Deploying Spatial-Stream Query Answering in C-ITS Scenarios

Thomas Eiter[1], Ryutaro Ichise[2,3], Josiane Xavier Parreira[4], Patrik Schneider[1,4], and Lihua Zhao[3]

[1] Vienna University of Technology, Vienna, Austria
[2] National Institute of Informatics, Tokyo, Japan
[3] National Institute of Advanced Industrial Science and Technology, Tokyo, Japan
[4] Siemens AG Österreich, Vienna, Austria

Abstract. Cooperative Intelligent Transport Systems (C-ITS) play an important role for providing the means to collect and exchange spatio-temporal data via V2X between vehicles and the infrastructure, which will be used for the deployment of (semi)-autonomous vehicles. The Local Dynamic Map (LDM) is a key concept for integrating static and streamed data in a spatial context. The LDM has been semantically enhanced to allow for an elaborate domain model that is captured by a mobility ontology, and for queries over data streams that cater for semantic concepts and spatial relationships. We show how this approach can be extended to address a wider range of use cases in the three C-ITS scenarios *traffic statistics*, *events detection*, and *advanced driving assistance systems*. We define for them requirements derived from necessary domain-specific features and report, based on them, on the extension of our query language with temporal relations, delaying, numeric predictions and trajectory predictions. An experimental evaluation of queries that reflect the requirements, using the real-world traffic simulation tool provides evidence for the feasibility/efficiency of our approach in the new scenarios.

1 Introduction

The development of (semi)-autonomous vehicles involves 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 static, semi-static, and dynamic information in a spatial context.

In previous work, we have semantically enhanced the LDM to allow for an elaborate domain model that is captured by a mobility ontology, and for queries over data streams that cater for semantic concepts and spatial relationships [14]. Our approach is based on ontology-mediated query answering (OQA) and features conjunctive queries (CQs) over DL-Lite$_A$ [10] ontologies that support window operators over streams and spatial relations between objects. We believe that OQA is well suited for C-ITS applications, as an ontology can be used to model vehicles, traffic, and infrastructure details, and map to scalable stream database technology adding dynamicity to the model. For example, the definition of a hazardous situation is complex, ranging from bad road conditions to traffic jams [2]. Therefore, an expressive query language is crucial to fulfill C-ITS specific requirements needed for retrieving dynamic data and expressing complex patterns regarding, e.g., event detection. Furthermore, scalability and swift response time are crucial since fast changing traffic demands a quick response time to avoid accidents [2].

In this paper, we continue the work in [13,14] with the goal of showing how spatial-stream OQA can be used to address a wider set of C-ITS scenarios. For achieving this, the approach in [14] is extended with new domain-specific features beyond "generic" spatial-stream OQA. In cooperation with ITS domain experts from Siemens and the Austrian Institute of Advanced Industrial Science and Technology (AIST), the C-ITS scenarios – *traffic statistics*, *events detection*, and *advanced driving assistance systems* (ADAS) – were defined and used to single out requirements derived from a domain-specific list of features. We then formulate for each use case, requirements that should be covered by our approach. The focus of the new, more specific features will be on *temporal relations*, e.g., $during$, as well as numerical and trajectory *predictions*. For the assessment, we provide a detailed report on the extension of the implementation with the new features such as the temporal relations. The implementation is evaluated in an experimental setting using queries matching to features, where a real-world traffic simulation is used to generate the data. The results provide evidence for the feasibility and efficiency of our approach in these scenarios. Our contributions are briefly summarized as follows:[5]

- we outline the field of V2X integration using LDMs and provide details on our ontology-based LDM (Section 2);
- we define three scenarios, use cases, desired features, and requirements (Section 3);
- we present our current approach including data model, query language, and outline the implemented features (Section 4 and 5);
- we evaluate our platform regarding the set of features/requirements based on a traffic simulation and assess the results (Section 6).

In Section 7, we discuss related work, and conclude with ongoing and future work.

2 C-ITS Data Integration and Query Answering

Our setting is the ongoing efforts in data integration and querying in the C-ITS domain. The base technologies for C-ITS are already available and experimentally deployed in infrastructure projects as in [2]. The communication technology is based on the IEEE 802.11p standard, and the data integration effort is the *Local Dynamic Map* (LDM), which are starting points for our work. IEEE 802.11p allows wireless access in vehicular environments, called V2X communications, which enables messaging between vehicles and the infrastructure. The messages are broadcast every 100ms by traffic participants, i.e., vehicles and roadside ITS stations, to update other participants about their current states [2]. The main standardized message types are *CAMs* (Cooperative Awareness Messages) for frequency status updates of participants, *MAPs* (Map Data Messages) for detailed intersection topologies, *SPATs* (Signal Phase and Timing Messages) for traffic light signal phases, and *DENMs* (Decentralized Environmental Notification Messages).

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 an LDM, which acts as an integration platform to combine static geographic information system (GIS) maps, with dynamic environmental objects (e.g., vehicles or pedestrians). The integration is motivated by advanced safety applications, which need an "overall" understanding of a traffic environment. The LDM consists of the four layers (see Figure 1a): *permanent static*, *transient static*, *transient dynamic*, and *highly dynamic*, ranging from dynamic (as V2X messages) to permanent static (as GIS maps) information. Recent research by Netten et al. [18], and Shimada et al. [21] suggested that an LDM can be built on top of a spatial relational RDBMS enhanced with streaming capabilities. Netten et al. recognize that an LDM should be represented by a world model, world objects, and data sinks on the streamed input [18]. However, an elaborate domain model captured by an LDM ontology, and extended query processing

[5] As to [14], Sec. 3, 5, 6 are entirely new content, and 4 changed with the focus on new features.

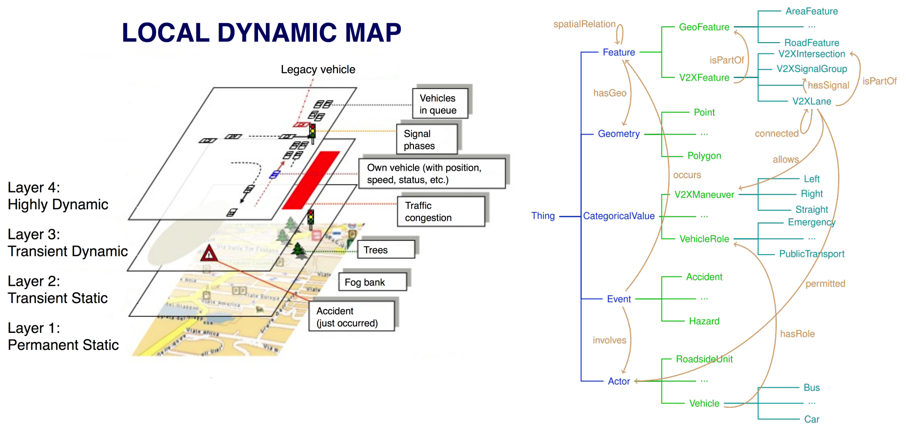

Fig. 1: (a) The four Layers of a LDM [2] and (b) LDM Ontology

or rule evaluation methods over spatial data streams, were still missing in the current approaches. An ontology-based LDM has advantages regarding the maintainability and understandability of the model, since dependencies between the concepts are clearly defined and easy extendable without altering the underlying database (DB).

