TPLP **19** (5–6): 841–856, 2019. © Cambridge University Press 2019
doi:10.1017/S147106841900022X

Online Event Recognition from Moving Vehicles: Application Paper

EFTHIMIS TSILIONIS
National Center for Scientific Research 'Demokritos', Athens, Greece,
(*e-mail:* eftsilio@iit.demokritos.gr)

NIKOLAOS KOUTROUMANIS, PANAGIOTIS NIKITOPOULOS and CHRISTOS DOULKERIDIS
University of Piraeus, Piraeus, Greece,
(*e-mail:* {koutroumanis,nikp,cdoulk,}@unipi.gr)

ALEXANDER ARTIKIS
National Center for Scientific Research 'Demokritos', Athens, Greece, and University of Piraeus, Piraeus, Greece,
(*e-mail:* a.artikis@unipi.gr)

submitted 25 July 2019; accepted 31 July 2019

Abstract

We present a system for online composite event recognition over streaming positions of commercial vehicles. Our system employs a data enrichment module, augmenting the mobility data with external information, such as weather data and proximity to points of interest. In addition, the composite event recognition module, based on a highly optimised logic programming implementation of the Event Calculus, consumes the enriched data and identifies activities that are beneficial in fleet management applications. We evaluate our system on large, real-world data from commercial vehicles, and illustrate its efficiency.

KEYWORDS: Event Pattern Matching, Event Calculus, Data Enrichment

1 Introduction

The European economy relies to a great extent on commercial vehicle fleets. According to the European Automobile Manufacturers Association[1], there were over 54 Million commercial vehicles in use in Europe in 2015, and this number is growing every year. Commercial vehicles are equipped with devices emitting information regarding their location and operational status, such as speed and fuel level. Fleet management applications collect the information emitted from moving vehicles in order to improve the management and planning of transportation services, and enable informed decision-making. Detecting composite events from such data streams can be beneficial for the drivers of commercial vehicles, since they can be informed about their performance, and even prevent dangerous situations. Additionally, the analysis of data generated by such a fleet of vehicles, can help the owners maximize the performance of the fleet.

[1] http://www.acea.be/statistics/article/vehicles-in-use-europe-2017

However, the data produced by a fleet of vehicles is not always sufficient on its own to support advanced vehicle monitoring. External data sources, such as weather information or proximity to points of interest (POIs), can have a significant effect on the movement of the vehicles. For example, fleet management applications can estimate better the fuel consumption of the fleet, by taking into consideration weather information. Furthermore, informing about the presence of locations of interest in a close distance, such as gas stations, can be a significant help both for drivers and fleet operators. Therefore, the integration of positional information with external data sources allows for improved monitoring.

In the context of the Track & Know project[2], we develop an online fleet management system for the recognition of composite events, that improves the operating efficiency of a commercial fleet. Our system utilizes the GPS (Global Positioning System) traces of moving vehicles along with information emitted by an installed accelerometer device, such as an abrupt acceleration, and information concerning the level of fuel in a vehicle's tank provided by a fuel sensor. These traces are enriched with weather and POI information by a dedicated component for data enrichment. The enriched data are provided as input to a composite event recognition (CER) component, which is based on the 'Event Calculus for Run-Time Reasoning' (RTEC). This is a logic programming implementation of the Event Calculus (Kowalski and Sergot 1986) with optimizations for continuous narrative assimilation on data streams (Artikis et al. 2015; Tsilionis et al. 2019). The contributions of this paper are then the following:

- We provide a high-throughput and scalable solution for the enrichment of mobility data with weather information and nearby POIs.
- We present a stream reasoning system integrating the component for data enrichment and a logic programming component for recognizing composite events.
- We illustrate our approach using large, real-world, heterogeneous data streams concerning commercial vehicles. The evaluation validates the robustness and scalability of the system as well as its capacity to operate in real-time.

The remainder of this paper is organized as follows. Section 2 discusses related work, while Sections 3 and 4 present the main system components. Section 5 presents our empirical evaluation. Finally, Section 6 discusses the challenges that we faced during the system development.

2 Related Work

Data enrichment is considered as part of a larger process known as *data integration*, which is a challenging topic, in particular in the context of data sources that provide large volumes of data, often in streaming mode, and in heterogeneous formats (Dong and Srivastava 2015). Unfortunately, despite the significance of integrating mobility data with weather, there is a lack of publicly available and reusable systems or tools; our work on weather data integration (Koutroumanis et al. 2019) aims to address this limitation. Regarding the enrichment of GPS traces with static locations, also known as points of interest, the problem is essentially known as *distance join*, a variant of spatial joins (Jacox and Samet 2007), where records from two data sets are joined if their distance is below a user-specified threshold. Parallel processing of distance join is typically

[2] https://trackandknowproject.eu/

performed in two ways: (a) by repartitioning both data sets to processors in a way that guarantees the correctness of the result, when partitions are processed independently, or (b) by partitioning one data set and broadcast the other to all processors. The latter technique is usually preferred when one of the data sets is relatively small, and we adopt this method here.

Composite event recognition (CER) systems accept as input a stream of time-stamped, 'simple, derived events', such as events coming from sensors of moving vehicles, and identify composite events (CE)s of interest — collections of events that satisfy some pattern. The definition of a CE imposes temporal and, possibly, atemporal constraints on its sub-events (simple, derived events or other CEs). Numerous CER systems and languages have been proposed in the literature. See (Cugola and Margara 2012; Alevizos et al. 2017; Giatrakos et al. 2019) for three surveys. These systems have a common goal, but differ in their architectures, data models, pattern languages and processing mechanisms (Grez et al. 2019). For example, many CER systems provide users with a pattern language that is later compiled into some form of automaton (Demers et al. 2007; Zhang et al. 2014; Schultz-Møller et al. 2009; Apache FlinkCEP[3]). The automaton model is used to provide the semantics of the language and/or as an execution framework for pattern matching. Apart from automata, some CER systems employ tree-based models (Liu et al. 2011; Mei and Madden 2009). Again, tree-based formalisms are used for both modeling and recognition, i.e., they may describe the event patterns and the applied recognition algorithm.

