Deploying Spatial-Stream Query Answering in C-ITS Scenarios[1]

Thomas Eiter [a], Ryutaro Ichise [b,d], Josiane Xavier Parreira [c], Patrik Schneider [a,c,*] and Lihua Zhao [b,d]
[a] *Vienna University of Technology, Vienna, Austria*
E-mails: eiter@kr.tuwien.ac.at, patrik@kr.tuwien.ac.at
[b] *National Institute of Informatics, Tokyo, Japan*
E-mail: ichise@nii.ac.jp
[c] *Siemens AG Österreich, Vienna, Austria*
E-mail: josiane.parreira@siemens.com
[d] *National Institute of Advanced Industrial Science and Technology, Tokyo, Japan*
E-mail: lihua.zhao@aist.go.jp

Keywords: Mobility, C-ITS, Query Answering, Ontology-based Data Access, Stream Reasoning, Temporal Relations

1. Introduction

The development in (semi)-autonomous vehicles leads to an extensive communication between vehicles and the infrastructure, which is covered by Cooperative Intelligent Transport Systems (C-ITS). These systems produce 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 to improve road safety by analyzing traffic scenes that could lead to accidents (e.g., red light violations), and to reduce emissions by optimizing traffic flow (e.g., dissolve traffic jams). A key technology for this is the Local Dynamic Map (LDM) [1] 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 [2]. Our approach is based on ontology-mediated query answering (OQA) and features conjunctive queries (CQs) over DL-Lite$_A$ [3] ontologies that support window operators over streams and spatial relations between objects. We believe that OQA and the related ontology-based data access (OBDA) [4] are 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 [1]. Therefore, an expressive query language is crucial to cover C-ITS specific requirements needed for retrieving dynamic data and expressing complex patterns regarding, event detection for example. Furthermore, scalability and swift response time are crucial since fast changing traffic demands a quick response time of ranging below 1s to avoid accidents [1].

In this paper, we continue the work in [2, 5] 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 [2] is extended with new domain-specific features beyond "generic" spatial-stream OQA. In cooperation with ITS domain experts from Siemens 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*

[1]This article is a revised and extended version of a paper presented at EKAW 2018.
[*]Corresponding author. E-mail: patrik@kr.tuwien.ac.at.

on a query window. For the qualitative assessment of the use cases, requirements, and domain-specific features, we conducted several interviews with ITS experts, which supported our assumptions that temporal relations and predictions are important extensions, but also gave raise to important extensions for future work such as capturing uncertainty in the ontology and the query language. For the quantitative assessment, we provide a detailed report on the improved implementation, which we extended with the features temporal relations and predictions, but we also improved the performance based on pre-compiling static elements of the query, and the parallel execution of stream atoms. The implementation is evaluated in an experimental setting using queries that match with the features, where a real-world traffic simulation is used to generate the data. The results provide evidence for the potential feasibility and efficiency of our approach in these scenarios. Our contributions are briefly summarized as follows:

- 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 conducted expert interviews to obtain feedback on the scenarios, use cases, and query features (Section 4);
- we present our current approach including data model, query language, and evaluation strategy (Section 5).
- we report on the implementation of our approach in a prototype and comment on the implementation details (Section 6);
- we evaluate our work regarding the set of features and requirements based on a traffic simulation and assess the results (Section 7).
- we list related work, and evaluate existing stream reasoning systems, wherein we compare the performance of our prototype to the systems C-SPARQL [6] and CQELS [7] (Section 8).

In Section 9, we discuss lessons learned 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 [1]. The communication technology is based on the IEEE 802.11p standard, and the data integration effort is the *Local Dynamic Map* (LDM); there are starting points for this 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 [1]. The main standardized message types are [8–10]:

- *CAMs* (Cooperative Awareness Messages) provide high frequency status updates of a vehicle's position, speed, and might include vehicle type, model, and turn signals;
- *MAPs* (Map Data Messages) describe the detailed topology of an intersection, including its lanes, their connections, and assigned traffic light (TL) signal groups;
- *SPATs* (Signal Phase and Timing Messages) contain the projected TL signal phases (e.g., green, yellow, and red) for each lane;
- *DENMs* (Decentralized Environmental Notification Messages) informing whether specific events like road works or a traffic jam occur in a designated area.

2.1. 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 [1] 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) [11, 12]. The integration is motivated by advanced safety applications, which need an "overall" understanding of a traffic environment. The LDM consists of the following four layers (see Figure 1a):

1. *Permanent static*: the first layer contains static information obtained from GIS maps and includes roads, intersections, and points-of-interest (POIs);
2. *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;
3. *Transient dynamic*: the third layer contains temporary regional information like weather, road or traffic conditions (e.g., traffic jams), and traffic light signal phases;

4. *Highly dynamic*: the fourth layer contains dynamic information of road users taken from V2X messages or in-vehicle sensors like the GPS module.

Recent research by Netten et al. [13], and Shimada et al. [14] 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 [13]. However, an elaborate domain model captured by an LDM ontology and extended query processing 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).

2.2. Ontology-based LDM

With the support of Siemens and AIST domain experts, we have worked on our LDM ontology [2] (shown partially in Figure 1b) 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$_A$ [3], 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 as follows:

- *V2XFeature* is the representation of V2X objects, such as the 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;
- *LDMLayer* is the representation of the four layers of an LDM, where each feature can be assigned to one layer by the role *isOnLDMLayer*;
- *Actor* is the concept that includes persons, vehicles, as well as roadside ITS stations, which are autonomous agents and are the main generator of streamed data;
- *Event* captures prototypical events that happen in the ITS domain. An important subclass of *Event* is *Hazard* that captures the different types of dangers, e.g., accidents that might occur;
- *CategoricalValues* specify the different categories such as signal phases, or vehicle roles used in the domain.

Besides the "domain specific" roles and attributes such as *speedLimit*, *hasRole*, *speed*, and *position*, we also introduced generic roles that have an inherent meaning, i.e., *isPartOf*, *connected*, and *intersects*.

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

2.3. Spatial-Stream Query Answering

The OQA component is central part regarding the usage of a semantically enhanced LDM, since it allows us to access the streamed data in the LDM.

Example 2.1. 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 8 secs window, projecting 3 secs into the future (represented as a negative distance), which move on lanes (in *x*) *during* the time intervals, where the signal phases of these lanes will turn to "Stop", i.e., red, in a 10 secs window with a 5 secs look into the future to capture the current/next changes in the signal phases:

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

Query q_1 exhibits the different dimensions that need to be combined:

(a) $Vehicle(y)$, $LaneIn(x)$, $SignalGroup(z)$ and $isManagedBy(x,z)$ (assigning traffic lights z to lanes x) are ontology atoms, which have to be unfolded with 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) $speed(y,v)[traject\_line, 5s, -3s]$ and $pos(y, p@i_p)[mov\_avg, 5s, -3s]$ define window operators that aggregate and *predict* the moving average of speed and positions (represented by a path) of the vehicle y over the streams *speed* and *pos*, respectively, and $hasState(z, s@i_s)[last, 5s, -5s]$ returns the traffic lights that have their last phase on "Stop";
(d) the relation $during(p@i_p, s@i_s)$ checks if "v happens during s", where p is all the occurrences of trajectories on the set of time intervals of i_p, and s are the traffic light phases that are on "Stop" in the set of time intervals i_s, were i_p and i_s are derived from

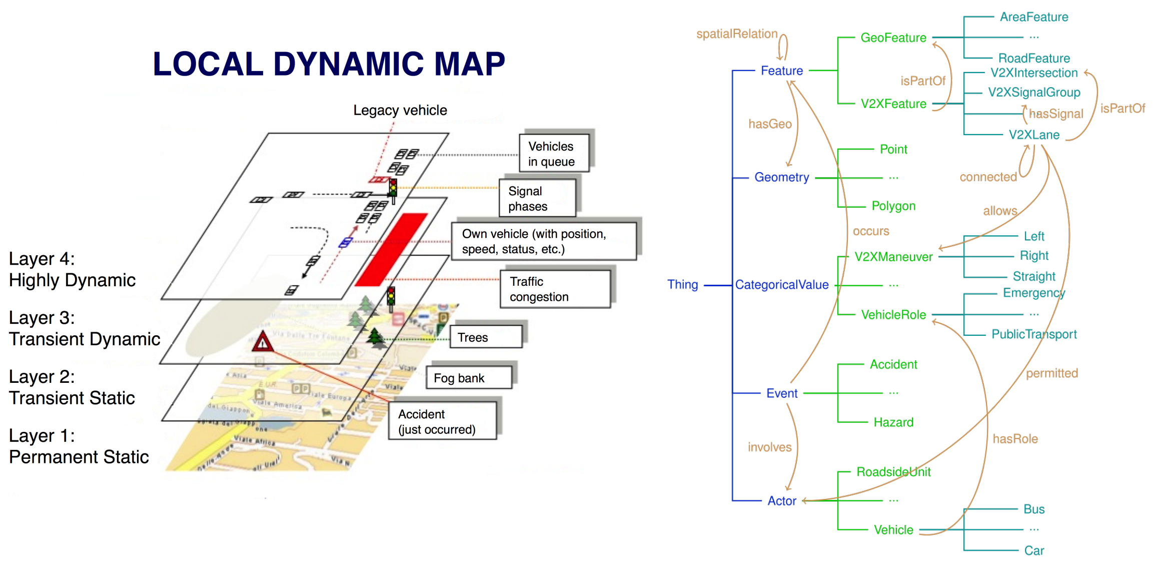

Figure 1. (a) The four Layers of a LDM [1] and (b) LDM Ontology

the trajectory aggregations and the phase duration of the traffic lights, respectively.

Note that we added for readability the implicit definitions of $p@i_p$ and $s@i_s$ to the original query syntax, where we would use solely p and s. The implicit variables i_p and i_s represents the time interval annotated to the aggregated values of variable p and s. For instance, the average speed of a vehicle in the interval $[1, 6]$ as $\langle car, speed, 20\rangle@[1, 6]$. A user formulating a query can ignore this notation and use p and s.

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 [15]. The sensor data can be linked to our ontology-based LDM and enables the system to represent the driving environments.

3.1. Scenario Description

First, we give an overview of the three scenarios.

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* [1], 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, and u-turns, which are complex maneuvers, composed of simpler maneuvers;
3. *Red-light violation*: we give in Example 2.1 a detailed outline on task of detecting red-light violation;
4. *Vehicle breakdown/accident*: this event is based on the stop maneuvers, 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. ADAS features are an important step towards fully automated 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*: this 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 crossing of the predicted trajectories of two vehicles has to be verified to provide a simple collision detection.
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.