Ontology-based LDM. With the support of Siemens and AIST domain experts, we have worked on our LDM ontology (shown partially in Figure 1b, available at http://www.kr.tuwien.ac.at/research/projects/loctrafflog/LocalDynamicMapITS-v0.4-Lite.owl) to capture the four levels of the LDM, as well as V2X-specific elements such as maneuvers. The LDM ontology is represented in DL-Lite$_A$ [10], which is the logical underpinning for the W3C standard OWL 2 QL. Apart from the restriction to DL-Lite$_A$, our methods are ontology-agnostic; hence other mobility ontologies could be used. We follow a layered approach starting with a simple separation between the top concepts of $V2XFeature$ that is the representation of V2X objects, such as details of an intersection topology including lanes ($V2XLane$) and traffic lights ($V2XSignalGroup$). $GeoFeature$ represents the GIS aspects of the LDM including POIs, areas like parks, and road networks with $Geometry$ as the geometrical representation of them. $Actor$ is the concept that includes persons, vehicles, as well as roadside ITS stations, which are autonomous and are the main generator of streamed data. $CategoricalValues$ specify the different categories such as signal phases, or vehicle roles used in the emergency domain. Besides "domain specific" roles and attributes like $speedLimit$, $hasRole$, $speed$, or $position$, we also introduced generic roles that have an inherent meaning, e.g., $isPartOf$.

Spatial-Stream Query Answering. The OQA component is central to the usage of a semantically enhanced LDM, since it allows us to access the streamed data in the LDM.

Example 1. The following query detects red-light violations on intersections by searching for vehicles (in y) with an *aggregated* trajectory and speed above 30km/h in a 8s window, projecting 3s into the future (represented as a negative time point), which move on lanes (in x) *during* these lanes signals will turn to "Stop", i.e., red, in a 10s window:

$$\begin{aligned} q_1(x,y) : \ & LaneIn(x) \wedge hasLoc(x,u) \wedge intersects(u,v) \wedge Vehicle(y) \\ & \wedge\ pos(y,v)[\boldsymbol{traject\_line},\ 5s, -3s] \wedge speed(y,r)[\boldsymbol{mov\_avg},\ 5s, -3s] \\ & \wedge\ (r > 30) \wedge \boldsymbol{during(v,s)} \wedge\ isManaged(x,z) \\ & \wedge\ SignalGroup(z) \wedge hasState(z,s)[last,\ 5s, -5s] \wedge (s = 'Stop') \end{aligned}$$

Query q_1 exhibits the different dimensions that need to be combined: (a) $LaneIn(x)$, $Vehicle(y)$ and $isManaged(x,z)$ (assigning traffic lights z to lanes x) are ontology atoms, which have to be unfolded in respect to the concept/role hierarchies of the LDM ontology;

(b) $intersects(u, v)$ and $hasLoc(x, u)$ are spatial atoms, where the first checks spatial intersection and the second returns the object geometries; (c) $speed(y, r)[traject\_line, 5s, -3s]$ and $pos(y, v)[mov\_avg, 5s, -3s]$ define window operators that aggregate and *predict* the moving average of speed and positions of the vehicles over $speed$ and pos, respectively, and $hasState(z, s)[last, 5s, -5s]$ returns the traffic lights that have their last phase on "Stop"; (d) the relation $during(v, s)$ checks if "v happens during s", where v is all the occurrences of trajectories on the set of time intervals $t1$, and s are the traffic light phases that are on "Stop" in the set of time intervals $t2$, were $t1$ and $t2$ are derived from the trajectory aggregations and the phase duration of the traffic lights.

3 Development of C-ITS Scenarios

In this section, we present three application scenarios that are used to define requirements and features split into three complexity levels. On the infrastructure side, we have C-ITS (roadside) stations that receive nearby V2X messages and send messages to inform other participants on their current state, i.e., the traffic light phases. Other participants such as vehicles share their states such as their current speed, acceleration, and position. On the vehicle side, ADAS perceive driving environments and make safe driving decisions to improve safety of autonomous vehicles. The ADAS use sensors such as Lidar/Radar or cameras, and process the sensor data to avoid accidents by detecting pedestrians, vehicles, or other obstacles [23]. The sensor data can be linked to our ontology-based LDM and enables the system to represent the driving environments.

S1: Traffic Statistics. The focus of this scenario is on the collection of statistical data that concerns 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 related challenges:

1. *Object level*: for a single vehicle or station, the average speed, acceleration, number of stops, or on a sensor data such as the temperature could be collected;
2. *Road/Intersection level*: on this level, besides calculating a summary of road/lane level indicators such as average throughput, waiting time, the amount of stops, also matrices regarding transfers (e.g. how many cars head straight on), modality, and type mix, (e.g. which vehicle classes are present) could be determined;
3. *Network level*: on the network level, intersections are represented by nodes connected by roads. We could collect statistical summaries of indicators on intersections. For instance, estimating the transfer times and traffic flow between intersections.

S2: Hazardous Events Detection. An important C-ITS application is *road safety* [2], where a reliable event detection is central to find unexpected, hazardous events. This is a more challenging case, since it requires the combination of the topology, vehicle maneuvers, and temporal relations that might be evaluated over longer and shorter periods. We identified the following events as possibly hazardous:

1. *Simple vehicle maneuvers*: the following maneuvers are relevant for this case and are directly extractable from trajectories: (1) quick slow down/speed up; (2) drive straight on, turn left, turn right; (3) stop, unload, park;
2. *Complex vehicle maneuvers*: the aim is to detect lane changes, overtakes, u-turn are complex maneuvers, which are a composite of simple maneuvers;
3. *Red-light violation*: as shown in Ex. 1, red-light violations can be detected by checking the spatial intersection of lanes that change to "Stop" and vehicles current trajectory taking their speed into account. This could be enhanced by trajectory predictions;
4. *Vehicle breakdown/accident*: this event is based on the stop maneuver, where we identify vehicles that are not moving and are inside a dangerous area of an intersection. This case can be extended to several vehicles;

5. *Traffic congestions*: this is a more complex event, where short and long term observations must be combined. Queuing cars could indicate a congestion 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.

S3: ADAS and Autonomous Driving. ADAS are an important step towards autonomous driving by enabling the vehicle to take control of speed or breaking, where drivers still have the "full" control over the vehicle. The following challenges come for ADAS:

1. *Self monitoring*: Self-monitoring is a central requirement of ADAS, where intelligent speed adaptation is an important feature to improve roadway safety;
2. *Obstructed view*: It concerns dangerous situations where a vehicle might collide with another vehicle, since they have no visual contact due to an obscured view (e.g., buildings). The overlap of predicted trajectories of two vehicles should be checked.
3. *Traffic rules*: The embedding of traffic rules like checking of traffic rules such as right-of-way rules could become an important requirement for autonomous driving.

Features for Spatial-Stream QA. The eight "standard" requirements: volume, velocity, variety, incompleteness, complex domain models, etc., as well as the three entailment levels for stream reasoning systems: stream-, window-, and graph-level entailment identified by [12] are not discussed here, but should hold for mobility stream systems as well. Besides the generic features *F1*, *F2*, *F3*, and *F9*, we also focus on domain specific features that are mapped to requirements crucial for enabling the above scenarios. For this, we distinguish for each feature the levels of fulfillment *basic* (L1), *enhanced* (L2), and *advanced* (L3). We have identified the following feature sets:

- *F1 - Time model*: possible time models are *point-based* (L1), and *interval-based* (L2), where L1 is the "simplest" representation. On point-based data, applying aggregations can be represented by intervals based on point-based data items. If we apply an interval-based model, temporal relations (L3) such as Allen's Time Interval Algebra [1] with operators like *before* can be used for querying and inference.
- *F2 - Process paradigm*: queries that are processed in a pull-based (L1) manner should be the baseline. Push-based processing (L2) in particular with sliding windows is already more challenging. If we allow a combined (L3) processing, we could treat high velocity (resp. low velocity) atoms as push-based (resp. as pull-based).
- *F3 - Query features*: these "basic" features include nesting of queries (L1), or unions of CQs (L2). Other feature relate to the computation of spatial relations using a simple point-set model (L1) or the more detailed *9-Intersection model* (L2).
- *F4 - Numerical aggregations*: aggregations can be "simple" functions such as *sum* or *average* on either a set or multiset (bag) of data items (L1). These could be extended by basic statistical function such as *median* (L2). Aggregation over multisets is important, since we often have data items of different objects in a single stream.
- *F5 - Spatial aggregations*: a wide range of spatial aggregations can be applied to geometric objects like points and lines (L1) and the aggregation functions need to take the peculiarities of geometries into account, e.g., convex vs. concave objects. Smoothing and simplification of complex objects could also be included (L2).
- *F6 - Numerical predictions*: predictions allow the generation of unknown data items projecting from the past into the future. Several prediction functions such as moving average (L1) or exponential smoothing (L2) regression should be available. Depending on the task, also more complex machine learning methods could be envisioned (L3).
- *F7 - Trajectory predictions*: we predict a vehicle's movement, by linearly projecting the trajectory into the future (L1). More accurate results could be achieved by (1) a

Table 1: Requirement Matrix (L1/L2/L3 is required, blank is not required, P is possibly)

Use Case	F1	F2	F3	F4	F5	F6	F7	F8	F9
S1.1 (Object statistics)	L1	L1		L2	P				
S1.2 (Road/Intersection statistics)	L2	L1	L2	L2	L1	L1	L1	P	
S1.3 (Network statistics)	L2	L1	L2	L2	L1	L2	L1	P	P
S2.1 (Simple maneuvers)	L1	L1	L1	L1	L1	P	P	P	
S2.2 (Complex maneuvers)	L2	L2	L2	L1	L1	L1	L1	L1	P
S2.3 (Red-light violation)	L2	L2	L2	L1	L1	L1	L1	L1	P
S2.4 (Vehicle breakdown)	L2	L1	L2	L1	L1	L1		P	P
S2.5 (Traffic congestion)	L2	L3	L2	L2	L1	L2		P	P
S3.1 (Self monitoring)	L1	L2	L1	L1	P	P	P		
S3.2 (Obstructed view)	L1	L2	L2	L1	L1	L1	L1	L1	P
S3.3 (Traffic rules)	L2	L2	L2	L2	L2	L1	L1	L1	L1

"point-to-curve" aggregation, and (2) calculating possible paths using a road graph (L2), and the usage of machine learning for trajectory predictions (L3).

- *F8 - Spatial matching*: basic spatial matching is the extraction of specific features such as angles from the objects (L1). Advanced features include the matching of complex geometries such as road graphs (L2).
- *F9 - Rules*: rules will reach beyond the expressivity of query answering and can include "simple" implications as $b(x, y) \wedge c(y, z) \rightarrow a(x, z)$ (L1), but also more advanced features such as aggregation, negation as failure (L2), and recursion (L3).

Requirements. In Table 1, we show the requirements that are derived by analyzing each scenario and use case regarding the necessary features. The requirements build the base line for the implementation and a later experimental assessment. In case of single features, we only distinguish between L1 to L3 for required, blank for not required, and "P" for possibly required. For instance, in *S2.2* for F1, a point-based time model (L1) suffices for detecting left/right turns, however, if we want to detect u-turns, interval-based model in combination with temporal relations (L2) might be needed. Furthermore, push-based queries are desired for swift reaction on changes. In [14], we partially support L1 for *F1* to *F5*, but we aim to push the level beyond that and need new features such as time intervals, temporal relations, and unions of CQs. *F6*, *F7*, and *F8* are entirely new features.

4 Approach for Spatial-Stream Query Answering

We start from previous work in [14], which introduced spatial ontology-mediated query answering over Mobility Streams using DL-Lite$_A$ [10]. We focus on pull-based queries that are evaluated at one single time point called the *query time* $\mathbb{T}_i$.

Data Model and Knowledge Base. 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. Importantly, while evaluating a query, the model can change (temporary) to an *interval-based* model that results from the window and aggregation functions. 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 $D{=}(\mathbb{T}, v, P)$, 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 $\mathbb{T}$ data items of $\langle \mathcal{F}, \mathcal{S}_{\mathcal{F}} \rangle$, where $\mathcal{F}$ (resp. $\mathcal{S}_{\mathcal{F}}$) is a *stream (resp. spatial-stream) DB*, and the integer P is called *pulse* defining the general interval of consecutive data items on the timeline (cf. [8]); this naturally induces a stream of data items. We always have a *main pulse* with a fixed interval length that defines the highest granularity of the validity of data points, and *larger pulses* for streams with lower frequency can be defined. The pulse also aligns the data items that arrive asynchronously in the DB to the timeline.

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:
$v(0) = \{speed(c_1, 30),\ pos(c_1, (5, 5)),\ speed(c_2, 10),\ pos(c_2, (4, 4)), speed(b_1, 10), ...\}$
$v(1) = \{speed(c_1, 29),\ pos(c_1, (6, 5)),\ speed(c_2, 0),\ pos(c_2, (5, 4)), speed(b_1, 5), ...\}, ...$
A "slower" stream $F_{SPaT} = (\mathbb{T}, v, 5)$ captures the next signal state of a traffic light: $v(0) = \{hasState(t_1, Stop)\}$ and $v(5) = \{hasState(t_1, Go)\}$. The static ABox contains assertions $Car(c_1)$, $Car(c_2)$, $Bike(b_1)$, and $SignalGroup(t_1)$. A different "annotated" representation by applying the function v on F_{CAM} yields $\{speed(c_1, 30)\ @t0, ...,$ $speed(c_1, 29)@t1\}$, which is better suited for an interval-based time model.

We consider a vocabulary of individual names Γ_I, domain values Γ_V (e.g., $\mathbb{N}$), and spatial objects Γ_S. A *spatial-stream knowledge base* is a tuple

$$\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{S}_\mathcal{A}, \langle \mathcal{F}, \mathcal{S}_\mathcal{F} \rangle, \mathcal{B} \rangle,$$

where $\mathcal{T}$ ($\mathcal{A}$, resp.) is a DL-Lite$_A$ TBox (ABox , resp.), $\mathcal{S}_\mathcal{A}$ is a spatial DB, and $\langle \mathcal{F}, \mathcal{S}_\mathcal{F} \rangle$ is a stream DB with spatial data support. Furthermore, $\mathcal{B} \subseteq \Gamma_I \times \Gamma_S$ is a partial function called the *spatial binding* from $\mathcal{A}$ to $\mathcal{S}_\mathcal{A}$ and $\mathcal{F}$ to $\mathcal{S}_\mathcal{F}$. The TBox $\mathcal{T}$ and the ABox $\mathcal{A}$ consist of finite sets of *inclusion assertions*, *functionality assertions*, and *membership assertions*. To specify the *localization* of atomic concepts and roles. we extended standard DL-Lite$_A$ (see [10]) with axioms $(loc\ A)$, $(loc_s\ A)$, and $(loc_s\ Q)$ that assign an unspecific or particular location to instances of atomic concepts A or basic roles Q. The extension with streaming consists of the axiom schemes

$$(stream_D\ C) \text{ and } (stream_D\ R),$$

where D is a particular stream over either complex concepts C or roles R in $\langle \mathcal{F}, \mathcal{S}_\mathcal{F} \rangle$. More details are given in [14].