Logic-based approaches to CER have also been attracting considerable attention, since they exhibit a formal, declarative semantics, and at the same time support efficient reasoning (Dousson and Maigat 2007; Cugola and Margara 2010; Paschke and Bichler 2008). We adopt the 'Event Calculus for Run-Time reasoning' (RTEC) for our CER engine (Artikis et al. 2015), a logic programming implementation of the Event Calculus (Kowalski and Sergot 1986), that has been used in various application domains, such as maritime monitoring (Patroumpas et al. 2017). CE patterns in RTEC are (locally) stratified logic programs. RTEC explicitly represents CE intervals (unlike e.g. Dousson and Maigat 2007; Cugola and Margara 2010; Beck et al. 2018) and thus avoids the related logical problems (Paschke 2006). Moreover, and in contrast to state-of-the-art recognition systems, such as the Esper[4] engine and SASE (Zhang et al. 2014), RTEC can naturally express hierarchical knowledge by means of well-structured specifications, and consequently employ caching techniques to avoid unnecessary re-computations.

Concerning the Event Calculus literature, a key feature of RTEC is that it includes a windowing technique. No other Event Calculus system (including Chittaro and Montanari 1996; Cervesato and Montanari 2000; Miller and Shanahan 2002; Paschke and Bichler 2008; Artikis and Sergot 2010; Montali et al. 2013) 'forgets' or represents concisely the data stream history.

3 Data Enrichment

The architecture of our system for online fleet management is depicted in Figure 1. The main input is streaming GPS traces from a fleet of moving vehicles, typically provided by

[3] `https://ci.apache.org/projects/flink/flink-docs-stable/dev/libs/cep.html`
[4] `http://www.espertech.com/esper/`

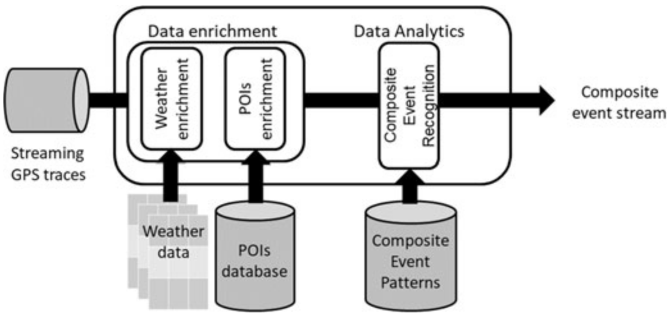


Fig. 1: The system architecture for online event recognition from moving vehicles.

a fleet management application. As this streaming data flows in the system, it is enriched with external information, mainly weather data and proximity to points of interest. The enrichment process augments the GPS traces with valuable information, which can be exploited for identifying patterns of composite events (CEs) that would otherwise remain hidden. Subsequently, a CER module consumes the stream of enriched GPS positions to identify CEs. Moreover, the system architecture is implemented on top of scalable big data frameworks (e.g., Kafka, Spark), thereby exploiting parallelism for data operations either at the level of a cluster of computers or at the level of a single computer (by means of multi-threading).

In the context of this work, data enrichment consists of two modules: weather data enrichment and point of interest (POI) enrichment. The input is GPS traces from a set of moving vehicles, which contain vehicle id ($v.id$), position $v.loc = (v.x, v.y)$ and timestamp ($v.t$), as well as other attributes (e.g., speed, acceleration, etc.). The output contains the same set of records, enriched with additional attributes. First, a set of weather attributes, selected according to needs of the application[5]. Essentially, for each position $v.loc = (v.x, v.y)$ and timestamp ($v.t$) of a vehicle, we retrieve the values of weather attributes. Second, each position $(v.x, v.y)$ is enriched with a set of POIs $\{p_i\}$ that are located within a user-specified distance threshold θ, i.e., $d(v.loc, p_i) \leq \theta$.

3.1 Weather Enrichment

The weather enrichment module operates in an online manner, by processing the GPS traces record-by-record, as they arrive in the stream. Internally, its logic is split in two sub-modules; the *Spatio-temporal parser*, which is responsible for extracting the position ($v.loc$) and timestamp ($v.t$) from the input record, and the *Weather data obtainer*, which is responsible for the retrieval of weather attribute values associated with the specific spatio-temporal position $(v.loc, v.t)$.

The Spatio-temporal parser, parses each record of the input data and performs some basic data cleaning operations. It checks the spatio-temporal part both for its existence (null or empty values) and validity (valid longitude and latitude values). Both checks are necessary, as GPS traces are typically noisy and may contain errors. If a value is not valid or missing, then the parser ignores the entire record, and continues with the next

[5] In this paper, we are mostly interested in events and their relationship to ice-related attributes.

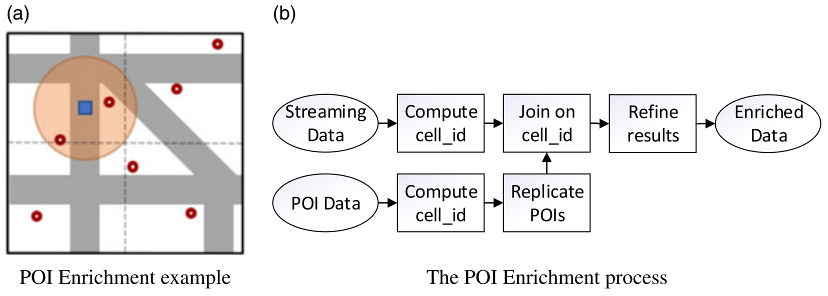


Fig. 2: POI Enrichment.

one. Each record with valid spatial and temporal information is passed to the Weather data obtainer sub-module, which is responsible of fetching the weather attribute values from the weather data source.

