Detecting Mobility Patterns using Spatial Query Answering over Streams

Thomas Eiter[1], Josiane Xavier Parreira[2], and Patrik Schneider[1,2]

[1] Vienna University of Technology, Vienna, Austria
[2] Siemens AG Österreich, Vienna, Austria

Abstract. The development of (semi)-autonomous vehicles and communication between vehicles and infrastructure (V2X) will aid to improve road safety and reduce emissions by identifying dangerous traffic scenes based on movement patterns. A key to this is the Local Dynamic Map (LDM), which acts as an integration platform for static, temporary, and dynamic information about traffic in a geographical context. We have semantically enhanced the LDM to allow an elaborate domain model, captured by a mobility ontology, and queries over data streams that allow for semantic concepts and spatial relationships. Our approach is in the context of ontology-mediated query answering (OQA), which features conjunctive queries over DL-Lite$_A$ ontologies that allow for window operators over streams having a pulse and for spatial relations between spatial objects. We have introduced in previous work simple aggregate queries over mobility streams, which often do not suffice to capture more complex movement patterns. Based on three scenarios, we define requirements derived from a domain-specific list of features. Further, we present an extension of the previous aggregate functions with statistical and predictive functionality, which allows us to query more complex mobility patterns such as road networks statistics, vehicles maneuvers, and temporal connected events such as (potential) accidents. We also give a more detailed report on our implementation and analyze it regarding the features and requirements.

1 Introduction

The development of (semi)-autonomous vehicles needs extensive communication between vehicles and the infrastructure, which is covered by Cooperative Intelligent Transport Systems (C-ITS). These systems collect temporal data (e.g., traffic light signal phases) and geospatial data (e.g., GPS positions), which are exchanged in vehicle-to-vehicle, vehicle-to-infrastructure, and combined communications (V2X). This aids (a) to improve road safety by analyzing traffic scenes that could lead to accidents (e.g., red light violations), and (b) to reduce emissions by optimizing traffic flow (e.g., dissolve traffic jams). A key technology for this is the Local Dynamic Map (LDM) [2] as an integration platform for (semi-)static, and dynamic information in a geographical context.

We have semantically enhanced the LDM towards an elaborate domain model, captured by a mobility ontology, with conjunctive queries (CQs) over data streams that allow for semantic concepts and spatial relationships to address safety applications, such as detection of red light violations on complex intersections managed by a roadside C-ITS station. To realize query answering (QA) over mobility streams, spatial and streaming data were lifted to ontology-mediated QA (OQA) with the ontology language DL-Lite$_A$. However, for monitoring more complex actions such as vehicle maneuvers (e.g., overtaking), this query language is too limited and needs to be extended with temporal, statistical, and predictive functions. To this end, we start with the three important scenarios "intersection statistics", "mobility patterns", and "event detection" and derive requirements for a suitable query language, which are based on a domain-specific set of features. Our contributions are briefly summarized as follows:

- we define our desired scenarios, features, and requirements (Section 2);
- we outline the field of V2X integration using LDMs and provide details on our LDM ontology that incorporates the features/requirements (Section 3);
- we present our current data model, query language, and outline the evaluation techniques with extensions regarding the features/requirements (Section 4);
- we give a detailed description of our query platform and analyze it regarding the implemented features and requirements (Section 5).

In Section 6, we discuss related work and conclude with ongoing and future work.

2 Scenarios, Features, and Requirements

In this section, we give three application scenarios that are used to define the features and derived the requirements. All scenarios are related to roadside C-ITS stations deployed on either a single road intersection or a network of intersections. The C-ITS stations receive any V2X message and can send signal phases and a local intersection topology messages, to inform other participants on its current state. The other participants such as vehicles (or pedestrians) might share their states such as their current speed, acceleration, and position using V2X messages or an IoT-based protocol (e.g., ZigBee, Bluetooth, or Z-Wave).[3] Furthermore, the C-ITS could have access to auxiliary streams providing news, public transport, and weather data.

S1: Intersection and Network Statistics. The focus of this scenario is on the collection of statistical data concerning stops, throughput, traffic distribution, or types of participants by aggregating the streaming data on specific intersections. Regarding this scenario, we have identified the following use cases and measurements:

1. *Object level*: for a single vehicle, the average speed, acceleration, number of stops, or on a sensor level average temperature or rainfall could be calculated ;
2. *Road/Lane level*: on the road/lane level, important measures could be the average throughput, waiting time, the amount of stops, but also traffic light related measurements such as the average signal phase length;
3. *Intersection level*: on this level, besides calculating a summary of road/lane level indicators, matrices regarding transfers (e.g. how many cars head straight on), modality, and type mix (e.g. relation between public/private transport, or car/truck) could be determined;
4. *Network level*: on the network level, intersections are represented by nodes connected by roads. In this setting, we could calculate summary of each intersections, but also transfer times and traffic flow between intersections could be calculated.

[3] http://www.zigbee.org/, https://www.bluetooth.com/, or http://www.z-wave.com/

S2: Vehicle Maneuvers. This scenario is concerned with the detection of maneuvers performed by a single car or a group of cars. The detection of maneuvers can be used for statistics (e.g., number of cars turning left), or as a basic pattern for event detection as defined below (e.g., crossing a double line). The following maneuvers could be extracted from the trajectories of vehicles:

1. *Slow down/speed up*: the detection of these maneuvers could be done by aggregating the acceleration;
2. *Drive straight on, turn left, turn right*: for the detection of turns, the geometry of a vehicle trajectory could be matched to geometries representing all types of turns.
3. *Stop, load/unload, park*: to determine whether a car has stopped is straightforward, but distinguishing the reason for the stop needs contextual information such as the location of the stop, e.g., parking;
4. *Lane change*: this maneuver requires the combination of trajectory-based geometries and spatial relations that are evaluated w.r.t. the trajectories;
5. *Overtake*: this maneuver is an extension of a "change lane", but needs additionally trajectory predictions.

Note that the calculation of the trajectory needs a preprocessing step of correcting frequent GPS errors in position data.