3.2. Features for Spatial-Stream QA

The eight "standard" requirements for stream processing identified by [16], namely volume, velocity, variety, incompleteness, noise, timely fashion, fine-grained information access, complex domain models, and user intention, as well as the three entailment levels identified by [17] for stream reasoning systems, namely stream-, window-, and graph-level entailment , are not discussed here; they should hold for C-ITS 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 three 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. Also belonging to L2, 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 [18] with operators like *before* that 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 by push-based, resp. as pull-based queries.

F3 - Query features: we consider "basic" query features that are include in standard query languages as in SPARQL or SQL such as selection, projection, join, filter as part of L1. Allowing time- or triple-based windows or event patterns belong also to L1. Combining the basic features by nesting of queries, unions of CQs, and allowing inter-stream joins belongs to 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). L3 extends the other levels by advanced functions by applying statistical function such as *median*, or qualitative spatial relations such as RCC5 and RCC8 [19]. Note that the aggregation is mainly over multisets, 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. L2 adds the computation of spatial relations using a simple point-set model [20] or the more detailed *9-Intersection model* (L2) based on the aggregated objects. Smoothing and simplification of complex objects could also be included,which leads to L3.

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 "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 - Advanced reasoning: include the different forms of reasoning ranging from rule-based to ontological reasoning that reaches beyond the expressivity of query answering in OWL2 QL [21]. The underlying language can include "simple" implications as $b(x,y) \wedge c(y,z) \rightarrow a(x,z)$, but also "advanced" features such as aggregation, recursion as in OWL2 RL [21], and all combined with negation as failure as present in Answer Set Programming (ASP) [22].

3.3. Requirements

In Table 1, we show the requirements that are derived by analyzing each scenario and use case with respect to the needed 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, an interval-based time model in combination with temporal relations (L2) will be needed. Furthermore, push-based queries are desired for swift reaction on changes. In [2], we already cover level L1 for the features *F1* to *F5*, but aim in this work to introduce new features such as time intervals, temporal relations, and unions of CQs. *F6*, *F7*, and *F8* are entirely new features.

4. Expert Interviews

We conducted four interviews with ITS experts, who play different roles in the field. The first two experts work in industry and the other two experts come from academia. The interviews were conducted as guideline-based interview [23]. Following the guidelines of [23], we left the interviewee the freedom to choose the topic he/she prefers to discuss. However, we had prepared a set of questions, which we asked if the interviewee heads into the direction of a particular topic, e.g., the query language. The goal of the interviews is to answer:

"How suitable is our spatial-stream QA for real-world ITS applications?"

This can be put into more specific questions as:

- *Q1*: "What are important technologies/developments for future ITS systems?"
- *Q2*: "How well is the LDM suited for the integration of vehicle/traffic control sensor data such as V2X messages?"
- *Q3*: "Is the presented ontology and the approach of ontology-based data access suitable to realise an LDM?"
- *Q4*: "We present three scenarios with several use cases, how relevant are they? Are different use cases needed?"
- *Q5*: "We present different query features, such as trajectory predictions, how important are these features in your opinion? Should they be extended?"
- *Q6*: "We present you an example for our queries for detecting red-light violations; how well is this query comprehensible for you? Do you believe another query language is better suited for this?"

Following, we give a summary of the interviews conducted with each expert, where we do not transcribe the full interview, which range from 30 minutes to 1.5 hours, but only present a summarized version of each interview.

Expert 1. The first expert is an initiators of the V2X technology development in Europe. He was responsible, in a large ITS company, for the standardization of V2X messages in different committees of standardization organizations such as ETSI, ISO, and SAE. Additionally, he was participating in several large research projects such as *DRIVE C2X*.[3]

First, the expert pointed out that there are not only CAM, SPAT, MAP, and DENM messages, but also

[3]http://www.drive-c2x.eu/project

Use Case	F1	F2	F3*	F4	F5*	F6	F7	F8*	F9*
S1.1 (Object statistics)	L1	L1	L1	L1	L1				RDFS
S1.2 (Road/Intersection statistics)	L2	L1	L2	L2	L2	L1	L1		OWL2 QL
S1.3 (Network statistics)	L2	L1	L2	L2	L2	L2	L1	P	OWL2 RL / ASP
S2.1 (Simple maneuvers)	L1	L1	L1	L1	L2	P	P		OWL2 QL
S2.2 (Complex maneuvers)	L2	L2	L2	L1	L2	L1	L1	L1	OWL2 QL
S2.3 (Red-light violation)	L2	L2	L2	L1	L2	L1	L1	L1	OWL2 QL
S2.4 (Vehicle breakdown)	L2	L2	L2	L1	L2	L1		P	OWL2 QL
S2.5 (Traffic congestion)	L2	L3	L2	L2	L3	L2		P	OWL2 RL / ASP
S3.1 (Self monitoring)	L1	L2	L 1	L1	L1	P	P		OWL2 QL
S3.2 (Obstructed view)	L1	L2	L2	L1	L2	L1	L1	L1	OWL2 QL
S3.3 (Traffic rules)	L2	L2	L2	L2	L3	L1	L1	L1	OWL2 RL / ASP

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

messages designed for public transport signal requests called SRM/SSN, as well as position correction messages for autonomous vehicles. The latter are important since in inner-cities, the exact postion in combination with high-resolution maps are crucial for safety. An important task arising form this challenge is the continuous matching of the vehicle position to the high-resolution map. He stated that autonomous vehicles, even with advanced sensors, will in the near future *not* be able to drive autonomous and safely in inner-cities with complex intersections, in particular when the weather is unpredictable. He then said that messages like the CAM or DENM need to be send near real-time to the surroundings, hence the existing 4G and upcoming 5G mobile standards are not suitable for this purpose, and the standard IEEE 802.11p (also called ITS-G5) is better suited, since it is based on WLAN technology and already available for low latency (in ms) communication. However, existing WLAN technology can not be used, since these protocols require sessions and authentication with a base station, which would cause a long delay for fast moving vehicles. He gave more details on DENM messages, which inform the surrounding on dangerous events, and noted that DENM messages a fired based on triggering conditions that have specific probabilities assigned. The triggering conditions are defined by the standardization bodies, and vehicle manufacturer have to implement them accordingly. Furthermore, he highlighted that more development is needed to protect vulnerable road users, which requires the integration of bluetooth-based communication such as ZigBee, and also the sharing of data with infrastructure sensors such as mounted Lidar/Radar stations.

Second, we discussed the LDM, and he identified the LDM as an integration platform for the V2X messages, where each vehicle has its own proprietary representation of the LDM, since the ETSI standard defines only the interfaces to access it, but not its internal structure. He doubts though that a common definition of the LDM is needed, since every manufacturer should implement it on their own, where only the dynamic elements could be encoded as a snapshot (in a standardized data model) and exchanged with the surrounding. One discussed use case is the exchange of snapshots, where legacy vehicles and/or vulnerable road users are on the road, hence only fixed Lidar/Radar stations could detect them sending their snapshots to other V2X-based vehicles, since the quality of fixed Lidar/Radar stations should be more accurate than of the moving ones. Currently several companies develop Collective Perception Messages (CPMs) [24], which figure as an exchange of the mentioned Lidar/Radar data. A future development could foster the exchange of sensor data by vehicles using the CPMs.

Third, we discussed the usage of ontologies and OQA for realizing the LDM, which he regarded initially sceptical since scalability might be an issue. After explaining the intention of OQA this skepticism was in parts redressed. Regarding the modelling of the ontology, he stated that users do not care how the (ontological) model is shaped, more important is the query language, since users will directly be confronted with it. He identified that the query engine and the stream DB could be migrated and deployed on roadside ITS stations, where the user could cover custom use cases by writing and applying queries.

Fourth, he reviewed the query features, where he explained that an interval-based time model and push-based processing is favourable since more information can be extracted, and in C-ITS applications due to low latency push-based queries should be supported, which could include some buffering techniques so not every change will be computed. He pointed out that "native" query languages such as SQL might be too complicated for writing queries, and a simplified, object-oriented representation (similar to CQ) could be favourable. While discussing the example query, we observed that he understands and reads queries as rules. He noted also that, if the query get complex as in the second scenario, rules might be easier usable to express problems, since they more aligned with "the way people think". He also stated that the aggregation feature can be also used to guarantee privacy, since on aggregated values single vehicles are not distinguishable anymore. Furthermore aggregations can be used to calculate journey times in a road network. Finally, he agreed that predictions are a crucial feature for accident prevention, hence it should be more elaborated.

Expert 2. The second interviewed expert is the head of a traffic engineering department in a large ITS company and is responsible for managing R&D projects.

The expert described that historically traffic management was a closed, autarkic system, where traffic data was collected in a slow process using radar, road loops, and cameras. She identified V2X as an important step towards collecting real-time data on traffic and vehicles, where besides the mentioned CAM, SPAT, and DENM messages, Collective Perception Messages (CPMs) will play an important role, since they allow one to exchange locally perceived objects by a vehicles sensors, and exchange the object data with other V2X vehicles. She believed that the LDM could be used in combination with CPMs, where a CPM could be aligned with a vehicle's own LDM. However, she saw no immediate demand for the use of an ontology-based extension of a LDM and the use of spatial-stream queries to access the (streaming) data, since they use their own tools and languages for processing V2X messages. She commented though, that an ontology-enhanced LDM could be used as a data integration platform, which could be used for future data analytics tasks.

Expert 3. The interviewed expert is a associate professor in an European university, where he works in the fields where reasoning and learning over streams is applied to safe autonomous systems including robots, boats, and drones.

First, we discussed the LDM, and he noted that in robotics similar techniques have been used for long, but with the additional challenge that these dynamic maps have to build cooperatively, on-the-fly. An interesting challenge arises if different agents have a local representation of an LDM, and for coordination a global perspective has to be constructed based upon them. He identified another important task in this context, which is the matching of identical entities (in the sense of a physical grounding of the same entity) detected by different agents/sensors, which they solved by using bridge-rules.