Weather data is provided as GRIB-formatted files that store gridded meteorological data in binary form. GRIB files are provided by the National Oceanic and Atmospheric Administration (NoAA), which contains data from computer-generated, numerical weather prediction models. Weather attributes are represented as values on a 2-dimensional (2D) spatial grid divided into cells, where each cell is mapped to a specific geographical area. We use GRIB files that provide the highest spatial resolution, namely $0.5° \times 0.5°$. Each day is composed of 4 GRIB files, which are based on the 4 distinct forecast models that run on a daily basis, with times 00:00, 06:00, 12:00, and 18:00. Each GRIB file contains weather attribute values whose validity is for 3 hours after the forecast, i.e., 03:00, 09:00, 15:00, and 21:00, respectively.

The Weather data obtainer maintains a tree data structure in-memory, organizing the references (paths) of each GRIB file based on their reference time. For example, a file with time 00:00 contains a forecast for 03:00. Given a timestamp $(v.t)$, the tree is searched to locate the nearest GRIB file in terms of its reference time. For instance, given a timestamp 05:10, the forecast at 03:00 is considered as nearest in time, rather than the one at 09:00. Then, this file is accessed in order to fetch the value corresponding to the location $(v.loc)$ at hand. Since there is an overhead when opening a GRIB file, a caching mechanism is used for maintaining handles to open files. This is beneficial for sequential requests that are served from the same GRIB file, as repeated open/close operations are avoided, thus saving processing time. For a detailed presentation of the architecture of the weather enrichment module refer to Koutroumanis et al. 2019.

3.2 POIs Enrichment

The POI enrichment is implemented as an Apache Spark Structured Streaming job to improve efficiency through parallelized processing. It takes as input (a) the streaming spatio-temporal data set of moving vehicles, (b) a set of POIs containing their spatial information, and (c) a distance threshold θ expressed in meters. The POI data set, provided by OpenStreetMap, refers to static points of interest described by their spatial location $p.loc = (p.x, p.y)$, name and type of POI. The POI enrichment aims to enrich the spatio-temporal GPS traces of moving vehicles with the information of POIs located at maximum distance θ from any trace. An example is depicted in Figure 2a, where the blue rectangle represents a vehicle moving through a city's road network, and the red small circles refer to various places of interest. The circle centered at the vehicle's location with radius θ encloses all POIs which are located at maximum distance θ from the vehicle. Hence, our goal is to efficiently identify these nearby POIs and add them to the trace information of the vehicle.

Essentially, the POI enrichment process evaluates a distance join query over the streaming spatio-temporal data set of GPS traces and the static data set of POIs with a maximum distance threshold θ. A naive solution to this problem would join the POI data set with the entire streaming spatio-temporal data set, and then filter out the records that have a joined distance higher than θ. This solution, however, inflicts high computation cost of $O(n \cdot m)$, where n is the number of POIs and m is the number of traces in the streaming spatio-temporal data set, thus reducing the efficiency of our solution.

We propose a more efficient algorithm for computing the distance join query, demonstrated in Figure 2b. Our premise is to employ a grid that partitions the spatial space into equally sized cells. All records from both data sets can be easily assigned a cell id, based on their associated spatial information. Since the POI data set is static (i.e. does not change while processing the streaming data set), we start by distributing its records to the available computing nodes, based on their corresponding cell ids. We keep the POI records in the nodes main memories to enable fast retrieval later. Then, we start processing the spatio-temporal data set, by distributing every streaming record to the corresponding node, based on the computed cell id. That node performs a join operation between the trace record and all the POI records based on their cell id values. The goal is to evaluate the trace's distance join result on a single node, thus reducing the communication complexity of the join operation. To this end, we opt to replicate all POI records to nearby cells, located at maximum distance θ from the POI. This results to a new POI data set where every cell id is associated with all the POIs located at maximum distance θ from the cell. Hence, the aforementioned join operation is guaranteed to process all candidate results, without needing any additional communication between the nodes. The last step is to refine the results, by filtering out the records that have a joined distance higher than θ. The computation complexity of this algorithm is significantly reduced to $O(c \cdot m)$, where c is the average number of POIs in a cell, and m is the number of traces in the streaming spatio-temporal data set. In the example of Figure 2a, the θ circle spans through three of the cells. The fourth cell (bottom-right) is located at a distance larger than θ from the location of the vehicle; by pruning the POIs of that cell, our algorithm achieves higher performance without compromising the correctness of the result.

4 Composite Event Recognition

The enriched data stream from moving commercial vehicles is transmitted to the CER module, in order to recognize various types of vehicle activity. All such activities have been formalized in collaboration with the domain experts of the Track & Know project. Table 1 presents the input and the output of the CER component, i.e. the Event Calculus for Run-Time reasoning (RTEC). In the following sections we present RTEC and illustrate its use for fleet management.

Table 1: Input and Output of CER. The first six input event types accompany the original GPS stream, while the remaining ones are the result of data enrichment. All input events are instantaneous, while all output CEs are durative.

	Events	**Description**
Input	$moving(V, S)$	Vehicle V is moving with a speed S
	$stopped(V)$	Vehicle V is not moving
	$abruptAcceleration(V)$	Vehicle V accelerates abruptly
	$abruptDeceleration(V)$	Vehicle V decelerates abruptly
	$abruptCornering(V)$	Vehicle V turns abruptly
	$fuelLevel(V, L)$	The level of fuel in tank of vehicle V is L
	$iceOnRoad(V)$	Vehicle V is moving in an icy road
	$closeToGas(V)$	Vehicle V is near a gas station
Output	$highSpeed(V)$	Vehicle V exceeds the user-specified speed limit
	$dangerousDriving(V)$	Vehicle V is potentially moving in a dangerous way
	$reFuelOpportunity(V)$	There is refueling opportunity for vehicle V

Table 2: Main predicates of RTEC.

