land D \leq 50
7. NEW ACTION: \textit{Send Notification}_2

The event e\textit{moving\_towards}(OID,LM,5) is a composite event that can be detected using a trigger TR1c, similar to TR1dyn (c.f. Section 3.2.2). The only difference that the action part of TR1c, instead of sending a notification, will execute an action which will induce the occurrence of an (composite) event. Observe that the user is interested in monitoring the proximity ("...closer than 50 miles...") only after the
event e\textit{moving\_towards}() has been detected. Thus, the \textit{child} trigger will become part of the system. However, along with the proximity, the user is still interested in the developments with respect to the \textit{moving\_towards} property. This is reflected in the <\textit{parent consumption = no}> which means that both the \textit{parent} (lines 1. - 3.) and the \textit{child} (lines 5. - 7.) co-exist after the first detection of e\textit{moving\_towards}().

![Diagram of Air Bases and Airplanes]

\textbf{Figure 5: Evolution of Events and Conditions}

An illustrating scenario is given in Figure 5. Clearly, at 5:16PM, the event e\textit{moving\_towards}(o_1, LM, 5) will be detected. However, the \textit{Notification} \_l will not be sent, because at that time, there are 11 airplanes in the Air Base B1. At 5:16PM, the system will also begin to monitor the proximity of the object o_1 to LM.

The main observation, which is the crux of this section, is that although the next \textit{location\_update}(o_1,(x,y,t)) (and, consequently, the detection of another e\textit{moving\_towards}(o_1,LM,5)) will occur at 5:18PM, we have an occurrence of a phenomenon of interest at 5:17PM. Namely, due to the take-off of three airplanes from B1, now the condition of (the parent-part of) the trigger TR2\textit{dyn} is satisfied. However, this will not be taken into effect until 5:18PM, which is the next time-point when the enabling event of TR2\textit{dyn} will be detected. Clearly, such a behavior is not desirable in many applications of interest. To cope with situations like this, a context-aware active database system needs a mechanism which, upon a specification of the user’s trigger, will automatically bring to attention of TR2\textit{dyn} that its condition (which was \textit{false} at the last consideration of TR2\textit{dyn}) became \textit{true}. Now we proceed with illustrating the main features of such a mechanism.

\section{The \textit{Condition} vs. \textit{Event} Duality and Meta-Triggers}

In reality, the information about the airplanes located in various Air Bases may be managed by a database which is independent from the Moving Objects Database (MOD) which manages the trajectories representing the motion plans (past and future) of the mobile entities. In such distributed heterogeneous settings, the processing of the query corresponding to the condition of a particular active rule may impose a significant overhead to the response time. Even if the database is not distributed, as we illustrated, the heterogeneity of the \textit{natures} (semantics) of the dynamic entities involved, may cause unacceptable delays in the response time of the desired reactive behavior. The main steps towards a possible solution for handling this kind of problems are:

- Whenever a change is detected which affects the condition part of a particular trigger, say TR_i, an event is generated which will notify TR_i of that change.
- Trigger TR_i is, in a sense, made aware that now it needs to react to events other than its original “enabling” event.

Our discussion illustrates the \textit{dual} nature of the role of a given \textit{condition} for a particular trigger TR_i. The condition’s specification becomes a source for specifying a new \textit{event}, the detection of which was (previously) not explicitly requested in the active database. This \textit{dual} event subsequently becomes an entity that should “awake” the trigger TR_i, although it was not stated in the original formulation of the TR_i, corresponding to
the user's request. However, the ability of TRi to recognize the proper role of the dual event corresponding to its original condition, may require some additional modifications to the specification of the TRi itself.

The key concept for enabling all of the above in our approach is that of the Meta-Trigger, which is a module (agent) that, given the specification of a particular trigger TRi, will generate two triggers:
1. The trigger DTRi which monitors the modifications pertaining to the condition part of TRi and generates the dual event of that condition.
2. The trigger ATRi which is the modified version of the original TRi that is aware of the impact of the dual event generated by DTRi.

The details of the Meta-Triggers are presented in the next section.

6 Domain Descriptions for \((ECA^2)\)

Based on the observations above, we now specify the syntactic elements for the \((ECA^2)\) paradigm. Specification and reasoning about activities in our approach is done using a high level language CAR, developed in the spirit of the action description language \(A\) [30]. The syntax of the CAR language is based on \(L_{active}\) in [5] which was used for logical formalization of active databases; and \(ADC\) [4]. The CAR language is based on action theories, which have been successfully used in reasoning about robot control programs, reasoning about parameterized actions, qualification and ramification constraints, concurrent execution of actions and facts exempt from common-sense law of inertia [31, 48].

We assume that there are four, possibly (countably) infinite, pairwise disjoint sets of symbols: \(A\) – action names, \(F\) – fluents, \(E\) – event names, and \(R\) – rule names; and a set of variables. Each symbol has an \textit{arity} associated with it and literals from each set are defined as usual. Atoms from \(A, F, E\) and \(R\) are called \textit{actions}, \textit{fluents}, \textit{events}, and \textit{rule ids} respectively. We will restrict the use of literals to fluent and event literals. We use the generic term \textit{fluent} to denote data items which can change their values as the MOD evolves. They can take different forms according to the kind of underlying DBMS. For example, in an object oriented database, they could be a name of an object attribute, together with a value from the domain of the attribute. In a relational model, fluents correspond to the tuples that can appear in the relations. Variables can be used in the literals and they represent parameters that can be replaced by any value from the underlying domain of the attributes. In general, actions are the operations provided by the system that can be applied to the data. Some of them may have no direct effect on the database, and there will not be causal laws associated with them. An example of such action could be the retrieve operations defined in Postgres [68]. Other actions could be commit, abort, User-Defined Functions (UDF), or methods in an Object-Oriented Database.