Second, we discussed the LDM ontology, where he identified that meta-data is important, and a possibility of capturing uncertainty should be part of an ontology. Uncertainty introduces the additional challenge that measured observations, e.g., speed, are not crisp anymore, but are inside confidence intervals were the observation holds. He also stated that by using an average or most-likely values instead of intervals, we encounter a loss of information. This uncertainty also can occur in the classification of objects, wherein an instance of a vehicle is detected, but it turns out to be a bike. After discussing our query rewriting technique, he described their approach in robotics, which aims at matching the data to the query (or formula), instead of rewriting queries and checking for instances in the DB. The data is processed/transformed until it matches the queries. If new streamed data appears, they continuously try to match the streamed data to the queries.

Third, he reviewed the query and could understand most under certain assumptions (i.e., that aggregations are grouped), and he believed that including predictions are an important extension, which they consider for their approach as well. Since predictions need an underlying model, he sees that it is important to re-evaluate and monitor the predictions to detect concept drifts. He also commented on the semantics of query language and windows, where it is important to define when to forget data in the stream, and how long a data item holds into the future. Regarding the language features, he noted that intervals are good way to also express temporal uncertainty. Further it would be crucial to respect the underlying time model, which could be dense continuous or discrete. If someone assumes the finite number of changes assumption, it is possible to map the observation from the continuous into the discrete space. Finally, he believed that numerical aggregations and predictions could be enriched by different types of filters, whereby he mentioned that Kalman or particle filters [25] are often used in robotics.

Expert 4. This interviewed expert is a senior researcher in the fields of stream reasoning, graph DBs, and autonomous driving in an European university, and has also been involved in the development of stream reasoning tools.

First, we discussed the LDM, which he saw as a suitable approach to integrate map and streaming data. He pointed out though that classical DB technology as B+ trees [26] are not suited for spatial-stream data, but advances in the field of moving object DBs [27] result in efficient query languages and spatio-temporal indexing methods. Furthermore, the LDM does not consider a 3D representation of maps, which is crucial if the map is used in the context of autonomous driving, since detected objects of a image recognition step (e.g., building), need to be anchored in a 3D map.

Second, he evaluated the LDM ontology, where he agreed on the modelling regarding map features, but criticized that the LDM ontology misses to concept of sensors, resp., observations, which is crucial in C-ITS applications. He noted that elements such as *Stimulus*, *Sensor*, and *Observation* pattern of the Semantic Sensor Network (SSN) ontology [28] could be incorporated in the LDM ontology.

Third, he reviewed the query language, and easily understood the presented query (for red-light violations). He noted that the definition of the ontology and spatial atoms are clear, but the stream atoms need a formal grammar to capture an atoms parameterization such as $speed[avg, 10s]$. Subsequently, the definition of $speed[last, 5, -5s]$, i.e., the last element in a 10 secs window projecting 5 secs to the future, was not clear to him. He further suggested that using a SPARQL-dialect, either C-SPARQL [6] or CQELS [7] could be suitable languages. Evaluating the language features, he believed that all presented features are important, but more focus in terms of research and development should be given numerical- (F6) and trajectory prediction (F7), as he saw them as crucial for autonomous driving, since these are the main techniques used, for instance in object recognition and motion planning. He also believed that the combination of (stream) reasoning and machine learning is in its early stages and needs more attention from the community. He further suggested that the aggregates (F4) could be extended with top-k aggregate functions [29].

Fourth, he criticized that the scenario S3 is too generic, and should narrowed to more specific tasks in the domain of autonomous driving; he suggested to include motion planing as a replacement for the generic scenario.

5. Approach for Spatial-Stream Query Answering

First, we introduce the data model and the definition of a spatial-stream knowledge base, which leads to our main focus of spatial-stream queries. Then, we introduce the “standard” query rewriting and extend it with temporal relations. Finally, we describe how the queries are evaluated putting the focus on stream aggregation and predictions. We start from previous work in [2], which introduced spatial ontology-mediated query answering over Mobility Streams using DL-Lite$_A$ [3]. We focus on pull-based queries that are evaluated at one single time point called the *query time* $\mathbb{T}_i$.

5.1. 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 $(\mathbb{N}, \leqslant)$. 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 $\mathbb{T}$ (called a *timestamp*) 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*; the integer P is called *pulse* defining the general interval of consecutive data items on the timeline (cf. [30]), which 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. A spatial or spatial-stream DB is a Data Stream Management System that support spatial (geographic) objects and operators (e.g., ODYSSEUS, PIPELINEDB or SQLSTREAM).[4]

Example 5.1. 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:

[4] http://odysseus.informatik.uni-oldenburg.de/, https://www.pipelinedb.com/, and https://sqlstream.com/

$$\begin{aligned}v(0) &= \{speed(c_1,30),\ pos(c_1,(5,5)),\ speed(c_2,10),\\ &\quad pos(c_2,(4,4)), speed(b_1,10),...\},\\ v(1) &= \{speed(c_1,29),\ pos(c_1,(6,5)),\ speed(c_2,0),\\ &\quad pos(c_2,(5,4)), speed(b_1,5),...\},...\end{aligned}$$

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. Given atomic concepts A, atomic roles P, and atomic attributes E, we define (a) *basic* concepts B, *basic roles* Q, and *basic value-domains* E (attribute ranges); (b) *complex* concepts C, *complex* role expressions R, and *complex attributes* V_C; and (c) value-domain expressions D:

$$\begin{aligned}Q &::= P \mid P^-\\ B &::= A \mid \exists Q \mid \delta(U_C)\\ E &::= \rho(U_C)\\ C &::= \top_C \mid B \mid \neg B \mid \exists Q.C'\\ D &::= \top_D \mid D_1 \mid \ldots \mid D_n\\ R &::= Q \mid \neg Q\\ V_C &::= U_C \mid \neg U_C\end{aligned} \tag{1}$$

where P^- is the *inverse* of P, $\top_D$ is the *universal* value-domain and $\top_C$ is the *universal* concept; furthermore, U_C is a given attribute with domain $\delta(U_C)$ (resp. range $\rho(U_C)$). A DL-Lite$_A$ *knowledge base* (KB) is a pair $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ where the TBox $\mathcal{T}$ and the ABox $\mathcal{A}$ consist of finite sets of axioms as follows:

– *inclusion assertions* of the form $B \sqsubseteq C$, $Q \sqsubseteq R$, $E \sqsubseteq D$, and $U_C \sqsubseteq V_C$; respectively
– *functionality assertions* of the form $funct\ Q$ and $funct\ U_C$;
– *membership assertions* of the form $A(a)$, $D(c)$, $P(a, b)$, and $U_C(a, c)$, where a, b are individual names in Γ_I and c is a value in Γ_V.

The semantics of DL-Lite$_A$ is as usual (see [3]).

We introduce the possibility to specify the *localization* of atomic concepts and roles. For this, we extend the standard DL-Lite$_A$ syntax as follows:

$$\begin{aligned}C &::= \top_C \mid B \mid \neg B \mid \exists Q.C' \mid (loc\ A) \mid (loc_s\ A)\\ R &::= Q \mid \neg Q \mid (loc\ Q) \mid (loc_s\ Q),\end{aligned} \tag{2}$$

where $s \in \Gamma_S$ and the concept and roles are as before. Intuitively, $(loc\ \ A)$ is the set of individuals in A that can have a spatial extension (e.g., $(loc\ \ Parks)$), and $(loc_s\ A)$ is the subset where it is s (e.g., $(loc_{(48.20,16.37)}\ \ Vienna)$). The extension with streaming consists of the axiom schemes

$$(stream_F\ C) \text{ and } (stream_F\ R), \tag{3}$$

where F is a particular stream over either complex concepts C or roles R of $\mathcal{T}$. Details of the spatial and temporal extension of DL-Lite$_A$ are given in [31] and [2].

Example 5.2. A TBox may contain $(stream_{CAM}\ speed)$, $(stream_{CAM}\ (loc\ \ pos))$, $(stream_{CAM}\ Vehicle)$, and $(stream_{SPaT}\ hasState)$; we have the axioms $Car \sqsubseteq Vehicle$, $Bike \sqsubseteq Vehicle$, $Ambulance \sqsubseteq Vehicle$, and $Ambulance \sqsubseteq \exists hasRole.Emergency$ describing vehicle classes.

Finally, 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 spatial-stream DB. 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 $\langle \mathcal{F}, \mathcal{S}_\mathcal{F} \rangle$. The binding function $\mathcal{B}$ does not relate to the *mapping function* known from OBDA mapping concepts/roles to an underlying DB, but *guarantees* that spatial objects in $\mathcal{A}$ have a spatial extension in $\mathcal{S}_\mathcal{A}$ and $\langle \mathcal{F}, \mathcal{S}_\mathcal{F} \rangle$.

5.2. Query Language

Our query language is based on conjunctive queries (CQs) and adds spatial-stream capabilities (see Example 2.1). A spatial-stream CQ $q(\mathbf{x})$ is a formula:

$$\begin{aligned}&\textstyle\bigwedge_{i=1}^{m} Q_{O_i}(\mathbf{x}, \mathbf{y}) \wedge \bigwedge_{j=1}^{n} Q_{S_j}(\mathbf{x}, \mathbf{y}) \wedge\\ &\textstyle\bigwedge_{k=1}^{o} Q_{D_k}(\mathbf{x}, \mathbf{y}) \wedge \bigwedge_{l=1}^{p} Q_{T_l}(\mathbf{x}, \mathbf{y})\end{aligned} \tag{4}$$

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 [38]. 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 $Q_{T_l}(\mathbf{x},\mathbf{y}) = (T_1(z_1, z'_1), \ldots, T_q(z_q, z'_q))$ 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 $\{<, \leqslant, =, \geqslant, >\}$; for intervals, we choose the relations of Allen's Time Interval Algebra [18], 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. [30] 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.

5.3. *Query Rewriting by Stream Aggregation*

For the evaluation of spatial-stream CQs, we have to extend OQA to handle spatial and streaming data, which is not considered in the standard approach as [3]. In detail, we aim at answering pull-based queries at *a single* time point T_i with stream atoms that define *aggregate functions* on different windows sizes relative to T_i. For this, we consider a semantics based on *epistemic aggregate queries* (EAQ) [33] over ontologies by dropping the order of time points inside a window and handle the streamed data items as *bags* (multi-sets). Roughly, we perform two steps, where we (1) calculate only "known" solutions, and (2) evaluate the rewritten query, which contains the rewritten TBox axioms, over these solutions. 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$_A$. Further details on the algorithm for evaluating EAQs are provided in [2].