Predicate	**Meaning**
happensAt$(E,\ T)$	Event E occurs at time T
holdsAt$(F = V,\ T)$	The value of fluent F is V at time T
holdsFor$(F = V,\ I)$	I is the list of maximal intervals for which $F = V$ holds continuously
initiatedAt$(F = V,\ T)$	At time T $F = V$ is initiated
terminatedAt$(F = V,\ T)$	At time T $F = V$ is terminated

4.1 Run-Time Event Calculus

The time model of RTEC is linear and includes integer time-points. If F is a fluent — a property that is allowed to have different values at different points in time — the term $F{=}V$ denotes that fluent F has value V. holdsAt$(F{=}V, T)$ is a predicate representing that fluent F has value V at time-point T. holdsFor$(F{=}V, I)$ represents that I is the list of maximal intervals for which $F = V$ holds continuously. holdsAt and holdsFor are defined in such a way that, for any fluent F, holdsAt$(F{=}V, T)$ if and only if T belongs to one of the maximal intervals of I for which holdsFor$(F{=}V, I)$. An *event description* in RTEC comprises rules that express: (a) event occurrences using the happensAt predicate, (b) the effects of events using the initiatedAt and terminatedAt predicates, (c) the values of fluents, with the use of the holdsAt and holdsFor predicates, as well as other, possibly atemporal, parameters. Table 2 presents the RTEC predicates available to the event description developer.

4.2 Pattern Representation

For a fluent F, $F = V$ holds at a particular time-point T if $F = V$ has been initiated by an event at some time-point earlier than T, and has not been terminated at some other time-point in the meantime. This is an implementation of the law of *inertia*. The time-points at which $F = V$ is initiated (respectively, terminated) are computed with the use of $\mathsf{initiatedAt}$ (resp. $\mathsf{terminatedAt}$) rules. $highSpeed(V)$, for example, is a Boolean fluent denoting that a vehicle V is moving with a speed greater than a user-specified threshold V_θ:

$$
\begin{aligned}
&\mathsf{initiatedAt}(highSpeed(V) = \mathsf{true},\ T) \leftarrow\\
&\quad \mathsf{happensAt}(moving(V, S),\ T),\\
&\quad threshold(V, speed,\ V_\theta),\ \ S > V_\theta.\\
&\mathsf{terminatedAt}(highSpeed(V) = \mathsf{true},\ T) \leftarrow\\
&\quad \mathsf{happensAt}(moving(V, S),\ T),\\
&\quad threshold(V, speed,\ V_\theta),\ \ S \leq V_\theta.\\
&\mathsf{terminatedAt}(highSpeed(V) = \mathsf{true},\ T) \leftarrow\\
&\quad \mathsf{happensAt}(stopped(V),\ T).
\end{aligned}
\tag{1}
$$

$moving(V, S)$ and $stopped(V)$ are input events, presented in Table 1. $threshold$ is an atemporal predicate recording the numerical thresholds of the patterns — in this case, the user-specified speed threshold of each vehicle in our knowledge base. Such a predicate supports code transferability, since the use of different thresholds in different applications may be realised by modifying $threshold$ only, and not the pattern specifications. Rule-set (1) states that $highSpeed(V) = \mathsf{true}$ is initiated if a $moving$ event is reported for vehicle V, and the speed S of V is greater than V_θ. Furthermore, $highSpeed(V) = \mathsf{true}$ is terminated if V is moving with a speed less or equal to V_θ, or when a $stopped$ event is reported for V. By using the $\mathsf{initiatedAt}$ and $\mathsf{terminatedAt}$ rules of rule-set (1), RTEC computes the maximal intervals I for which $highSpeed(V) = \mathsf{true}$ holds continuously, i.e. $\mathsf{holdsFor}(highSpeed(V) = \mathsf{true}, I)$. This is achieved by first finding all time-points T_s at which $highSpeed(V) = \mathsf{true}$ is initiated, and then, for each T_s, retrieving the first time-point T_f after T_s at which $highSpeed(V) = \mathsf{true}$ is terminated. Note that, in this formulation of the Event Calculus, $\mathsf{initiatedAt}(F = V, T)$ does not necessarily imply that $F \neq V$ at T. (This is similar to the 'weak interpretation' of initiation of the Cached Event Calculus, Chittaro and Montanari 1996). Similarly, $\mathsf{terminatedAt}(F = V, T)$ does not necessarily imply that $F = V$ at T. Suppose that $F = V$ is initiated at time-points 100 and 110 and terminated at time-points 125 and 135 (and at no other time-points). In that case $F = V$ holds at all T such that $100 < T \leq 125$.

$highSpeed(V)$ is useful indicator on its own, but can also be used to define potentially dangerous driving:

$$
\begin{aligned}
&\mathsf{initiatedAt}(dangerousDriving(V) = \mathsf{true},\ T) \leftarrow\\
&\quad \mathsf{happensAt}(abruptAcceleration(V),\ T),\\
&\quad \mathsf{holdsAt}(highSpeed(V) = \mathsf{true},\ T).\\
&\mathsf{initiatedAt}(dangerousDriving(V) = \mathsf{true},\ T) \leftarrow\\
&\quad \mathsf{happensAt}(abruptDeceleration(V),\ T),\\
&\quad \mathsf{holdsAt}(highSpeed(V) = \mathsf{true},\ T).\\
&\mathsf{initiatedAt}(dangerousDriving(V) = \mathsf{true},\ T) \leftarrow\\
&\quad \mathsf{happensAt}(abruptCornering(V),\ T),
\end{aligned}
\tag{2}
$$

$$
\begin{aligned}
&\quad \mathsf{holdsAt}(\mathit{highSpeed}(V) = \mathsf{true},\ T).\\
&\mathsf{initiatedAt}(\mathit{dangerousDriving}(V) = \mathsf{true},\ T) \leftarrow\\
&\quad \mathsf{happensAt}(\mathit{iceOnRoad}(V),\ T),\\
&\quad \mathsf{holdsAt}(\mathit{highSpeed}(V) = \mathsf{true},\ T).\\
&\mathsf{terminatedAt}(\mathit{dangerousDriving}(V) = \mathsf{true},\ T) \leftarrow\\
&\quad \mathsf{happensAt}(\mathsf{end}(\mathit{highSpeed}(V) = \mathsf{true}),\ T).\\
&\mathsf{terminatedAt}(\mathit{dangerousDriving}(V) = \mathsf{true},\ T) \leftarrow\\
&\quad \mathsf{happensAt}(\mathit{stopped}(V),\ T).
\end{aligned}
$$

