International Journal of Intelligent Transport System Research manuscript No.
(will be inserted by the editor)

Towards a Semantically Enriched Local Dynamic Map

Thomas Eiter · Herbert Fuereder · Fritz Kasslatter · Josiane Xavier Parreira · Patrik Schneider

Received: date / Accepted: date

Abstract With the increasing availability of Cooperative Intelligent Transport Systems, the *Local Dynamic Map* (LDM) is becoming a key technology for integrating static, temporary, and dynamic information in a geographical context. However, existing ideas do not leverage the full potential of the LDM approach, as an LDM contains streaming data and varying *implicit* information which are not captured by current models. We aim to provide a *semantically enriched* LDM that applies Semantic Web technologies, in particular ontologies, in combination with spatial stream databases. This allows us to define an enhanced world model, to derive model properties, to infer new information, and to offer expressive query capabilities over streams. We introduce our envisioned architecture which includes an LDM ontology, an integration and annotation framework, and a stream query answering component. We also sketch three application scenarios that illustrate the usability and benefits of our approach, thus we provide an in-depth validation of the scenarios in an experimental prototype.

1 Introduction

For Cooperative Intelligent Transport Systems (ITS), the integration of static, temporary, and dynamic information in a geographical context is a crucial feature for the understanding and processing of complex traffic scenes. Different ITS systems collect (sensor) data and exchange it by vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), or combined (V2X) communication, which naturally contains temporal data (e.g., traffic light signal phases) and geospatial data (e.g., GPS location) of traffic participants. Different types of messages are broadcast at most every 100 ms by traffic participants and roadside C-ITS stations to inform other participants about the current local state, e.g. about position and speed. The main types of V2X messages are:

- *Cooperative Awareness Messages* (CAM) provide high frequency status updates of a vehicle's position, speed, vehicle type, etc.;
- *Map Data Messages* (MAP) describe the detailed topology of an intersection, including its lanes and their connections;
- *Signal Phase and Timing Messages* (SPaT) give the projected signal phases (e.g., green) for each lane; and
- *Decentralized Environmental Notification Messages* (DENM) inform whether specific events like road works occur in a designated area.

Since V2X communication is a key technology to enable autonomous driving, many projects regarding this technology have been carried out. The SAFESPOT [1] and CVIS [29] projects are of particular interest, since they are motivated by improving road safety. In these projects, the concept of the *Local Dynamic Map (LDM)* was introduced, which acts as an integration platform to combine static digital maps, also called geographic information system (GIS) maps, with dynamic environmental objects representing, e.g., vehicles or pedestrians. As shown in Figure 1, the LDM consists of the following four layers:

(1) *Permanent Static:* the first layer contains static information obtained from GIS maps that include roads, intersections, and points-of-interest (POIs);
(2) *Transient Static:* the second layer extends the static map by traffic attributes, roadside ITS stations, landmarks, and intersection features such as more detailed topological data that include lanes and their connections;

T. Eiter, and P. Schneider
Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9–11, A-1040 Vienna, Austria
E-mail: {eiter,patrik}@kr.tuwien.ac.at

H. Fuereder, F. Kasslatter, J. X. Parreira, and P. Schneider
Siemens AG, Austria
Siemensstrasse 90, 1210 Vienna, Austria
E-mail: first.last@siemens.com

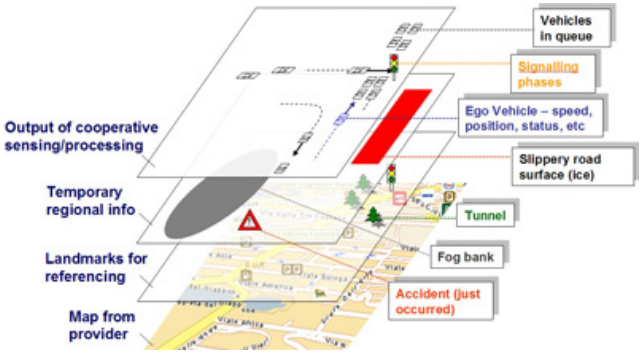


Fig. 1: The four layers of an LDM [1]

(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 fast-changing information, mainly V2X messages of road users including their GPS position, heading, and speed.

We recognize that the LDM is a key technology for data integration in cooperative ITS systems, which is indicated by its initial standardization as ETSI [24, 23] and ISO ([26, 27]) technical recommendations. Influenced by the ongoing standardization efforts, there is a common understanding that the LDM should include a high-level API, a GIS database (DB), and have SQL as a query language. Thus, the LDM is a conceptual data store in an ITS station, which integrates GIS maps, sensor data, and V2X messages.

To realize the LDM, the authors of [1] and [33] suggest an object-oriented schema, called world model, a topology of geospatial objects, and an object associator. The object associator connects different objects like vehicles and roadside units to the world model. Yet, the existing ideas do not address the "streaming" nature of the data in combination with the complex world model, which would allow to leverage the full potential of the LDM approach. The LDM can become a powerful integration tool for sensor data and V2X messages in general, allowing complex scene recognition by utilizing implicit information in the streaming data. Such an advanced integration can be used to provide new or enhanced functionality for ITS applications. For instance, by combining the lane direction, the trajectory, and the role of a vehicle (e.g., ambulances) a wrong-way driver can be detected by querying the stream of V2X messages. By implicit information, we mean all the data that is not directly represented either by V2X messages or by static information from the GIS map. We identified the following list of important implicit information in an LDM:

– *Part-Whole* relations, e.g., a traffic light is part of an intersection;
– *Spatial* relations, e.g., a car is crossing a stop lane that is within an intersection;
– *Connectivity*, e.g., one intersection is connected to another intersections via a road;
– *Functionality*, e.g., if two objects have the same identifier (ID), they are the same.

In this paper, we aim to provide a semantically enriched LDM that applies Semantic Web technologies to the standard LDM approach, which include ontologies (an enhanced world model), spatial-stream databases, the related stream processing, and ontology-based data access (OBDA) [39, 14] for connecting the ontology with the database. Essentially, OBDA is the technique for accessing databases through the ontology by queries (typcially, conjunctive queries). Although adding another layer increases complexity, but we gain the following advantages from a semantically enriched LDM:

– *World Model:* our notion of a world model is captured by an LDM ontology, which is based on the W3C standard OWL [30] and simply modifiable and extendable. Extensions can be made without altering the database and its relational schema.
– *Model Properties:* the formal models of an ontology and the data have defined properties, which can be used for verification, simplification, and optimization on the conceptual level. For instance, by defining constraints in the ontology, inconsistencies in the data can be found; e.g., by stating disjointness of the concepts car and bicycle, that a an object can not be both a car and a bicycle.
– *Inference:* OBDA allows us to infer new information at query time (e.g., class hierarchies), which reveals implicit information and keeps the amount of stored data small.
– *Expressive Queries:* the queries are posed through the ontology extending the vocabulary beyond database relations. The query language of conjunctive queries[1] (CQ) is simple and yet powerful. Furthermore, by examining the structure of the ontology, we can obtain meaningful combinations of query atoms, which aid in building and validating user queries.

By semantically enriching the LDM, we highlight the following contributions and related challenges, which we aim to address in this paper:

– *Modeling:* besides the mentioned ETSI/ISO standards, mobility vocabularies are defined in Schema.org and Mobivoc.[2] Yet, there are no comprehensive ontologies available that allow to capture an LDM and related V2X messages. The latter are standardized and thoroughly specified, but based on a different (modeling) language, namely the Abstract Syntax Notation One (ASN.1). The specifications of V2X messages in ASN.1 are tree-like

[1] CQs are a lean representation of SQL select-project-join queries: $q(x) = \texttt{MAPLane}(x) \wedge \texttt{isIngress}(x, T)$ is rewritten into SELECT a.x FROM MAPLane AS a, isIngress AS b WHERE a.x = b.x AND b.y = T
[2] http://schema.org/ and http://www.mobivoc.org/

and must be converted into triples of an RDF-graph, as needed by the ontology standards in the Semantic Web.
- *Integration and Annotation:* after the conversions to triples is completed, the V2X messages and the GIS database have to be mapped to the given vocabulary of the ontology. Due to the tree-like structure of the messages, not all relations between objects are available, hence we need to calculate the missing relations, e.g., spatial relations. The integration and annotation steps have to handle static and streaming data in a uniform way and should be easily extendable and maintainable.
- *Stream query answering:* the combination of the methods and respective techniques for query answering (QA) over streams are challenging regarding performance and scalability. With OBDA, there is a trend to lightweight query answering over ontologies; we thus can benefit from recent results which improve performance and scalability (cf. [39, 14]). On the other hand, for query evaluation on stream database systems such as PipelineDB[3], most implementations are designed towards efficiency, but not for complex query evaluation using ontologies and/or geospatial data; both aspects add complexity and diminish scalability.