5.4. *Query Rewriting with Temporal Relations*

At first sight, spatial and temporal relations could be treated similarly. As shown in [2], we evaluate spatial relations regarding their *Point-Set Topological Relations*. This 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 $p = (p_1, \ldots, p_n)$ 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) [18] 13 relations can hold between two intervals. The domain of IA relations is the set of intervals $\mathbb{I} = \{[p_1], \ldots, [p_k]\}$ 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 [18]:

$$\begin{aligned}
before(x, y) &= \{(x, y) : \underline{x} < \overline{x} < \underline{y} < \overline{y}\}\\
meets(x, y) &= \{(x, y) : \underline{x} < \overline{x} = \underline{y} < \overline{y}\}\\
overlaps(x, y) &= \{(x, y) : \underline{x} < \underline{y} < \overline{x} < \overline{y}\}\\
starts(x, y) &= \{(x, y) : \underline{x} = \underline{y} < \overline{x} < \overline{y}\}\\
finishes(x, y) &= \{(x, y) : \underline{y} < \underline{x} < \overline{x} = \overline{y}\}\\
during(x, y) &= \{(x, y) : \underline{y} < \underline{x} < \overline{x} < \overline{y}\}\\
equal(x, y) &= \{(x, y) : \underline{y} = \underline{x} < \overline{x} = \overline{y}\}
\end{aligned} \qquad (5)$$

Interpretation of IA relations. 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 match 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

[18]. 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.

From timestamps to intervals. Time intervals are not directly represented in our streamed data, but are an intermediate product of the EAQ evaluation and are used to annotate the aggregated objects. After evaluating an EAQ, the results (also called answers) are aggregations of data items, hence their single timestamps are not "meaningful" anymore; we need to assign time intervals to the answers, which can be taken either from the window itself, or be extracted from the aggregations themselves.

We already have introduced the function v that assigns to each element of $\mathbb{T}$ the data items of $\langle \mathcal{F}, \mathcal{S}_{\mathcal{F}} \rangle$. Now, we define the function $v^* : \mathcal{E}_\phi \rightarrow \mathbb{I}$, where $\mathcal{E}_\phi$ is the set of epistemic (certain) answers that result from evaluating the EAQ q_ϕ (as described in [2]) and $\mathbb{I}$ is the set of intervals $\{[p_1], \dots, [p_k]\}$ over the linear order of $\mathbb{T}$; q_ϕ represents a single query with the atom $\phi[agr, b, e]$, where agr, b, and e are defined as before. The function v^* assigns to each answer of q_ϕ in $\mathcal{E}_\phi$ one interval of $\mathbb{I}$. For instance, we assign the window size $[2, 6]$ to an answer as follows: $v^*(speed(c_1, 50)) = [2, 6]$. Note that we also have a shorter notation using @, which would be in our example: $speed(c_1, 50)@[2, 6]$.

The function v^* can be defined in different ways. In a first approach, we use $\mathbb{T}_i$ and the window sizes b and e for generating the interval for the annotation. 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 and aggregated answer of an EAQ the upper and lower bounds of the timestamps of each data item that is part of the group/aggregation.

More sophisticated approaches might include a segmentation of the data items, thus creating different fragmented subintervals.

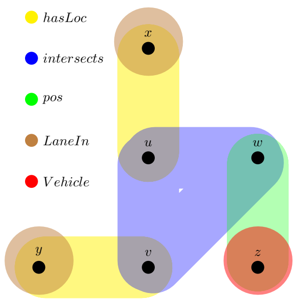

Figure 2. Hypergraph of q_2

5.5. Query Evaluation by Hypertree Decomposition

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 [3]. 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 summarized 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 [2], 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, based on the query hypergraph and the derived join plan, 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 the standard DB literature such as [34].

The main steps of our query evaluation strategy are 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.

Thus we have sub CQs that share the same aggregation/prediction functions and the 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 (a temporary table) R_{ϕ_i}. 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 cache the results in memory for future queries.

Example 5.3. The following example is a simplified version of q_1, where the layers distinguish between ontology (first), stream/temporal (second), and spatial (third line) atoms:

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

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

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

5.6. 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 of items*: *first*, *last*, where they give the first or last element in the stream, respectively;
– *Simple aggregations*: *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) *lin_reg* calculates the log-linear regression model;
  - (b) *mov_avg* calculates the moving average of the past values;
  - (c) *exp_smooth* applies simple exponential smoothing; and
  - (d) *grad_boost* uses gradient boosting with regression trees.

Note that the order of items is lost due to the bag semantics, the temporal annotations (e.g., $\mathit{speed}(c_1, 50)@10$) 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 p_b that represent geometries. As with *first* or *last*, we must rearrange them to create a valid geometry, i.e., a sequence $p_o = (p_1, \dots, p_n)$ of points. We allow these functions to derive new geometries (among others):

– *point*: we evaluate the function *last* to get the last data item p_n of the sequence p_o;
– *line*: we create a sequence of points p_o representing a path by calculating a total order on the bag of points p_b, 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 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 the start- and endpoints are the same, i.e., $p_1 = p_n$, by (1) determining the *convex hull* of the bag of points, and (2) extracting all pairs of points representing the convex hull;
– *traject_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 latter 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.

6. Implementation

We have implemented a prototype of our spatial-stream OQA approach in JAVA 1.8 using the stream RDBMS PIPELINEDB 9.8.1.[5] The system architecture is shown in Figure 3. We chose PIPELINEDB, as it is built on top of PostgreSQL[6] and PostGIS[7] 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 mapping 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.

6.1. 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 [35] implementation.[8] 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, window 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 already cached; (2) if not, it instantiates one of the evaluators according to the sub CQ type.

[5]https://www.pipelinedb.com/

[6]https://www.postgresql.org/

[7]http://postgis.net/

[8]https://www.dbai.tuwien.ac.at/proj/hypertree/

Ontology evaluator: this evaluator uses the DL-Lite$_A$ query rewriter OWLGRES 0.1 [36], but a more efficient implementation as in ONTOP [37] is planned.
Stream evaluator: for each stream sub CQ q_i, it detemporalizes the streams by grouping/aggregating the data items and performs 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.[9]
Spatial evaluator: it handles the different spatial relations. For performance reasons, we do not compile the spatial relation to SQL, but evaluate them in-memory using the functions of the JTS TOPOLOGY SUITE.[10]
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 relation 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.

We designed the prediction function being an integrated part of the stream evaluator. However, predictions could be treated as separate atoms, generating data items continuously. This would be an appealing extension, but would change the query evaluation process and needs further investigation. The same considerations apply to the trajectory predictions. The aggregate 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 apply a simple straight-line projection to create new points. However, taking the curvature into account is a desired extension.

[9]https://xgboost.readthedocs.io/en/latest/

[10]https://github.com/locationtech/jts

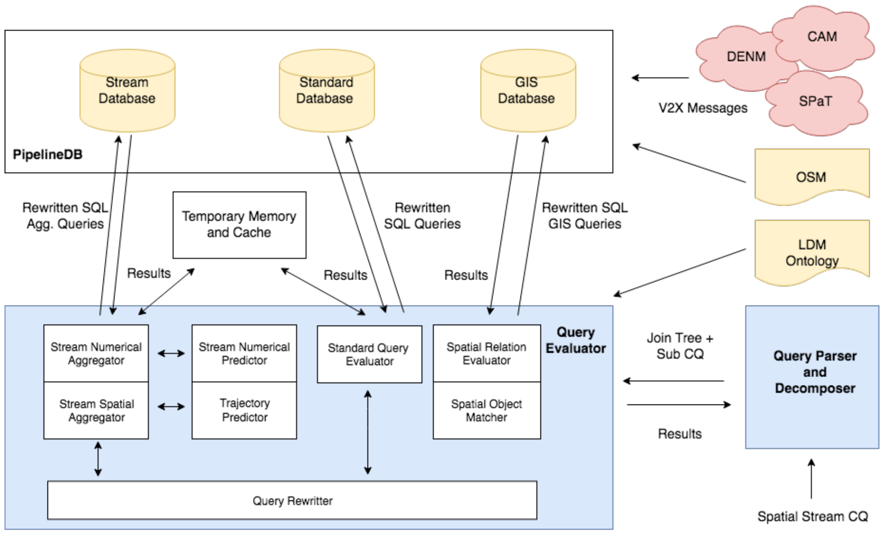

Figure 3. System Architecture

7. 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 the evaluation website.[11]

7.1. Scenario Data

For having realistic traffic data, we generated our streaming data with the microscopic traffic simulation tool, PTV VISSIM[12] 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. A log player is provided to replay the different simulations by feeding the log data to PIPELINEDB. For varying the data throughput, we adjusted the following parameters: (a) replayed 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 simulated light, medium, and heavy traffic in our scenario, where we have approx. 20, 50, and 150 vehicles, respectively, simultaneously on the road network. We modeled a real-world scenario shown in Fig. 4, 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.

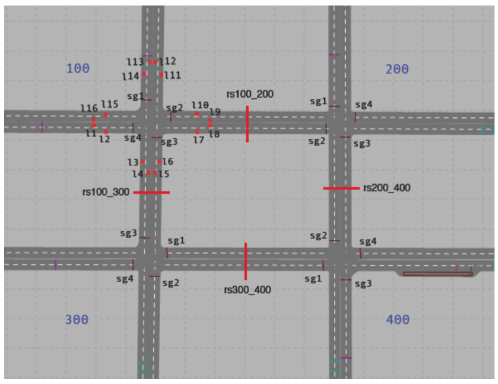

Figure 4. Four Intersection Scenario

[11]http://www.kr.tuwien.ac.at/research/projects/loctrafflog/ekaw2018

[12]http://vision-traffic.ptvgroup.com/en-us/products/ptv-vissim/

7.2. *Queries for Experiments*

Based on the requirements, we derived a set of queries to assess each scenario, where each query aims at answering a specific problem of the use case taking the set of features into account. We use a more compact representation, where the commas between atoms are conjunctions and 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:

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

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 us to check the vehicle's position 7s later, and the temporal relation *before* ensures that a vehicle first passes $i100$ and then $i200$.

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

For the use case *S2.1* (simple maneuvers), query $q_{2.1}$ returns all vehicles x that turned left or right in the last 6s, where the function *match* is an extension of the aggregate function *line_angle*, which incorporates a filter with a predefined interval on the results of *line_angle*, e.g., they have to be between -175 and -15. Then both results are combined by unions of CQs resulting in all vehicles performing the two maneuvers.

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

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 passed for a certain period from an in-lane to an out-lane or vice versa.