abruptAcceleration, *abruptDeceleration* and *abruptCornering* are instantaneous input events provided by the accelerometer device installed in each commercial vehicle (see Table 1). $\mathit{iceOnRoad}(V)$ is a weather event emitted by the data enrichment module and states that in the location of V the road is slippery due to ice. $\mathsf{end}(F{=}V)$ is a built-in RTEC event indicating the ending points of each maximal interval for which $F = V$ holds continuously. According to rule-set (2), therefore, $\mathit{dangerousDriving}(V) = \mathsf{true}$ is initiated when a vehicle V is engaged in a harsh driving event, such as abrupt acceleration, breaking or cornering, or when there is ice on the road and V has speed above the user-specified threshold. $\mathit{dangerousDriving}(V) = \mathsf{true}$ is terminated when the speed of vehicle V goes below the user-specified threshold, or when it stops moving. $\mathit{dangerousDriving}(V)$ is thus useful for driver behavior analysis and safety.

Companies owning commercial fleets place emphasis on fuel consumption. One way to achieve this, is detecting opportunities for refueling. Consider the formalisation below:

$$
\begin{aligned}
&\mathsf{initiatedAt}(\mathit{reFuelOpportunity}(V) = \mathsf{true},\ T) \leftarrow\\
&\quad \mathsf{happensAt}(\mathit{closeToGas}(V),\ T),\\
&\quad \mathsf{holdsAt}(\mathit{highSpeed}(V) = \mathsf{true},\ T),\\
&\quad \mathsf{happensAt}(\mathit{fuelLevel}(V, L),\ T),\\
&\quad \mathit{threshold}(V, \mathit{fuel},\ V_{tank}),\ \ L < \frac{V_{tank}}{2}.\\
&\mathsf{terminatedAt}(\mathit{reFuelOpportunity}(V) = \mathsf{true},\ T) \leftarrow\\
&\quad \mathsf{happensAt}(\mathit{fuelLevel}(V, L),\ T),\\
&\quad \mathit{threshold}(V, \mathit{fuel},\ V_{tank}),\ \ L \geq \frac{V_{tank}}{2}.
\end{aligned}
\tag{3}
$$

$\mathit{closeToGas}(V)$ is a spatial relation computed by the data enrichment module (see Table 1), indicating that a vehicle V is close to a gas station, which is a type of point of interest. $\mathit{fuelLevel}(V, L)$ is an instantaneous input event emitted by the fuel sensor of each vehicle. $\mathit{threshold}(V, \mathit{fuel}, V_{tank})$ records the tank size of vehicles. According to rule-set (3), our system starts flagging that vehicle V should refuel when it is close to a gas station, its speed is above the user-specified threshold (implying uneconomic driving), and the fuel level is lower than half of the tank size. Moreover, we stop flagging the need to refuel when the fuel is more than half of the tank size.

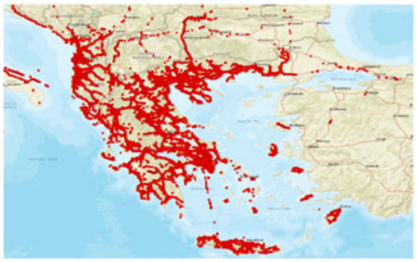


Fig. 3: Vehicle position signals of the data set.

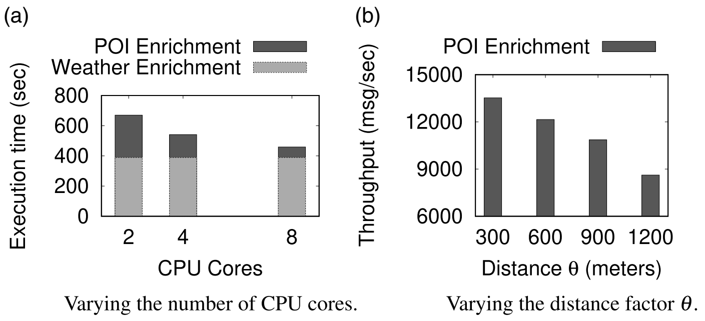


Fig. 4: Performance of enrichment process.

5 Implementation and Empirical Analysis

5.1 Experimental Setup

We evaluated our system on real-world positional data of vehicles, provided by Vodafone Innovus[6], our partner in the Track & Know project, which offers fleet management services. Figure 3 illustrates the geographical coverage of the data, practically covering Greece and some surrounding countries, for a temporal duration of 1 month. The fleet data contains approximately 4M records and is 527 MB in the form of CSV files. We replayed these records, according to their timestamps, in order to simulate a streaming environment. The records are enriched with weather information, acquired by 120 GRIB files with total size 7.4 GB. The POIs were retrieved from OpenStreetMap; we selected only the POIs referring to gas stations which resulted to approximately 140K POIs. The data enrichment module operated on a VM running the CentOs 7.6.1810 operating system, on a hardware with Intel Xeon Processor and 4GB RAM. The CER module operated on a computer with 8 cores (Intel(R) Core(TM) i7-7700 CPU @ 3.6GHz) and 16 GB of RAM, running Ubuntu 16.04 LTS 64-bit and YAP Prolog 6.2.2.

5.2 Data Enrichment

The vehicle position signals were loaded in an Apache Kafka topic, consisting of 6 partitions sorted by the date field. The topic was consumed by the weather enrichment component, and the poll timeout was set to 1 sec[7]. The POI enrichment runs as a sepa-

[6] https://www.vodafoneinnovus.com/
[7] This is the time that defines a batch of messages fetched for processing.