To provide a description of the \((ECA^2)\) paradigm, we have the following propositions in the CAR language:

- **Effects** – We assume that an execution of a particular action in an environment in which certain conditions hold, will have effects on the fluent literals\(^8\). For that, we have the following \textit{effect proposition}:

\[
a(\mathcal{X}) \textit{ causes } f(\mathcal{Y}) \textit{ if } p_1(X_1), \ldots, p_n(X_n), e_1(Z_1), \ldots, e_m(Z_m)\]  

(1)

The intuitive meaning is that the execution of the action \(a\) in a state of MOD in which each fluent \(p_i(i \in \{1, \ldots, n\})\) is true and each (primitive and/or composite) event \(e_j(j \in \{1, \ldots, m\})\) has occurred, ensures that the fluent \(f(\mathcal{Y})\) is true in the next state (resulting from the execution of the action).

- **Event Definitions** – We have two kinds of events definitions:

  1. Events induced by the execution of an action:

\[
a(\mathcal{W}) \textit{ induces } e(\mathcal{X}) \textit{ if } e_1(Y_1), \ldots, e_m(Y_m), q_1(Z_1), \ldots, q_n(Z_n)\]  

(2)

---

\(^8\)Throughout this work, we will not discuss the issues pertaining to \textit{binding} of the variables. We assume that the rules are \textit{safe} (c.f. [73]).
Intuitively, this proposition says that the execution of the $a(W)$ in a state in which each of the fluent literals $q_1(Z_i)$ is true and each of the event literals $e_j(Y_j)$ is true (i.e., the event in the event literal belongs to the current set of events if the event literal is positive, or it does not belong to the set if the event literal is negative) generates the event literal $e(X)$, i.e., it is added to the set of current events if the event literal is positive, or removes the event from the set of current events if the event literal $e(X)$ is negative.

The main motivation behind this type of event definitions is to enable the method invocations and UDF executions to raise an event, besides the common insert, delete and update statements.

2. The other kind of event definition is of the form:

$$\text{Expr}_{\text{events}}(W) \text{ induces } e_{\text{composite}}(X) \text{ if } e_1(Y_1), \ldots, e_m(Y_m), q_1(Z_1), \ldots, q_n(Z_n)$$

(3)

This kind of propositions allow one to specify the occurrence of composite events in a particular state. The $\text{Expr}_{\text{events}}(W)$ is an expression in the respective Event Algebra (c.f. Section 2) provided by the system.

We assume that there is a set of consumption policies (c.f. Section 3) provided by the system, however, along with the common approach where the policy has to be tied to a particular composite event $[15]$, we choose a more flexible approach where the consumption policy is either a part of the events definition or the specification of the active rule (see below) which is triggered by the composite event $$. This way, the same composite event which triggers two different rules may use different policies for its consumption for the particular rule.

- **Definitions of Active Rules** – An active rule in the context of the $(ECA)^2$ paradigm is the proposition of the following form:

$$\text{TR}(\Delta\sigma)(A_C) : <\leftrightarrow\ e_t(X_t) \ \text{consumption}\_\text{scope}\ (C.s)$$

if $p_1(X_1), \ldots, p_n(X_n)$ at $e_c(X_c)$

initiates [$\sigma$] at $e_a(X_a)$

Duality($C_d$)

ELSE

$$e_t^{\text{new}}(X_t^{\text{new}}) \ \text{consumption}\_\text{scope}\ (C.s^{\text{new}})$$

if $p_1^{\text{new}}(X_1^{\text{new}}), \ldots, p_n^{\text{new}}(X_n^{\text{new}})$ at $e_c^{\text{new}}(X_c^{\text{new}})$

initiate [$\alpha^{\text{new}}$] at $e_a^{\text{new}}(X_a^{\text{new}})$

Duality($C_d$)

(4)

where $\text{TR}(\Delta\sigma)$ is the rule identifier atom, and the $A_C$ variable can take values yes or no, indicating whether the trigger is available for consideration or not. $e_t(X_t)$, $e_c(X_c)$, and $e_a(X_a)$ denote the triggering event, condition-evaluation event and the action-execution event, respectively. They are used to distinguish between the different coupling modes of event detection vs. condition evaluation (vs. the execution of the action part) $[28, 57]$. For example, under the immediate coupling, $e_t(X_t) = e_a(X_a) = e_c(X_c)$. The optional symbol ↓ is used to denote that the particular trigger has a high-enough priority so that it can interrupt the regular processing of the sequence of database actions and, without waiting for a rule processing point, it requests to be considered immediately in a state in which its enabling event $e_t(X_t)$ is detected. $p_1(X_1), \ldots, p_n(X_n)$ are fact-literals, constituting the condition of the rule and $\alpha = [a_1, \ldots, a_k]$ ($a_i \in A$) is a sequence of actions, representing the action part of the rule. Observe that by using the ↑ symbol in the sequence $\alpha$ we allow for recursive processing of the active rules. $C.s$ is one of the symbols no, local or global denoting various
modes of which a given active rule can consume its enabling event and $C_{\text{parent}}$ can assume only the values yes or no, depending on whether the parent trigger will co-exist with the child trigger, or not. Again, we need the safety of the expression with respect to the variables. The ones appearing in $\overline{X}$, in any of the input arguments of the actions in $\alpha$ or in any negative literal in the condition must also appear in $\overline{X}$ or in a positive literal in the condition. Variables appearing in $\overline{X}$ or $\overline{X}$ must also appear in $\overline{X}$. Lastly, $C_d$ can assume only the values yes or no, depending on whether the trigger is allowed to be processed by the Meta-Trigger module or not.