S3: Event Detection. An important C-ITS application is "road safety" [2], which can be enabled by event detection. It is the most complex scenario, since it requires the combination of statistical measurements, vehicle maneuvers, and temporal relations that are evaluated over a longer period. The following events could be detected:

1. *Red-light violation*: as shown in [18], this event can be detected by checking the spatial intersection of lanes changing to a red phase, the vehicle's speed and current trajectory. This could be enhanced by predicting possible trajectories;
2. *Obstructed view*: this event concerns dangerous situation on intersections where two vehicles have no visual contact because of an obscured view due to buildings or trees. The overlap of the possible trajectories for the two vehicles has be checked, and the information (by tagging) that the intersection might have obscured view situations (based on historic data on accidents) has to be taken into account;
3. *Vehicle breakdown/accident*: this event is based on a "stop" maneuver, where we identify vehicles that are not moving and are inside a dangerous area of an intersection. This event can be extended for the case where several vehicles are involved;
4. *Traffic rule violations*: traffic rule violations cover a wide range of events. For instance, speeding or red-light violations, but also crossing double lines or forbidden overtaking could be detected for enforcement;
5. *Traffic congestions*: this is the most complex event, as short and long term observations must be combined. Queuing cars could indicate a congestions and be detected by checking the "stop" maneuvers of several vehicles that are behind each other, but not stopped by a longer red light phase.

Features and Requirements. The eight resp. nine central requirements, e.g., volume, velocity, variety, incompleteness, complex domain models, etc. identified by [35] resp. [13] for stream reasoning systems are not discussed here, but should hold for mobility stream systems as well. In this paper, we only focus on domain specific features that are mapped to requirements crucial for enabling the above scenarios. For this, we have identified the following feature sets:

- *F1 - Time model*: a *point-based* (PO), an *interval-based* (IT), or combined (CM) model of time is desired. In the CM model, aggregations could lead to intervals based on point-based data items;
- *F2 - Process paradigm*: queries are processed in a push-based (PS) or pull-based (PL) manner. A combined (CM) method is possible, where the high velocity atoms are push-based (for caching) and the low velocity atoms are still pull-based evaluated.
- *F3 - Spatial relations*: different sorts of relations are desired, starting from a point-set model (PSM) as we have used in [18] to the more detailed *9-Intersection model* (DE9) of [15] or qualitative spatial reasoning as in RCC8 [31];
- *F4 - Temporal relations*: several formalisms for temporal relations are suitable, Linear Temporal Logic (LTL)[29] with the operators *next*, *global*, or *until*; Allen's Time Interval Algebra [1] with operators like *before*, *meets*, and *during*, or Metric Temporal Logic (MTL) [22] with LTL operators plus timing constraints.
- *F5 - Numerical aggregations*: these can be "simple" aggregations such as *sum* or *average* on either a set or multiset (bag) of data items. Note that the aggregation over a multiset, e.g., $G_{c_1}=\{|30, 29, 34|\}$ and $G_{c_2}=\{|20, 0, 0|\}$, is crucial, since we often have data items of different objects in a single stream. In case the data items are not ordered, functions like *first* or *last* are desired.
- *F6 - Spatial aggregations*: a wide range of spatial aggregations (e.g., convex hull) can be applied to geometric objects like points and lines. The aggregation functions need to take the peculiarities of geometries into account. For instance, the dimension of the object changes depending on the function used, e.g., building a line segments of points. Furthermore, this feature could include smoothing and simplification of complex objects.
- *F7 - Numerical predictions*: predictions allow the generation of not known data times projected into the past or future. Several prediction functions such as multiple linear or k-nearest-neighbour regression should be available. Depending on the task, even more complex machine learning methods could be envisioned.
- *F8 - Trajectory computations*: instead of projecting numbers into the future, we predict a vehicle's movement. This could be achieved by (1) a "point-to-curve" aggregation, and (2) calculating possible paths using an underlying road graph.
- *F9 - Spatial matching*: checking identical geometries does not suffice in all cases, hence geometric objects must be compared for their similarity. E.g., left or right turns are represented by prototypical curves, which are then compared to calculated trajectories of the vehicles.
- *F10 - Advanced features*: the set of "advanced" features increase the computational complexity of query answering considerably, with the effect that our "inherent" property of direct FO-rewritability is lost. These features are more generic and include graph connectivity (CN), Negation as Failure (NA), and repairing w.r.t. constraints in the ontology of inconsistent data items (RE).

We have derived the requirements (shown in Table 1) by analyzing each scenario and use case regarding the necessary features. In case of a single feature, e.g., trajectory computation, we only distinguish between "Y" for required, "N" for not required, and

Table 1: Requirement Matrix (Y is required, N is not required, P is possibly, other abbreviations as described in F1 to F10)

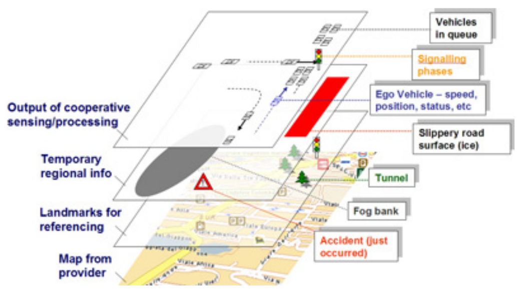
Case	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
S1.1	PO	PL	N	N	Y	P	P	P	N	
S1.2	PO	PL	Y	N	Y	Y	P	P	N	
S1.3	PO	PL	Y	N	Y	Y	Y	Y	Y	
S1.4	PO	PL	Y	N	Y	Y	Y	Y	Y	CN
S2.1	PO	PL	N	N	Y	P	P	N	N	
S2.2	PO	PL	Y	N	Y	Y	P	Y	Y	NA
S2.3	PO	PL	Y	P	Y	Y	Y	N	N	NA
S2.4	PO	PL/PS	Y	N	Y	Y	P	P	P	
S2.5	PO/IT	PL/PS	Y	N	Y	Y	Y	Y	P	NA
S3.1	PO/IT	PL/PS	Y	P	Y	Y	P	Y	Y	
S3.2	PO/IT	PL/PS	Y	P	Y	Y	P	Y	Y	
S3.3	CM	PL/PS	Y	Y	Y	Y	Y	Y	Y	NA
S3.4	CM	CM	Y	Y	Y	Y	Y	Y	Y	NA, CN
S3.5	CM	CM	Y	Y	Y	Y	Y	Y	Y	NA, CN

"P" possibly required. For instance in *S3.1*, a point-based time model suffices, however, intervals might be helpful to represent signal phases of traffic lights. In case of longer timed queries (such as *S3.5*), intervals are needed to process aggregations in combination with long-term temporal relations. Further in *S3.1*, both pull- or bushed-based queries are desired, and the "standard" features like spatial relations, aggregations, and predictions (including trajectories) are all necessary.