Example 3. A TBox may contain $(stream_{CAM}\ speed)$, $(stream_{CAM}\ (loc\ pos))$, $(stream_{CAM}\ Vehicle)$, and $(stream_{SPaT}\ hasState)$, and we have further axioms $Car \sqsubseteq Vehicle$, $Bike \sqsubseteq Vehicle$, and $Ambulance \sqsubseteq \exists hasRole.Emergency$.

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

$$\textstyle\bigwedge_{i=1}^{m} Q_{O_i}(\mathbf{x}, \mathbf{y}) \wedge \bigwedge_{j=1}^{n} Q_{S_j}(\mathbf{x}, \mathbf{y}) \wedge \bigwedge_{k=1}^{o} Q_{D_k}(\mathbf{x}, \mathbf{y}) \wedge \boldsymbol{\bigwedge_{l=1}^{p} Q_{T_l}(\mathbf{x}, \mathbf{y})} \qquad (1)$$

where $\mathbf{x}$ are the *distinguished* (*answer*) variables, $\mathbf{y}$ consists of *non-distinguished* (*existentially quantified*) variables, objects, and constant values:
- each atom $Q_{O_i}(\mathbf{x}, \mathbf{y})$ 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 $\mathbf{x}$ or $\mathbf{y}$;
- 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' represents geometries matched by S, where S is one of the following relations: S=$\{intersects,\ contains,\ next,\ equals,\ within,\ disjoint,\ outside\}$;
- each atom $Q_{D_k}(\mathbf{x}, \mathbf{y})$ is similar to $Q_{O_i}(\mathbf{x}, \mathbf{y})$ but adds stream operators that relate to Continuous Query Language operators. We have a window $[agr, b, e]$ over a stream D_k, where b and e are the bounds of the window in time units (positive for past, negative for future) and an *aggregate function agr* applied to the data items in the window:
  - $[agr, b]$ represents the aggregate of last or next b time units of stream D_k;
  - $[b]$ represents the single tuple of F_j at index b with $b = 0$ if it is the current tuple;
  - $[agr, b, e]$: represents the aggregate of a window $[b, e]$ in the past/future of D_k.
- each atom $Q_{T_l}(\mathbf{x}, \mathbf{y}) = (T_1(z_1, z_1'), \dots, T_q(z_q, z_q'))$ represents a disjunction of *temporal* relations, where the variables z_i, z_i' represent matches, i.e., individuals annotated with time points/intervals, which are filtered by the temporal relation T_i. For points, $T_i = T_i^P$ is from $\{<, \leq, =, \geq, >\}$; for intervals, we choose the relations of Allen's Time Interval Algebra [1], i.e., $T_i = T_i^I$ is from $\{before,\ equal,\ meets,$

$overlaps, during, starts, finishes\}$ and the set of inverses, e.g., $during^-$, which filter variable matches according to the start/end points of the intervals.

The "historic" window operator $[agr, b, e]$ is derived from Brandt et al. [8] and allows us to query logs represented by data streams. Details on handling the temporal relations and aggregate functions are given below. We also have added a limited form of disjunction in our temporal relations; in general this would move the language beyond CQs.

Query Rewriting with Spatio-Temporal Relations. We consider answering pull-based queries at *a single* time point $\mathbb{T}_i$ with stream atoms that define *aggregate functions* on different window sizes relative to $\mathbb{T}_i$. For this, we consider a semantics based on *epistemic aggregate queries* (EAQ) over ontologies [11] by dropping the order of time for the data and handling 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 *filtered and merged temporary* ABoxes. The filtering and merging, relative to the window size and $\mathbb{T}_i$, creates for each EAQ a temporary ABox $A_{\boxplus_\phi}$, which is the union of the static ABox A and the filtered streaming data items from the stream DB. The EAQs 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$_A$. For details see [14].

At first sight, spatial and temporal relations could be treated similarly. As shown in [14], we evaluate spatial relations regarding their *Point-Set Topological Relations*. It amounts to pure set theoretic operations on point sets using the function $points(p)$, which defines the (infinite) set of points of a geometry p that is a sequence $p = (p_1, \dots, p_n)$ of (defined) points. For instance, the relation $inside(x, y)$ between geometries is defined as $\{(x, y) : points(y) \subseteq points(x)\}$. However, for temporal relations, we distinguished point-based relations that can be encoded as simple arithmetic filters, from interval-based relations, where in Allen's Time Interval Algebra (IA) [1] 13 relations can hold between two intervals. The domain of IA relations is the set of intervals $\{[p_1], \dots, [p_k]\}$ over the linear order of $\mathbb{T}$ defined as $[p_i] = [\underline{p_i}, \overline{p_i}]$ with $\underline{p_i} < \overline{p_i}$. The binary *basic IA* relations are defined according to their start/end points as follows [1]:

$before(x,y) = \{(x,y) : \underline{x} < \overline{x} < \underline{y} < \overline{y}\}$; $meets(x,y) = \{(x,y) : \underline{x} < \overline{x} = \underline{y} < \overline{y}\}$;
$overlaps(x,y) = \{(x,y) : \underline{x} < \underline{y} < \overline{x} < \overline{y}\}$; $starts(x,y) = \{(x,y) : \underline{x} = \underline{y} < \overline{x} < \overline{y}\}$;
$finishes(x,y) = \{(x,y) : \underline{y} < \underline{x} < \overline{x} = \overline{y}\}$; $during(x,y) = \{(x,y) : \underline{y} < \underline{x} < \overline{x} < \overline{y}\}$;
$equal(x,y) = \{(x,y) : \underline{y} = \underline{x} < \overline{x} = \overline{y}\}$.

IA relations can be interpreted over the sets of intervals $\mathbb{I}_A$ and $\mathbb{I}_B$ in two ways: (a) *IA filtering*, where each relation is treated as a single binary constraint. In that sense, the temporal relation acts as a filter on all intervals in $\mathbb{I}_A \times \mathbb{I}_B$ that matching the relations regarding their start/end points; (b) *IA reasoning*, which requires the computation of the path consistency of all temporal relations over the intervals in $\mathbb{I}_A \cup \mathbb{I}_B$ using the predefined composition table of [1]. The composition table is defined as a set of transitive rules on basic relations, which are applied until no new *general* relations can be inferred. For instance, if we have the edges $during(I_1, I_2)$ and $during(I_2, I_3)$, we can infer a new relation $during(I_1, I_3)$. Note that only with approach (b) all possible (chained) relations between intervals are derivable. A well-known representation for IA relations are IA graphs (also called IA networks), which are directed graphs, where the vertices are the intervals of $\mathbb{I}_A$ and $\mathbb{I}_B$ and the edges represent the IA relations that hold between two intervals. Hence, an IA graph (closed by transitive rules) is a materialization of all relations that can hold between intervals, and can be used to check the relations if a directed edge exists.

Our intervals are an intermediate product of the EAQ evaluation and annotate the resulting objects. As mentioned, for each stream atom we have a temporary ABox derived from $\mathbb{T}_i$ and the window $[agr, b, e]$. In a first approach, we directly use $\mathbb{T}_i$ and the window size for the interval generation. For instance, having $\mathbb{T}_5$ and $speed[avg, 3, -1]$, we would annotate each grouped/aggregated match with the interval $[2, 6]$. In a second, approach, we extract for each grouped/aggregated match of an EAQ the upper and lower bounds of the time points annotated to the data items in that window, where the window size is the outer bound. More sophisticated approaches might include the segmentation of the data items, thus creating different fragmented subintervals.