The remainder of the paper is organized as follows. Section 2 describes the state-of-the-art of the LDM approach. In Section 3 we introduce Semantic Web and stream-processing technologies. Section 4 presents our envisioned architecture to illustrate how Semantic Web technologies can be used for the LDM. Section 5 describes the stream query answering component, which is a central component of the architecture. In Section 6 we present three application scenarios to show the benefits of a semantically enriched LDM, which is evaluated in Section 7. Section 8 concludes with possible future works and refinements.

2 State-of-the-art of the LDM

In this section, we have a closer look at state-of-the-art efforts regarding the LDM approach.

SAFESPOT Project. The SAFESPOT project initiated the term and definition of the LDM in work package D 7.3.1 [1]. The authors recognized that the data model "has a hierarchical structure using associations between classes to describe their relationships". However, they dropped an object-orient model in favor of a relational model tailored to a Relational Database Management System (RDBMS) due to performance concerns. The authors also suggested two implementations based on commercial geospatial RDBMS: *PG-LDM* and *NAVTEQ-LDM*. PG-LDM is developed by Tele Atlas and built on top of PostGIS.[4] This is a natural choice, as the first and second layer contain largely static GIS maps. NAVTEQ-LDM is built by Navteq[5] and uses SQLite[6] as its geospatial RDBMS. It is thus well-suited for the first and second LDM layer and already targets the deployment on mobile devices. In SAFESPOT, also a relational database schema was introduced, which represents the four layers with different groups of tables that include static features, moving objects, conceptual objects, and relationships. Finally, it defined an API that supports custom functions such as to access all lanes of a road element (called *getLanesForRoadElement*) and a direct interface to submit SQL queries.

ETSI/ISO Standards. The initial standard was by the ETSI TR 102 863 (V1.1.1) report [24], where an LDM was defined as "a conceptual data store located within an ITS station ... containing information that is relevant to the safe and successful operation of ITS applications." The report locates the LDM in the facilities layer of the ITS station reference architecture and connects the four layers with possible ITS applications. For example, speed limitation is defined in the third layer and can be used for co-operative speed management. The report also recognizes that the LDM architecture is made of a management and a data store that can be accessed through an API with three interfaces, called *AF-SAP* for applications, *NF-SAP* for networking, and *SF-SAP* for security. It also addresses the topic of how the LDM can be linked to the road network of a static GIS map (the first layer), called *dynamic location referencing*. In the ETSI EN 302 895 (V1.1.0) final draft [23], the work of the previous report was extended with new functionalities, introducing LDM Data Objects, which are compositional data structures, and LDM Data Providers/Customers. Also a new interface for LDM services and maintenance was defined. Via the interface Data Objects can be fetched with SQL-like filtering and selection statements. We view a semantically enriched LDM as an extension of LDM Data Objects and Providers. With a more international focus, the ISO/TS 17931:2013 [26] and ISO/TS 18750:2015 [27] reports defined comparable standards to ETSI, which include an LDM architecture, data models, and an embedding into the ITS architecture.

Recent Research. Netten et al. [33] introduced *DynaMap*, which is an extension to the LDM architecture. They recognized that previous work on the LDM was car-centric and thus focussed on roadside ITS stations. Netten et al. defined a novel architecture that includes data sources, a world model, world objects, and data sinks. World objects are created by the world object associator based on the streamed input from the different data sources which includes V2X messages and sensor readings. The world model resembles an ontology and defines the relationships between all the objects including their hierarchical relations and a running history of data items.

[3] https://www.pipelinedb.com/
[4] http://postgis.net/
[5] now part of https://here.com/
[6] https://www.sqlite.org/

They also recognize that each object has a reference position that connects it to a spatial topology.

Koenders et al. [28] developed an "open Local Dynamic Map", where they point out that the LDM cannot store all objects and their data items permanently. Hence, they introduced a "simple" streamed filtering technique, by deleting the objects that are too far from the ITS station. They designed their own relational schema having tables for areas, objects, and roads including a spatial topology. They also provided additional functions to the LDM, which include map-matching and a security layer. Shimada et al. [41] implemented the initial (RDBMS-centric) approach by SAFESPOT and evaluated it in a complex collision detection application scenario. For the evaluation, a traffic simulation tool was used to generate V2X messages for different numbers of vehicles. The authors also recognized that GIS maps and tools can be open-source and extracted the road graph from OpenStreetMap (OSM).

Ulbrich et al. [44] introduced a graph-based context representation of the static and dynamic environment used by an ego vehicle. For the context representation, they developed an ontology that includes classes for actions, traffic objects, and situations. The ontology is part of an "overall" context model that includes (a) a geometric layer, (b) a topological layer, and (c) a semantic layer, which are all linked to each other. Besides the context model, the authors described an approach for information aggregation in order to enrich the model. Furthermore, they provided a quantitative evaluation of the approach in the context of their *Stadtpilot* project.

Zoghby et al. [46], investigated an approach to improve the environmental perception of vehicles by cooperative perception. They extended the (local) LDM of each vehicle with an extension called *Dynamic Public Map (DPM)* that is based on Dynamic Distributed Maps (DDM), which are exchanged between the vehicles. They provided an algorithm for calculating the DPM using (i) a discounting step with confidence evaluation, (ii) a prediction step using spatial and temporal alignment, and (iii) a fusion step using an existing association algorithm. An extensive experimental evaluation showed that the DPM improves the detection of surrouding vehicles and their classfication.

Current Shortcomings. The above efforts are already mature and allow an elaborate usage of the LDM. However, they are of limited use in a complex, fast-changing environment with vehicles that update information at high frequency. We believe the following shortcomings are still present:

- *Database-centric:* besides DynaMap, all other efforts have a database-centric model of the LDM using a static schema, where the LDM objects are directly mapped to relational tables. New types of objects need a modification of the schema, which makes it harder to add new domains, e.g., traffic regulations. Furthermore, the database schema cannot simply capture and query class hierarchies and the dependencies between the different objects as it is not graph-based.
- *Stream processing:* except DynaMap, other efforts are designed to work on top of a GIS database (e.g., PostGIS) and neglect the streaming nature of the LDM data, since it should allow for real-time queries over large amounts of data "in-stream", i.e., without storing. Stream processing needs a clear data model (e.g., a point-based model) and a defined query language that supports window operators (e.g., having sliding windows), stream joins, and aggregation over streams (e.g., average speed over 30 s). These features are either entirely missing or only considered in external components.
- *Sound integration:* the integration of all layers and the model of a complex intersection could lead to incomplete or inconsistent data. A MAP message allows integrity constrains to check for wrongly connected lanes. However they are only implicit in the database definitions and do not cover for all possible integrity cases (e.g., disjointness between classes).

As already mentioned, DynaMap and Stadtpilot address to some extent the above shortcomings, but differs from our work in three points: (a) our world model is based on a standard language using ontologies, thus existing approaches and algorithms can be applied directly; (b) their streaming support is based on "monitors" or "vector of points" that do not provide the power of a full query language, hence, this makes a flexible processing, optimization, and integration harder; and (c) checking inconsistencies is not an aim of both projects. Thus, our emphasis is more on query answering over streams with ontologies, whereas DynaMap and Stadtpilot have an object-oriented data model and use custom processing techniques such as a point-in-lane-segment algorithm.

3 Background Technology and Methods

In this section, we give a brief introduction to the methods and technologies that we envision to use for achieving a semantically enriched LDM.

Semantic Web Technologies. Semantic Web technologies provide a common framework for sharing and reusing data across boundaries. We refer to the seminal article of Berners-Lee et al. [9] for an outline of the ideas and architectural overview to the Semantic Web stack[7] The Resource Description Framework (RDF) [40, 12] serves as a flat, graph-based unified data model which is based on URIs as identifiers for objects and relations. An RDF graph is represented by triples $\langle \mathsf{S}, \mathsf{P}, \mathsf{O} \rangle$ of a subject S, a predicate P, and an object O. Ontologies are used for modeling knowledge domains,

[7] A recent version is at https://commons.wikimedia.org/wiki/File:Semantic_web_stack.svg

by expressing relations between terms with a restricted vocabulary and by modeling them as class hierarchies. In the Semantic Web context, OWL [30] plays a central role as the standard modeling language of ontologies with its (formal) logical underpinning of Description Logics (DL) [6]. The vocabulary of a DL consists of objects (called individuals), classes (called concepts), and properties (called roles). Furthermore, a knowledge base (KB) consists of a terminological box (TBox), which contains axioms about relations between classes and properties, and instance data in an assertion box (ABox), which contains factual knowledge about individuals by the following assertions represented as N-triples (cf. [6]):

- Class assertions ⟨id1 type Class⟩:
  states object id1 is of type Class, e.g., Vehicle;
- Object property assertions ⟨id1 *property1 id2*⟩:
  states object id1 is related by property1 to object id2, e.g., ⟨lane1 isPartOf intersection2⟩;
- Data property assertions ⟨id1 *property2* value⟩:
  states object id1 has a property2 with a numeric value, e.g., ⟨car1 hasSpeed 20⟩.