6.1 The Meta-Trigger Description

Given a description of an active rule $\text{TRr}$ in the $\text{CAR}$ language, if the condition part $p_1(\overline{X_1}), \ldots, p_n(\overline{X_n})$ evaluates to false when $\text{TRr}$ is considered (i.e., the very first time the condition is evaluated upon the detection of the enabling event $e_i(\overline{X_i})$, the Meta-Trigger module is invoked. The effects of the Meta-Trigger $\text{MT}_{\text{CAR}}$ can be described as follows:

1. If $C_d =$ no then the effect of $\text{MT}_{\text{CAR}}$ is null;
2. Else If $C_d =$ yes
3. Generate a set of triggers $\{\text{DTRr}_1, \ldots \text{DTRr}_n\}$, one for each $p_i(\overline{X_i})$ ($i \in \{1, \ldots, n\}$) from the condition part of $\text{TRr}$ such that:
   
   3.1. The triggering event of $\text{DTRr}_i$ is any modification that can affect $p_i$ (insertion, deletion or update in relational settings; or a method invocation in object-oriented settings);
   
   3.2. The condition of each $\text{DTRr}_i$ is $p_1(\overline{X_1}), \ldots, p_n(\overline{X_n})$ (same as the condition of the original trigger $\text{TRr}$);
   
   3.3. The action part of each $\text{DTRr}_i$ generates an event which indicates that the condition of $\text{TRr}$ is satisfied, e.g., $\text{action}_{\text{DTRr}_i}$ induces $e_{\text{condition satisfied}_{\text{TRr}}}(\text{ground})$, where $\text{(ground)}$ indicates the ground values substituted for the variables in the condition of $\text{TRr}$ ($\overline{X_1}, \ldots, \overline{X_n}$);

4. Generate the Aware version of $\text{TRr}$ – the $\text{ATRr}$ for which:
   
   4.1. The event part is the composite event $e_i(\overline{X_i}) \land e_{\text{condition satisfied}_{\text{TRr}}}(\overline{X_{\text{Es}}})$;
   
   4.2. The condition part is empty;
   
   4.3. The action part is same as the action part of $\text{TRr}$;

5. End If

Along with the Meta-Trigger translation and generation of the $\text{DTRr}_i$ and $\text{ATRr}$ rules, we have the Meta-Trigger Control Module ($\text{MTCM}$) which acts as follows

1. When a trigger $\text{TRr}$ is considered and its condition is false
   
   1.1. Set the $A_C$ value of $\text{TRr}$ to no;
   
   1.2. Set the $A_C$ values of each $\text{DTRr}_i$ and $\text{ATRr}$ to yes;

2. When the triggering event $e_i$ of $\text{TRr}$ is explicitly deleted
   
   2.1. Set the $A_C$ values of each $\text{DTRr}_i$ and $\text{ATRr}$ to no;
   
   1.1. Set the $A_C$ value of $\text{TRr}$ to yes;

6.2 Reasoning About Dynamics of $\text{CAR}$ Specifications

For the rest of this section we assume that $D_{\text{CAR}}$ is the set of ground instances of the propositions in the domain description under consideration. We will refer to any set of facts as a fact state and any set of events as an event state. We say that a fact $f$ holds in a fact state $\Sigma$ if $f \in \Sigma$, and $\neg f$ holds in $\Sigma$ if $f \notin \Sigma$. Similarly, an event $e$ holds in an event state $\Gamma$ if $e \in \Gamma$, and $\neg e$ holds in $\Gamma$ if $e \notin \Gamma$.

Let $\Gamma$ be an event state and $\Sigma$ a fact state. Let $\Theta$ be a set of (triggered) rules. Let $\Omega$ be set of (considered) rules (to be fired). We refer to a tuple of the form $< \Sigma, \Gamma, \Theta, \Omega >$ as an active database state, or state for brevity.
The central concept in our declarative semantics of the \( \mathcal{CAR} \) are the definition of transition functions called causal interpretations. A causal interpretation is a partial function \( \Psi \) that maps a (possibly empty) sequence of actions \( \alpha \) and a state \( < \Sigma, \Gamma, \Theta, \Omega > \) into a new state. Given a domain description \( D \), we would like to identify the causal interpretations that model the behavior of \( D \) given any initial state. We will do that through four auxiliary functions that will describe how an action, when executed in a state \( < \Sigma, \Gamma, \Theta, \Omega > \), affects each component of the state. We will also need an action selection function. An action selection function \( S \) is a total function that takes a set of events \( \Gamma \) and a set of considered rules \( \Omega \), and returns the sequence of actions appearing in some rule \( r_i \) in \( \Omega \), such that action execution event \( e_i^\alpha \) of \( r_i \) is in \( \Gamma \). If such a rule does not exist it returns a special null action \( \mu \). Each selection function \( S \) has an associated function \( S' \) that when applied to \( \Gamma \) and \( \Omega \), returns a singleton set with the rule \( \{ r_i \} \) which contains the sequence \( S(\Gamma, \Omega) \) if it is not the null action; otherwise it returns an empty set. Action selection functions will be used to determine which actions’ sequence will be selected for execution when several active rules in \( D \) are ready to be executed. However, if a particular active rule \( TR_r \) has a \( \downarrow \) priority, then the sequence of actions in its action part will be selected by \( S \).