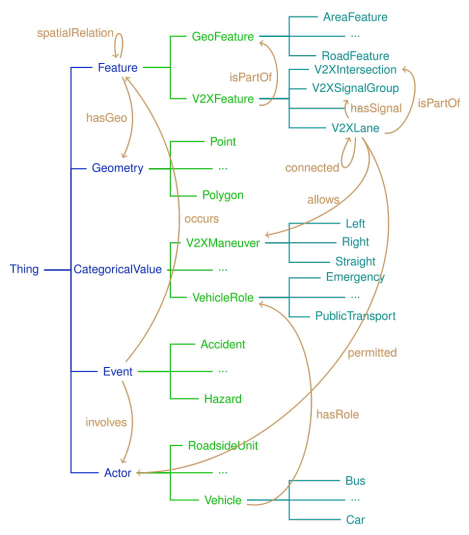
3 V2X Integration and the Local Dynamic Map

The base communicate technologies (i.e., the IEEE 802.11p standard) allow wireless access in vehicular environments, which enables messaging between vehicles themselves and the infrastructure, also called V2X communications. The messages are broadcast every 100ms by traffic participants (e.g. vehicles, roadside ITS stations) to update other participants about their current state [2]. The main messages types are:
- *CAMs* (Cooperative Awareness Messages) provide high frequency status updates of a vehicle's position, speed, but could include vehicle type, etc.;
- *MAPs* (Map Data Messages) describe the detailed topology of an intersection, including its lanes and their connections;
- *SPATs* (Signal Phase and Timing Messages) give the projected signal phases (e.g., green) for each lane;
- *DENMs* (Decentralized Environmental Notification Messages) informing if specific events like road works or a traffic jam occur in a designated area.

Local Dynamic Map. The V2X technology does not yet consider the integration of the different types of messages. As a comprehensive integration effort, the EU SAFESPOT project [2] introduced the concept of a *Local Dynamic Map*, which acts as an integration platform to combine static geographic information system (GIS) maps, with dynamic


Output of cooperative sensing/processing
Temporary regional info
Landmarks for referencing
Map from provider
Vehicles in queue
Signalling phases
Ego Vehicle – speed, position, status, etc
Slippery road surface (ice)
Tunnel
Fog bank
Accident (just occurred)

(a) The four layers of a LDM [2]



(b) LDM ontology (partial rendering)

Fig. 1: Local Dynamic Map

environmental objects (e.g., vehicles, pedestrians). The integration was motivated by advanced safety applications, which need an "overall" picture of traffic environment. The LDM consists of the following four layers (see Figure 1a):

- *Permanent static*: The first layer contains static information obtained from GIS maps and includes roads, intersections, and points-of-interest (POIs);
- *Transient static*: The second layer extends the static map by detailed local traffic informations such as fixed ITS stations, landmarks, and intersection features like lanes;
- *Transient dynamic*: The third layer contains temporary regional information like weather, road or traffic conditions (e.g., traffic jams), and traffic light signal phases;
- *Highly dynamic*: The fourth layer contains dynamic information of road users detected by V2X messages, in-vehicle sensors like the GPS module.

Recent research ([25], [33]) suggested that an LDM can be built on top of a spatial RDBMS enhanced with streaming capabilities. It was recognized in [25] that an LDM should be represented by a world model, world objects, and data sinks on the streamed input. However, an elaborate domain model, captured by an LDM ontology, and extended queries over data streams allowing spatial relations (e.g., contains), were still missing.

Extended LDM Ontology. With the support of domain experts, we have extended our LDM ontology (shown partially in Figure 1b) to capture the four levels of the LDM, the elements of a traffic scene, but also scenario-specific elements such as maneuvers.[4] The LDM ontology is represented in DL-Lite$_A$ [11], which is the logical underpinning for the W3C standard OWL 2 QL. Besides the restriction to DL-Lite$_A$, our methods are

[4] Available at http://www.kr.tuwien.ac.at/research/projects/loctrafflog /LocalDynamicMapITS-v0.4-Lite.owl

ontology-agnostic; hence other mobility ontologies could be used as well. We follow a layered approach starting with a simple separation between the following classes:
- $V2XFeature$: is the spatial representation of V2X objects, such as details of an intersections including lanes and traffic lights;
- $GeoFeature$: represents the GIS aspects of the LDM including POIs, areas like parks, and road networks with $Geometry$ as the geometrical representation of features;
- $Actor$: is the class that includes persons, vehicles, as well as roadside ITS stations. Its objects have autonomous behavior and are the main generator of streamed data;
- $Event$: describes prototypical events that happen in a mobility scene, an important subclass of $Event$ is $Maneuver$ that captures possible maneuver such as left-turns, U-turns, or stops by vehicles;
- $StatisticalValues$: describes possible measurements such as the average speed or throughput that could be collected on an intersection for a lengthy period of time;
- $CategoricalValues$: specify the different categories such as signal phases, or vehicle roles used in the domain. Vehicle roles are crucial for emergency detection, since it informs the type of an emergency vehicle (e.g., police).

Further, we also introduced the following properties:
- properties for partonomies, e.g., $isPartOf$;
- spatial relations, e.g., $intersects$;
- connectivity, e.g., $connected$; and
- standard properties, e.g., $isAllowed$, $hasRole$, $isManaged$, or $positions$.

The sub-classes of $GeoFeature$ are linked to GeoOWL and GeoNames for embedding it into existing ontologies.[5] $V2XFeature$ is the domain specific modeling of the MAP (Map Data Message) topology and describes the details of an intersection including its lanes, allowed maneuvers, and traffic lights.

4 Spatial-Stream Query Answering

The QA component is central to the usage of a semantically enriched LDM, since it allows us to access the streamed data in the LDM. We focus on pull-based queries that evaluate a query at a single point in time called the query time $\mathbb{T}_i$.