In the light of data-intensive applications, there is a trend to move from the expressive OWL 2 language towards more scalable and tractable fragments called OWL 2 Profiles [32]. These research efforts have been focused on efficient query answering techniques over lightweight ontology languages, such as OWL2 QL [14] and OWL2 EL [5]. Conjunctive query evaluation over OWL2 QL ontologies can be delegated by SQL query rewriting to a RDBMS, which facilitates scalable query processing. Ontologies modeled with OWL2 QL are well-suited for defining the conceptual level. The Ontop system [39] is an example for an ontology-based data access (OBDA) system, where a global schema is defined as an OWL2 QL ontology, and the source schemas are mapped to the global schema by SQL queries. OBDA has been extended to non-standard data formats such as GIS [19,8], temporal data [4,10], and streamed data [13,34].

Stream Processing and Stream Reasoning.
Relational stream processing has been investigated for long. An important step was the Continuous Query Language (CQL) [3] with the design goals of a clear syntax based on SQL-99 and a multi-set (bag) semantics. The authors defined streams and relations as data types and use the following operators that map between them: (1) stream-to-relation, (2) relation-to-relation, and (3) relation-to-stream. For (1), they introduced various window operators n order to select tuples rom a stream.Among them are time-based sliding, tuple-based sliding, and partitioned window operators.

(2) is taken directly from the relational database setting; for (3), they defined insert stream, delete stream, and relation stream operators. Based on the operators, they developed a method for query execution and benchmarked a prototype implementation of a highway tolling application scenario in order to show the applicability of their approach.

Stream reasoning studies how to introduce reasoning processes into scenarios that involve streams of continuously produced information. In that, domain models provide background knowledge for the reasoning and lift streams to a "semantic" level. Particular aspects of stream reasoning are incremental and repeated evaluation, either push-based, i.e. on data arrival, or pull-based at given points in time, and using data snapshots (called windows) to reduce the data volume. Windows can be obtained by selecting data based on temporal conditions (called time-based windows) or data counts. Besides the seminal CQL, many formalisms and languages for stream reasoning exist. Among them are

(1) extensions of the SPARQL web query language, e.g., Morph-streams [13], and CQELS [35];
(2) extensions of ontology languages to streams e.g. by Ren and Pan [38], STARQL [34], and Cayuga [38]; and
(3) rule-based formalisms e.g., ETALIS [2], Reactive ASP [25], Teymourian et al. [43], LARS [7], metric Datalog [11], and Streamlog [45].

LARS [7] provides a rule-based formalism with generic windows as first class citizens. Windows can be nested, and modal operators allow to reason about points in a window. LARS provides a monotonic and an answer set programming (ASP) semantics that generalizes the standard ASP semantics. It offers nondeterminism and nonmonotonicity to deal with missing information. The ETALIS system [2], which was applied to traffic management, offers a rule-based language to express complex event patterns and a background KB. The patterns are expressed with predicates like $during(event1,5)$ which in combination with the KB and causal parts can be used for query answering. The standard Prolog query evaluation is altered to an event-driven backward chaining (EDBC) of rules. A Prolog system is then triggered to evaluate a query and the EDBC rules when new data is passing by. The STARQL framework [34] is an effort for streamifying ontology-based data access (OBDA) by introducing an extensible query language and uses temporal reasoning over sequences of ABoxes. The framework extends the first-order query rewriting of DL-Lite with Intra-ABox reasoning.

4 Architecture of a Semantically Enriched LDM

Our envisioned architecture is shown in Figure 2a and illustrates how Semantic Web technologies can be used to enrich the LDM. In this paper, we focus on the main methods and related techniques the architecture and leave parts of the technical analysis to (existing) in-depth work (see [21] and [22]). Also, we also show how the architecture is implemented.

We apply and extend two different methods for the semantic enrichment: first, rule-based methods are used to

(semantically) complete the LDM by materializing newly derived data. This is applied to slower changing data of the first, second, and third layers of the LDM, but is is also needed for more complex tasks like consistency checking and diagnosis. Second, OBDA-based methods are applied to (semantically) complete queries on demand by encoding axioms (e.g., subclass axioms) of the ontologies into the query, called query rewriting. This approach is suitable for large amounts of fast changing data that occur in the fourth layer. The queries can be used for data aggregation and (complex) event detection (details are shown in Section 5). The starting points for the methods are the works on rule-based spatial data extraction [20], spatial query answering (QA) [19], and spatial-stream QA [22]. Next we describe the LDM ontology, as well as the integration, annotation, and linking framework.

LDM Ontology. With the support of ITS domain experts, we have modeled an ontology to capture the four levels of the LDM architecture as well as elements of a traffic scene.[8] The LDM ontology is represented by the W3C standard OWL2 QL [14] for OBDA and partially rendered in Figure 2b. Besides the restriction to OWL2 QL, our methods are ontology-agnostic, hence other ontologies could be used as well. We follow a layered approach starting at the bottom with a simple separation between the following classes:

- V2XFeature is the spatial representation of V2X objects;
- GeoFeature represents the GIS aspects of the LDM including POIs and the road networks;
- Geometry is the geometrical representation of features;
- Actor is an individual actor involved in a transport scene;
- Event describes events that happen in a transport scene;
- CategoricalValues such as allowed maneuvers or vehicle roles (e.g., emergency vehicle).

We also introduce the following properties:

- properties for partonomies, e.g., isPartOf;
- spatial relations, e.g., intersects;
- connectivity, e.g., connected; and
- standard properties, e.g., isAllowed, hasRole, isManaged, or positions.

The sub-classes of GeoFeature are linked to the GeoOWL and GeoNames ontologies for embedding it into existing work.[9] V2XFeature is the domain specific modeling of the MAP topology and describes the details of an intersection including its lanes, allowed maneuvers, and traffic lights. The Actor class includes persons, vehicles, as well as roadside ITS stations. Objects of the Actor class have their autonomous behavior and are the main generator of streamed data.

[8] Available at http://www.kr.tuwien.ac.at/research/projects/loctrafflog/LocalDynamicMapITS-v0.1-Lite.owl

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

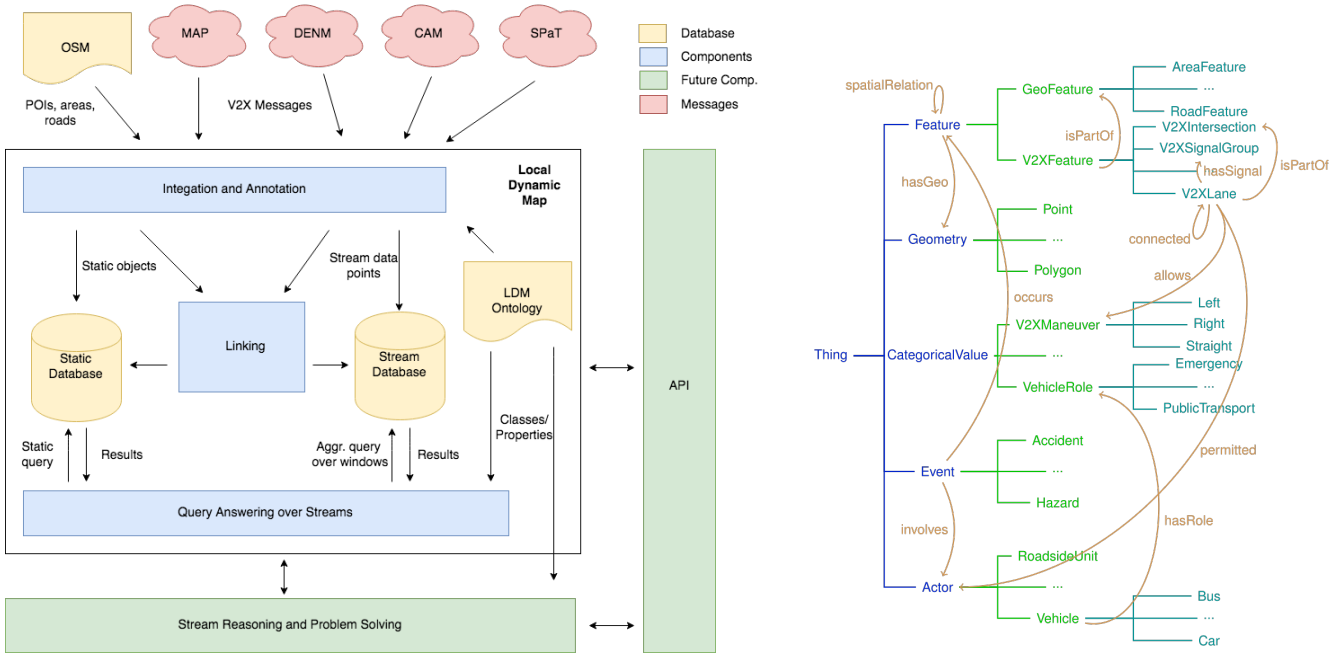
Static and Stream Databases. In an LDM, we have to deal with spatial-relational data that never or infrequently change (the first and second layer), and the streaming data (third and fourth layer) that changes at higher frequency (at most 100 ms). Besides the TBox T that consists of the ontology axioms, we distinguish between a (standard) static ABox A that is a static database, a stream database F, a static spatial database S_A, and a spatial database with stream support S_F. These sources can be combined in different ways as shown below, where angled brackets show that the sources are the same physical and logical entity:

- (i) $\langle A, F, S_A, S_F \rangle$ as a "universal" database, which allows for streamed and spatial data;
- (ii) $A, F, \langle S_A, S_F \rangle$ as a normal database for A, a stream database for F, and a spatial database which allows for time stamps;
- (iii) $A, S_A, \langle F, S_F \rangle$ as a normal database for A, a spatial database for S_A, and a stream database with limited support for spatial data;
- (iv) A, F, S_A, S_F as separate databases that are only connected by query atoms at query time.