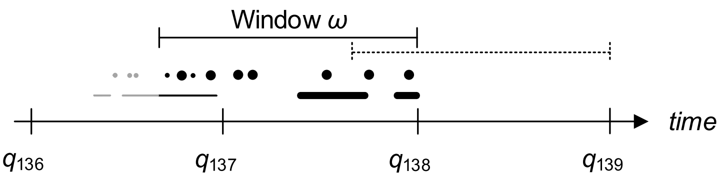


Fig. 5: Composite event recognition in RTEC.

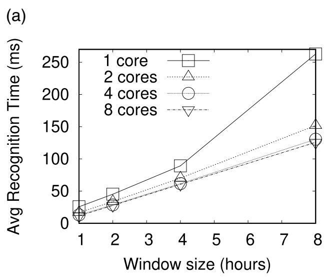
rate job, and consumes weather-enriched data that were output in an intermediate Kafka topic, and was configured to use 2 GB of RAM. In this set of experiments, we report on the performance of data enrichment, using total execution time and throughput as main metrics.

Figure 4 depicts the performance results of data enrichment. Figure 4a reports the total execution time for data enrichment when increasing the number of CPU cores. The weather enrichment was completed in 389 sec, corresponding to throughput values of 9,792 messages/sec. Notice that this value is constant in the figure, since weather enrichment does not use parallelization. The cache of the weather data obtainer reached 99.96% hit ratio. This high ratio was expected, since the records are temporally sorted. The POI enrichment results correspond to distance threshold $\theta = 300$m. As shown in the figure, the total execution time drops when increasing the parallelism from 2 to 8 CPU cores. This result shows that the POI enrichment component can exploit the availability of more CPUs and scale based on the available resources.

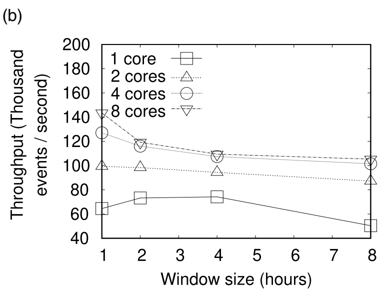
Figure 4b shows the throughput of POI enrichment obtained from increasing the value of θ, while fixing the number of cores to 2. Higher values of θ result in having more POIs associated with positions of vehicles, as θ practically defines what is considered as proximity. Also, higher values of θ imply that a larger spatial area around each vehicle's position needs to be examined, leading to decreased performance. However, when increasing θ by a factor of 4, the performance is decreased less than 50%. Hence, our POI enrichment component is efficient for even higher θ values. Also, it can achieve even better performance by exploiting more CPU resources, as already shown in Figure 4a.

5.3 Composite Event Recognition

In RTEC, the CER process involves the computation of the maximal intervals of fluents. This process takes place at specified query-times $q_1, q_2, \dots$. CER at each query-time q_i is performed over the input events that fall within a specified interval, the 'working memory' or window ω. All input events outside the window are discarded and not considered during recognition. This means that at each query-time q_i, CER depends only on the events that took place in the interval $(q_i - \omega, q_i]$. The size of ω as well as the temporal distance between two consecutive query-times — the slide step $(q_i - q_{i-1})$ — are user-specified. Figure 5 illustrates the recognition process of RTEC. Occurrences of instantaneous input events are displayed as dots and those of durative input events as line segments. For CER at query-time q_{138}, only the events marked in black are considered, whereas the greyed out ones are neglected. Assume that the events marked in bold arrive after q_{137}. Therefore, two input events are delayed and by using a window size larger than the slide step, these two events are not lost and considered at q_{138}. In the analysis that follows, we restrict attention to overlapping windows, i.e. windows longer than the slide step.



Average recognition time.



Throughput.

Fig. 6: CER under varying window sizes and parallel configurations.

At each query-time q_i, RTEC computes from scratch the intervals of CEs, without considering the computations of previous windows. In the case of significant delays in the input stream, this simple approach is the best option. However, in cases where CEs are unaffected by delays, computing their intervals from scratch is redundant. To address this issue, we recently developed a process for computing incrementally the maximal intervals of a CE (Tsilionis et al. 2019). Consider the first initiation rule of rule-set (2) again, and assume that at query-time q_i a delayed arrival of *abruptAcceleration*(V) arrived to the CER system or/and a new interval was computed for *highSpeed*(V). In both cases, a new initiation may have to be computed for *dangerousDriving*. To calculate new initiation points, we use the following *delta* rules (the remaining initiation rules and the termination rules of rule-set (2) are handled similarly):

$$\begin{aligned}
&\mathsf{initiatedAt}(\mathit{dangerousDriving}(V) = \mathsf{true},\ T) \leftarrow \\
&\quad \Big[\mathsf{happensAt}(\mathit{abruptAcceleration}(V),\ T)\Big]^{\mathit{ins}}, && \text{(a)} \\
&\quad \Big[\mathsf{holdsAt}(\mathit{highSpeed}(V) = \mathsf{true},\ T)\Big]^{\mathsf{Q_i}}. \\
& && && (4) \\
&\mathsf{initiatedAt}(\mathit{dangerousDriving}(V) = \mathsf{true},\ T) \leftarrow \\
&\quad \Big[\mathsf{happensAt}(\mathit{abruptAcceleration}(V),\ T)\Big]^{\mathsf{Q_i} \setminus \mathit{ins}}, && \text{(b)} \\
&\quad \Big[\mathsf{holdsAt}(\mathit{highSpeed}(V) = \mathsf{true},\ T)\Big]^{\mathit{ins}}.
\end{aligned}$$

The superscripts of these rules express the evaluation set of the time argument T. In rule (4)(a), *abruptAcceleration*(V) is evaluated only over the occurrences that arrived to the CER system between q_{i-1} and q_i, i.e. the occurrences in set *ins*. The time-points in *ins* are examined against all the intervals of *highSpeed*$(V) = \mathsf{true}$ overlapping the current window (set $\mathsf{Q_i}$). Rule (4)(b) is similar to (4)(a), but has a small modification which ensures that derivations are not repeated. In this rule, only the intervals computed at q_i are considered for *highSpeed*$(V) = \mathsf{true}$ (set *ins*). For *abruptAcceleration*(V) the occurrences within the current window that arrived by the previous query-time q_{i-1} (set $\mathsf{Q_i} \setminus \mathit{ins}$) are used.