Example 1. The following query detects red-light violations on intersections by searching for vehicles y at speed above 30km/h on lanes x whose signals will turn red in 4s:

$$\begin{aligned} q_1(x,y) : \ & LaneIn(x) \wedge hasLocation(x,u) \wedge intersects(u,v) \wedge pos_{(line,\ 4s)}(y,v) \\ & \wedge Vehicle(y) \wedge speed_{(avg,\ 4s)}(y,r) \wedge (r > 30) \wedge \ isManaged(x,z) \\ & \wedge SignalGroup(z) \wedge hasState_{(first,\ -4s)}(z, Stop) \end{aligned}$$

Query q_1 exhibits the different dimensions which need to be combined:
- $Vehicle(y)$ and $isManaged(x,z)$ are ontology atoms, which have to be unfolded in respect to the ITS domain modelled in the LDM ontology;
- $intersects(u,v)$ and $hasLocation(x,u)$ are spatial atoms, where the first checks spatial intersection and the second the assignment of a geometry to an object;
- $speed[avg, 4s](y,v)$ resp. $pos[line, 4s](y,r)$ defines a window operator that aggregates the average speed resp. positions (as points) of the vehicles over the streams

[5] see http://www.w3.org/2005/Incubator/geo/XGR-geo/ and http://www.geonames.org/ontology/

$speed$ and pos; $hasState[first, -4s](z, Stop)$ gives us the traffic lights, which switch in 4s to the state "Stop".

For query evaluation, we have adapted OQA to handle spatial and streaming data, which standard approaches as [11] do not consider. We extended the work of [16] with a window operator that (a) collects a set of data items (e.g., positions or speed) for each query atom from the underlying stream; and (b) calculates aggregation functions on the set of numerical (e.g., sum), sequential (e.g., first), and spatial (e.g., line) data items.

Data Model and Query Language. Our data model is *point-based* and captures the *valid time*, extracted from the V2X messages, saying that some *data item* is valid at that time point. To capture streaming data, we introduce the *timeline* $\mathbb{T}$, which is a *closed* interval of $(\mathbb{N}, \leq)$. A (data) *stream* is a triple $F = (\mathbb{T}, v, P)$, where $\mathbb{T}$ is a timeline, $v: \mathbb{T} \rightarrow \langle \mathcal{F}, \mathcal{S}_{\mathcal{F}} \rangle$ is a function that assigns to each element of $\mathbb{T}$ (called *timestamp*) data items of $\langle \mathcal{F}, \mathcal{S}_{\mathcal{F}} \rangle$, where F (resp. S_F) is a *stream (resp. spatial with streams) database*, and P is an integer called *pulse* defining the general interval of consecutive data items on the timeline (cf. [9]); this naturally induces as stream of data items. We always have a *main pulse* with a fixed interval length (usually 1) that defines the lowest granularity of the validity of data points. The pulse also aligns the data items, which arrive asynchronously in the database (DB), to the timeline. We allow additional *larger pulses* that generate streams with a lower frequency, which can be utilized to perform optimizations such as caching.

Example 2. For the timeline $\mathbb{T} = [0, 100]$, we have the stream $F_{CAM} = (\mathbb{T}, v, 1)$ of vehicle positions and speed at the assigned time points for the individuals c_1, c_2 and b_1:
$v(0) = \{speed(c_1, 30),\ pos(c_1, (5,5)),\ speed(c_2, 10),\ pos(c_2, (4,4)),$
$speed(b_1, 10), pos(b_1, (1,1))\},$
$v(1) = \{speed(c_1, 29),\ pos(c_1, (6,5)),\ speed(c_2, 0),\ pos(c_2, (5,4)),$
$speed(b_1, 5),\ pos(b_1, (2,1))\}$, and
$v(2) = \{speed(c_1, 34),\ pos(c_1, (7,5)),\ speed(c_2, 0),\ pos(c_2, (5,4))\}$.

A "slower" stream $F_{SPaT} = (\mathbb{T}, v, 5)$ captures the next signal state of a traffic light: $v(0) = \{hasState(t_1, Red)\}$ and $v(5) = \{hasState(t_1, Green)\}$. Further, the static ABox contains assertions $Car(c_1)$, $Car(c_2)$, $Bike(b_1)$, and $SignalGroup(t_1)$.

Our query language is based on CQs and adds spatial-stream capabilities (see Example 1). A spatial-stream CQ $q(\mathbf{x})$ is a formula:

$$\bigwedge_{i=1}^{l} Q_{O_i}(\mathbf{x}, \mathbf{y}) \wedge \bigwedge_{j=1}^{n} Q_{S_j}(\mathbf{x}, \mathbf{y}) \wedge \bigwedge_{k=1}^{m} Q_{F_k}(\mathbf{x}, \mathbf{y}) \qquad (1)$$

where $\mathbf{x}$ are the *distinguished* (*answer*) variables, $\mathbf{y}$ consists of *non-distinguished* (*existentially quantified*) variables, objects, and constant values:

- each $Q_{O_i}(\mathbf{x}, \mathbf{y})$ has the form $A(z)$ or $P(z, z')$, where A is a class name, P is a property name of the LDM ontology, and z, z' are from $\mathbf{x} \cup \mathbf{y}$;
- each atom $Q_{S_j}(\mathbf{x}, \mathbf{y})$ is from the vocabulary of spatial relations and of the form $S(z, z')$, where z, z' is as before and S is one of the following spatial relation $rel \in \{intersects,\ contains,\ nextTo,\ equals,\ inside,\ disjoint,\ outside\}$;
- $Q_{F_j}(\mathbf{x}, \mathbf{y})$ is similar to $Q_{O_i}(\mathbf{x}, \mathbf{y})$ but adds the vocabulary for stream operators, which are taken from [7] and relate to CQL operators [4]. We have a window $[agr, l]$ over a stream F_j, where l is the window size in time units (positive for past, or negative

for future). The *aggregate function* $agr \in \{count, sum, first, \ldots\}$ (see below for details) is applied to the data items in the window:[6]
- $[agr, l]$ represents the aggregate of last or next l time units of stream F_j;
- $[l]$ represents the single tuple of F_j at index l with $l = 0$ if it is the current tuple;
- $[agr, b, e]$: represents the aggregate on a window between b and e in the past/future of a stream.

The window $[agr, b, e]$ is derived from the work of [9] and allows us to query historic data (e.g., logs), if they are stored as streams.