Query Evaluation. The four types of query atoms need different evaluation techniques over separate DB entities. Ontology atoms are evaluated over the static ABox A using a "standard" DL-Lite$_A$ query rewriting, i.e., PerfectRef [10]. For spatial atoms, we need to dereference the bindings to the spatial ABox S_A and evaluate the spatial relations to filter spatial objects. Stream atoms are computed as EAQ to group and summarize over the temporary ABoxes of the different streams. For temporal atoms, we consider three techniques. For time points, we simply add the filter conditions to the rewritten query. For intervals, two techniques are suitable: (a) IA filtering, hence we can rewrite each IA relation of Q_{T_l} into a filter that encodes the equation with the start/end points (as defined before); (b) IA reasoning, where the closed IA graph is constructed applying the transitive rules on all intervals derived from an EAQ. We then extract all derived intervals with the annotated objects from the IA graph that hold according to the queried relations in Q_{T_l}.

In [14], we introduced two spatial query evaluation strategies assuming that *no bounded variables* occur in spatial atoms and the CQ is *acyclic* (roughly has no proper cycle between join variables). One strategy is based on the query hypergraph and the derived join plan and is well-suited for implementing spatial-stream CQs, as it gives us fine-grained caching, full control over the evaluation, and possibly handling different DB entities. Details are given in standard DB literature such as [17].

Example 4. The following example of a simplified q_1, where the layers distinguish between ontology (first), stream/temporal (second), and spatial (third line) atoms:

$$\begin{aligned} q_2(x,y) :\ & LaneIn(x) \wedge isManaged(x,z) \wedge SignalGroup(z) \wedge Vehicle(y) \\ & \wedge pos(y,v)[line,\ 10s] \wedge during(v,s) \wedge hasState(z,s)[last,\ 5s, -5s] \\ & \wedge hasLoc(x,u) \wedge intersects(u,v) \wedge (s = {'Stop'}) \end{aligned}$$

Based on the hypergraph decomposition, we have the following evaluation order:

(1) $q_{2_{F1}}(y, v@i_v) : Vehicle(y) \wedge pos(y, v@i_v)[line, 10s]$; (2) $q_{2_{N1}}(x,u) : LaneIn(x) \wedge hasLoc(x,u)$;
(3) $q_{2_{F2}}(z, s@i_s) : SignalGroup(z) \wedge\ hasState(z, s@i_s)[last, 5s, -5s] \wedge (s = {'Stop'})$;
(4) $q_{2_{T1}}(y,v) : q_{2_{F1}}(y, v@i_v) \wedge during(v@i_v, s@i_s) \wedge q_{2_{F2}}(z, s@i_s)$;
(5) $q_2(x,y) : q_{2_{T1}}(y,v) \wedge intersects(u,v) \wedge q_{2_{N1}}(x,u)$.

Stream Aggregation and Predictions. For *normal objects* and *constant values*, we allow the aggregate functions $count$, $first$, and $last$ on the stream data items. For $last$ and $first$, we need to search the bag of data items, as the sequence of time is lost. This is achieved by iteratively checking if we have a match at one of the points in time. In the implementation, the first and last match can be simply cached while processing the stream. For *individuals* and *constant (numerical) values*, we allow a range of aggregation and prediction functions on the streamed data items:

- *order*: $first,\ last$, where they give the first or last element in the stream, respectively;
- *simple*: $count,\ min,\ max,\ sum$, and avg;
- *descriptive statistics* (DS): $mean,\ sd,\ var,\ median$, where each function calculates the mean, standard deviation, variance, and median as expected;
- *predictions*: We apply predefined regression methods to predict values from existing (time-series) data items inside a window. Model building (i.e., the training) and prediction should be fast, hence we support the following lightweight methods: (a) mov_avg

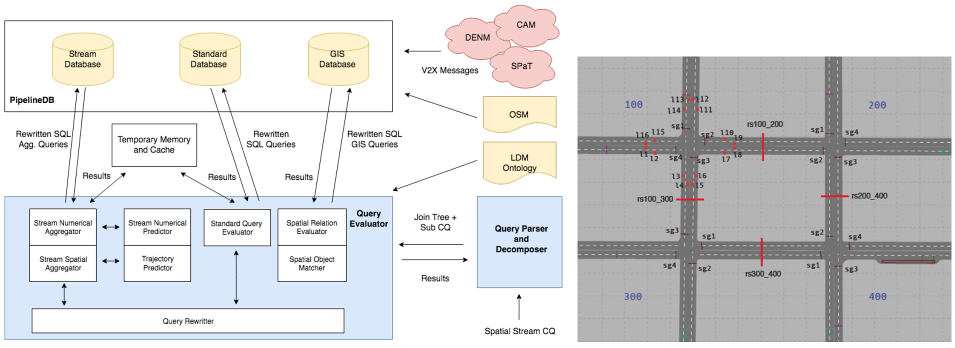

Fig. 2: (a) System Architecture and (b) Four Intersection Scenario

calculates the moving average of the past values; (b) exp_smooth applies simple exponential smoothing; and (c) $grad\_boost$ uses gradient boosting with regression trees. Note, since the order of items is lost due to the bag semantics, the temporal annotation (e.g., $speed(c_1, 50)@10$) are needed in the prediction functions as the second dimension. We allow different regression methods with increasing complex models. On small windows with a required fast response time, mov_avg and exp_smooth is preferable, while on larger windows, e.g., for traffic predictions, $grad\_boost$ could be applied.

For *spatial objects*, geometric aggregate functions are applied to the bag of data items that represent geometries, i.e., the sequence of points $p = (p_1, \ldots, p_n)$. We allow these functions to derive new geometries (among others):

- $point$: we evaluate the function $last$ to get the last data item p_n on the stream;
- $line$: we create a sequence of points representing a path by calculating 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*;
- $line\_angle$: the angle (in degrees) of $line$ regarding a reference system is calculated;
- $traject\_line$ and $traject\_heading$ are simple techniques to project possible trajectories from past points. The former is linearly projecting the trajectory based on the previous points and the current speed. The later calculates the trajectory based on the last point and the last heading of the vehicle.

For the trajectory computation, besides a simple linear also a curvature-based models could be applied. To improve the accuracy of the model, we could use the speed of the last data points, so a speed-up or slow down would be taken into account.

5 Implementation

We have implemented a prototype of our spatial-stream OQA approach in JAVA 1.8 using the stream RDBMS PIPELINEDB 9.8.1 (https://www.pipelinedb.com/). The system architecture is shown in Figure 2a. We chose PIPELINEDB, as it is built on top of PostgreSQL (https://www.postgresql.org/) and PostGIS (http://postgis.net/) and thus supporting stream and spatial data. It distinguishes between *streams* and *continuous views*, where streams are write-only, so the query evaluator has to access the read-only continuous views. We created an 1-to-1 mappings from streams to continuous views, and further to the TBox concepts and roles; e.g., vehicle positions are fed into the stream *stream_pos*(id, pos, tp), where id is the vehicle id, pos its position, and tp the time point of adding; *stream_pos* is accessed via the continuous view *view_pos*, which is mapped to the property pos. We also provide an integration framework that constantly receives V2X messages and adds the raw message data either to normal tables of the static DB, spatial tables of the GIS DB, or the *streams* of the stream DB.