We performed two sets of experiments. First, since the data set is temporally sorted, we evaluated the performance of RTEC without incremental reasoning, on varying window sizes and parallel executions. Second, we injected artificial delays to the data set,

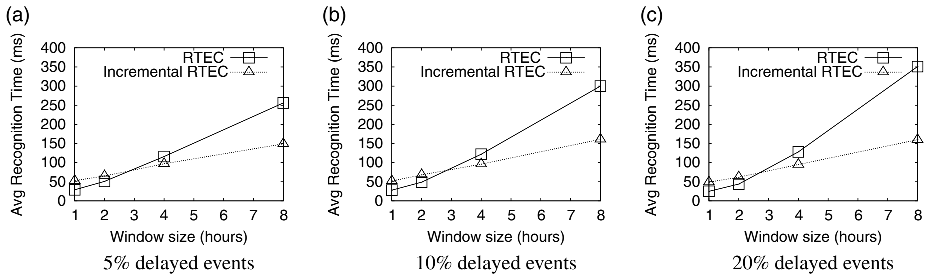


Fig. 7: Effects of incremental reasoning.

to simulate online processing, and thus compared RTEC with and without incremental reasoning. Figure 6 shows the results of the first set of experiments. Initially, we used a single processor to perform CER. Then, we run RTEC in parallel, by launching different instances of the engine, each one operating on a different processing core. Each RTEC instance performed CER for a different set of vehicles. For example, in the case of four processing cores each RTEC instance was responsible for one quarter of operating vehicles. In all sets of experiments the input was the same, that is, there was no data distribution.

We varied the window size ω from 1 to 8 hours and the slide step was always equal to the size of the window. In the absence of delays, it is redundant to have overlapping windows. Figure 6(a) presents the recognition times of RTEC in CPU milliseconds (ms), while Figure 6(b) presents the throughput. The empirical analysis shows that RTEC is capable of real-time CER even when operating on a single core. Additionally, running RTEC in parallel leads to significant performance gains even without data distribution.

The second set of experiments concerns out-of-order streams where we compared the performance of RTEC and its incremental extension. We injected artificially delays into the data set. We performed three experiments, each time varying the amount of input events being delayed. We selected uniformly 5%, 10% and 20% of the total events to be delayed. We used a uniform distribution for selecting events, since we assume that each event has the same probability to be delayed. In order to mimic reality as much as possible, we used a Gamma distribution to choose the extent of delay. (The Gamma distribution has a shape parameter $k = 2$ and a scale parameter $\theta = 2$.) Thus, a delay small in time has a higher probability to be imposed in a selected event. The average delay time, in all settings, is approximately 8 hours.

Figures 7(a-c), display the average recognition times in CPU milliseconds for windows ranging from 1 hour to 8 hours, and a slide step of 1 hour. As shown in Figures 7(a-c), the incremental version of RTEC outperforms the non-incremental one in the largest windows, i.e. those of 4 and 8 hours. In other words, the performance improvement becomes more profound as the overlap between consecutive windows increases.

6 Discussion

We presented a stream reasoning system for online fleet management. We opted for a separation of activities among different modules. Delegating data enrichment to a separate module allows for the effective integration of spatial reasoning with temporal reasoning for online CER. Additionally, the use of a dedicated module for data enrichment allows to combine heterogeneous data sources in an efficient way. The empirical evaluation on real-world data illustrated the scalability of our system as well as its capacity to operate in real-time.

A challenge that we faced in the development and deployment of our system is the memory leak of various Prolog implementations, such as YAP and SWI-Prolog, on continuous queries. To address this issue, we sometimes had to store the recognised CEs in order to restart the engine, which is suboptimal in online processing.

Similar to other Big Data projects (Artikis et al. 2015; Patroumpas et al. 2017; Artikis et al. 2013), the datasets of the Track & Know project did not come with a ground truth of CEs. One way to address this issue is to construct the CE patterns in close collaboration with domain experts. This is what we did in Track & Know. However, although the domain experts of the project have some idea about the CEs of interest, the precise conditions in which a CE should be recognized are not always clear. The use of RTEC facilitated the interaction of CE pattern developers and domain experts. Patterns in the language of RTEC were understood, and sometimes directly modified by the domain experts. To facilitate this process further, we have been developing a simple language for RTEC, with the aim of supporting people who are not familiar with the Event Calculus or (logic) programming (Vlassopoulos and Artikis 2017). A compiler translates, in a process transparent to the user, a specification in the simple language to an RTEC event description that may be subsequently used for continuous query computation.

To allow for accuracy evaluation, we implemented a way of visualising our recognised CEs by means of videos[8]. Our aim is to enable domain experts offer feedback on our recognised events, i.e. classify them as true or false positives. Using such videos, domain experts were able to perform a preliminary accuracy assessment. The findings of this assessment indicated that some CE intervals ended later than anticipated. This is due to the fact that the position signals of vehicles can be sparse. For example, there are some extreme cases in which there are 24 hours between two consecutive positional signals of the same vehicle, most likely indicating different trips. In some of these cases, the CE was terminated on the signal of the subsequent trip, i.e. the termination was delayed. In order to deal with this issue, we can directly use the 'deadlines' mechanism of RTEC, according to which a CE is automatically terminated after a designated number of time-points since the last initiation. A systematic accuracy evaluation based on expert feedback, using the aforementioned visualisations, is part of our current work.

We also aim to refine the manually constructed CE patterns by means of a recently developed technique for semi-supervised learning (Michelioudakis et al. 2019). The input of this technique will be the expert feedback as described above, as well as a small set of labels that may be provided with minimal resources by domain experts.