Example 3. The following query counts the vehicles y that speed above 40km/h in the last 12 hours on intersection x and compares the count of vehicles z to the previous day:

$$\begin{aligned} q_2(x, y, z) : \ & Intersection(x) \wedge hasLocation(x, u) \wedge intersects(u, v) \wedge \\ & pos_{(line,\ 60s)}(y, v) \wedge Vehicle_{(count, 12h)}(y) \wedge speed_{(max,\ 60s)}(y, r) \wedge \\ & (r > 40) \wedge pos_{(line,\ 60s)}(z, v) \wedge Vehicle_{(count, 36h, 24h)}(z) \wedge \\ & speed_{(line,\ 60s)}(z, s) \wedge (s > 40) \end{aligned}$$

Query Rewriting by Stream Aggregation. We aim at answering pull-based queries at *a single* time point $\mathbb{T}_i$ with stream atoms that define *aggregate functions* on different windows sizes relative to $\mathbb{T}_i$. For this, we consider a semantics based on *epistemic aggregate queries* (EAQ) over ontologies [12] by dropping the order of time points for the data and handle the (streamed) data items as *bags* (multi-sets). Roughly, we perform two steps: (1) calculate only “known” solutions, and (2) evaluate the rewritten query, which includes the TBox axioms as well, over them. Each EAQ is evaluated over one or more *filtered and merged temporal* ABoxes. The filtering and merging, relative to the window size and $\mathbb{T}_i$, creates for each EAQ one (so-called) *windowed* ABox $A_{\boxplus_\phi}$, which is the union of the static ABox A and the filtered streaming data items from the stream database. The EAQ are then applied on $A_{\boxplus_\phi}$ by grouping and aggregating the normal objects, constant values, and spatial objects. We use a *bag-based epistemic semantics* for the queries, in which we locally close our world for the specific window and avoid “wrong” aggregations due to the open world semantics of DL-Lite$_A$. Further details on the algorithm for evaluating EAQs are provided in [17,18].

For *normal objects* and *constant values*, we allow three aggregate functions such as $count,\ first,\ last$ on the data items of a stream. In case of objects, e.g. vehicle c_1 other functions such as sum are not meaningful. For $last$ and $first$, we need to search the bag of data items, as the sequence of time points is lost. This is achieved by iteratively checking if we have a match at one of the time points. In the implementation, the first and last match can be simply cached while processing the stream.

Similar to [9] for *constant (numerical) values*, we allow a range of aggregation and prediction functions on the streamed data items:
- *simple*: $count,\ min,\ max,\ sum,\ first,\ last$, where $last$ and $first$ give the first or last element in the stream;
- *descriptive statistics* (DS): $mean,\ sd,\ var,\ median$, where sd (resp. var) calculate the standard deviation (resp. variance) and $median$ as expected;
- *predictions*: $line\_reg$ calculates the linear regression of tuples of data items and predicts future/past values.

[6] This would be represented in CQL as `R[Range L]`, `R[Now]`, `R[Range L Slide D]`, etc.

The DS and the predictions can be performed on the bag of data items or the result of a previous aggregations. For predictions and DS, we need to keep the temporal information as the order is lost using bags, as the prediction functions need the time dimension. Hence we tag the exact timestamp to each data item, e.g., $speed(c_1, 50)_{[10:05]}$ and $speed(c_1, 45)_{[10:07]}$ states that c_1 had a speed of 50 (resp. 45) at 10:05 (resp. 10:07).

Aggregation of Spatial Objects. For *spatial objects*, geometric aggregate functions are applied to the bag of data items that represent geometries. As with $first$ or $last$, we must rearrange them to create a valid geometry $g(s)$, i.e., a sequence $p = (p_1, \dots, p_n)$ of points p_i. We allow the following aggregate functions to derive new geometries:

- $point$: we evaluate $last$ to get the last available position p_1;
- $line$: we create $p = (p_1, \dots, p_n)$, where $p_1 \neq p_n$ and determine a total order on the bag of points, such that we have a starting point using $last$ and iterate backwards finding the next point by *Euclidean distance*;
- $line\_angle$: this aggregate function determines angles (in degrees) in a geometry by (1) applying the function $line$, (2) obtaining a simplified geometry using smoothing, and (3) calculating the angles between the lines of the simplified geometry;
- $polygon$: similar to $line$, but we create a polygon $(p_1, \dots, p_n)$, where $p_1 = p_n$ by: (1) determining the *convex hull* of the bag of points, and (2) extracting all pairs of points representing the convex hull;
- $traject$: The simplest approach to calculate trajectories is by using the function $line$. However, this is often not sufficient, and more complex smoothing and map matching functions are needed. For the prediction of the future paths, we allow different projections such a linear or curvature-based models. As an additional information, we need to include the maximal distance and step size for each time point. In a simple approximation the current speed can be taken, but also more elaborate models using velocity profiles could be applied.

Example 4. This query returns all vehicles y that will pass the crossing $C1$ in 10s:

$$\begin{aligned} q_3(x, y) : {} & Crossing(x) \wedge hasLocation(x, u) \wedge intersects(u, v) \wedge \\ & pos_{(traject,\ 10s)}(y, v) \wedge Vehicle(y) \wedge (x = "C1") \end{aligned}$$

Query Evaluation by Hypertree Decomposition. The main challenge is to handle three types of query atoms that need different evaluation techniques over possibly separate databases. Ontology atoms are evaluated over the static ABox A using a “standard” DL-Lite$_A$ query rewriting, i.e., PerfectRef [11]. For spatial atoms, we need to dereference the bindings to the spatial ABox S_A and evaluate the spatial relations (e.g., $inside$) on the spatial objects. Stream atoms are computed as EAQ over the windowed ABoxes of the different streams.

In [16], we introduced two spatial query evaluation strategies based on the assumptions that *no bounded variables* occur in spatial atoms and the CQ is *acyclic* (roughly, no proper cycle between join variables). As shown in [16], one of them is based on the query hypergraph and the derived join plan. This strategy is well-suited for lifting to spatial-stream CQs, as it gives us fine-grained caching, full control over the evaluation, and possibly handling different database entities. The details of decomposing an acyclic query into a hypergraph and the related join tree, we refer to [24].

