ulm university universität
uulm

UNIVERSITÄT ULM
FAKULTÄT FÜR INGENIEURWISSENSCHAFTEN, INFORMATIK UND PSYCHOLOGIE
INSTITUT FÜR KÜNSTLICHE INTELLIGENZ

Ontology-Based Map Modelling and Processing for Autonomous Vehicles

Dissertation

zur Erlangung des Doktorgrades Dr. rer. nat. der Fakultät für Ingenieurwissenschaften, Informatik und Psychologie der Universität Ulm

vorgelegt von

Haonan QIU
geboren in Jiangsu, China

2021

October 15, 2021

This thesis has been written by Haonan Qiu in partial fulfilment of the requirements for a doctoral degree of the faculty of engineering, computer science and psychology at Ulm University. It has been submitted on October 15, 2021.

Amtierender Dekan:	Prof. Dr.-Ing. Maurits Ortmanns
Erstgutachter:	Prof. Dr. Birte Glimm
Zweitgutachter:	Prof. Dr. Markus Krötzsch
Tag der Promotion:	July 6, 2022.

Contact
Haonan Qiu
Institute of Artificial Intelligence
Faculty of Engineering, Computer Science and Psychology
Ulm University
Albert-Einstein-Allee 43
89081 Ulm, Germany

This document was set by the author using the LaTeX 2ε typesetting system.

"…the happiness and greatness, the rank and station, the pleasure and peace, of an individual have never consisted in his personal wealth, but rather in his excellent character, his high resolve, the breadth of his learning, and his ability to solve difficult problems…."

'ABDU'L-BAHÁ

UNIVERSITÄT ULM

Abstract

Engineering, Computer Science and Psychology
Institute of Artificial Intelligence

Doctor of Philosophy

Ontology-Based Map Modelling and Processing for Autonomous Vehicles

by Haonan QIU

Autonomous Driving (AD) systems use digital maps as a virtual sensor to anticipate the road ahead and make decisions. The evolution of digital maps has posed a range of challenges to the current mapping ecosystem. First, different standards and formats make maps lack interoperability. Second, the regular update and increasing size of the maps data make it largely impossible to store and process a complete detailed map in a navigation system of a car. Finally, errors introduced in any step of map creation may cause unavailability of AD functions. Currently, there are separate software components trying to solve each challenge, however, the integration and maintenance of these components is a difficult task due to the challenges of the maps' evolution. This thesis investigates an ontology-based approach with the goal of shifting from map data-oriented functional design to knowledge-centered ontology design. Thus, we contribute a knowledge-spatial architecture with an embedded quality assurance mechanism to achieve efficient dynamic map provision with quality assurance, providing flexible query answering.

Our first contribution is two levels of ontological abstraction to solve the map data integration problem. The developed low-level ontologies represent the specific map data formats. A single high-level ontology is designed with the prerequisites and requirements of a self-driving vehicle in mind resulting in a light(er) weight generic map ontology. Mapping rules are designed to unify low-level map ontologies to the generic map ontology. As our second contribution, we develop and implement an efficient dynamic map update strategy to provide continuous road knowledge ahead. To achieve efficient map updates, we design a spatial-sliding window on top of the light(er) weight generic map ontology and process map data streams via reasoning based on a pre-fetching mechanism. To ensure map data quality and preventing AD mode degradation, we conduct our third contribution which is the design of a workflow to detect and fix the map data violation including semantic enrichment, violation detection, and violation handling. To facility violation detection, we develop a Map Quality Violation Ontology and a set of constraint rules. Violation handling is realized based on a real-world map data error.

To evaluate the proposed methods, we rely on empirical evaluations as well as on the development of concrete use cases. The attained results provide evidence that an ontology-based approach enables effective map integration and processing with ensured data quality. This allows engineers to focus more on developing AD functions on the knowledge level rather than on data processing and integration. The proposed two-level ontology development methodology shades the light for ontology practitioners to build ontologies in situations where the data is dynamic, and the computation resources are limited.

Acknowledgements

This research was funded by BMW's PhD ProMotion programme from 2018 to 2021 and created under the roof of a cooperation with Artificial Intelligence Institute of Ulm University, Faculty of Informatics and BMW Cart IT, Ulm.

Undertaking this PhD has been a truly life-changing experience for me and it would not have been possible to do without the support and guidance that I received from many people.