Observe that if \( S \) selects a sequence of actions which does not have a processing point at the end of the list, no new rules will be allowed to fire at the end of executing the selected sequence (i.e. the rules in \( \Omega \) will have to wait until a new processing point is encountered). With minor modifications to the definition of models we could assume that rules by default are processed each time we get into an empty sequence of actions (so that rules will be processed at least at the end of the transaction) in addition to the explicit processing points. Furthermore, we can put a restriction on a syntax which will require that every sequence of actions must end with a \( \uparrow \) symbol.

The query language associated with \( L_{active} \) consists of hypothetical facts of the form:

\[
  f \text{ after } \alpha \text{ at } \Sigma
\]

where \( f \) is a fact literal, \( \alpha \) a sequence of actions, and \( \Sigma \) a fact state. For a query \( q \), we will denote by \( \neg q \) the query \( g \text{ after } \alpha \text{ at } \Sigma \) if \( f \) in the query is \( \neg g \). If \( f \) is positive, it denotes \( \neg f \text{ after } \alpha \text{ at } \Sigma \).

**Definition 1** We say that a query \( q \) of the form (5) is true in a model \( \Psi \) of an active database description \( D_{\mathcal{CAR}} \) iff \( f \) holds in the fact state of the state \( \Psi(\alpha, < \Sigma, \emptyset, \emptyset, \emptyset >) \).

**Definition 2** An active database description \( D_{\mathcal{CAR}} \) entails a query \( q \) (written as \( D_{\mathcal{CAR}} \models q \)) iff \( q \) is true in all models of \( D_{\mathcal{CAR}} \). The set of all facts entailed by \( D_{\mathcal{CAR}} \) will be denoted by \( \text{Con}(D_{\mathcal{CAR}}) \).

7 Declarative Semantics of \( \mathcal{CAR} \)

Recall that (c.f. Section 4), we assume that we have a set of ground instances (i.e., each variable is substituted with a proper constant value) of the propositions in the domain description under consideration. Also, recall that \( \Sigma \) is an event state, \( \Phi \) a fact state, \( \Pi \) is the set of active rules which are enabled (i.e., their triggering event is present in the current state), \( \Theta \) is the set of enabled (triggered) rules which are available (eligible) for consideration and \( \Omega \) is the set of considered rules whose condition evaluated to true and are ready to be fired (i.e., to have the sequence of their action executed). A state \( S \) is a quintuple of the form: \( S = < \Phi, \Sigma, \Pi, \Theta, \Omega > \), affects each component of the state. Furthermore, we will also need two selection functions \( S_C \) and \( S_E \). The selection function \( S_C \) is a total function that takes a given state \( S \) as its input and returns one of the rules \( R_{SC} \) from \( \Theta \). On the other hand, the selection function \( S_E \), applied to a given state \( S \), returns an active rule from \( \Omega \). \( S_E \) is accompanied by an action function \( S_E \) which, when applied to a given rule (the one selected by \( S_E \)) returns a sequence of actions corresponding to the action part of that rule. Selection functions will be used to determine which rules and actions sequence will be selected.
for *consideration* and *execution* when a choice has to be made among several active rules from the domain $D_{CAR}$. They can be implemented using the *choice* operator (c.f. [81]) and/or based on the *priority* assigned to the rules upon their creation.

We start with the definition of the function that describes the effects of an action $a$ when executed in a given state. Actually, this function only depends on $\Phi$ and $\Sigma$ – the other parts of the state are irrelevant. First we need the following definitions. We say that a fact literal $f$ is an (immediate) *effect* of (executing) $a$ in a state $\mathcal{S}$ if there is an effect proposition $a$ *causes* $f$ if $p_1, \ldots, p_n$ in $D$ whose preconditions $p_1, \ldots, p_n, e_1, \ldots, e_m$ hold in the $< \Phi, \Sigma >$ part of $\mathcal{S}$. Let

$$F^+_a(\Phi) = \{ f : f \in \mathcal{F} and f is an effect of a in < \Phi, \Sigma > \},$$

$$F^-_a(\Phi) = \{ f : f \in \mathcal{F} and \neg f is an effect of a in < \Phi, \Sigma > \}$$

However, in the presence of deductive rules, we must take into consideration the changes of the IDB due to the changes of the EDB as a result of the immediate consequences of executing the action $a$. As we mentioned in Section 2, some facts may be removed from $\Phi$ due to the negation, however, we assume that the computation eventually terminates at a given fixpoint. Let $F^{aTota}_{a}$ denote all the positive literals from $\mathcal{F}$ which resulted after the termination of the fixpoint computation following the action of the action $a$ in a given state $< \Phi, \Sigma >$. Similarly, let $F^{-aTota}_{a}$ denote all the negative literals from $\mathcal{F}$ which resulted after the termination of the fixpoint computation following the application of the action $a$ in a given state $< \Phi, \Sigma >$. We have:

$$ResF(a, \Phi) = (\Phi \cup F^{aTota}_{a}(< \Phi, \Sigma >)) \setminus F^{-aTota}_{a}(< \Phi, \Sigma >).$$

$ResF$ is referred to as the *facts transition function*.