The main steps of our query evaluation strategy is as follows. First, we construct the acyclic hypergraph H_q from q and label each hyperedge in H_q with l_O (ontology edge), l_S (spatial edge), and l_F (stream edge), where l_F gets the window size assigned. Then, we build the join tree J_q of H_q and extract the subtrees J_{ϕ_i} in H_q, such that each node is covered by the same labels, so we have sub-CQs that share the same aggregation/prediction functions and same window size l. For each subtree J_{ϕ_i}, we perform the detemporalization of the stream CQ q_{ϕ_i} by extracting and computing the results, which are stored in a virtual relation R_{ϕ_i} (a temporary table). Finally, we traverse J_q bottom up, left-to-right, to evaluate q_{ϕ_i} for each subtree J_{ϕ_i} (without stream atoms) and keep the results in memory for caching in future queries.

Example 5. The following query returns all lane change of vehicles z by checking if two lanes were crossed in the last 4s. The coloring distinguishes between ontology (orange), spatial (blue), and stream (green) atoms:

$$q_4(z) : LaneIn(x) \wedge hasLoc(x,u) \wedge intersects(u,w) \wedge LaneIn(y) \wedge hasLoc(y,v) \\ \wedge intersects(v,w) \wedge pos_{(line,\ 4s)}(z,w) \wedge Vehicle(z)$$

In Figure 2, we show the derived hypergraph that leads to the following decomposition and evaluation order:

(1) $q_{F1}(z,w) :\ Vehicle(z) \wedge pos_{(line,\ 4s)}(z,w)$
(2) $q_{N1}(x,u) :\ LaneIn(x) \wedge hasLoc(x,u)$; (3) $q_{N2}(y,v) :\ LaneIn(y) \wedge hasLoc(y,v)$
(4) $q_4(z) :\ q_{F1}(z,w) \wedge intersects(u,w) \wedge q_{N1}(x,u) \wedge intersects(v,w) \wedge q_{N2}(y,v)$

5 Stream Query Platform and Evaluation

We have implemented a prototype for our spatial-stream QA approach in JAVA 1.8 using the PIPELINEDB 9.6.1[7] as the stream RDBMS. We present the system architecture in Figure 3 and give the details on the components below.

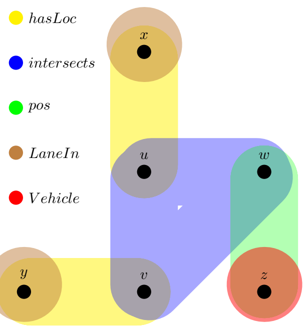


Fig. 2: Hypergraph of q_4

PipelineDB. We chose the open-source stream RDBMS PIPELINEDB 9.6.1, as it is built on top of PostgreSQL and PostGIS,[8] and allows us to support spatial data. PIPELINEDB distinguishes between *streams* and *continuous views*, where streams are write-only, so the query evaluator has to access the read-only continuous views. The database is modeled such that there are one-to-one mappings from streams to continuous views, and further to the TBox classes and properties; e.g., vehicle positions are fed into the stream *stream_pos* that is accessed via the continuous view *view_pos*, which is mapped to the property *pos*.

We also provide a simple integration framework that constantly receives V2X messages and adds the raw message data either to normal tables of the static database, to spatial tables of the GIS database, or the *streams* of the stream database.

[7] https://www.pipelinedb.com/
[8] http://postgis.net/ and https://www.postgresql.org/

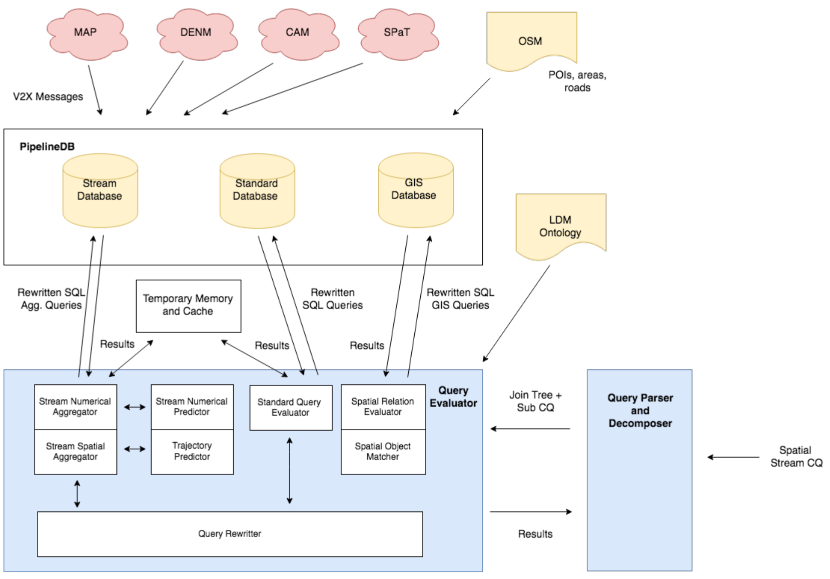


Fig. 3: System Architecture

Query Parser and Decomposer. After parsing the input spatial-stream CQ, the hypertree is decomposed using Gottlob et al.'s implementation [14].[9] Depending on the size of the CQ, the decomposition can be expensive, hence it is performed as a preprocessing step, whereas the decompositions are cached locally. The component gives us the joint tree J_q and the sub-CQs assigned to each tree node. For each node, we also keep the label that includes the subquery type, windows size, and aggregation/prediction function.

Query Evaluator. The query evaluator traverses J_q bottom up, left-to-right, and perform the steps: (1) check if the tuples resulting from a sub-CQ are in the cache; (2) if not, instantiate one of the evaluators according to the sub-CQ type. We have (partially) implemented the following sub-CQ evaluators:
Ontology evaluator: we use the standard query evaluator, which is based on the (simple) query rewriter of OWLGRES 0.1 [34]; we plan to replace it with more efficient implementations such as ONTOP [32].
Spatial evaluator: the spatial evaluator is responsible for handling the spatial relations such as $intersects$. As mentioned in [16], we evaluate the spatial relations in-memory instead of compiling them into a large rewritten SQL. Each spatial relation is evaluated as a join using the functions of the JTS TOPOLOGY SUITE.[10] Furthermore, this component calculates the matches between a trajectories and geometries (e.g., extracted from the road graph) using the map matching algorithm as described by Newson and Krumm [26].[11]

[9] https://www.dbai.tuwien.ac.at/proj/hypertree/
[10] https://github.com/locationtech/jts
[11] https://github.com/graphhopper/map-matching

Stream evaluator: this evaluator is responsible for detemporalizing the streams by grouping and aggregating the data items. For a stream sub-CQ q_i, we perform the following steps:

(1) extract the data items according to the window size derived from l, b, and e;
(2) evaluate the query q_i without rewriting and store the "known solutions" in memory as $R_{i,1}$;
(3) evaluate the rewritten query q'_i over q_i and store it in memory as $R_{i,2}$;
(4) apply the prediction function on $R_{i,2}$ and add the newly predicted data items;
(5) apply the grouping and aggregation function on $R_{i,2}$, and produce the outcome $R_{i,3}$.