First and foremost, I would like to thank my supervisor, Prof. Birte Glimm. She has supported and encouraged me from the very first meeting we had in April 2018 to discuss my research ideas, 2 months before the start of my PhD. Since then she has never stopped inspiring me. The weekly meetings she offered in the last three years has been tremendously helpful for me to plan and reflect on the research work. Technical discussions with her helped me immensely to tackle some major challenges in this work. I am very grateful for her strong support and scientific guidance throughout this long walk with all its "ups and downs". She is my mentor and a better advisor for my doctorate study beyond the imagination.

Secondly, I would like to sincerely thank Adel Ayara for proposing the topic of this thesis and giving me the opportunity to conduct this research at BMW Car IT. His profound domain knowledge and consistent support have been essential for the conduct of the research. The regular meetings with him helped me better understand the background of the research field. His professional planning and organizing advice are some very valuable inputs for the completion of this work. Many thanks also to Dr. Sebastian Winter, who made my access simpler to a map data set and gave me an opportunity to become an intern in his team. I appreciate the support received from Christian Enchelmaier.

I also appreciate the support received through the collaborative work with the RDFox team in Oxford— thank you to Prof. Boris Motik for both academic insights and technical analyses; thank you to Valerio Cocchi for the encouragement and technical support.

Furthermore, I would like to thank my friends and family. Thank you to Ruya, Sapna, Arian, and the Bahai community in Ulm for their open doors and heartwarming support throughout this whole journey. Their companionship has uplifted my spirit. Finally, I thank from the bottom of my heart to my mom and Alex. With their unwavering support they believed in me since day one, and allowed me to embark into this wonderful and challenging adventure. I would simply not be here without them.

Contents

List of Figures

List of Tables

List of Abbreviations

BASt	German Federal Road Research Institute (Bundesanstalt Für Straßenwesen)
AD	Autonomous Driving
ADAS	Advanced Driver Assistance Systems
ADASIS	Advanced Driver Assistance Systems Interface Specifications
ABS	Anti-lock Braking Systems
ESC	Electronic Stability Control
EBA	Emergency Brake Assist
GNSS	Global Navigation Satellite Systems
GPS	Global Positioning System
HD	High Definition
SD	Standard Definition
HDLM	HERE HD Live Map
NDS	Navigation Data Standard
LDM	Local Dynamic Map
SAE	Society Of Automotive Engineers
BLOB	Binary Large Object
Protobuf	Protocol Buffers
eHorizon	Electronic Horizon
ETSI	European Telecommunications Standards Institute
RDS	Relational DataScript
ORSD	Ontology Requirements Specification Document
DL	Description Logic
FOL	First Order Logic
SHACL	Shapes Constraint Language
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF schema
SPARQL	SPARQL Protocol and RDF Query Language
SWRL	Semantic Web Rule Language
Turtle	Terse RDF Triple Language
URI	Uniform Resource Identifier
IRI	Internationalized Resource Identifier
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSD	XML Schema Definition
OWA	Open World Assumption
CWA	Close World Assumption
KB	Knowledge Base
KR	Knowledge representation
AI	Artificial Intelligence
TBox	Terminological Box
ABox	Assertional Box

ODP	Ontology Design Pattern
OBDI	Ontology Based Data Integration
OBDA	Ontology Based Data Access
GAV	Global-as-View
Graffoo	Graphical Framework for OWL Ontologies

Chapter 1

Introduction

Almost every vehicle on the market now comes with driver assistance systems such as anti-lock braking (ABS) and electronic stability control systems (ESC) [75]. High-class cars are equipped with a variety of advanced driver assistance systems (ADAS) features for added safety such as emergency brake assist (EBA). According to statistical results, although such technologies have contributed to a decrease in the number of fatal incidents, more than 90% of accidents have been caused by human mistakes or fatigue [46, 40]. As a result, reducing the number of human-caused accidents by implementing the next generation of ADAS for vehicles, also known as automated driving systems (ADS), is critical for the European Union.

Automated driving requires an intelligent control system consisting of high-performance sensors and robotic technologies. The requirements for a technical system must reflect the capabilities humans need to drive a car. Through several major recent advances in many different technological areas, such as sensing systems and computational power, the autonomous vehicle is now on the verge of becoming a reality [117]. The autonomous car relies on a multitude of sensors, for example, radar (radio detection and ranging), ultrasonic sensors, cameras, LiDAR (light detection and ranging) as well as Global Navigation Satellite Systems (GNSS) [192] and its intelligent fusion. Therefore, it can prevent and anticipate critical driving situations, achieving response times and viewing capabilities that go far beyond humans.