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

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.

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

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).

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

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".

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

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 the vehicle is currently driving on.

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

In use case *S3.2* (obstructed view) we compute query $q_{3.2}$, where our prototype (as part of the ego vehicle) aims to detect vehicles that very likely will collide in 2s on an intersection by checking whether their predicted trajectories will cross another vehicle's trajectory.

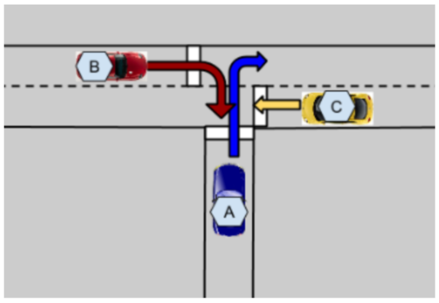

Figure 5. Use Case Traffic Rules

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

In *S3.3* (traffic rules), our ego vehicle approaches an uncontrolled intersection at the same time with other vehicles. According to (local) traffic rules, preference (shown Fig. 5) is given to (1) the vehicle on the main road, (2) the one that is not changing lanes, and (3) the vehicle approaching from the right. Vehicle *C* has preference over *A* and *B*, where *B* would have preference over *A*, but the preference can not be given, since *A* is on a single-lane road. We can express the traffic rules with the following Datalog rules:

$$willCross(x, y) \land straightOn(x) \land turnRight(y) \land onMainRoad(x) \land onMainRoad(y) \rightarrow giveWay(y, x) \quad (1)$$
$$willCross(x, y) \land onMainRoad(x) \land onTributRoad(y) \rightarrow giveWay(y, x) \quad (2)$$
$$willCross(x, y) \land giveWay(x, y) \land onSingleLaneRoad(x) \rightarrow giveWay(y, x) \quad (3)$$
$$vehicle(x) \land vehicle(y) \land giveWay(y, x) \rightarrow stop(y) \quad (4)$$

The atom $willCross(x, y)$ matches all vehicles that might collide and can be evaluated by $q_{3.2}(x, y)$ (modified without $isEgo(x)$). The atoms $turnRight(x)$ and $straightOn(x)$ can be evaluated by the queries $q_{2.1r}(x, y)$ assuming the queries are treated as atomic rules with $q(x, y)$ as the head. The atoms $onMainRoad(x)$, $onTributRoad(y)$, and $onSingleLaneRoad(x)$ can be evaluated in the spirit of $q_{1.2a}(x, y)$ checking spatial containment. Then, the rules of *S3.3* can be expressed as unions of CQs, but with the difficulties that the order of the query evaluation effects the completeness of the results, Rule (2) as to be evaluated before (3).

7.3. Results

We conducted our experiments on a Mac OS X 10.14.4 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 shown 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 density and update delay in ms. The new experiments confirm results of [2] 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 artificially delaying the next stream atom evaluation by 180ms. The artificial delay is empirically determined and needed for PIPELINEDB to set up the *continuous views* (CVs); ignoring this would lead to missing results.

As shown with the queries that are marked by an asterisk, we could improve the baseline time by an AET ranging between 0.49s and 1.05s. This can be attributed to the following optimizations: (a) pre-compiling the static part of the query, which shortens evaluation by approx. 25%, and (b) parallel execution of the stream atoms, which improves performance by approx. 20%.

The added functions for statistics, matching, and predictions, i.e., *mov_avg* and *traject_line* do not affect performance, since they are applied on small windows with few data items. If we would apply *gradboost* for predictions on large windows for long-term traffic predictions, our query time can rise considerably, since prediction time (without a preprocessed training step) can be above 20s.

7.4. 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*

	$\#Q$	$\#A$	(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
$q^*_{1.1}$	3(2)	42	0.46	0.45	0.38	0.37	0.48	0.46	0.40	0.38	0.50	0.47	0.41	0.39
$q_{1.2}$	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
$q^*_{2.3}$	7(3)	50	0.66	0.63	0.56	0.54	0.69	0.65	0.57	0.55	0.67	0.66	0.59	0.55
$q_{2.4}$	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, where the * marks results from warm starts

for long-term traffic forecasting. For trajectory prediction (*F7*), we have implemented a method based on a simple linear path calculation. However, 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.

7.5. Summary of Expert Evaluation

Overall, the experts confirmed that (a) the extension to time intervals and temporal relations, and (b) the inclusion of prediction capabilities are important extensions of the initial spatial-stream OQA approach. However, it turned out that the experts identified several limitations and interesting extensions.

The first limitation regards the *LDM ontology*, which should be aligned with the SSN ontology [28] to include patterns such as *Stimulus*, *Sensor*, and *Observation*. Furthermore, it should also capture 3D maps. The second limitation regards the unclear definition of the *grammar* of stream atoms, which should be clarified in this work. The third limitation is more generic and captures the assumption that rules would be better suited to capture the complex use cases, which is most apparent in the traffic rules use case.

The first extension regards the Collective Perception Messages (CPM) [24], which could be added as new type of message to share local sensor data. The second extension addresses Scenario 3, which could be extended with use cases that are taken from motion planning tasks in autonomous driving. The third extension identifies Kalman filters [25] and top-k aggregates [29] as more powerful aggregation functions. The fourth extension is likely the furthest reaching, since the ontology and query language (with the underlying semantics) could support uncertainty on the level of (a) data items and (b) of TBox assertions, e.g., inclusion assertions. The fifth extensions describes the handling of data items inside windows, where adaptably forgetting data items and extending their validity into future could be added to have a more flexible query language. The last extension is again more generic and captures the use of LDMs in an autonomous agent-based scenario, which would lead to the challenge of aligning different LDMs to a global dynamic map.

8. Related Work and System Comparison

First, we give a general overview on related work, and will discuss in-depth related systems in the sections afterwards, where we provide an qualitative and quantitive comparison with selected systems.

8.1. Overview

Data stream management systems (DSMSs) such as STREAM [38], Aurora [39], and TelegraphCQ [40] were built supporting streaming applications by extending RDBMS. More recently, RDF stream processing engines, such as Streaming SPARQL [41], C-SPARQL [6], SPARQLstream [42], and CQELS [7], were proposed for processing RDF streams integrated with other Linked Data sources and background KBs. EP-SPARQL [43], resp., LARS [44] proposes a language that extends SPARQL, resp., CQ with stream reasoning, but translate KBs into expressive (less efficient) logic programs. Regarding spatio-temporal RDF stream processing, a few SPARQL extensions were proposed, such as SPARQL-ST [45] and st-SPARQL [46]. Closest to this work are (i) [47], which extends CQELS and supports spatial operators as well as aggregate functions over temporal features (ii) [42], which allows the rewriting of SPARQL queries over stream RDBMS, and (iii) [48], 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.

Our approach is situated in-between "classical" stream processing approaches that handle the streaming data as bags in windows, and temporal QA over DL-Lite using temporal operators like LTL in [49], which are evaluated over a (two-sorted) model separating the object and temporal domain. We believe that detemporalization with its bag semantics suffices for the C-ITS case, since the order of V2X messages is not guaranteed, and for most of the normal as well as spatial aggregates it can be ignored (e.g., sum) or is implicit in the data (e.g., Euclidian distance of points). Besides [49], similar temporal QA is investigated in [50] and [51], which are all on the theoretical side and provide no implementation yet. Finally, we build on the results for EAQs in [33], but we introduce spatial streams and more complex queries. Temporal QA is also investigated in [49] and [50], both are on the theoretical side and provide no implementation yet.

8.2. Comparison with Existing Systems

There is a wide range of systems for stream processing, stream reasoning, and event detection available. For a comparison with our approach, we focus on systems that fulfill the following criteria:
- ability to deal with streaming or temporal data, either by having a window operator, or supporting incremental updates;
- ability to provide reasoning, either based on ontology-, rule-, or query rewriting-based methods;
- a maintained implementation of the system should be available.

As commented in Section 3, we will not re-evaluate the eight "standard" requirements of [16], and the three entailment levels for stream reasoning systems of [17]. Still, this work overlaps on some features with the comprehensive study of Margara et al. [52], where the authors compare systems according to the criteria:
- continuous queries (F2/F3),
- background data (F9),
- time model (F1),
- reasoning (F9), and
- temporal operators (F1).

The other criteria in [52], namely data transformation, uncertainty management, historical data, quality of service, and parallel/distributed processing are not investigated in the context of this work.

The comparison in Table 3 shows the evaluation of the selected systems on the generic features *F1* (Time model), *F2* (Process paradigm), *F3* (Query features), and *F9* (Advanced reasoning), as well as the ITS domain specific features *F4* (Numerical aggregations), *F5* (Spatial aggregations), *F6* (Numerical predictions), *F7* (Trajectory predictions), and *F8* (Spatial matching). We separate the table into two groups, where the first group represents query-based systems, and the second group rule-based systems.

8.2.1. C-SPARQL

C-SPARQL was introduced by Della Valle et al. [6] and was one of the first contributions in the area of stream processing with reasoning extensions. The main intention of this work lies in bridging the gap between the world of stream processing systems, i.e., stream DB systems, and the Semantic Web, i.e., RDF and SPARQL. C-SPARQL includes (a) a language for continuous queries over streams of RDF data, (b) an evaluation engine for this language, whereby C-SPARQL has the distinguishing features of (i) supporting timestamped RDF triples, (ii) supporting continuous queries over streams, and (iii) defining of ad-hoc, explicit operators for data aggregation. The results of C-SPARQL queries are continuously updated as new data items appear on the stream, hence an efficient evaluation of sliding windows is possible.

8.2.2. CQELS

Similar to C-SPARQL, CQELS [7] also offers a query language and processing engine to answer queries over a combination of static and stream RDF data.