In our current evaluator, the prediction function is a preprocessing step for the aggregation, and is not yet considered as an independent step. If we would consider predictions independently, the prediction atoms could be considered as streams themselves, which generate new data points. However this would change the query evaluation process and would need to be investigated further. The function $traject$ is designed with the same intention; we take the existing points (as coordinates) and project a single path into the future. Currently, we ignore the curvature and use a simple straight-line projection to create new points. However, taking the curvature into account is a desired extension.

Evaluation of Features. We evaluate our platform regarding the list of features and rely in the evaluation on our previous results of [18], since we already have implemented and evaluated parts of the platform. We point out that the implementation and benchmarking is ongoing work. Following, the detailed comments on the evaluation:

- *F1*: the work in [18] is based on the point-based model, where we have shown the feasibility for a set of nine queries in two scenarios. We aim to add an *interval-based* model as an additional layer to the query evaluation. However, this is not natively supported by PIPELINEDB, and the intervals need to be evaluated (internally) in-memory. Intervals are a natural extension, if we want to deal with the results of aggregations over a window, since these aggregations can be expressed as intervals, e.g., the average speed in the last 20 seconds.
- *F2*: our initial prototype is based on the evaluation of pull-based queries which is in the nature of OQA. An appealing extension are push-based queries in the OQA setting, where the query decomposition and *pulses* are suitable technologies to be used, since they allow the partial evaluation of a query including the arrival estimation off the next data items. However, our prototype's database PIPELINEDB would need to be replaced by an event-driven (reactive) RDBMS such as ESPER.[12]
- *F3*: also in [18], we have evaluated the (spatial) point-set model, and believe that the extension to the 9-Intersection model is straightforward. Spatial relations using in RCC8 is beyond our current work. It was shown in [27] that an extension for DL-Lite is feasible.
- *F4*: the extension of DL-Lite with LTL resp. MTL has been thoroughly investigated in [5] resp. [9]. For our work, the mentioned temporal logics are interesting extension, and we aim to combine them with the existing query language.
- *F5*: numerical and spatial aggregations are a central element of our approach, and we have implemented the mentioned numerical and spatial aggregation functions.

[12] http://www.espertech.com/esper/

New to this work are the functions for descriptive statistics, which extend the existing functions of [18].
- *F6* and *F8*: a new extension is the calculation of trajectories including the prediction of future paths. We still have to implement and evaluate the trajectory calculation thoroughly and add more advanced spatial functions as smoothing and simplification of objects. The initial implementation of trajectory computations with predictions is based on a simple linear path calculation.
- *F7*: we calculate the linear regression of (time-annotated) data items, which is a standard statistical method that does not need any parameterization or training. If other methods, e.g., k-nearest-neighbour regression, would be added, our query language and evaluator would need to be extended, since the parameterization and training steps is needed, taking the changing nature of streamed data into account. Further, we believe that tying the predictions directly to the aggregation function is not very intuitive, and a more flexible method (using the hypergraph decomposition) of finding the related prediction and aggregation functions is desired.
- *F9*: important for detecting maneuvers, we have implemented a map matching algorithm that links the points of a trajectory to the underlying (road) graph. However, this functions could be extended by to similarity detect between two trajectories.
- *F10*: the calculation of graph connectivity that requires transitivity, NAF, and repairing change the semantics and increase computational complexity of our language, and are longer-ranged goals.

6 Related Work and Conclusions

Data stream management systems (DSMSs) such as STREAM [4], were built supporting streaming applications by extending RDBMS [19]. More recently, RDF stream processing engines, such as C-SPARQL [6], SPARQLstream [10], and CQELS [23], were proposed for processing RDF streams integrated with other linked data sources. Besides C-SPARQL, most of them follow the DSMSs paradigm and do not support stream reasoning. EP-SPARQL [3] and LARS [7] propose languages that extend SPARQL respectively CQs with stream reasoning, but translate KBs into more expressive (less efficient) logic programs. Regarding spatio-temporal RDF stream processing, a few SPARQL extensions were proposed, such as SPARQL-ST [28] and st-SPARQL [21]. Closest to our work are (i) [30], which supports spatial operators as well as aggregate functions over temporal features (ii) [10], which allows evaluating OQA queries over stream RDBMS, and (iii) [9], which extends SPARQL with aggregate functions (using advanced statistics) evaluated over streamed and ordered ABoxes. This work differs regarding (a) the evaluation approach using EAQ with aggregates on the query and not ontology level, (b) hypergraph-based query decomposition, and (c) the main focus of querying streams of spatial data in an OQA setting. In [5], temporal QA over DL-Lite using temporal operators of MTL and LTL are evaluated over a (two-sorted) model separating the object and temporal domain. Similar, temporal QA is investigated in [8] and [20], which lack implementations yet. Finally, we build on the results for EAQs in [12], but we introduce spatial streams and more complex queries. The authors of [13] define three entailment levels for stream reasoning: stream-, window-, and graph-level entailment. Our approach is a mix of graph- and window-level entailment, where we complete the static and the stream items w.r.t. the domain ontology before being aggregated and evaluated. If we would extend it to a push-based paradigm, we also would have to take incremental reasoning into account.

This work is sparked by the need of detecting mobility patterns based on the LDM for V2X communications, where the LDM serves as an integration effort for streaming data (e.g., vehicle movements) in a spatial context (e.g., intersections) over a complex domain (e.g., a mobility ontology). In previous work, we have introduced simple aggregate queries over mobility streams, which often do not suffice to capture more complex movement patterns. Hence, we present an extension of the previous aggregation functions with statistical and predictive functionality, which allows us to query more complex mobility patterns such as road networks statistics, vehicles maneuvers, and temporal connected events such as (potential) accidents. The extension is based on the newly developed scenarios traffic statistics, vehicle maneuvers, and event detection. We have defined a set of domain-specific features such as trajectory computation and spatial matching, which are matched with the use cases and scenarios to define the requirements.