Acknowledgments

This work was funded by European Union's Horizon 2020 research and innovation programme Track & Know "Big Data for Mobility Tracking Knowledge Extraction in Urban Areas", under grant agreement No 780754.

[8] See `http://cer.iit.demokritos.gr/`

References

ALEVIZOS, E., SKARLATIDIS, A., ARTIKIS, A., AND PALIOURAS, G. 2017. Probabilistic complex event recognition: A survey. *ACM Comput. Surv. 50,* 5, 71:1–71:31.

ARTIKIS, A. AND SERGOT, M. J. 2010. Executable specification of open multi-agent systems. *Logic Journal of the IGPL 18,* 1, 31–65.

ARTIKIS, A., SERGOT, M. J., AND PALIOURAS, G. 2015. An event calculus for event recognition. *IEEE Trans. Knowl. Data Eng. 27,* 4, 895–908.

ARTIKIS, A., WEIDLICH, M., GAL, A., KALOGERAKI, V., AND GUNOPULOS, D. 2013. Self-adaptive event recognition for intelligent transport management. In *Proceedings of the IEEE International Conference on Big Data*. 319–325.

BECK, H., DAO-TRAN, M., AND EITER, T. 2018. LARS: A logic-based framework for analytic reasoning over streams. *Artif. Intell. 261*, 16–70.

CERVESATO, I. AND MONTANARI, A. 2000. A calculus of macro-events: Progress report. In *Seventh International Workshop on Temporal Representation and Reasoning, TIME 2000, Nova Scotia, Canada, July 7-9, 2000*. 47–58.

CHITTARO, L. AND MONTANARI, A. 1996. Efficient temporal reasoning in the cached event calculus. *Computational Intelligence 12*, 359–382.

CUGOLA, G. AND MARGARA, A. 2010. TESLA: a formally defined event specification language. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems, DEBS 2010, Cambridge, United Kingdom, July 12-15, 2010*. 50–61.

CUGOLA, G. AND MARGARA, A. 2012. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv. 44,* 3, 15:1–15:62.

DEMERS, A. J., GEHRKE, J., PANDA, B., RIEDEWALD, M., SHARMA, V., AND WHITE, W. M. 2007. Cayuga: A general purpose event monitoring system. In *CIDR 2007, Third Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*. 412–422.

DONG, X. L. AND SRIVASTAVA, D. 2015. *Big Data Integration*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

DOUSSON, C. AND MAIGAT, P. L. 2007. Chronicle recognition improvement using temporal focusing and hierarchization. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*. 324–329.

GIATRAKOS, N., ALEVIZOS, E., ARTIKIS, A., DELIGIANNAKIS, A., AND GAROFALAKIS, M. 2019. Complex event recognition in the big data era. *VLDB Journal*.

GREZ, A., RIVEROS, C., AND UGARTE, M. 2019. A formal framework for complex event processing. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*. 5:1–5:18.

JACOX, E. H. AND SAMET, H. 2007. Spatial join techniques. *ACM Trans. Database Syst. 32,* 1, 7.

KOUTROUMANIS, N., SANTIPANTAKIS, G. M., GLENIS, A., DOULKERIDIS, C., AND VOUROS, G. A. 2019. Integration of mobility data with weather information. In *Proceedings of the Workshops of the EDBT/ICDT 2019 Joint Conference, EDBT/ICDT 2019, Lisbon, Portugal, March 26, 2019.*

KOWALSKI, R. A. AND SERGOT, M. J. 1986. A logic-based calculus of events. *New Generation Comput. 4,* 1, 67–95.

LIU, M., RUNDENSTEINER, E. A., GREENFIELD, K., GUPTA, C., WANG, S., ARI, I., AND MEHTA, A. 2011. E-cube: multi-dimensional event sequence analysis using hierarchical pattern query sharing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*. 889–900.

MEI, Y. AND MADDEN, S. 2009. Zstream: a cost-based query processor for adaptively detecting composite events. In *Proceedings of the ACM SIGMOD International Conference on Man-*

agement of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009. 193–206.

Michelioudakis, E., Artikis, A., and Paliouras, G. 2019. Semi-supervised online structure learning for composite event recognition. *Machine Learning 108,* 7, 1085–1110.

Miller, R. and Shanahan, M. 2002. Some alternative formulations of the event calculus. In *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II.* 452–490.

Montali, M., Maggi, F. M., Chesani, F., Mello, P., and van der Aalst, W. M. P. 2013. Monitoring business constraints with the event calculus. *ACM TIST 5,* 1, 17:1–17:30.

Paschke, A. 2006. Eca-ruleml: An approach combining ECA rules with temporal interval-based KR event/action logics and transactional update logics. *CoRR abs/cs/0610167.*

Paschke, A. and Bichler, M. 2008. Knowledge representation concepts for automated SLA management. *Decision Support Systems 46,* 1, 187–205.

Patroumpas, K., Alevizos, E., Artikis, A., Vodas, M., Pelekis, N., and Theodoridis, Y. 2017. Online event recognition from moving vessel trajectories. *GeoInformatica 21,* 2, 389–427.

Schultz-Møller, N. P., Migliavacca, M., and Pietzuch, P. R. 2009. Distributed complex event processing with query rewriting. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, DEBS 2009, Nashville, Tennessee, USA, July 6-9, 2009.*

Tsilionis, E., Artikis, A., and Paliouras, G. 2019. Incremental event calculus for run-time reasoning. In *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems, DEBS 2019, Darmstadt, Germany, June 24-28, 2019.* 79–90.

Vlassopoulos, C. and Artikis, A. 2017. Towards A simple event calculus for run-time reasoning. In *Proceedings of the Thirteenth International Symposium on Commonsense Reasoning, COMMONSENSE 2017, London, UK, November 6-8, 2017.*

Zhang, H., Diao, Y., and Immerman, N. 2014. On complexity and optimization of expensive queries in complex event processing. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014.* 217–228.