Despite its sensing capabilities, the autonomous vehicle may encounter hazardous driving circumstances that may be avoided only through foresight that extends beyond the car's sensor range. To address such situations and to ensure human safety and trust in the technical systems, the autonomous vehicles also rely on an additional "virtual" sensor for their driving task, the so-called High Definition Map (HD Map) [161, 121, 44, 101, 190]. This map provides detailed and critical information to the autonomous vehicles regarding their current surrounding traffic situation and environment. The provided information is highly accurate and georeferenced up to the sub-meter level of precision. Current standard navigation map data, on the other hand, only reaches a localisation accuracy of a few meters. As a result, the HD map may be seen as a highly precise virtual 3D representation of the real world [101]. That way, autonomous vehicles can compare their sensor readings with a virtual reference, facilitating the driving task. Almost all players in the area of highly automated driving, e.g., Google, HERE, TomTom, Baidu, BMW, or Toyota, rely on HD maps to further improve the driving capabilities of their autonomous vehicles.

Recently, collective efforts have been made towards standardising HD maps, such as the Geographic Data File (GDF) 5.1 at the international level [104] or the Navigation Data Standard (NDS) and the ADASIS protocol V3 at industrial level [141]. There are also national efforts for developing HD map standards, such as the China Industry Innovation Alliance for the Intelligent and Connected Vehicles

(CAICV). However, as of now, there is no single, authoritative format or standard for HD maps [121]. As a result, map data integration poses major challenges in practice for providing autonomous driving functions.

Compared to conventional digital navigation maps such as Standard Definition (SD) maps, HD maps are extremely detailed, including a large amount of highly precise information. As a result, HD maps need more regular updates than SD maps to maintain the map's content. Additionally, HD maps need a substantial increase in processing capacity and computing resources. This effectively eliminates the possibility of storing a complete, detailed map inside a vehicle's navigation system. Due to the regular updates and storage limitations associated with digital map data, the existing map processing for digital map data is in many ways inadequate for HD Maps.

Making HD maps is a complex process where data collected from various sensors goes through data fusion, data processing, layer abstraction, and format converting based on selected map formats. As a result, it is very challenging to create an error-free HD map [9]. According to a study carried out by the European Commission (EC) on the integrity and reliability of HD maps, the map data are obtained from multiple sources of varying trustworthiness [57]. As the HD market continues to expand, inconsistent approaches using potentially poor data may increase the risk of hazardous events occurring on the road network. Therefore, map quality checking is required for autonomous driving functions.

1.1 Research Challenges

Autonomous vehicles, as an important future transportation means, now require new approaches to address the challenges mentioned earlier that had not been required before. Therefore, i) new methodologies have to be developed so that the driving functions are supported by a generic map representation independent of any specific map format. This includes ii) a constant map request and process strategy based on a vehicle route. To ensure the availability of autonomous driving functions, iii) data quality checking and handling errors is an additional key aspect, as addressed in this thesis.

Challenge 1: Representing a generic and unified map model for autonomous driving functions.
HD maps are provided in different logical models and formats with respect to different map providers. Current implementations of map data processing, however, are tightly coupled to the underlying raw data formats and specific map models. Applications and protocols developed based on one specific map cannot be easily extended to handle other maps, which results in poor extensibility. Therefore, a solution that is able to easily integrate various map data sources into a generic and unified map model which can be easily accessed by high-level driving functions is required.

Challenge 2: Efficient map processing for autonomous driving functions.
Autonomous driving systems need HD maps to provide adequate information for providing vehicle localisation, planning, and guidance functions. HD maps are highly detailed at the centimetre level, which requires frequent updates. Additionally, HD maps contain big amounts of data and high demand for computation resources. The available computation resources of embedded devices in the vehicle, however, is limited. This makes the HD map process for answering autonomous

driving functions in time an especially challenging task, considering frequent updates and computational resource constraints. As a result, a method for effectively processing dynamic map data is required to address autonomous driving tasks in real-time.

Challenge 3: Ensuring map data quality for autonomous driving functions.
The autonomous driving functions rely on the quality of map data. It, however, continuously proves to be very difficult to produce error-free maps. This is due to the high complexity of the real world and its dynamic nature, as well as quality management complexities when combining and matching large datasets from various sources. Currently, there is no standard procedure for assessing HD map data quality. Often the quality requirements are encoded in unstructured formats, e.g., plain text, which is not machine-readable. Therefore, a mechanism to ensure map data quality using knowledge in a machine-readable form should be part of the map process for providing autonomous driving functions.

1.2 Research Questions & Contribution

Following the discussion in the previous sections, the following research questions are defined.

> RQ1: Can an ontology-based approach solve the map data integration problem and provide a generic and unified map model?