Our work builds on combination (iii), therefore we enhance a stream database (in our case PipelineDB) with the support for spatial data. The spatial data includes the static spatial objects of the GIS maps such as POIs (e.g., petrol stations). Furthermore, V2X objects like intersection topologies are also kept in the spatial database. After integration and annotation of a V2x message, most of the objects have a geometrical representation, i.e., polygons or points. Objects of the first layer are initially added from OSM instances (e.g., Vienna) and stored in the spatial database. Objects of the second layer might be predefined for a roadside unit or frequently updated by incoming MAP messages.

The third and fourth layer are represented by a stream database and include relations such as *streams* and *continuous views* (named as in PipelineDB). The database is modeled such that there are one-to-one mappings from stream relations to continuous views, and further to the TBox classes and properties. For instance, vehicle positions are fed into the relation *stream_pos* that is accessed by *view_pos*, which then is mapped to the property positions. The stream relations keep the raw message data, which are only accessible via the continuous views over the streams.

Integration and Annotation Framework. The integration framework represents the initial layer that is responsible for (continuously) receiving the V2X messages. The raw message data is extracted and added either to normal relations (i.e., relational tables) of the static database or the stream relations (called *streams*) of the stream database, which are managed by PipelineDB. Note that in PipelineDB streams are write-only, so the query component has to create (read-only) continuous views on-demand for querying the streams of data. Redundancy and duplication of received message data



(a) System architecture

(b) Ontology (partial rendering)

Fig. 2: LDM system architecture and ontology

is handeled in this layer. Since we have a one-to-one mapping between classes (e.g., vehicles) resp. properties (e.g., speed) and database relations, we split the message content into tuples and add them to the respective relations. For instance, we add the tuples $\langle$car1$\rangle$ resp. $\langle$car1 hasSpeed 50$\rangle$ to the unary table Vehicle resp. binary stream hasSpeed.

Furthermore, we align the data items to our system timeline using the time injected by the data sources (called application time). If the application time is missing, we add the (implicit) arrival time. In addition to the alignment, we erase duplicate data items in the working storage based on the unique identification by object ID, message ID, and application time. Duplicate data items might occur due to the multi-hop propagation of V2X messages. Note that we do not have distributed and redundant LDMs here, as we consider in our system design only single roadside ITS stations.

For the first three layers, we can take advantage of our work on an OSM data extraction tool, which combines Datalog rules with simple extract, transform, and load (ETL) features [20]. Each rule is a mapping from a source to target with an optional transformation step in between. The sources include geospatial data sources like OSM, while the target is designed to produce triples.

Example 1 The following rules create class and property triples for police stations in a city, where `GeoPoint` and `TagPolice` extract the OSM geometries (e.g, points) that are tagged as "police":[10]

$$\texttt{GeoPoint}(x) \wedge \texttt{TagPolice}(x) \rightarrow \texttt{PoliceStation}(x).$$
$$\texttt{GeoPoint}(x) \wedge \texttt{TagPolice}(x) \wedge \texttt{GeoPolygon}(y) \wedge \texttt{TagCity}(y) \wedge \texttt{Inside}(x,y) \rightarrow \texttt{hasPoliceStation}(y,x).$$

Linking Framework. The linking framework (LF) computes and stores links between objects that are not directly represented in the data. The linking could be computed offline or on-demand using similar Datalog rules as in the annotation framework. For the linking, spatial relations are the main focus, where we follow the standard approach called Dimensionally Extended Nine-Intersection Model (DE-9IM), which supports relations like *touches*, *intersects*, or *disjoint*. For instance, one important link is the adjacency of lanes.

Example 2 The following rule calculates which lanes are adjacent to other lanes:

$$\texttt{MAPLane}(x) \wedge \texttt{hasLocation}(x,u) \wedge \texttt{hasLocation}(y,v) \wedge \texttt{MAPLane}(y) \wedge \texttt{touches}(u,v) \rightarrow \texttt{adjacentLane}(x,y).$$

The atom `MAPLane` extracts the lane objects from the incoming MAP message and `hasLocation` returns the polygons of these objects, which are checked in `touches` if the polygons touch each other. Another important link is the connection of the MAP intersection object to its representation in OSM, which is a node in the road network. For this, the OSM representation has to be matched to the closest MAP intersection, thus anchoring the V2X features in the OSM road network.

Stream Reasoning and Problem Solving. Simple events can be detected by the stream QA component (details are given in Section 5), but complex events like multiple-vehicle

[10] $\texttt{PoliceStation}(x)$ and $\texttt{hasPoliceStation}(x,y)$ are legible renderings of the N-Triples $\langle x$ type PoliceStation$\rangle$ and $\langle x$ hasPoliceStation $y\rangle$

collisions, which are a chain of simple events satisfying specific temporal relations, are not expressible by simple queries. This requires a more powerful stream reasoning component, which could by enabled by a more expressive rule-based formalism such as Answer Set Programming (ASP) [18] that offers nondeterminism and nonmonotonicity. Adding a long-term memory for detected events (called observations), the stream reasoning component could also be geared towards *model-based diagnosis* [37,36], which would allow one to find the cause for complex traffic problems like traffic jams. This is a topic for future work.

API. The LDM is defined as a data integration platform that provides services to external applications. We aim to support different types of APIs on top of our approach. First we aim to support the standard API requirement by the ETSI TR 102 863 [24]. We assume that the SF-SAP is handled by the ITS station; thus it is not in the scope of our work. As described in the standard, the NF-SAP interface connects the LDM to the communication functions of the ITS station and receives the V2X messages, which are then forwarded to the integration layer.

5 Query Answering over Streams

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

Example 3 The following query should illustrate the component as it detects red-light violations on intersections by searching for vehicles y with speed above 30km/h on lanes x whose signals will turn red in 4s:

$$\begin{aligned}\mathsf{q}(\mathsf{x},\mathsf{y}) : \ & \mathsf{LaneIn}(\mathsf{x}) \wedge \mathsf{hasLocation}(\mathsf{x},\mathsf{u}) \wedge \mathsf{intersects}(\mathsf{u},\mathsf{r}) \wedge \\ & \mathsf{pos}[\mathsf{line},4\mathsf{s}](\mathsf{y},\mathsf{r}) \wedge \ \mathsf{Vehicle}(\mathsf{y}) \wedge \mathsf{speed}[\mathsf{avg},4\mathsf{s}](\mathsf{y},\mathsf{v}) \wedge \\ & (\mathsf{v} > 30) \wedge \ \mathsf{isManaged}(\mathsf{x},\mathsf{z}) \wedge \ \mathsf{SignalGroup}(\mathsf{z}) \wedge \\ & \mathsf{hasState}[\mathsf{first},-4\mathsf{s}](\mathsf{z},\mathsf{s}) \wedge (\mathsf{s} = \mathsf{Stop})\end{aligned}$$

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

- $\mathsf{Vehicle}(\mathsf{y})$ and $\mathsf{isManaged}(\mathsf{x},\mathsf{z})$ are ontology atoms, which have to be unfolded with respect to the ITS domain that is modelled in the LDM ontology;
- $\mathsf{intersects}(\mathsf{u},\mathsf{v})$ and $\mathsf{hasLocation}(\mathsf{x},\mathsf{u})$ are spatial atoms, where the first checks spatial intersection and the second the assignment of geometries to objects;
- $\mathsf{speed}[\mathsf{avg},4\mathsf{s}](\mathsf{y},\mathsf{v})$ resp. $\mathsf{pos}[\mathsf{line},4\mathsf{s}](\mathsf{y},\mathsf{r})$ defines a window operator that aggregates the average speed resp. positions (as points) of the vehicles over the streams *speed* and *pos*; $\mathsf{hasState}[\mathsf{first},-4\mathsf{s}](\mathsf{z},\mathsf{Stop})$ gives us the traffic lights, which switch in 4s to the state "Stop".