The second function defines the changes on the set of events. We say that an event literal $e$ is an (immediate) *effect* of (executing) $a$ in a state $< \Phi, \Sigma >$ if there is an event effect law $e$ after a if $e_1, \ldots, e_m, q_1, \ldots, q_n$ in $D_{CAR}$ whose preconditions $q_1, \ldots, q_n$ hold in $\Phi$ and $e_1, \ldots, e_m$ hold in $\Sigma$. Let

$$E^+_a(< \Phi, \Sigma >) = \{ e : e \in \mathcal{E} and e is an effect of a in < \Phi, \Sigma > \},$$

$$E^-_a(< \Phi, \Sigma >) = \{ e : e \in \mathcal{E} and \neg e is an effect of a in < \Phi, \Sigma > \}.$$

These two sets identify the events directly generated or removed by $a$. However, similarly to the application of the deductive rules for inference of the facts from the IDB, now we have to apply the expressions from the Event Algebra which detect the occurrences of composite events from the $CE$ part of $\Sigma$ (c.f. Section 2). Let

$$E^+_{aCE}(< \Phi, \Sigma >) = \{ e_c : e_c \in (\mathcal{E} \cup CE) and there is an event-algebra expression whose LHS = e_c and RHS is satisfied in $\Sigma$ \}$$

$$E^-_{aCE}(< \Phi, \Sigma >) = \{ e_c : e_c \in (\mathcal{E} \cup CE) and there is an event-algebra expression whose LHS = e_c and RHS is not satisfied in $\Sigma$ \}$$

We have: $ResE(a, < \Phi, \Sigma >) = (\Sigma \cup E^+_a(< \Phi, \Sigma >) \cup E^+_{aCE}(< \Phi, \Sigma >)) \setminus (E^-_a(< \Phi, \Sigma >) \cup E^-_{aCE}(< \Phi, \Sigma >))$

We will refer to $ResE(a, < \Phi, \Sigma >)$ as event transition function and we will use the abbreviations

$$E^{aTota}_{a}(< \Phi, \Sigma >) = E^+_a(< \Phi, \Sigma >) \cup E^+_{aCE}(< \Phi, \Sigma >)$$

$$E^{-aTota}_{a}(< \Phi, \Sigma >) = E^-_a(< \Phi, \Sigma >) \cup E^-_{aCE}(< \Phi, \Sigma >).$$

At this point, based on the values of $E^{aTota}_{a}(< \Phi, \Sigma >)$ which are used as enabling (triggering) events, the *Meta-Trigger Control Module* changes the value of the $A_C$ parameter of each active rule $TRr_{rk}$ whose “dual-counterparts” $DTTRr_{rk}$ and $ATTRr_{rk}$ were being available for consideration, as specified in Section 4.

So far we did not consider the effects that the execution of the action $a$ in a state $\mathcal{S}$ has on the rules which are *enabled* (i.e. the set $\Pi$). Once an action is executed, a rule may become *enabled* (eligible for consideration) and, consequently, added to $\Pi$, if its triggering event $e_t$ became part of $\Sigma$. On the other hand,
if an event was removed from $\Sigma$ (i.e., $\neg e$ is the result of executing the action $a$), then a rule may have to be removed from $\Pi$. Let $R_i$ denote an arbitrary rule from $R$. We have:

$$P^+(E_a^+ \text{total}(< \Phi, \Sigma, \Pi >)) = \{ R_i : e_i^R_i \in E_a^+ \text{total}(< \Phi, \Sigma, \Pi >) \}$$

$$P^-(E_a^- \text{total}(< \Phi, \Sigma, \Pi >)) = \{ R_i : e_i^{-} \in E_a^- \text{total}(< \Phi, \Sigma, \Pi >) \}$$

The enabled rules transition function is specified as:

$$Resp(a, < \Phi, \Sigma, \Pi >) = (P \cup P^+(E_a^+ \text{total}(< \Phi, \Sigma, \Pi >))) \setminus P^-(E_a^- \text{total}(< \Phi, \Sigma, \Pi >))$$

Now we are ready to proceed towards taking into account the effects which the execution of an action $a$ in a given state $\mathcal{S}$ has on the rules that are to be considered (i.e., whose condition part may be evaluated). Given a rule $\text{“}R^j(A_C) : e_i^C$ consumed $$(C_s) \text{ if } p_1^e, \ldots, p_n^e \text{ at } e_i^C$ \text{ initiates } [a]$$ at $e_i^{S_a}$"$, it is available for consideration if $e_i^C \in E_a^+ \text{total}(< \Phi, \Sigma >)$, and $A_C^j = \text{yes}$. Similarly, regardless of the presence of its condition-evaluation event, if the value of $A_C^j$ is no, then it will not be eligible for consideration. Moreover, if its condition-evaluation event $e_i^C$ is removed from $\Sigma$, the rule $R^j$ is no longer available for consideration. Thus, we have:

$$T^+(a, < \Phi, \Sigma, \Pi, \Theta >) = \{ R^j : R^j \in Resp(a, < \Phi, \Sigma, \Pi >) \text{ and } e_i^C \in Resc(a, < \Phi, \Sigma >) \text{ and } A_C^j = \text{yes} \}$$

$$T^-(a, < \Phi, \Sigma, \Pi, \Theta >) = \{ R^j : R^j \notin Resp(a, < \Phi, \Sigma, \Pi >) \text{ or } e_i^C \notin Resc(a, < \Phi, \Sigma >) \}$$