Implementation Details. The parser/decomposer component is used for parsing the input spatial-stream CQ, and then decomposing the query hypertree using Gottlob et al.'s (https://www.dbai.tuwien.ac.at/proj/hypertree/) implementation. 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 in-memory. The decomposer gives us the join 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. The query evaluator traverses J_q bottom up, left-to-right, and (1) checks if the result of a sub CQ are cached; (2) if not, instantiates one of the evaluators according to the sub CQ type.
Ontology evaluator: this evaluator uses the DL-Lite$_A$ query rewriter OWLGRES 0.1 [22], but a more efficient implementation as in ONTOP [20] is planned.
Stream evaluator: for each stream sub CQ q_i, it detemporalizes the streams by grouping/aggregating the data items by performing the following steps:
(1) extract the data items according to the defined window size;
(2) evaluate q_i (no rewriting) and store the "known solutions" in memory as $R_{i,1}$;
(3) evaluate q'_i (with rewriting) over $R_{i,1}$ and store it in memory as $R_{i,2}$;
(4) apply the prediction function on $R_{i,2}$ and add the predicted data items;
(5) apply the grouping/aggregation function on $R_{i,2}$, and produce the outcome $R_{i,3}$.
Predictors: the prediction function is an integrated part of the stream evaluator, where we apply the predictions on the aggregated data items. We provide a standard implementation for the functions mov_avg and exp_smooth. For $grad\_boost$, we use the state-of-art library XGBOOST (https://xgboost.readthedocs.io/en/latest/).
Spatial evaluator: it handles the different spatial relations. For performance reasons, we do not compile them to SQL, but evaluate the spatial relations in-memory using the functions of the JTS TOPOLOGY SUITE (https://github.com/locationtech/jts).
Temporal evaluator: this evaluator supports the mentioned *IA filtering* technique, since temporal relations can be directly rewritten into SQL by encoding the relations as joins, where each relations is encoded as a filter on the start/end points of the aggregated data items. The second technique *IA reasoning* is planned for future work.

6 Experiments and Evaluation

We evaluated our platform regarding the requirements/features (cf. Table 1) derived from the use cases. The requirements are encoded into a set of queries that include the desired features. The ontology, queries, experimental setup, logs, results, and the implementation are available on http://www.kr.tuwien.ac.at/research/projects/loctrafflog/ekaw2018.

Scenario Data. For having realistic traffic data, we generated our streaming data with the microscopic traffic simulation tool PTV VISSIM (http://vision-traffic.ptvgroup.com/en-us/products/ptv-vissim/), which allows us to simulate realistic driving and traffic light behavior, as well as the possibility to create unexpected events like accidents. We extract the actual state of each Vissim simulation step, and store the result as JSON in a log, and provide a log player that replays the simulation by feeding the data to PIPELINEDB. For varying the data throughput, we adjust the following parameters: (a) replay with 5ms, 10ms, 50ms, 100ms delay, where 5ms are the fastest updates (i.e., simulating sensors) and 100ms is the real-time speed of the Vissim simulation; (b) we simulate light, medium, and heavy traffic in our scenario, where we have approx. 20, 50, and 150 vehicles respectively. We modeled a real-world scenario shown in Fig. 2b, which is based on a grid layout with four intersections of four roads crossing, and two incoming and outgoing lanes per street. The two incoming lanes of each side have traffic light controllers assigned; all maneuvers (turn left/right, straight on) to outgoing lanes are allowed. The main traffic flow is from north to south and west to east. We encode the structure of the full intersections into static

ABox instances as follows: (a) intersections, roads, lanes, signal groups, and vehicles as concept assertions; (b) geometries for each lane, road, etc. as attribute assertions; and (c) lane connectivity, signal group assignments, etc. as role assertions.

Queries for Experiments. Based on the requirements, we derived a set of queries to assess each scenario. Each query aims at answering a specific problem of the use case taking the set of features into account. Note that commas between atoms are conjunctions, disjunctions are explicitly stated using or. For the use case *S1.1* (object statistics), query $q_{1.1}$ determines the average and max speed of BMWs and VWs in the last 10 secs.

$$\begin{aligned} q_{1.1}(x,u,v) :\ & \mathit{Vehicle}(x), \mathit{vehicleMaker}(x,z), (z{=}'\mathit{BMW}'\ \mathit{or}\ z{=}'\mathit{VW}'),\\ & \mathit{speed}(x,u)[\mathit{avg}, \mathit{10s}], \mathit{speed}(x,v)[\mathit{max}, \mathit{10s}] \end{aligned}$$

For the use case *S1.2* (intersection statistics), we count vehicles according to their engine type. Sub-queries $q_{1.2a}$ and $q_{1.2b}$ select cars with either diesel or petrol engine that pass intersection $i100$. Query $q_{1.2}$ aggregates the sub-queries and returns the count of diesel in y and petrol vehicles in z, respectively:

$$\begin{aligned} q_{1.2a}(x,y) :\ & \mathit{Vehicle}(y), \mathit{pos}(y,z)[\mathit{line}, \mathit{10s}], \mathit{vehicleEngine}(y,m), (m{=}'\mathit{Petrol}'),\\ & \mathit{intersects}(z,u), \mathit{hasLoc}(x,u), \mathit{Intersection}(x), x = '\mathit{i100}'\\ q_{1.2b}(x,y) :\ & \mathit{Vehicle}(y), \mathit{pos}(y,z)[\mathit{line}, \mathit{10s}], \mathit{vehicleEngine}(y,m), (m{=}'\mathit{Diesel}'),\\ & \mathit{intersects}(z,u), \mathit{hasLoc}(x,u), \mathit{Intersection}(x), x = '\mathit{i100}'\\ q_{1.2}(x,y,z) :\ & q_{1.2a}(x,y)[\mathit{count}, \mathit{10s}],\ q_{1.2b}(x,z)[\mathit{count}, \mathit{10s}] \end{aligned}$$

For the use case *S1.3* (network statistics), we have two linked intersections $i100$ and $i200$. Query $q_{1.3}$ traces the vehicles that start at $i100$ and counts those passing through $i200$. A delay of 7s allows to check the vehicle's position 7s later, and the temporal relation $before$ ensures that a vehicle first passes $i100$ and then $i200$.

$$\begin{aligned} q_{1.3a}(x,v) :\ & \mathit{Vehicle}(x), \mathit{pos}(x,v)[\mathit{line}, \mathit{6s}], \mathit{intersects}(v,u), \mathit{Intersection}(r),\\ & \mathit{hasLoc}(r,u), (r = '\mathit{i100}')\\ & \mathit{delay}(\mathit{7s})\\ q_{1.3b}(x,z) :\ & \mathit{Vehicle}(x), \mathit{pos}(x,z)[\mathit{line}, \mathit{6s}], \mathit{intersects}(z,w), \mathit{Intersection}(r),\\ & \mathit{hasLoc}(r,w), (r = '\mathit{i200}')\\ q_{1.3c}(x) :\ & q_{1.3a}(x,v), before(v,z), q_{1.3b}(x,z) \end{aligned}$$

For the use case *S2.1* (simple maneuvers), query $q_{2.1}$ returns all vehicles x and y that turned left or right in the last 6s. Then both results are combined by unions of CQs resulting in all vehicles performing the two maneuvers.

$$\begin{aligned} q_{2.1l}(x,z) :\ & \mathit{Vehicle}(x), \mathit{pos}(x,y)[\mathit{line}, \mathit{6s}], \mathit{match}(y,z)[\mathit{angle}, -\mathit{175}, -\mathit{15}],\\ & \mathit{intersects}(y,u), \mathit{hasLoc}(r,u), \mathit{Intersection}(r), (r = '\mathit{i100}')\\ q_{2.1r}(x,z) :\ & \mathit{Vehicle}(x), \mathit{pos}(x,y)[\mathit{line}, \mathit{6s}], \mathit{match}(y,z)[\mathit{angle}, \mathit{15}, \mathit{175}],\\ & \mathit{intersects}(y,u), \mathit{hasLoc}(r,u), \mathit{Intersection}(r), (r = '\mathit{i100}')\\ q_{2.1}(x,z) :\ & q_{2.1l}(x,z)\ \mathit{or}\ q_{2.1r}(x,z) \end{aligned}$$