System	F1	F2	F3	F4	F5	F6	F7	F8	F9
C-SPARQL	Point	Pull	SPARQL + windows	Yes	Pre	Pre	Pre	No	RDFS
CQELS	Point	Push	SPARQL + windows	Yes	Pre	Pre	Pre	No	RDF
INSTANS	Point	Push	SPARQL + event patterns	Yes	Pre	Pre	Pre	No	RDFS + Rete
SPARQLstream / Morph-streams	Point	Pull	SPARQL + windows	Yes	Pre	Pre	Pre	No	OWL2 QL
*ONTOP (DatalogMTL)**	Point + MTL	Pull	SPARQL + windows	Yes	Yes	Yes	Pre	No	DatalogMTL
*ONTOP (STARQL)**	Point	Pull	SPARQL + windows	Yes	Yes	Yes	Pre	No	OWL2 QL
TrOWL	Point	Pull	CQ	Lmt	Pre	Pre	Pre	No	OWL2 EL or approx. OWL2 DL
*Clingo (Multi-shot ASP)**	Point	Pull	Rules + ext. atoms	Yes	Ext	Ext	Ext	Ext	ASP
ETALIS (EP-SPARQL)	Interval + full AIA	Push	Rules or SPARQL + windows	Yes	Lmt	Pre	Pre	No	Prolog
RDFox	Point	Pull	Rules	Yes	Pre	Pre	Pre	No	Datalog / OWL2 RL (incremental)
*Laser (LARS)**	Point + LTL	Pull	Rules + windows	Yes	Pre	Pre	Pre	No	Stratified LARS
*Ticker (LARS)**	Point + LTL	Pull	Rules + windows	Yes	Pre	Pre	Pre	No	LARS
Spatial-stream OQA	Point/Interval with limited AIA	Pull	CQ + windows	Yes	Yes	Yes	Lmt	Lmt	OWL2 QL

Table 3

Qualitative comparison of F1 - F9 on selected systems (* indicates comments on systems, *Ext*, *Pre*, and *Lmt* means evaluation by external atoms, preprocessing needed, and limited coverage)

While C-SPARQL adopts a “black box” approach, i.e. static and stream query elements are first divided and sent to the respective underlying stream and RDF query engines, CQELS, on the other hand, takes a “white box” approach by natively implementing the required query operators for the triple-based data model, both for streams and static data. This native approach enables better performance and can dynamically adapt to changes in the input data.

Another difference is that CQELS takes an eager query execution strategy: input data is processed as soon as it arrives in the system, contrary to the periodic evaluation of C-SPARQL which is triggered periodically, regardless of the input data throughput.

8.2.3. ETALIS with EP-SPARQL

The ETALIS system [43, 53] was applied to the ITS domain and offers the combination of Datalog-style rules with a background knowledge base. In ETALIS a Prolog-based language is used to express complex event patterns with predicates like $during(event1, 5)$ or $begins(event1, event2)$. The background knowledge is also encoded into the rule language, which in combination with the temporal and causal parts can be used for query answering. The standard Prolog query evaluation (based on a request-response paradigm) is altered to an event-driven backward chaining (EDBC) of rules. A standard Prolog system is then triggered to evaluate a query and additional EDBC rules when new data is arriving at the system. EP-SPARQL [43] is an approach that extends ETALIS and introduces windows and the handling of RDF streams for lifting the rule-engine to a Semantic Web/Linked Data settings.

8.2.4. INSTANS

INSTANS [54] is an event processing engine based on handling multiple interconnected SPARQL queries with updates. It supports continuous evaluation of incoming RDF data using an encoding of SPARQL queries into Rete-like structures. INSTANS supports stateless/stateful filters using its internal memory, enrichment, (de-)composition, aggregation and pattern matching on the streamed events. The authors have implemented their approach, where they provide a conversion of the SPARQL queries to Rete rules, which then are evaluated on the Rete [55] rule engine JESS [56].

8.2.5. SPARQLstream/Morph-streams

SPARQLstream [42] extends standards SPARQL with time windows over streams similar to C-SPARQL, but adds the relation-to-stream operator for dealing with relational streams. SPARQLstream was further extended to Morph-streams [57], where OBDA techniques are applied to access the underlying streams, which then are stored in a stream processing system (SPS). R2RML mappings are used to create virtual streams on-the-fly, which can be accessed in their SPARQL extension. The authors implemented and tested their query rewriting techniques for different SPS, namely for SNEE, Esper, and Global Sensor Networks (GSN).

8.2.6. *Clingo with Multi-shot ASP*

Multi-shot Answer Set Programming (ASP) is an extension of existing ASP solving techniques, which deals in a reactive fashion if new information arrives at the logic program, instead of solving the program from scratch. Clingo 4 [58] supports natively multi-shot solving by offering high-level constructs and control capacities via the scripting languages Lua and Python. The authors introduced the *#external* directive, which allows a flexible handling of undefined atoms. Additionally, the solver needs to keep account for the sequence of system states, which is defined using a simple operational semantics, where operations such as *create*, *add*, or *assignExternal* can modify the states.

8.2.7. *LARS with Ticker/Laser*

The LARS framework [59] is a recent development in stream reasoning that considers lifting of ASP to streams with generic windows for capturing data snapshots, so as to generalize time-, tuple- and partition-based window functions. To this end, the ASP syntax is extended with window operators and with temporal operators for evaluating truth at every, some, and a specific (exact) time point in a stream; variables ranging over domain constants or time points are allowed in rules. Semantically, answer sets of ASP programs are naturally generalized to answer streams of LARS programs. Fragments of LARS programs have been implemented in Ticker [60] and Laser [61], based on plain LARS rules.

In Ticker,[13] full negation is supported, with sliding time-based and tuple-based windows. It comes with two evaluation modes, viz. a static ASP encoding, which employs Clingo for repeated solving, and an incremental ASP encoding, which performs model update by truth maintenance techniques; the latter is usually faster.

Laser[14] is geared towards high performance and thus focuses on positive and stratified programs with sliding windows. It aims at fast model update by efficient substitution management, for which it extends semi-naive evaluation of Datalog by incorporating the temporal dimension and tracks intervals of guaranteed formula validity. This avoids redundant re-derivations and allows for efficient removal of expired derivations. Laser was shown to outperform Ticker, C-SPARQL and CQELS in micro-benchmarks.

[13]https://github.com/hbeck/ticker

[14]https://github.com/karmaresearch/laser

8.2.8. *ONTOP with STARQL and DatalogMTL*

The STARQL framework of Özçep et al. [62] is an effort of streamifying ODBA by introducing an extensible query language, hence it is closely related to our approach. It uses a first-order logic fragment for temporal reasoning over sequences of ABoxes. The framework extends the first-order query rewriting of DL-Lite with intra-ABox reasoning. In a second extension of ONTOP, the authors added Metric Temporal Logic (MTL) to allow querying of log data using LTL operators that are extended with time intervals [63, 64]. For this purpose, they introduced datalogMTL, which combines non-recursive Datalog with the MTL operators.

8.2.9. *RDFox*

RDFox [65] is the combination of a scalable main-memory RDF store that supports materialisation and parallel Datalog reasoning, which also includes SPARQL query answering. The Datalog materialisation is based on a novel parallel reasoning algorithm extending the well-known DRed algorithm, computing incremental updates on its internal triple store.

8.2.10. *TrOWL*

TrOWL [66] is an incremental DL reasoner over the expressive OWL2 DL language. It handles streams of KBs instead of using fixed time windows over streams, hence allowing to add and remove axioms from the KB on-the-fly. The authors applied syntactic approximation to reduce the reasoning complexity, which guarantees soundness but looses completeness in certain cases. TrOWL provides justifications for the following entailments: atomic concept subsumption, atomic class assertion and atomic object property assertion.

8.3. *Feature Comparison*

Based on the literature review and discussions with the system developers, we conducted a feature evaluation of the above systems. In Table 3, we summarized the reviews and discussions, where we use the underlying specifications for the three levels of fulfillment: *basic, enhanced,* and *advanced.* For F1, the basis level matches to a point-based time model, the enhanced level would relate to an interval-based model, and the advanced level would include AIA over an interval-based model. For F2, the basic level includes pull-based, the enhanced level push-based queries, and the advanced level the combination of pull- and push-based queries. In F3 and F9, we list the query, rule, or ontology language and the possibility to allow windows, but do not classify the fulfillment level. For F4 to F8, we evaluate how a specific feature is covered by the *basic* fulfillment levels, since most systems are generic reasoners, and are not intended to support ITS-specific features.

As shown in Table 3, for F1 the ONTOP- and LARS-based systems offer similar or richer query- and ontology languages, since these systems allow the use of LTL and MTL operators. For F2, our work is on a par with most presented systems, except the two push-based systems CQELS and INSTANS, which support a higher reactiveness, since they support an eager query execution strategy. For F4 and F5, our work covers the widest range of numerical aggregation and prediction functions, with the exception of STARQL, which covers similar functionalities. One of the motivation of this work are the coverage of F6 to F8; hence our work is the only approach that supports spatial aggregations and predictions.

8.4. Performance Comparison

We conducted our experiments on a Mac OS X 10.14.4 system with an Intel Core i7 2.9GHz, 8GB of RAM, and a 250GB SSD. We calculated the average of 50 runs for query evaluation time with warm starts, hence, we did not restart the systems over 50 runs in each experiment.

For the experiments, we compared our prototype on two selected queries with the state-of-the-art systems C-SPARQL and CQELS, which support limit reasoning but are designed to deal with high velocity and volume streams. The comparison of all presented queries is not feasible, since C-SPARQL and CQELS do not support natively the features spatial/temporal relations, spatial aggregation, and inline predictions. Hence, we selected the two queries $q_{1,.1}$ and $q_{2,.3}$, where the first is our baseline comparison and the second is our running example. We pre-calculated the missing features such as the spatial relations in the log player and materialized the outcome as streamed data items. Furthermore, we adapted our CQs to the SPARQL dialects of each system; in Figure 6, we give the encoding of $q_{1.1}$ as an example.

The results of our experiments are shown in Table 4 where t is the AET in seconds for different traffic densities and update delays in ms. The baseline systems C-SPARQL and CQELS outperform our prototype in the range between 70ms in $q_{1.1}$ (C-SPARQL on heavy traffic with 100ms delay) and 657ms in $q_{2.3}$ (CQELS on light traffic with 5ms delay). The results are an important indicator for a lower bound of QA over streams. Yet, the results are not fully comparable since CQELS or C-SPARQ respectively, do not support RDFS or OWL2 QL, respectively; there is a trade-off between performance and expressivity, where in CQELS the full TBox is omitted, and in C-SPARQL axioms with existentially-quantified variables on the right-hand side of inclusion assertions are ignored leading to incomplete results. Furthermore, both systems do not support directly spatial relations, aggregates, and predictions, which amount to approx. 100ms evaluation time, since these functions are precomputed by the log player in the experiments, which is not reflected in the AET. The results also indicate that by increasing traffic density, the performance seems to align in $q_{1.1}$, where the grouping/aggregation might be the most demanding operation for C-SPARQL and CQELS.