The consideration-available transition function is specified as:

$$Rest(a, < \Phi, \Sigma, \Pi, \Theta >) = (\Theta \cup T^+(a, < \Phi, \Sigma, \Pi, \Theta >)) \setminus T^-(a, < \Phi, \Sigma, \Pi, \Theta >)$$

Now we are ready to select one of the rules from $\Theta$, which will have its condition part evaluated and, if true, will be available for executing its action part. Recall that we rely on the function $S_C$ for it. Let $S_C(< \Phi, \Sigma, \Pi, \Theta >) = R^{Sc}$, and assume that the rule $R^{Sc}$ has the format $\text{“}R^{Sc}(A_C) : e_i^{Sc}$ consumed $$(C_s) \text{ if } p_1^{Sc}, \ldots, p_n^{Sc} \text{ at } e_i^{Sc}$ \text{ initiates } [a]$$ at $e_i^{S_a}$. Depending on the value of the parameter describing the consumption scope $C_s$ for the triggering event of $R^{Sc}$, some rules will have to be removed from both of the sets $\Pi$ of the enabled rules, and $\Theta$ - the rules eligible for consideration. If $C_s = \text{local}$, then $R^{Sc}$ is de-triggered (no longer enabled) and, consequently, not available for subsequent considerations. On the other hand, if $C_s = \text{global}$, then the set of rules which are removed from $\Pi$ and $\Theta$ becomes:

$$\{ R_i : e_i^{R_i} = e_i^{Sc} \}$$

Let $\Theta_g$ (respectively, $\Pi_g$) denote the set of all the rules which have to be removed from $\Theta$ (respectively, $\Pi$) due to a selection for consideration of a rule $R^{Sc}$ whose consumption mode is $C_s = \text{global}$. Similarly, let $\Theta_l$ (respectively, $\Pi_l$) denote the set of all the rules which have to be removed from $\Theta$ (respectively, $\Pi$) due to a selection for consideration of a rule $R^{Sc}$ whose consumption mode is $C_s = \text{local}$ - both of which are singleton sets. As a result of applying an action $a$ in a given state $\mathcal{S}$, we have the following removal transitions due to the values of the consumption scope of a given rule selected for consideration:

$$REM^{H}_{Sc}(a, < \Phi, \Sigma, \Pi, \Theta >) = (\Theta_g \cup \Theta_l)$$

$$REM^{G}_{Sc}(a, < \Phi, \Sigma, \Pi, \Theta >) = (\Pi_g \cup \Pi_l)$$

Thus, the result of applying the selection function $S_C$ for consideration of a rule, after executing an action $a$ in a given state $\mathcal{S}$ is:

$$S_C(a, < \Phi, \Sigma, \Pi, \Theta, \Omega >) =$$

$$< Resp(a, \Phi),$$

$$Resc(a, < \Phi, \Sigma, \Pi >),$$

$$Resp(a, < \Phi, \Sigma, \Pi >) \setminus REM^{H}_{Sc}(a, < \Phi, \Sigma, \Pi, \Theta >),$$

$$Rest(a, < \Phi, \Sigma, \Pi, \Theta >) \setminus REM^{G}_{Sc}(a, < \Phi, \Sigma, \Pi, \Theta >),$$

$$\Omega >$$

Once the rule $R^{Sc}$ is selected for consideration, its condition part is evaluated. If it evaluates to false, the Meta-Trigger Control Module is invoked to set the value of $A^{Sc}_C$ to no, provided that rule was specified
as the one which intends to utilize the dual-events of its condition (i.e. \( C_D = \text{yes} \)). Subsequently, the Meta-Trigger Control Module will set the values of the \( A_C \) parameters for each of the \( DR^S_C \) and the \( AR^S_C \) to yes. However, if the condition of the selected rule \( R^S_C \) evaluates to true, then it becomes eligible for executing its action part and, consequently, it is added to the set \( \Omega \). Observe that some rules may be removed from \( \Omega \) if their action-execution events are no longer in the set \( \Sigma \) after \( a \) is executed. Thus, we have:

\[
O^+_a(\Phi, \Sigma, \Pi, \Theta, \Omega) = \{ R^S_C \}
\]

\[
O^-_a(\Phi, \Sigma, \Pi, \Theta, \Omega) = \{ R^j : e^j_a \not\in Res_E(a, < \Phi, \Sigma > ) \}
\]

Consequently, the action-execution transition function is specified as:

\[
Res_O(a, < \Phi, \Sigma, \Pi, \Theta, \Omega>) = Res_O(a, < \Phi, \Sigma, \Pi, \Theta, \Omega>) \cup O^+_a(\Phi, \Sigma, \Pi, \Theta, \Omega) \setminus O^-_a(\Phi, \Sigma, \Pi, \Theta, \Omega)
\]

Given a quintuple \(< \Phi, \Sigma, \Pi, \Theta, \Omega >\), the selection function \( S_E \) selects a rule from \( \Omega \) for execution, only if its action event is part of the events set \( \Sigma \). In other words \( S_E(\Phi, \Sigma, \Pi, \Theta, \Omega) = R^S_E \), such that \( e^R_E \in \Theta \).