In use case *S2.2* (complex maneuvers), query $q_{2.2}$ detects illicit lane changes in terms of crossing the middle marker (i.e., a white line). This is detected by evaluating whether a vehicle has moved from in-lane, temporally to an out-lane or vice versa.

$$\begin{aligned} q_{2.2}(x,y) :\ & \mathit{LaneIn}(z), \mathit{hasLoc}(z,u), \mathit{intersects}(u,v), \mathit{pos}(x,v)[\mathit{line},\ \mathit{6s}, \mathit{3s}],\\ & \mathit{Vehicle}(x), \mathit{pos}(x,w)[\mathit{line}, \mathit{3s}, \mathit{0s}],\ \mathit{intersects}(t,w), \mathit{hasLoc}(y,t), \mathit{LaneOut}(y) \end{aligned}$$

For the use case *S2.3* (red-light violation), we modified Ex. 1 by taking trajectory and speed prediction into account, which allows us a more precise detection of violations, since we can rule out vehicles that are slowing down or are about to change lanes.

$$\begin{aligned} q_{2.3}(x,y) :\ & \mathit{LaneIn}(x), \mathit{hasLoc}(x,u), \mathit{intersects}(u,v), \mathit{pos}(y,v)[\mathit{traject\_line}, \mathit{5s}, -\mathit{3s}],\\ & \mathit{Vehicle}(y), \mathit{speed}(y,r)[\mathit{mov\_avg}, \mathit{5s}, -\mathit{3s}],\ (r > \mathit{10}),\ \mathit{hasSignalGroup}(x,z),\\ & \mathit{SignalGroup}(z), \mathit{hasState}(z, \mathit{Stop})[\mathit{last}, 5, -5] \end{aligned}$$

For the use case *S2.4* (vehicle breakdown), we check with $q_{2.4}$, if a car has stopped for longer than 30s, while (using the $during$ relation) it is located inside our intersections, but not on one of the park lanes (using the $disjoint$ relation).

$$\begin{aligned} q_{2.4}(x,y):\ & \mathit{Vehicle}(x), \mathit{speed}(x,r)[\mathit{avg}, \mathit{30s}], (r < \mathit{1}), \mathit{pos}(x,v)[\mathit{line}, \mathit{15s}], \mathit{inside}(v,u), \\ & \mathit{hasLoc}(y,u), \mathit{Intersection}(y), \mathit{during}(v,r), \mathit{disjoint}(v,z), \mathit{hasLoc}(p,z), \mathit{ParkLane}(p) \end{aligned}$$

The use case *S2.5* (traffic congestion) can be evaluated by a query similar to *S2.4*, but with the extension that stop-and-go traffic can be excluded by checking if there is no movement while the traffic light phases are on “Go”.

$$\begin{aligned} q_{2.5}(x,y):\ & \mathit{Vehicle}(x), \mathit{speed}(x,r)[\mathit{avg}, \mathit{30s}], (r < \mathit{1}), \mathit{pos}(x,v)[\mathit{line}, \mathit{30s}], \mathit{intersects}(v,u), \\ & \mathit{hasLoc}(y,u), \mathit{LaneIn}(y), \mathit{hasSignalGroup}(y,z), \mathit{SignalGroup}(z), \\ & \mathit{hasState}(z,s)[\mathit{last}, \mathit{30s}), (s = '\mathit{Go}'), \mathit{during}(s,r) \end{aligned}$$

For the use case *S3.1* (self monitoring), we aim to detect with $q_{3.1}$, if our ego vehicle is exceeding the speed limit that is assigned to the lane our car is driving on.

$$\begin{aligned} q_{3.1}(x,y):\ & \mathit{LaneIn}(y), \mathit{hasLocation}(y,u), \mathit{intersects}(u,v), \mathit{pos}(x,v)[\mathit{line}, 5s], \\ & Vehicle(x), \mathit{isEgo}(x), \mathit{speed}(x,r)[\mathit{max}, 5s], \mathit{speedLimit}(y,s), (r > s) \end{aligned}$$

In use case *S3.2* (obstructed view) we compute query $q_{3.2}$, where our system (as part of the ego vehicle) aims to detect cars that very likely will collide in 2s on a busy intersection by checking if our predicted trajectories will cross another car.

$$\begin{aligned} q_{3.2}(x,y):\ & \mathit{Vehicle}(y), \mathit{isEgo}(y), \mathit{pos}(y,v)[\mathit{traject\_line}, \mathit{2s}, -\mathit{1s}], \mathit{intersects}(v,w), (r > \mathit{10}), \\ & \mathit{Vehicle}(x), \mathit{speed}(x,r)[\mathit{mov\_avg}, \mathit{5s}, -\mathit{2s}],\ \mathit{pos}(x,w)[\mathit{traject\_line}, \mathit{2s}, -\mathit{1s}] \end{aligned}$$

In *S3.3* (traffic rules), our ego vehicle approaches an uncontrolled intersection at the same time with other vehicles. According to traffic rules in Austria, preference is given to the vehicles on the main road. We can express these traffic rules in positive Datalog rules as:

$$\begin{aligned} & \mathit{willCross}(x,y) \wedge \mathit{straightOn}(x) \wedge \mathit{turnLeft}(y) \rightarrow \mathit{giveWay}(y,x) \\ & \mathit{willCross}(x,y) \wedge \mathit{turnRight}(x) \wedge \mathit{turnLeft}(y) \wedge \mathit{crossOpposLane}(y) \rightarrow \mathit{giveWay}(y,x) \\ & \mathit{vehicle}(x) \wedge \mathit{vehicle}(y) \wedge \mathit{giveWay}(y,x) \rightarrow \mathit{stop}(y) \end{aligned}$$

The atom $\mathit{willCross}(x,y)$ matches all vehicles that might collide and can be evaluated by $q_{3.2}(x,y)$ (modified without $\mathit{isEgo}(x)$. The atoms $\mathit{turnLeft}(x)$, $\mathit{turnRight}(x)$, and $\mathit{straightOn}(x)$ can be evaluated by the queries $q_{2.1l}(x)$, $q_{2.1r}(x)$, assuming the queries are atomic rules with $q(x)$ as the head. Then, the rules of *S3.3* can be expressed as unions of CQs, but this approach is not feasible if we need rule chaining and transitive rules.

Results. We conducted our experiments on a Mac OS X 10.13.3 system with an Intel Core i7 2.9GHz, 8GB of RAM, and a 250GB SSD. The average of 21 runs for query rewriting time and evaluation time was calculated. The results are in Table 2 presenting the number of subqueries $\#Q$ with stream queries in brackets, the size of rewritten atoms $\#A$, and t as the average evaluation time (AET) in seconds for different traffic densities and update delay in ms. The new experiments confirm results of [14] with closer to “real-world” queries and simulation data. The AET ranges between 0.86s and 2.06s with the exception of use case *S3.3*, which emulates rules using unions of CQs. Query $q_{3.1}$ shows the highest delay of 2.06s, since the join condition of $(r > s)$ is evaluated inline and not on the DB, which adds a delay of $0.4s$ with larger windows. Our baseline query is $q_{1.1}$ tested with 100ms delay and low traffic. It has an AET of 0.86s, where 0.23s is the time-to-load (TOL), 0.63s is needed for query evaluation of two stream atoms, where we added an artificial delay of 0.18s before each stream atom evaluation. The artificial delay is empirically determined and needed for PipelineDB to set up the *continuous views* (CVs); ignoring this would lead to missing results. Note that the baseline time still could be reduced by (a) pre-compiling the program, which shortens evaluation by 0.2s, and (b) parallelization of stream atom evaluation that improves performance by approx. 20%; details are available on the results website. The added functions for statistics, matching, and predictions, i.e., $\mathit{mov\_avg}$ and $\mathit{traject\_line}$ do not affect performance, since they are applied on small windows with few data items. However, if we would apply $\mathit{gradboost}$ for predictions on larger windows, our query time could rise considerably, since prediction time (without a preprocessed training step) can be above 20s.