Given by the new features, we extend our previous LDM ontology, the current spatial-stream query language, and adjust our methods to take the features into account. Further, we redesigned our system architecture and give insights in the new components used for prediction and trajectory calculation.

Future work. Currently, we are implementing the platform thoroughly and will provide benchmarks based on a set of queries related to the three scenarios. As discussed in the previous section, our ongoing and future research is directed to extend our language, methods, and the platform to fulfill the defined requirements, which will allow to use the platform in the mentioned scenarios.

References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM 26(11), 832–843 (1983)
2. Andreone, L., Brignolo, R., Damiani, S., Sommariva, F., Vivo, G., Marco, S.: Safespot final report. Tech. Rep. D8.1.1 (2010), available online.
3. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: a unified language for event processing and stream reasoning. In: Proc. of WWW 2011. pp. 635–644 (2011)
4. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. VLDB J. 15(2), 121–142 (2006)
5. Artale, A., Kontchakov, R., Kovtunova, A., Ryzhikov, V., Wolter, F., Zakharyaschev, M.: First-order rewritability of temporal ontology-mediated queries. In: Proc. of IJCAI 2015. pp. 2706–2712 (2015)
6. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: C-sparql: a continuous query language for rdf data streams. Int. J. Semantic Computing 4(1), 3–25 (2010)
7. Beck, H., Dao-Tran, M., Eiter, T., Fink, M.: LARS: A logic-based framework for analyzing reasoning over streams. In: Proc. of AAAI 2015. pp. 1431–1438 (2015)
8. Borgwardt, S., Lippmann, M., Thost, V.: Temporalizing rewritable query languages over knowledge bases. J. Web Sem. 33, 50–70 (2015)
9. Brandt, S., Kalayci, E.G., Kontchakov, R., Ryzhikov, V., Xiao, G., Zakharyaschev, M.: Ontology-based data access with a horn fragment of metric temporal logic. In: Proc. of AAAI 2017. pp. 1070–1076 (2017)

10. Calbimonte, J., Mora, J., Corcho, Ó.: Query rewriting in RDF stream processing. In: Proc. of ESWC 2016. pp. 486–502 (2016)
11. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *dl-lite* family. J. Autom. Reasoning 39(3), 385–429 (2007)
12. Calvanese, D., Kharlamov, E., Nutt, W., Thorne, C.: Aggregate queries over ontologies. In: Proc. of ONISW 2008. pp. 97–104 (2008)
13. Dell'Aglio, D., Valle, E.D., van Harmelen, F., Bernstein, A.: Stream reasoning: A survey and outlook. Data Science, IOS Press, (to appear) (2017)
14. Dermaku, A., Ganzow, T., Gottlob, G., McMahan, B.J., Musliu, N., Samer, M.: Heuristic methods for hypertree decomposition. In: Proc. of MICAI 2008: Advances in Artificial Intelligence. pp. 1–11 (2008)
15. Egenhofer, M.J., Franzosa, R.D.: Point set topological relations. International Journal of Geographical Information Systems 5(2), 161–174 (1991)
16. Eiter, T., Krennwallner, T., Schneider, P.: Lightweight spatial conjunctive query answering using keywords. In: Proc. of ESWC 2013. pp. 243–258 (2013)
17. Eiter, T., Parreira, J.X., Schneider, P.: Towards spatial ontology-mediated query answering over mobility streams. In: Proc. of Stream Reasoning Workshop 2016 (2016)
18. Eiter, T., Parreira, J.X., Schneider, P.: Spatial ontology-mediated query answering over mobility streams. In: Proc. of ESWC 2017. To appear (2017)
19. Golab, L., Özsu, M.T.: Issues in data stream management. SIGMOD Rec. 32(2), 5–14 (2003)
20. Klarman, S., Meyer, T.: Querying temporal databases via OWL 2 QL. In: Proc. of RR 2014. pp. 92–107 (2014)
21. Koubarakis, M., Kyzirakos, K.: Modeling and querying metadata in the semantic sensor web: The model strdf and the query language stsparql. In: ESWC 2010. pp. 425–439 (2010)
22. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems 2(4), 255–299 (1990)
23. Le-Phuoc, D., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: ISWC 2011. pp. 370–388 (2011)
24. Maier, D.: The Theory of Relational Databases. Computer Science Press (1983)
25. Netten, B., Kester, L., Wedemeijer, H., Passchier, I., Driessen, B.: Dynamap: A dynamic map for road side its stations. In: Proc. of ITS World Congress 2013 (2013)
26. Newson, P., Krumm, J.: Hidden markov map matching through noise and sparseness. In: Proc. of ACM SIGSPATIAL 2009. pp. 336–343 (2009)
27. Özçep, Ö.L., Möller, R.: Scalable geo-thematic query answering. In: The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I. pp. 658–673 (2012)
28. Perry, M., Jain, P., Sheth, A.P.: SPARQL-ST: extending SPARQL to support spatiotemporal queries. Geospatial Semantics and the Semantic Web 12, 61–86 (2011)
29. Pnueli, A.: The temporal logic of programs. In: Proc. of Annual Symposium on Foundations of Computer Science 1977. pp. 46–57 (1977)
30. Quoc, H.N.M., Le Phuoc, D.: An elastic and scalable spatiotemporal query processing for linked sensor data. In: Proc. of SEMANTICS 2015. pp. 17–24. ACM (2015)
31. Randell, D.A., Cui, Z., Cohn, A.G.: A spatial logic based on regions and connection. In: KR. pp. 165–176 (1992)
32. Rodriguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Ontology-based data access: Ontop of databases. In: Proc. of ISWC 2013. pp. 558–573 (2013)
33. Shimada, H., Yamaguchi, A., Takada, H., Sato, K.: Implementation and evaluation of local dynamic map in safety driving systems. J. Transportation Technologies 5(2), 102–112 (2015)
34. Stocker, M., Smith, M.: Owlgres: A scalable owl reasoner. In: Proc. of OWLED 2008 (2008)
35. Stonebraker, M., Çetintemel, U., Zdonik, S.B.: The 8 requirements of real-time stream processing. SIGMOD Record 34(4), 42–47 (2005)