On the other hand, given a (selected for execution) rule \( R^S_E \), the function \( S'_E \) returns the sequence of the actions from the action-part of \( R^S_E \). In other words, \( S'_E(S^R_E) = [\alpha R^S_E] \).

Now, we are ready to describe the dynamics of the evolution of the environment specified in the CAR language, when it is subjected to an execution of a sequence of actions.

**Definition 3** A causal interpretation \( \Psi \) is a model for a domain description \( D_{CAR} \) iff for any state \( S = < \Phi, \Sigma, \Pi, \Theta, \Omega > \), there exist action selection function \( S_C \) and \( S_E \), such that for any sequence of actions \( \phi \)

1. if \( \phi = [] \) then \( \Psi(\phi, S) = S \).
2. if \( \phi = \uparrow \circ \beta \) then
   \[
   \Psi(\phi, S) = \Psi(\beta, \Psi(S_E(S_E(S), < \Phi, \Sigma, \Pi, \Theta, \Omega \setminus S_E(S) > ))).
   \]
3. if \( \phi = a \circ \beta \) and \( a \not\uparrow \) then
   \[
   \Psi(\phi, S) = \Psi(\beta, < Res_F(a, < \Phi, \Sigma >),
   Res_E(a, < \Phi, \Sigma >),
   Res_P(a, < \Phi, \Sigma >),
   Res_O(a, < \Phi, \Sigma, \Theta, \Omega >))
   \]

To avoid contradiction, \( \Psi \) is defined only for the states in which \( F^+_{a}(\Phi) \cap F^-_{a}(\Phi) = \emptyset \) and \( E^+_{a}(\Phi, \Sigma) \cap E^-_{a}(\Phi, \Sigma) = \emptyset \). Otherwise is undefined for the state.

8 Conclusions, Related Work and Future Goals

We introduced two novel spatio-temporal (or, dynamic topological) predicates used in Requests for Notification and presented efficient algorithms for their processing. Along with that, we addressed the issue of the efficiency of the overall (dynamics of the) reactive behavior in MOD settings. Motivated by the analysis of the issues involved in the processing of the two particular predicates, we demonstrated that for some requests, besides the dynamics of the (location, time) information of the objects, another dimension of the dynamics of the environment may have to be taken into consideration, although it is not explicitly stated in the particular request. To address this, we demonstrated the inadequacy of the “classical” ECA paradigm for specifying the reactive behavior in MOD settings and we proposed the \((ECA)^2\) paradigm.

To the best of the authors knowledge, this is a first attempt to formalize this kind of reactive behavior in MOD settings. We believe that we have just scratched a surface of a topic that can generate a very fruitful research both in theoretical and practical aspects.
We cannot really compare this work with any existing literature, however, there are many results for which the line of research that we initiated is complementary, and vice versa. Thus, in the sequel, we will intertwine the survey of the relevant literature and the future goals.

The topic of active databases has been extensively studied for a long time [12, 20, 28, 56, 79] and various aspects have been investigated: – termination and confluence [75]; – coupling modes between transactions which generated events vs. condition evaluation and actions execution [28]; – event processing and consumption [15, 54]; – expressiveness issues [58] and semantics of the active rules behavior, for which several formalisms have been used, e.g., action theories [5] and temporal logic [67]. However, at the time when the research in the field of Active Databases was at its peak, the research on Moving Objects Databases was barely at its infancy. Due to the specific nature of the spatio-temporal domain, none of the works can be applied directly to the MOD settings, however, as we indicated throughout the paper, they provide a solid foundation for extension of the management of reactive behavior in MOD settings. In particular, the Extended Event-Condition-Action (EECA) model in [28] takes a step towards more “event-aware” active rules. However, although it utilizes the notion of querying over the history of the evolution based on specification of the proper events, the work does not have the full concept of composite events (i.e., it only considers a disjunction of primitive events). However, none of the works addressed the problem of evolution of the triggers and their context-awareness.

In the last few years the MOD research has brought many interesting results and the recent collection [45] provides an extensive list of references. However, none of these works has explicitly addressed the problem of (management of) continuous requests in a reactive manner. The continuous queries in MOD settings were introduced in [66], however, the work also introduced the category of persistent queries for which, we believe, our work provides a good paradigm for their management.

The works [10, 53, 80] are addressing the problem of efficient maintenance of the answers of continuous queries in the settings in which the Motion Plan of the moving objects is obtained by a sequence of location updates in time. Each of them tries to minimize the computational overhead, in the sense that they focus on calculating the changes to the answer-sets of the continuous queries due to the updates of the moving objects’ locations in time, thus avoiding their complete re-evaluations. The main focus is on scalability [53, 80], which is, efficient processing in a presence of multiple query requests and location updates. In particular [10] investigates the trade-off between location updates and server processing costs while maintaining accuracy of the query results. Although we did not address the issues of scalability, our work is, in a sense, complementing the approaches in [10, 53, 80] because our paradigm can be applied for processing continuous queries in which the conditions evolve too.