For the evaluation of this query, we have to adapt OBDA to handle spatial and streaming data, which is not considered in the standard approaches like [14]. For that, we extend our preliminary work [19] with a window operator, which (a) collects a set of data items (e.g., positions or speed) for each query atom from the underlying stream; and (b) calculates different aggregation functions on the set of numerical (e.g., sum), sequential (e.g., first), and spatial (e.g., line) data items.

Data Model and Query Language. Our data model is *point-based* (in contrast to an *interval-based* model) and captures the *valid time*, extracted from the V2X messages, saying that some *data item* is valid at that time point. To capture streaming data, we introduce the *timeline* $\mathbb{T}$, which is a *closed* interval of $(\mathbb{N},\leq)$. A (data) *stream* is a triple $F = (\mathbb{T}, v, P)$, where $\mathbb{T}$ is a timeline, $v : \mathbb{T} \to \langle F, S_F \rangle$ is a function that assigns to each element of $\mathbb{T}$ (called *timestamp*) data items of $\langle F, S_F \rangle$, where F (resp. S_F) is a *stream (resp. spatial with streams) database*, and P is an integer called *pulse* defining the general interval of consecutive data items on the timeline (cf. [34]). A pulse generates a stream of data items with the frequency derived from the interval length. We always have a *main pulse* with a fixed interval length (usually 1) that defines the lowest granularity of the validity of data points. The pulse also aligns the data items, which arrive asynchronously in the database (DB), to the timeline. We allow additional *larger pulses* that generate streams at lower frequency and thus of larger intervals; this can be utilized to perform optimizations such as e.g. caching.

Our query language is based on ordinary conjunctive queries (CQs) and adds spatial-stream capabilites. Thus, queries may contain ontology, spatial, and stream atoms. A spatial-stream CQ $q(\mathbf{x})$ is a formula:

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

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

- each $Q_{O_i}(\mathbf{x},\mathbf{y})$ has the form $A(z)$ or $P(z,z')$, where A is a class name, P is a property name of the LDM ontology, and z, z' are from $\mathbf{x} \cup \mathbf{y}$;
- each atom $Q_{S_j}(\mathbf{x},\mathbf{y})$ is from the vocabulary of spatial relations and of the form $S(z,z')$, where z, z' is as before and S is one of the following spatial relation $rel \in \{intersects, contains, nextTo, equals, inside, disjoint, outside\}$;
- $Q_{F_j}(\mathbf{x},\mathbf{y})$ is similar to $Q_{O_i}(\mathbf{x},\mathbf{y})$ but adds the vocabulary for stream operators, which are taken from [7] and relate to CQL operators [3]. We have a window $[agr, l]$ over a stream F_j, where l is the window size in time units (positive for past, or negative for future). The *aggregate function* $agr \in \{count, sum, first, \ldots\}$ (see below for details) is applied to the data items in the window:[11]
  - $[agr, l]$: represents the aggregate of last or next l time units of stream F_j;

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

- $[l]$: represents the single tuple of F_j at index l with $l = 0$ if it is the current tuple;
- $[agr, b, e]$: represents the aggregate on a window between b and e in the past/future of a stream. This extension is inspired by [11] and allows to query historic data (e.g., logs), if they are stored as streams.

Query Rewriting by Stream Aggregation. We aim at answering pull-based queries at *a single* time point $\mathbb{T}_i$ with stream atoms that define *aggregate functions* on different windows sizes relative to $\mathbb{T}_i$. For this, we consider a semantics based on *epistemic aggregate queries* (EAQ) over ontologies [15] by dropping the order of time points for the data and handling the streamed data items as *bags* (multi-sets). This is similar to classic stream processing approaches. Roughly, we perform two steps, where we (1) calculate only "known" solutions, and (2) evaluate the rewritten query, which includes the TBox axioms as well, over these solutions. Each EAQ is evaluated over one or more *filtered and merged temporal* ABoxes. The filtering and merging, relative to the window size and $\mathbb{T}_i$, creates for each EAQ ϕ one (so-called) *windowed* ABox $A_{\boxplus_\phi}$, which is the union of the static ABox A and the filtered streaming data items from $\langle F, S_F \rangle$. The EAQ are then applied on $A_{\boxplus_\phi}$ by grouping and aggregating the normal objects, constant values, and spatial objects.

We introduce a *bag-based epistemic semantics* for the queries, where we locally close the world for the specific window and avoid "wrong" aggregations due to the open world semantics of OWL2 QL. Further details on the algorithm for EAQ evaluation are provided in [17,22].

For *normal objects* and *constant values*, we allow different aggregate functions such as *count*, *min*, *max*, *sum*, *avg*, $first$, *last* on the data items of a stream. For *last* and $first$, we need to search the bag of data items as the sequence of time points is lost. This is achieved by iteratively checking whether we have a match at one of the time points. In the implementation, the first and last match can be simply cached while processing the stream.

For *spatial objects*, geometric aggregate functions are applied to the bag of data items, which are usually geometries. As with *last*, the order of the items is lost, hence, we need to rearrange them to create a valid geometry $g(s)$, which is a sequence $p = (p_1, \ldots, p_n)$ of points p_i. We also introduce new aggregate functions to create new geometries:

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

Besides the above aggregate functions, more functions such as computing a minimum spanning tree could be applied.

Query Evaluation by Hypertree Decomposition. A main challenge relates to the handling of three types of query atoms that need different evaluation techniques over possibly separate databases. Ontology atoms are evaluated over the static ABox A using a "standard" OWL2 QL query rewriting, i.e., PerfectRef [14]. For spatial atoms, we need to dereference the bindings to the spatial ABox S_A and evaluate the spatial relations (e.g., *inside*) on the spatial objects. Stream atoms are computed via EAQs over the windowed ABoxes extracted from F and the spatial ABox with stream support S_F .

In [19], we introduced two spatial query evaluation strategies based on the assumptions that *no bounded variables* occur in spatial atoms and that the CQ is *acyclic* (roughly no proper cycle between join variables exists). As shown in [19], one of them is based on the query hypergraph and the derived join plan. This strategy is well-suited for a lifting to spatial-stream CQs, since it allows us fine-grained caching, the full control over the evaluation, and possibly the handling of different database entities. We omit here decomposing an acyclic query into a hypergraph and the related join tree (details in [31]). The main steps of our query evaluation strategies are:

(1) construct the acyclic hypergraph H_q from q and label each hyperedge in H_q with l_O, l_S, and l_F, if it represents an ontology, spatial, or stream atom, resp. the combination of them; l_F gets the window size assigned, e.g., $l_{F,2}$ for $\mathsf{speed}[\mathsf{avg}, 2\mathsf{s}](\mathsf{y}, \mathsf{v})$.
(2) 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 label $l_{F,n}$. The intention is to extract subtree CQs that share the same window size l (where static queries have $l = 0$); they can be jointly evaluated and cached for future query evaluations.
(3) apply detemporalization as described above, where for each subtree J_{ϕ_i} the stream CQ q_{ϕ_i} is extracted and com-

puted. The results are stored in a (virtual) relation R_{ϕ_i}, and each J_{ϕ_i} is replaced with a query atom pointing to R_{ϕ}.

(4) traverse J_q bottom up, left-to-right, to evaluate the CQ q_{ϕ_i} for each subtree J_{ϕ_i} (now without stream atoms) and keep the results in memory for future steps. Ontology atoms and spatial atoms are evaluated as described before.

Caching for future queries is achieved by storing the intermediate results in memory with an expiration time according to l; static results never expire. Implementation details are given in Section 7.

6 Application Scenarios

In this section, we give three application scenarios that illustrate the usability and benefits of our approach. The scenarios are related to the deployment of roadside ITS stations on complex road intersections. The ITS stations receive arbitrary V2X messages and are capable of sending signal phases (SPaT) and the local intersection topology (MAP) messages. The first scenario is concerned with static data, where the consistency of a topology is validated offline. The second and third scenario deal with streaming data arising from vehicle movements (CAM) and SPaT messages.

Application Scenario 1 (S1) - Consistency Checks. The LDM is defined as a data integration platform that provides services such as message validation to external applications. Hence, this scenario is related to the first and second layer of the LDM and shows that data inconsistency can be detected using Datalog rules. Inconsistencies occur in individual messages, but also the integration of different messages such as MAP and SPaT might lead to it, due to wrongly generated or not up-to-date information. The following rules show possible consistency checks.