After profiling the runtime of our prototype, we noticed that approx. 200ms are lost by establishing a connecting to PIPELINEDB, which could be mitigated by pooling the connections using a persistent connection pool such as provided by PG BOUNCER.[15]

9. Conclusion and Future Work

This work was sparked by applying spatial-stream query answering as an effort to integrate and access streamed mobility data, e.g., vehicle movements, in a spatial context over the complex C-ITS domain. In [2] we have introduced simple aggregate queries over streams, which often do not suffice to capture more complex use cases. In this paper, we presented 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 detecting (potential) accidents. Based on the newly developed scenarios of *traffic statistics*, *event detection*, and *advanced driving assistance systems* (ADAS), we have defined a set of domain-specific features such as trajectory computation, which are matched with the

[15]http://pgfoundry.org/projects/pgbouncer

```
PREFIX :     <http://www.kr.tuwien.ac.at/its/ldm/items#
SELECT       ?vehicle (AVG(?val1) AS ?as) (MAX(?val1) AS ?ms)
FROM STREAM <http://www.kr.tuwien.ac.at/its/streams/vehicles>
             [RANGE 10s STEP 1s]
WHERE        { ?vehicle :observed ?obs . ?obs :speed ?val1 .
             ?vehicle :vehicleMaker ?maker . FILTER
              ( regex(str(?maker), 'BMW', 'i') ||
              regex(str(?maker), 'VW', 'i') ) }
GROUP BY     ?vehicle
```

```
PREFIX : <http://www.kr.tuwien.ac.at/its/ldm/items#
SELECT   ?vehicle (AVG(?val1) AS ?as) (MAX(?val1) AS ?ms)
WHERE    {STREAM <ws://localhost:8124> [RANGE 10s]
         {?vehicle :observed ?obs . ?obs :speed ?val1 .
         ?vehicle :vehicleMaker ?maker .
         FILTER ( regex(str(?maker), 'BMW', 'i') ||
          regex(str(?maker), 'VW', 'i') ) } }
GROUP BY ?vehicle
```

Figure 6. C-SPARQL encoding (left) and CQELS encoding (right) of query $q_{1.1}$

		(l) with ms delay				(m) with ms delay				(h) with ms delay			
Query	System	5	10	50	100	5	10	50	100	5	10	50	100
$q_{1.1}$													
	C-SPARQL	0.019	0.032	0.031	0.027	0.132	0.134	0.113	0.110	0.252	0.347	0.324	0.314
	CQELS	0.034	0.044	0.043	0.038	0.029	0.038	0.043	0.041	0.056	0.043	0.035	0.043
	Spatial-stream QA	0.464	0.446	0.384	0.370	0.480	0.463	0.404	0.376	0.499	0.474	0.414	0.389
$q_{2.3}$													
	C-SPARQL	0.045	0.025	0.058	0.070	0.050	0.090	0.048	0.077	0.307	0.253	0.265	0.353
	CQELS	0.001	0.003	0.013	0.034	0.001	0.002	0.006	0.017	0.001	0.003	0.009	0.020
	Spatial-stream QA	0.658	0.637	0.561	0.542	0.692	0.653	0.574	0.548	0.670	0.660	0.585	0.554

Table 4

Results (t in secs) for query $q_{1.1}$ and $q_{2.3}$ with the scenarios (l)ow, (m)edium, and (h)eavy traffic

scenarios/use cases to define the requirements. Given the new features, we extended the LDM ontology, the spatial-stream query language, and extended the methods used for query answering accordingly. We also redesigned the architecture and optimized the system by pre-compiling the static query elements and executing stream atoms parallelly. The experimental evaluation provides evidence for an improved performance of approx. 40% and an evaluation time below 700ms. This indicates that potentially the feasibility and efficiency of our approach in the mentioned scenarios is given.

Lessons Learned. The presented approach of spatial-stream QA is well-suited for data integration and query answering in the C-ITS domain. The concept of a LDM was a good starting point, since it has been developed and standardized by the C-ITS community and was already extended by Netten et al. to an ontology-like model [13]. In particular, Semantic Web Technologies play to their strengths in easily modelling a complex domain such as C-ITS, and allowing the (expert) user to formulate powerful queries on top of the streams that are integrated by the ontology. This can be seen by the new scenario ADAS, where small modifications of the ontology and new queries open up a new application field. However, our approach of using OQA revealed some limitations that are discussed in expert summary.

Using spatial-stream CQs for capturing the scenarios worked out to our satisfaction in most of the use cases. But as illustrated with use cases *S2.4*, *S2.5*, and *S3.3*, our language reached its limits regarding “usability” and also “expressivity”. If we have a larger set of rules as in *S3.3* (even without transitivity), the conversion to unions of CQs becomes cumbersome and inefficient (AET is between 2.46s and 3.34s), hence rule engines such as in Ticker [60] or Laser [67] might be better suited. In *S1.3* and *S2.5*, we can see that qualitative temporal relations like $before$ are convenient, so we can avoid the implicit encoding of temporal relation using (fixed) shifted windows, and use the join operation to merge them. This solution needs an a-priory definition of window sizes and reference of the sizes to each other, which makes it inflexible and prone to errors if the times between the window sizes change. Furthermore, the usage of CQ with sub-queries can be directly transferred to SPARQL queries, the underlying evaluation system would have to be adjusted to deal with the mobility-specific features.

Outlook. We believe that stream processing/reasoning methods could well be applied to the mentioned C-ITS scenarios, which was confirmed by the experts; they also acknowledged that the new features time intervals with temporal relations and prediction capabilities are important extensions of the initial spatial-stream OQA approach.

Nevertheless, the experts identified practical and theoretical extensions that should be addressed in future research. On the practical side, they suggested that the LDM ontology could be combined with the SSN ontology [28] , and Collective Perception Messages[24] could be used to integrate local sensor data of other vehicles. This could be further elaborated such that the different LDMs (of each vehicle) could be aligned to a single global dynamic map. Furthermore, the experts suggested that we could integrate Kalman filters [25] and top-k aggregates [29] to provide more powerful aggregates. On the theoretical side, our methods could be extended to capture uncertainty on the level of data items and also of TBox assertions, which would lead to a change in the underlying semantics and the computational properties. Furthermore, our methods could be extended to handle windows in a more flexible manner by flexibly forgetting data items or extending their validity into the future. Finally, the experts suggested that Scenario 3 could be focuses more on motion planning instead of ADAS.

In addition to the suggestions of the experts we believe that efforts on following issues would be beneficial: (a) allowing the integration of external (domain-specific) modules such as functions for advanced trajectory prediction, which would be similar to external atoms in ASP [22]; (b) the possibility of analytic queries over longer periods, hence an extension with a transient cache and variable window sizes would be needed; (c) the full integration of OQA with IA relations, which would include the representation of IA networks and the rewriting of subsets of the composition table; and (d) handling more complex queries and rules while still maintaining scalability, which bring our approach closer to Ticker [60] and Laser [61].

As discussed in the section above, ongoing and future research should be directed to extend the languages, methods, and the platform to fulfill the defined requirements, which will enable us to apply our approach and prototype to more complex scenarios such as motion planing.

Acknowledgements. This work has been supported by the Austrian Research Promotion Agency project LocTraffLog (FFG 5886550) and DynaCon (FFG 861263), as well as by the European Commission through IOTCrawler (H2020 contract 779852).

References

[1] L. Andreone, R. Brignolo, S. Damiani, F. Sommariva, G. Vivo and S. Marco, SAFESPOT Final Report, Technical Report, D8.1.1, 2010, Available online..

[2] T. Eiter, J.X. Parreira and P. Schneider, Spatial Ontology-Mediated Query Answering over Mobility Streams, in: *Proc. of ESWC 2017*, 2017, pp. 219–237.

[3] D. Calvanese, G.D. Giacomo, D. Lembo, M. Lenzerini and R. Rosati, Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family, *J. Autom. Reasoning* **39**(3) (2007), 385–429.

[4] R. Kontchakov, M. Rodriguez-Muro and M. Zakharyaschev, Ontology-Based Data Access with Databases: A Short Course, in: *Reasoning Web. Semantic Technologies for Intelligent Data Access - 9th International Summer School 2013, Mannheim, Germany, July 30 - August 2, 2013. Proceedings*, 2013, pp. 194–229.

[5] T. Eiter, J.X. Parreira and P. Schneider, Detecting Mobility Patterns using Spatial Query Answering over Streams, in: *Proc. of Stream Reasoning Workshop 2017*, 2017.

[6] D.F. Barbieri, D. Braga, S. Ceri, E.D. Valle and M. Grossniklaus, C-SPARQL: a Continuous Query Language for RDF Data Streams, *Int. J. Semantic Computing* **4**(1) (2010), 3–25.

[7] D. Le-Phuoc, M. Dao-Tran, J.X. Parreira and M. Hauswirth, A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data, in: *ISWC 2011*, 2011, pp. 370–388.

[8] ETSI EN 302 637-2 (V1.3.2), Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service, Technical Report, ETSI, 2014.

[9] ETSI EN 302 637-3 (V1.2.2), Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service, Technical Report, ETSI, 2014.

[10] ETSI TS 103 191-3 (V1.1.1), Intelligent Transport Systems (ITS); Testing; Conformance test specifications for Signal Phase And Timing (SPAT) and Map (MAP); Part 3: Abstract Test Suite (ATS) and Protocol Implementation eXtra Information for Testing (PIXIT), Technical Report, ETSI, 2015.

[11] ETSI TR 102 863 (V1.1.1), Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Local Dynamic Map (LDM); Rationale for and Guidance on Standardization, Technical Report, ETSI, 2011.

[12] ETSI EN 302 895 (V1.1.0), Intelligent transport systems - Extension of map database specifications for Local Dynamic Map for applications of Cooperative ITS, Technical Report, ETSI, 2014.

[13] B. Netten, L. Kester, H. Wedemeijer, I. Passchier and B. Driessen, DynaMap: A Dynamic Map for road side ITS stations, in: *Proc. of ITS World Congress 2013*, 2013.

[14] H. Shimada, A. Yamaguchi, H. Takada and K. Sato, Implementation and Evaluation of Local Dynamic Map in Safety Driving Systems, *J. Transportation Technologies* **5**(2) (2015), 102–112.

[15] L. Zhao, R. Ichise, Z. Liu, S. Mita and Y. Sasaki, Ontology-Based Driving Decision Making: A Feasibility Study at Uncontrolled Intersections, *IEICE Trans.* **100-D**(7) (2017), 1425–1439.