	$\#Q$	$\#A$	(l) with ms delay				(m) with ms delay				(h) with ms delay			
			5	10	50	100	5	10	50	100	5	10	50	100
$q_{1.1}$	3(2)	42	1.35	1.18	0.95	0.86	1.45	1.30	0.99	0.88	1.46	1.35	1.14	0.99
$q_{1.2}$	6(2)	43	1.30	1.20	1.01	0.96	1.33	1.24	1.04	1.00	1.41	1.38	1.07	1.01
$q_{1.3}$	8(2)	44	1.44	1.35	1.15	1.08	1.47	1.37	1.23	1.09	1.45	1.44	1.30	1.20
$q_{2.1}$	6(2)	43	1.31	1.20	1.01	0.98	1.43	1.29	1.09	0.99	1.48	1.40	1.13	1.02
$q_{2.2}$	7(2)	45	1.36	1.26	1.05	1.00	1.47	1.29	1.08	1.03	1.51	1.43	1.13	1.06
$q_{2.3}$	7(3)	50	1.57	1.50	1.27	1.21	1.63	1.53	1.30	1.22	1.72	1.65	1.37	1.27
$q_{2.4}$	5(2)	46	1.24	1.21	0.98	0.92	1.28	1.24	1.06	0.97	1.28	1.29	1.13	0.99
$q_{2.5}$	7(3)	43	1.44	1.38	1.16	1.08	1.50	1.41	1.20	1.11	1.55	1.47	1.26	1.17
$q_{3.1}$	5(2)	43	1.85	1.72	1.40	1.32	1.89	1.79	1.48	1.35	2.06	2.04	1.57	1.38
$q_{3.2}$	5(3)	63	1.41	1.34	1.23	1.17	1.48	1.43	1.27	1.20	1.56	1.51	1.31	1.21
$q_{3.3}$	12(5)	43	3.02	2.80	2.42	2.39	3.26	2.98	2.58	2.38	3.36	3.20	2.66	2.44

Table 2: Results (t in secs) for scenario with (l)ow, (m)edium, and (h)eavy traffic

Feature Coverage. As shown with the queries, we covered in the implementation all initial levels (L1) of features that are defined in the scenarios/uses cases. We support temporal relations and a (partial) interval-based data model (*F1*) evaluated by pull-based queries (*F2*). Then, we allow temporal relations and nested queries that include unions of CQs (*F3*). But, we have not yet implemented the *IA reasoning* for temporal relations, since an in-memory evaluation of the transitive rules completing the IA graph needs further investigation. Regarding *F4* and *F5*, we have implemented the initial set of numerical, descriptive statistical, and spatial aggregation functions. For *F6*, we covered mov_avg and exp_smooth for fast, simple predictions, and support $grad\_boost$ for long-term traffic forecasting. For trajectory prediction (*F7*), we have implemented a method based on a simple linear path calculation. But, more accurate trajectory predictions would be desired. Feature *F8* is covered by the atom $match(y, z)[angle, 0, 15]$, and *F9* is partially covered by unions of CQs, but transitive rules are out of scope for this work.

7 Related Work and Conclusions

RDF stream processing engines, such as C-SPARQL [5], SPARQLstream [9], and CQELS [15], were proposed for processing RDF streams integrated with linked data sources. EP-SPARQL [3] and LARS [6] introduce languages that extend SPARQL respectively CQs with stream reasoning, but translate KBs into more expressive, less efficient logic programs. Closest to our spatial-stream QA approach is the work of (i) [19] that supports spatial operators as well as aggregate functions over temporal features, (ii) [9] that allows evaluating OQA queries over stream RDBMS, and (iii) [8] that extends SPARQL with aggregate functions and statistic methods and is evaluated over streamed and ordered ABoxes. Temporal QA is also investigated in [4] and [7], both are on the theoretical side and provide no implementation yet. The work of Netten et al. [18] and Lécué et al. [16] focus on longer-term diagnosis, but neglect the streaming nature of C-ITS data.

This work is sparked by applying spatial-stream QA as an integration and QA effort for streamed mobility data, e.g., vehicle movements, in a spatial context over the complex mobility domain. In [14], we have introduced simple aggregate queries over streams, which often do not suffice to capture more complex use cases. We present an extension with temporal relations, and numerical/trajectory predictions, which allows us to query complex mobility patterns such as traffic statistics or complex events such as (potential) accidents. Based on the newly developed scenarios of *traffic statistics*, *event detection*, and *ADAS*, we have defined a set of domain-specific features such as trajectory computation, which are matched with the scenarios/use cases to define the requirements. Given the new features, we adjusted our LDM ontology, our spatial-stream query language, and extended our methods accordingly. We also redesigned our system architecture and give insights on the new components for temporal relations, prediction and trajectory calculation.

The experimental evaluation provides evidence for the feasibility and efficiency of our approach in the mentioned scenarios.

As discussed in feature coverage, ongoing and future research should be directed to extend the languages, methods, and the platform to fulfill the defined requirements, which will allow us to apply them to more scenarios such as logistics.

Acknowledgements. This work has been supported by the Austrian Research Promotion Agency project LocTraffLog (FFG 5886550) and DynaCon (FFG 861263).

References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. Com. 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. 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)
5. 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)
6. 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)
7. Borgwardt, S., Lippmann, M., Thost, V.: Temporalizing rewritable query languages over knowledge bases. J. Web Sem. 33, 50–70 (2015)
8. 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)
9. Calbimonte, J., Mora, J., Corcho, Ó.: Query rewriting in RDF stream processing. In: Proc. of ESWC 2016. pp. 486–502 (2016)
10. 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)
11. Calvanese, D., Kharlamov, E., Nutt, W., Thorne, C.: Aggregate queries over ontologies. In: Proc. of ONISW 2008. pp. 97–104 (2008)
12. Dell'Aglio, D., Valle, E.D., van Harmelen, F., Bernstein, A.: Stream reasoning: A survey and outlook. Data Science, IOS Press 1(1-2), 59–83 (2017)
13. Eiter, T., Parreira, J.X., Schneider, P.: Detecting mobility patterns using spatial query answering over streams. In: Proc. of Stream Reasoning Workshop 2017 (2017)
14. Eiter, T., Parreira, J.X., Schneider, P.: Spatial ontology-mediated query answering over mobility streams. In: Proc. of ESWC 2017. pp. 219–237 (2017)
15. 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)
16. Lécué, F., Tallevi-Diotallevi, S., Hayes, J., Tucker, R., Bicer, V., Sbodio, M.L., Tommasi, P.: Smart traffic analytics in the semantic web with STAR-CITY: scenarios, system and lessons learned in dublin city. J. Web Sem. 27, 26–33 (2014)
17. Maier, D.: The Theory of Relational Databases. Computer Science Press (1983)
18. 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)
19. 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)
20. Rodriguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Ontology-based data access: Ontop of databases. In: Proc. of ISWC 2013. pp. 558–573 (2013)
21. 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)
22. Stocker, M., Smith, M.: Owlgres: A scalable owl reasoner. In: Proc. of OWLED 2008 (2008)
23. Zhao, L., Ichise, R., Liu, Z., Mita, S., Sasaki, Y.: Ontology-based driving decision making: A feasibility study at uncontrolled intersections. IEICE Trans. 100-D(7), 1425–1439 (2017)