Example 4 This rule checks if each ingress lane x connects to an outgress lane y:

$$\texttt{LaneIn}(\texttt{x}) \wedge \textit{ not } \texttt{connected}(\texttt{x},\texttt{y}) \wedge \texttt{LaneOut}(\texttt{y}) \wedge \texttt{isPartOf}(\texttt{x},\texttt{z}) \wedge \texttt{Intersection}(\texttt{z}) \rightarrow \texttt{inconsistent}(\texttt{x},\texttt{z}).$$

Note that *not* is the *negation as failure* for $\texttt{connected}(\texttt{x},\texttt{y})$ saying that the clause will succeed, if there is no match of x and y in the relation `connected`.

Example 5 The following rule checks if each ingress lane x has at least one maneuver, e.g., turn left:

$$\texttt{LaneIn}(\texttt{x}) \wedge \#\texttt{count}\{\texttt{allowsManeuver}(\texttt{x},\texttt{y}) \wedge \texttt{Maneuver}(\texttt{y})\} = 0 \wedge \texttt{isPartOf}(\texttt{x},\texttt{z}) \wedge \texttt{Intersection}(\texttt{z}) \rightarrow \texttt{inconsistent}(\texttt{x},\texttt{z}).$$

Application Scenario 2 (S2) - Data Aggregation. The focus of this stream scenario is the collection of statistical data and vehicle moving patterns by observing the streaming data on a specific intersection. Data aggregation can be accomplished as an independent task, but is also a preliminary step for more complicated reasoning task such as diagnosis. The aggregation will be often based on the CAM messages, but also SPaT aggregation is interesting, as monitoring of traffic lights could be desired to detect faulty behavior.

Example 6 The following query returns the last signal phase u of each lane x on intersection I_1 in an interval of 20 seconds:

$$\mathsf{q_a}(\mathsf{x},\mathsf{u}) : \mathsf{LaneIn}(\mathsf{x}) \wedge \mathsf{Intersection}(\mathsf{y}) \wedge (\mathsf{y} = \mathsf{I_1}) \wedge \mathsf{isPartOf}(\mathsf{x},\mathsf{y}) \wedge \mathsf{isManaged}(\mathsf{x},\mathsf{z}) \wedge \mathsf{V2XSignal}(\mathsf{z}) \wedge \mathsf{hasState}[\mathsf{last},20\mathsf{s}](\mathsf{z},\mathsf{u})$$

A more challenging task is the analysis of driving patterns based on vehicles maneuvers (e.g., u-turns), which then can be used for event detection or gathering local traffic statistics.

Example 7 The following query returns all vehicles x and their brands y that are moving above 30km/h and have been heading straight during the last 5 seconds:

$$\mathsf{q_b}(\mathsf{x},\mathsf{y}) : \mathsf{Vehicle}(\mathsf{x}) \wedge \mathsf{speed}[\mathsf{avg},5\mathsf{s}](\mathsf{x},\mathsf{v}) \wedge (\mathsf{v} > 30) \wedge \mathsf{vehicleMaker}(\mathsf{x},\mathsf{y}) \wedge \mathsf{position}[\mathsf{line\_angle},5\mathsf{s}](\mathsf{x},\mathsf{a}) \wedge (\mathsf{a} > 10) \wedge (\mathsf{a} < 10)$$

Application Scenario 3 (S3) - Event Detection. This stream scenario deals with the detection of *emergency vehicles* and *red light violation*. Emergency detection aims at giving preference to emergency vehicles, e.g., ambulances or fire trucks, by switching to a green phase for the incoming vehicle. For this case, we need to integrate CAM and MAP messages and use the LDM to detect whether a vehicle is an emergency vehicle and is moving on one of the incoming lanes of an intersection.

Example 8 This query returns the emergency vehicles z that are the last 10 seconds on incoming lanes of intersection I_1:

$$\mathsf{q_c}(\mathsf{z}) : \mathsf{LaneIn}(\mathsf{x}) \wedge \mathsf{isPartOf}(\mathsf{x},\mathsf{y}) \wedge \mathsf{Intersection}(\mathsf{y}) \wedge \mathsf{hasGeo}(\mathsf{x},\mathsf{u}) \wedge \mathsf{intersects}(\mathsf{u},\mathsf{v}) \wedge \mathsf{position}[\mathsf{line},10\mathsf{s}](\mathsf{z},\mathsf{v}) \wedge (\mathsf{y} = \mathsf{I_1}) \wedge \mathsf{MotorVehicle}(\mathsf{z}) \wedge \mathsf{hasRole}(\mathsf{z},\mathsf{s}) \wedge (\mathsf{s} = \mathsf{EmergencyRole})$$

In this example, we can see the interaction of all mentioned atoms. The stream atom $\mathsf{position}[\mathsf{line},10\mathsf{s}](\mathsf{z},\mathsf{v})$ specifies a time-based window of 10s at query time, where the GPS positions are used to construct a path using the aggregate function line; $\mathsf{intersects}(\mathsf{u},\mathsf{v})$ checks whether the path crosses the bounding box of any incoming lane.

The detection of red light violations is driven by improving road safety on a heavily frequented intersection with bad visibility. This is our most challenging case, as all types of V2X messages have to be combined. We use the LDM to detect, whether a vehicle is moving above 30km/h on a lane, whose current signal phase will turn red in 4 seconds. Note that the parameter -4s relies on the capabilities of the signal controller to predicts its future state.

Example 9 This query is a small extension of Ex. 3 by limiting it to a single intersection. It returns all vehicles y that are

/* S1: cars with brands, travelling above 30km/h */
$q_1(x,y,z)$: Car(x), speed(x,y)[avg,10], vehicleMaker(x,z), $y > 30$
/* S1: lanes and signal groups switched to red */
$q_2(x,y)$: LaneIn(x), hasSignal(x,y), SignalGroup(y), signalState(y,z)[last,15], $z =$ "R"
/* S1: vehicles on incoming lanes */
$q_3(x,v)$: Vehicle(x), pos(x,y)[line,10], inside(y,u), hasGeo(v,u), LaneIn(v)
/* S2: vehicles with crossed paths */
$q_4(x,y)$: Vehicle(x), pos(x,w)[line,30], intersects(w,z), pos(y,z)[line,30], Car(y)
/* S2: vehicles travelling above 30km/h heading straight */
$q_5(x,y)$: Vehicle(x), speed(x,z)[avg,15], pos(x,y)[line_angle,15], $z > 30, y > -10, y < 10$
/* S2: detection of red-light violation */
$q_6(x,y)$: Taken from Ex. 9
/* Synthetic: testing many ontology atoms */
$q_7(x,z)$: LaneIn(x), isPartof(x,u), Intersection(u), $u =$ "I1", hasSignal(x,y), SignalGroup(y), signalState(y,r)[last,15], $r =$ "R", connect(x,q), connect(q,v), Lane(v), hasSignal(v,z), SignalGroup(z), signalState(z,s)[last,15], $s =$ "R"
/* Synthetic: testing many spatial atoms */
$q_8(x,y)$: Vehicle(x), pos(x,y)[line,20], intersects(y,u), LaneIn(r), hasGeo(r,u), intersects(y,v), LaneIn(s), hasGeo(s,v), intersects(y,w), LaneIn(t), hasGeo(t,w), within(y,z), hasGeo(q,z), Intersection(q)
/* Synthetic: testing many stream atoms */
$q_9(x,q,r,s,t,u)$: Vehicle(x), speed(x,q)[avg,1], speed(x,r)[avg,5], speed(x,s)[avg,10], speed(x,t)[avg,25], speed(x,u)[avg,50]

Table 1: Benchmark queries (windows size in seconds)

moving above 30km/h and violate the signal phase stop of a lane x:

$$\mathsf{q_d}(x,y) : \mathsf{LaneIn}(x) \wedge \mathsf{hasLocation}(x,u) \wedge \mathsf{intersects}(u,v) \wedge \mathsf{pos}[\mathsf{line},4\mathsf{s}](y,v) \wedge \mathsf{Vehicle}(y) \wedge \mathsf{speed}[\mathsf{avg},4\mathsf{s}](y,r) \wedge (r > 30) \wedge \mathsf{isManaged}(x,z) \wedge \mathsf{SignalGroup}(z) \wedge \mathsf{hasState}[\mathsf{first},-4\mathsf{s}](z,s) \wedge (s = \mathsf{Stop}) \mathsf{isPartOf}(y,w) \wedge \mathsf{Intersection}(w) \wedge (w = \mathsf{I}_1)$$

The examples above are just samples of possible queries, since our approach could also be used for other scenarios as collecting road tolls, surveillance, or route planning.

7 Implementation and Experiments

We have implemented a prototype for our spatial-stream query answering approach in JAVA 1.8 using the open-source PIPELINEDB 9.6.1[12] as the spatial-stream RDBMS. As a preprocessing step for each query, the hypertree decomposition is computed using the implementation of Gottlob et al. [16].[13] Based on it, each sub CQ is evaluated separately and (spatial) joined in-memory. For the query rewriting of OWL 2 QL, we used the implementation of OWLGRES 0.1 [42]; more recent (and more efficient) implementations for query rewriting (e.g., [39]) are available.