Many works on Event-based and Context-Aware Notification systems like, for example the GUIDE project [18, 19], TIP system [43], CATIS system [55] (to name a few) are concerned with the filtering, formatting and delivering the desired information to a mobile user based on his current location, preferences and communication device. However, these works mostly match static conditions with the dynamically changing information of the user’s whereabouts, unlike this work where, along with the user’s location, the condition are also changing dynamically.

Our earlier work [70, 69] on CAT system has addressed the problem of managing continuous spatio-temporal range queries using triggers. However, the proposed framework assumed that the Motion Plan of each mobile user was represented as a complete (trajectory) and the main focus of the context-awareness was to correctly identify and update the trajectories that may be affected by abnormal traffic conditions and, subsequently, update the answer sets of the queries which may change due to the update of the affected trajectories. The works did not consider any conditions that may change in a dynamic manner.

In this work we only presented three specific Requests for Notification and in all of them we assumed a model which represents the user’s Motion Plan as a sequence of GPS-like (location,time) update points. What is the impact of the different models of the mobility (e.g., what if one considers the existing road networks [22, 76]) on the types of requests and their processing? That is part of our ongoing work. Another
A challenging aspect for our future work is considering a comprehensive type system and carefully identifying the set of operators as well as their processing aspects for various argument signatures, similar in spirit to the sequence of works \cite{27, 35, 47}.

The implementation of the \((ECA)^2\) paradigm will need mechanisms for efficient querying over a history (sequence) of database states which, sometimes, may have to search for patterns of, say, occurrences of events. Two recent results that could be useful along these lines are: \cite{62}, where the authors extend the Knuth, Morris and Pratt text-search algorithm to handle complex queries (expressed in SQL-TS) over database sequences; and \cite{25, 24} where the authors propose classification model for trajectories based on a multi-scale map and describe a query language based on regular expressions for specifying mobility patterns, along with the processing algorithm (based on a finite state automata). The work of \cite{62} cannot be directly applied to the domain of problems that we addressed because, despite the efficiency of querying the history, it is not capable of a reactive behavior. \cite{25, 24} are more “reactive-aware” in the sense that they detect patterns of trajectories “on-the-fly”. However, they did not address the issue of monitoring other types of conditions and executing any actions in turn.

The notion of evolution over time of various spatial objects (e.g., county boundaries, rivers) along with the changes of the attributes in the populations (e.g., the salary or address of a person) is addressed in \cite{23}. The work presents various bindings of inter/intra history querying over heterogeneous data as well as cross-time querying for certain patterns of change. The underlying platform is TRIPOD, subsequently presented in \cite{32, 33}. However, these works are not addressing the issue of a reactive behavior and are not targeted towards environments which are highly dynamic. As such, they are not applicable to notification systems in MOD settings.

Although we used examples from the MOD settings, we believe that the \((ECA)\) paradigm is attractive for any dynamic environment, like sensor networks, where the data is generated in a stream-like manner and the system needs to monitor continuous and/or persistent queries of interest and react efficiently. Due to the specific nature of this domain, the \((ECA)^2\) paradigm will have to be made robust to errors/uncertainty and, in particular, be able to deal with non-instantaneous events \cite{7} and QoS issues (e.g., how to detect and remove triggers which are obsolete). On the theoretical aspect, it is an interesting challenge to investigate the expressive power of the \((ECA)^2\) rules (in the spirit of \cite{59}) and to try to formally characterize the active rules which allow arbitrary levels nesting of the ELSE parts and/or the option for ELSEIF. When considering novel paradigms, one has to think about how the solutions can be “brought to life” in an actual implementation. A recent survey of spatio-temporal database management systems is presented in \cite{8} and, along the terminology used there, we believe that, to some extent, the paradigm the we proposed in this work can be implemented on top of an extensible ORDBMS. Clearly, this has some advantages – to say the least, pushing the processing of the rule as much as possible into the underlying ORDBMS (c.f. \cite{78}). However, in this case, one carefully needs to investigate the limits that the underlying ORDBMS enforce on the semantics and expressiveness issues of the \((ECA)^2\) paradigm. Another possible approach is to implement the reactive behavior from the scratch. For example, from the perspective of flexibility of type systems, the SECONDO \cite{38} seems like a possible choice of a development platform (which, in turn, could itself benefit from having a framework for managing reactive behavior). At the current stage, due to our experiences with the CAT system \cite{70} we are focusing on extending the trigger capabilities of the Oracle server.

For the sake of historic honesty, part of the motivation (along with the coincidence in part of the name) for the \((ECA)^2\) paradigm came from the research on Evolving Algebras \cite{34} as a tool to formalize the operational semantics of programming languages\footnote{Subsequently, the project became Abstract State Machines and is integrated in the Microsoft .net platform research.microsoft.com/jsa/asml as an executable specification language.}. Lastly (but not “least-ly”) we would like to point to, what seems at this moment, an interesting theoretical research challenge. What is the limit of the \((ECA)^2\)? In the lieu of the monadic metalanguages, where the goal is arbitrary interleaving of computation and code generation steps \cite{52}, it seems tempting to investigate a full-fledge \((ECA)^2\) where triggers can be used to modify the
events and actions and, moreover, completely specify other triggers. What are the MOD-based constraints on it, what would be the expressive power of such a paradigm and what are the application domains where it could be useful are part of the questions that, we hope, will be addressed in the future.

References