[16] M. Stonebraker, U. Çetintemel and S.B. Zdonik, The 8 requirements of real-time stream processing, *SIGMOD Record* **34**(4) (2005), 42–47.

[17] D. Dell'Aglio, E.D. Valle, F. van Harmelen and A. Bernstein, Stream reasoning: A survey and outlook, *Data Science, IOS Press* **1**(1–2) (2017), 59–83.

[18] J.F. Allen, Maintaining Knowledge about Temporal Intervals, *Com. ACM* **26**(11) (1983), 832–843.

[19] D.A. Randell, Z. Cui and A.G. Cohn, A Spatial Logic based on Regions and Connection, in: *KR*, 1992, pp. 165–176.

[20] R.H. Güting, Geo-Relational Algebra: A Model and Query Language for Geometric Database Systems, in: *EDBT 1988*, 1988, pp. 506–527.

[21] R. Kontchakov and M. Zakharyaschev, An Introduction to Description Logics and Query Rewriting, in: *Reasoning Web. Reasoning on the Web in the Big Data Era - 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings*, 2014, pp. 195–244.

[22] T. Eiter, G. Ianni and T. Krennwallner, Answer Set Programming: A Primer, in: *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures*, 2009, pp. 40–110.

[23] A. Bognerand, B. Littig and W. Menz, *Das Experteninterview*, VS Verlag für Sozialwissenschaften, 2009. ISBN 978-3-531-16259-1.

[24] A. Rauch, F. Klanner, R.H. Rasshofer and K. Dietmayer, Car2X-based perception in a high-level fusion architecture for cooperative perception systems, in: *2012 IEEE Intelligent Vehicles Symposium, IV 2012, Alcal de Henares, Madrid, Spain, June 3-7, 2012*, 2012, pp. 270–275.

[25] G. Welch and G. Bishop, An Introduction to the Kalman Filter, Technical Report, 1995.

[26] D. Comer, Ubiquitous B-Tree, *ACM Comput. Surv.* **11**(2) (1979), 121–137.

[27] R.H. Güting and M. Schneider, *Moving Objects Databases*, Morgan Kaufmann, 2005. ISBN 0-12-088799-1.

[28] A. Haller, K. Janowicz, S.J.D. Cox, M. Lefrançois, K. Taylor, D.L. Phuoc, J. Lieberman, R. García-Castro, R. Atkinson and C. Stadler, The modular SSN ontology: A joint W3C and OGC standard specifying the semantics of sensors, observations, sampling, and actuation, *Semantic Web* **10**(1) (2019), 9–32.

[29] I.F. Ilyas, W.G. Aref and A.K. Elmagarmid, Supporting top-k join queries in relational databases, *The VLDB Journal* **13**(3) (2004), 207–221.

[30] S. Brandt, E.G. Kalayci, R. Kontchakov, V. Ryzhikov, G. Xiao and M. Zakharyaschev, Ontology-Based Data Access with a Horn Fragment of Metric Temporal Logic, in: *Proc. of AAAI 2017*, 2017, pp. 1070–1076.

[31] T. Eiter, T. Krennwallner and P. Schneider, Lightweight Spatial Conjunctive Query Answering Using Keywords, in: *Proc. of ESWC 2013*, 2013, pp. 243–258.

[32] A. Arasu, S. Babu and J. Widom, The CQL continuous query language: semantic foundations and query execution, *VLDB J.* **15**(2) (2006), 121–142.

[33] D. Calvanese, E. Kharlamov, W. Nutt and C. Thorne, Aggregate queries over ontologies, in: *Proc. of ONISW 2008*, 2008, pp. 97–104.

[34] D. Maier, *The Theory of Relational Databases*, Computer Science Press, 1983.

[35] A. Dermaku, T. Ganzow, G. Gottlob, B.J. McMahan, N. Musliu and M. Samer, Heuristic Methods for Hypertree Decomposition, in: *Proc. of MICAI 2008: Advances in Artificial Intelligence*, 2008, pp. 1–11.

[36] M. Stocker and M. Smith, Owlgres: A Scalable OWL Reasoner, in: *Proc. of OWLED 2008*, 2008.

[37] M. Rodriguez-Muro, R. Kontchakov and M. Zakharyaschev, Ontology-Based Data Access: Ontop of Databases, in: *Proc. of ISWC 2013*, 2013, pp. 558–573.

[38] A. Arasu, S. Babu and J. Widom, The CQL continuous query language: semantic foundations and query execution, *VLDB J.* **15**(2) (2006), 121–142.

[39] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul and S. Zdonik, Monitoring streams: a new class of data management applications, in: *Proc. of VLDB 2002*, 2002, pp. 215–226.

[40] S. Madden, M. Shah, J.M. Hellerstein and V. Raman, Continuously adaptive continuous queries over streams, in: *2002 ACM SIGMOD International Conference on Management of Data*, 2002, pp. 49–60.

[41] A. Bolles, M. Grawunder and J. Jacobi, Streaming SPARQL - Extending SPARQL to Process Data Streams, in: *Proc. of ESWC 2008*, 2008, pp. 448–462.

[42] J. Calbimonte, J. Mora and Ó. Corcho, Query Rewriting in RDF Stream Processing, in: *Proc. of ESWC 2016*, 2016, pp. 486–502.

[43] D. Anicic, P. Fodor, S. Rudolph and N. Stojanovic, EP-SPARQL: a unified language for event processing and stream reasoning, in: *Proc. of WWW 2011*, 2011, pp. 635–644.

[44] H. Beck, M. Dao-Tran, T. Eiter and M. Fink, LARS: A Logic-Based Framework for Analyzing Reasoning over Streams, in: *Proc. of AAAI 2015*, 2015, pp. 1431–1438.

[45] M. Perry, P. Jain and A.P. Sheth, SPARQL-ST: Extending SPARQL to Support Spatiotemporal Queries, *Geospatial Semantics and the Semantic Web* **12** (2011), 61–86.

[46] M. Koubarakis and K. Kyzirakos, Modeling and Querying Metadata in the Semantic Sensor Web: The Model stRDF and the Query Language stSPARQL, in: *ESWC 2010*, 2010, pp. 425–439.

[47] H.N.M. Quoc and D. Le Phuoc, An Elastic and Scalable Spatiotemporal Query Processing for Linked Sensor Data, in: *Proc. of SEMANTICS 2015*, ACM, 2015, pp. 17–24.

[48] Ö.L. Özçep, R. Möller and C. Neuenstadt, Stream-Query Compilation with Ontologies, in: *Proc. of AI 2015*, 2015, pp. 457–463.

[49] A. Artale, R. Kontchakov, A. Kovtunova, V. Ryzhikov, F. Wolter and M. Zakharyaschev, First-Order Rewritability of Temporal Ontology-Mediated Queries, in: *Proc. of IJCAI 2015*, 2015, pp. 2706–2712.

[50] S. Borgwardt, M. Lippmann and V. Thost, Temporalizing rewritable query languages over knowledge bases, *J. Web Sem.* **33** (2015), 50–70.

[51] S. Klarman and T. Meyer, Querying Temporal Databases via OWL 2 QL, in: *Proc. of RR 2014*, 2014, pp. 92–107.

[52] A. Margara, J. Urbani, F. van Harmelen and H.E. Bal, Streaming the Web: Reasoning over dynamic data, *J. Web Semant.* **25** (2014), 24–44.

[53] D. Anicic, S. Rudolph, P. Fodor and N. Stojanovic, Stream reasoning and complex event processing in ETALIS, *Semantic Web* **3**(4) (2012), 397–407.

[54] M. Rinne and E. Nuutila, Constructing Event Processing Systems of Layered and Heterogeneous Events with SPARQL, in: *On the Move to Meaningful Internet Systems: OTM 2014 Conferences - Confederated International Conferences: CoopIS, and ODBASE 2014, Amantea, Italy, October 27-31, 2014, Proceedings*, 2014, pp. 682–699.

[55] C.L. Forgy, Rete: A fast algorithm for the many pattern/many object pattern match problem, *Artificial Intelligence* **19**(1) (1982), 17–37.

[56] E. Friedman-Hill, *Jess in Action: Rule-Based Systems in Java*, Manning Publications, 2003. ISBN 978-1-930-11089-2.

[57] J.-P. Calbimonte, H. Jeung, Ó. Corcho and K. Aberer, Enabling Query Technologies for the Semantic Sensor Web, *Int. J. Semantic Web Inf. Syst.* **8**(1) (2012), 43–63.

[58] M. Gebser, R. Kaminski, B. Kaufmann and T. Schaub, Clingo = ASP + Control: Preliminary Report, *CoRR* **1405.3694** (2014).

[59] H. Beck, M. Dao-Tran and T. Eiter, LARS: A Logic-based framework for Analytic Reasoning over Streams, *Artif. Intell.* **261** (2018), 16–70.

[60] H. Beck, T. Eiter and C. Folie, Ticker: A system for incremental ASP-based stream reasoning, *TPLP* **17**(5–6) (2017), 744–763.

[61] H.R. Bazoobandi, H. Beck and J. Urbani, Expressive Stream Reasoning with Laser, in: *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I*, 2017, pp. 87–103.

[62] Ö.L. Özçep, R. Möller and C. Neuenstadt, A Stream-Temporal Query Language for Ontology Based Data Access, in: *KI 2014: Advances in Artificial Intelligence - 37th Annual German Conference on AI, Stuttgart, Germany, September 22-26, 2014. Proceedings*, 2014, pp. 183–194.

[63] S. Brandt, E.G. Kalayci, R. Kontchakov, V. Ryzhikov, G. Xiao and M. Zakharyaschev, Ontology-Based Data Access with a Horn Fragment of Metric Temporal Logic, in: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, 2017, pp. 1070–1076.

[64] S. Brandt, E.G. Kalayci, V. Ryzhikov, G. Xiao and M. Zakharyaschev, Querying Log Data with Metric Temporal Logic, *J. Artif. Intell. Res.* **62** (2018), 829–877.

[65] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu and J. Banerjee, RDFox: A Highly-Scalable RDF Store, in: *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*, 2015, pp. 3–20.

[66] Y. Ren and J.Z. Pan, Optimising ontology stream reasoning with truth maintenance system, in: *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, 2011, pp. 831–836.

[67] H.R. Bazoobandi, H. Beck and J. Urbani, Expressive Stream Reasoning with Laser, in: *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I*, 2017, pp. 87–103.