The experiments are based on our scenarios of data aggregation and event detection located (a) on a single intersection and (b) on a network of locally connected intersections, both managed by a single roadside C-ITS station. The ontology, nine queries (see Table 1), the experimental setup with logs, and the implementation are available on our project website.[14] The nine queries are composed of three queries of Scenario 2, three queries of Scenario 3, and three queries that

	Type	#Q	#A	(a) t with #vehicles 10	100	500	1000	2500	5000	(b) t with ms sim. delay 0	100	250	500
q_1	O,F	1	1	0.85	0.82	0.91	1.05	1.22	1.58	0.78	0.74	0.73	0.71
q_2	O,F^*	1	6	0.83	0.83	0.83	0.83	0.83	0.83	0.77	0.77	0.72	0.71
q_3	O,S,F	3	23	0.89	0.87	1.00	1.25	1.39	1.74	0.83	0.81	0.77	0.75
q_4	O,S,F	3	22	1.10	1.09	1.24	1.53	1.81	2.32	1.02	1.00	0.95	0.93
q_5	O,S,F	3	42	1.11	1.10	1.26	1.39	1.90	1.92	1.05	1.00	0.98	0.96
q_6	O,S,F	7	52	1.39	1.39	1.49	1.69	2.36	2.28	1.40	1.28	1.26	1.25
q_7	O,F^*	6	69	1.16	1.16	1.16	1.16	1.16	1.16	1.15	1.12	1.11	1.09
q_8	O,S	9	73	0.92	0.94	1.30	1.43	1.72	2.19	0.99	0.98	0.92	0.91
q_9	O,F	9	105	1.67	1.73	1.99	2.06	2.49	2.97	1.71	1.68	1.66	1.63

Table 2: Results (t in seconds) for scenario (a) and (b), items marked with * are signal streams

are synthetic and test specific aspects of the query evaluation. We use our LDM ontology (see Section 4) with 119 concepts having 113 inclusion assertions (e.g., sub-classes); 34 roles and 28 data roles having 31 inclusion assertions.

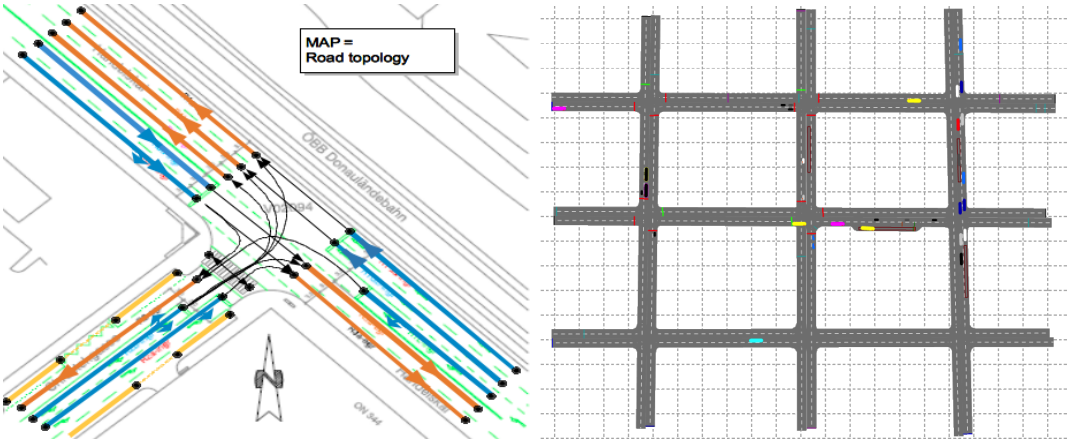
For (a), we have a T-shaped intersection as shown in Fig. 3a. It represents a real-world deployment of an experimental C-ITS station in Vienna, and connects two roads with 13 lanes and 3 signal groups linked to the lanes. We have developed a synthetic data generator that simulates the movement of a varying number of vehicles (10, 100, 500, 1000, 2500, and 5000) on a single intersection updating the streams every 50 ms on average. This allows us to generate streams with up to 10000 data items per stream and second.[15] We chose random starting points and simulated linear movements on a constant pace, creating a stream of vehicle positions. We also simulated simple signal phases for each traffic light that toggle between red and green every three seconds. This scenario aims to show the scalability of our approach with many vehicles that have simple driving patterns.

For (b), we use a realistic traffic simulation of 9 intersections in a grid (see Fig. 3b), developed with the microscopic traffic simulation tool PTV VISSIM,[16] which allows us to simulate realistic driving behavior and signal phases. The intersection structure, driving patterns and signal phases are more complex, but the number of vehicles is smaler (max. 300) than in (a), as quickly traffic jams emerge. We developed an adapter to extract the actual state of each simulation step, allowing us to replay the simulation from the logs. To vary data throughput, we ran the replay with 0ms (no delay), 100ms (real-time), 250ms and 500ms delay.

Results. We conducted our experiments on a Mac OS X 10.6.8 system with an Intel Core i7 2.66GHz, 8GB of RAM, and a 500GB hard disk drive. The average of 11 runs for query rewriting time and evaluation time was calculated, where the largest outlier was dropped. The results are shown in Table 2, where we present the query type (O for ontology, F for stream, and S for spatial atoms), the number of subqueries

[12] https://www.pipelinedb.com/
[13] https://www.dbai.tuwien.ac.at/proj/hypertree/
[14] http://www.kr.tuwien.ac.at/research/projects/loctrafflog/eswc2017

[15] The intervals vary due to the number of vehicles, so we scale the DB updates up to 12 generators.
[16] http://vision-traffic.ptvgroup.com/en-us/products/ptv-vissim/



Scenario (a) Scenario (b)

Fig. 3: Schematic representation of the scenarios

#Q, size of rewritten atoms #A, and the average evaluation time (AET) t in seconds for n vehicles or the delay in ms.

The baseline spatial-stream query is q_3 for 500 vehicles, where we have a loading time of 0.22s, an evaluation time for the stream (resp. ontology) atom of 0.54s (resp. 0.03s), and a spatial join time of 0.05s. Clearly, 50% of the AET is used to evaluate the stream atoms (including rewriting steps). The loading time could be reduced by pre-compiling the program; this shortens evaluation by roughly 0.2s. Initial evaluation of the queries q_4, q_5, q_6 and q_9 show that with each new stream subquery the number of results dropped down to zero, which seems an implementation issue of PIPELINEDB with *Continuous Views* on the same stream with different window sizes. We found a workaround by adding a delay of 0.2s that again increases the number of results. This delay increases the AET, e.g. by 0.76s in q_9, and might be ignored with future versions of PIPELINEDB and other stream RDBMS. The synthetic queries with predominant ontology (q_6), spatial (q_7), and stream atoms (q_9) clearly show that the real challenging aspect of query evaluation are stream aggregates. The good performance of PIPELINEDB allows us to work on condensed results (reducing the join sizes); however, stream aggregates could be further accelerated by continuously calculating inline aggregates on the DB, which are skimmed by our queries. Notably, PIPELINEDB does not always keep the order of inserted data items; this does not affect our bag semantics, since an order on the data items is not needed.

In conclusion, the results show that our experimental prototype for up to 500 vehicles manages evaluation within 1.5 s (except query q_9). This suggests that with optimizations such as the ones mentioned, the quick detection of red-light violations on complex intersections is feasible.

8 Conclusion

In this paper, we have presented an extension of the LDM approach with Semantic Web technologies and stream processing. The technologies allow us to define a "semantic" world model, i.e., the LDM ontology, an expressive spatial-stream query language, derived model properties, and the inference of new information over streams. Our envisioned architecture is designed to show how these technologies can be applied in the context of the LDM and V2X integration. The architecture consists of an ontology, an integration and annotation framework for static, spatial and stream databases, a query answering component over streams, and a more expressive problem solving component. We have also worked on an initial version of an LDM ontology, and show how the architecture is implemented based on existing work in spatial data extraction [20] and spatial-stream query answering [19] using PIPELINEDB. Apart from the restriction to OWL2 QL, our methods are ontology-agnostic, hence other ontologies or evolved versions of the initial one could be used as well.

Furthermore, we have developed application scenarios for (i) consistency checking, (ii) data aggregation, and (iii) event detection to show the usability and benefits of our design. Finally, based on the scenarios we have conducted experiments utilizing the microscopic traffic simulation PTV VISSIM, to show the feasibility of query answering component.

Outlook Applications. The three application scenarios are good starting points; however more complex scenarios such as signal phase optimization of traffic lights or stream-based routing could be considered. As the next step, we aim to develop a *model-based diagnosis* component that includes a clear definition and encoding of observations O, a (fault) model S, and a list of system components C applied to the C-ITS domain. Based on the encoding, we aim to apply a standard rule-based evaluation with a solver for calculating a

diagnosis. The diagnosis component could further be complemented with a repair and/or re-configuration service for the system components, which would allow the dynamic adjustment and optimization of signal phases on a network of intersections.

Outlook Framework. Future work is directed to extend the components of the framework. The LDM ontology is only an initial draft and may be refined to capture more elements of the LDM and a traffic scene. An evolved version of the ontology could be taken as a first step towards standardization. The query answering component over streams could be further elaborated by realizing push-based querying using query decomposition and caching, handling of inconsistency in data items, and new aggregate functions.

Also, further investigations regarding expressivity and efficiency are needed. For instance, we have not considered distributed LDMs as introduced by [46], which would require alignment at the ontology level (if two ITS stations use different schemas), and data fusion including consistency checks at the stream level. Distributed LDMs would enable us to apply our methods to larger settings of distributed ITS stations. Finally, the framework can be directed towards methods for complex event detection and model-based diagnosis.

Acknowledgements We thank the reviewers for their comments, which helped to improve this article. This work has been supported by the Austrian Research Promotion Agency (FFG) project Industrienahe Dissertationen and the Austrian Science Fund (FWF) projects P26471 and P27730.

References

1. Andreone, L., Brignolo, R., Damiani, S., Sommariva, F., Vivo, G., Marco, S.: Safespot final report. Tech. Rep. D8.1.1 (2010). Available online.
2. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: a unified language for event processing and stream reasoning. In: Proc. of WWW 2011, pp. 635–644 (2011)
3. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. VLDB J. **15**(2), 121–142 (2006)
4. Artale, A., Kontchakov, R., Wolter, F., Zakharyaschev, M.: Temporal description logic for ontology-based data access. In: Proc. of IJCAI 2013, pp. 711–717 (2013)
5. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathscr{EL}$ envelope. In: Proc. of IJCAI 2005, pp. 364–369. Morgan-Kaufmann Publishers (2005)
6. Baader, F., Horrocks, I., Sattler, U.: Description logics. In: S. Staab, R. Studer (eds.) Handbook on Ontologies, pp. 21–43. Springer (2009)
7. Beck, H., Dao-Tran, M., Eiter, T., Fink, M.: LARS: A logic-based framework for analyzing reasoning over streams. In: Proc. of AAAI 2015, pp. 1431–1438 (2015)
8. Bereta, K., Koubarakis, M.: Ontop of geospatial databases. In: Proc. of ISWC 2016, pp. 37–52 (2016)
9. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American **284**(5), 34–43 (2001)
10. Borgwardt, S., Lippmann, M., Thost, V.: Temporalizing rewritable query languages over knowledge bases. J. Web Sem. **33**, 50–70 (2015)
11. Brandt, S., Kalayci, E.G., Kontchakov, R., Ryzhikov, V., Xiao, G., Zakharyaschev, M.: Ontology-based data access with a horn fragment of metric temporal logic. In: Proc. of AAAI 2017. To appear (2017)
12. Brickley, D., Guha, R., (eds.): RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, W3C (2004)
13. Calbimonte, J., Mora, J., Corcho, Ó.: Query rewriting in RDF stream processing. In: Proc. of ESWC 2016, pp. 486–502 (2016)
14. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *dl-lite* family. J. Autom. Reasoning **39**(3), 385–429 (2007)
15. Calvanese, D., Kharlamov, E., Nutt, W., Thorne, C.: Aggregate queries over ontologies. In: Proc. of ONISW 2008, pp. 97–104 (2008)
16. Dermaku, A., Ganzow, T., Gottlob, G., McMahan, B.J., Musliu, N., Samer, M.: Heuristic methods for hypertree decomposition. In: Proc. of MICAI 2008: Advances in Artificial Intelligence, pp. 1–11 (2008)
17. Eiter, T., Füreder, H., Kasslatter, F., Parreira, J.X., Schneider, P.: Towards a semantically enriched local dynamic map. In: Proc. of ITSWC 2016 (2016)
18. Eiter, T., Ianni, G., Krennwallner, T.: Answer set programming: A primer. In: Proc. of Reasoning Web Summer School 2009, pp. 40–110 (2009)
19. Eiter, T., Krennwallner, T., Schneider, P.: Lightweight spatial conjunctive query answering using keywords. In: Proc. of ESWC 2013, pp. 243–258 (2013)
20. Eiter, T., Pan, J.Z., Schneider, P., Simkus, M., Xiao, G.: A rule-based framework for creating instance data from openstreetmap. In: Proc. of RR 2015 (2015)
21. Eiter, T., Parreira, J.X., Schneider, P.: Towards spatial ontology-mediated query answering over mobility streams. In: Proc. of Stream Reasoning Workshop 2016 (2016)
22. Eiter, T., Parreira, J.X., Schneider, P.: Spatial ontology-mediated query answering over mobility streams. In: Proc. of ESWC 2017 (2017)
23. 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)
24. 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)
25. Gebser, M., Grote, T., Kaminski, R., Obermeier, P., Sabuncu, O., Schaub, T.: Stream reasoning with answer set programming: Preliminary report. In: Proc. of KR 2012 (2012)
26. ISO/TS 17931:2013: Intelligent transport systems - Extension of map database specifications for Local Dynamic Map for applications of Cooperative ITS. Technical report, ISO (2013)
27. ISO/TS 18750:2015: Intelligent transport systems - Cooperative systems - Definition of a global concept for Local Dynamic Maps. Technical report, ISO (2015)
28. Koenders, E., Oort, D., Rozema, K.: An open local dynamic map. In: Proc. of ITS European Congress 2014 (2014)
29. Kompfner, P.: Cvis final activity report. Tech. Rep. D.CVIS.1.3 (2010). Available online.
30. Krötzsch, M., Patel-Schneider, P.F., Rudolph, S., Hitzler, P., Parsia, B.: OWL 2 web ontology language primer. Tech. rep., W3C (2009)
31. Maier, D.: The Theory of Relational Databases. Computer Science Press (1983)
32. Motik, B., Fokoue, A., Horrocks, I., Wu, Z., Lutz, C., Grau, B.C.: OWL 2 web ontology language profiles. W3C recommendation, W3C (2009)

33. Netten, B., Kester, L., Wedemeijer, H., Passchier, I., Driessen, B.: Dynamap: A dynamic map for road side its stations. In: Proc. of ITS World Congress 2013 (2013)
34. Özçep, Ö.L., Möller, R., Neuenstadt, C.: Stream-query compilation with ontologies. In: Proc. of AI 2015, pp. 457–463 (2015)
35. Phuoc, D.L., Dao-Tran, M., Tuán, A.L., Duc, M.N., Hauswirth, M.: RDF stream processing with CQELS framework for real-time analysis. In: Proc. of the 9th ACM International Conference on Distributed Event-Based Systems, DEBS 2015, pp. 285–292 (2015)
36. Poole, D.: A methodology for using a default and abductive reasoning system. Int. J. Intell. Syst. **5**(5), 521–548 (1990)
37. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. **32**(1), 57–95 (1987)
38. Ren, Y., Pan, J.Z.: Optimising ontology stream reasoning with truth maintenance system. In: Proc. of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, pp. 831–836 (2011)
39. Rodriguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Ontology-based data access: Ontop of databases. In: Proc. of ISWC 2013, pp. 558–573 (2013)
40. Schreiber, G., Raimond, Y., Manola, F., Miller, E., McBride, B.: Rdf 1.1 primer. W3C working group note, W3C (2014)
41. Shimada, H., Yamaguchi, A., Takada, H., Sato, K.: Implementation and evaluation of local dynamic map in safety driving systems. J. Transportation Technologies **5**(2), 102–112 (2015)
42. Stocker, M., Smith, M.: Owlgres: A scalable owl reasoner. In: Proc. of OWLED 2008 (2008)
43. Teymourian, K., Paschke, A.: Plan-based semantic enrichment of event streams. In: Proc. of ESWC 2014, pp. 21–35 (2014)
44. Ulbrich, S., Nothdurft, T., Maurer, M., Hecker, P.: Graph-based context representation, environment modeling and information aggregation for automated driving. In: Proc. of 2014 IEEE Intelligent Vehicles Symposium Proceedings, pp. 541–547 (2014)
45. Zaniolo, C.: Logical foundations of continuous query languages for data streams. In: Proc. of Datalog 2.0 Workshop 2012, pp. 177–189 (2012)
46. Zoghby, N.E., Cherfaoui, V., Denoeux, T.: Evidential distributed dynamic map for cooperative perception in vanets. In: Proc. of 2014 IEEE Intelligent Vehicles Symposium Proceedings, pp. 1421–1426 (2014)