In order to answer this research question, an ontology-based data integration approach, in particular, Global as View (GAV) architecture proposed by Wache et al. [181] is used for integrating different low-level maps into a generic high-level map model. To meet the need for data integration, ontologies are used for representing the different map specific low-level ontologies and the generic high-level map ontology, and rules are used to process the low-level map ontologies as well as transferring them to the generic high-level map ontology.

Contribution: An ontology-based approach to semantically integrate various map formats is proposed. A practical methodology for building the generic high-level ontologies and low-level ontologies is presented. Based on this methodology, we build the generic HLM (high-level map) ontology and two low-level ontologies – the LNDS ontology using the NDS map format and the LHERE ontology based on the HERE HD Live Map format. A categorisation of the types of rules that are needed for knowledge abstraction and spatial reasoning is described. By using the proposed approach, semantic integration over different map formats using a generic map model is resolved.

> RQ2: Can an ontology-based approach perform efficient knowledge processing and spatial reasoning while the knowledge base is continuously changing?

To respond to this question, spatial sliding windows and parallel knowledge processing with layered datastores are used. The vehicle spatial window is operated on the high-level datastore, responsible for prefetching future map tiles and determining expired map objects based on the received GPS position and available high-level

map knowledge. The prefetching decision made by the high-level datastore triggers parallel low-to-high map knowledge processes running in the low-level datastores. Subsequently, the high-level datastore is incrementally updated via newly generated high-level map knowledge.

Contribution: A knowledge-spatial architecture realised in a practical solution for dynamic map processing is outlined. The knowledge dimension is responsible for the parallel knowledge abstraction process from the format-specific and detailed low-level ontologies to the generic high-level ontology. The spatial dimension is orthogonal to the knowledge dimension and correlates facts that are true within a certain space and time at the high-level ontology. It describes the continuous spatial reasoning process with respect to the updated vehicle position and dynamic road environmental knowledge. We adopt the notion of a spatial window with a fixed width or region in terms of geographic elements shifting (sliding) over a path line. Within the spatial window, the prefetching mechanism incrementally updates the high-level ontology to ensure sufficient road knowledge ahead. The proposed approach is evaluated over a developed JAVA prototype *SmartMap*, and the result shows that efficient knowledge processing and spatial reasoning over dynamic data is achieved. Additionally, we provide a discussion about two alternative deployment choices over the proposed knowledge-spatial architecture, namely, client deployment and client-server deployment.

> RQ3: Can an ontology-based approach be utilised to ensure map data quality for road knowledge consistency?

For addressing this question, an ontology describing map quality violations is developed. Together with developed constraint rules, the occurrence of map data violation can be captured, providing necessary information to locate the troublesome data. With the graph-like spatial relationships among map objects, RDF graph aggregation and decomposition are applied to handle detected violations.

Contribution: We present a workflow for ensuring map data quality based on OWL 2 RL ontologies [93] and Datalog rules [79]. A Map Quality Violation (MQV) ontology and a set of constraint rules for violation detection are developed to record the occurrence of map data violations. We demonstrate violation handling strategies via violation tolerance and resolution. The performance of violation detection and the correctness of violation resolution is evaluated over the developed *SmartMap* application. By using the proposed approach, map data quality is ensured.

1.3 Publications

The work presented in this thesis has already been published as conference or demo articles. The publications building this thesis are outlined as follows:

1. **Haonan Qiu**, Adel Ayara, and Birte Glimm. *A knowledge-Spatial Architecture for Processing Dynamic Maps in Automated Driving*. In Proceedings of the ISWC 2020 Demos and Industry Tracks: From Novel Ideas to Industrial Practice colocated with 19th International Semantic Web Conference (ISWC 2020)

2. **Haonan Qiu**, Adel Ayara, and Birte Glimm. *Ontology-Based Processing of Dynamic Maps in Automated Driving*. In 12th International Joint Conference on

Knowledge Discovery, Knowledge Engineering and Knowledge Management (KEOD 2020), volume 2. (**Best Paper Award**, acceptance rate: 21%)).

3. **Haonan Qiu**, Adel Ayara, and Birte Glimm. *A Knowledge Architecture Layer for Map Data in Autonomous Vehicles*. In 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), pages 1–6. IEEE, 2020. (**Best Paper Nominated**, among some 570 conference papers, 40 papers are nominated)

4. **Haonan Qiu**, Adel Ayara, and Birte Glimm. *Ontology-Based Map Data Quality Assurance*. In The Semantic Web — 18th International Conference, ESWC 2021,Virtual Event, June 6-10, 2021, Proceedings, volume 12731 of Lecture Notes in Computer Science, pages 73–89. Springer, 2021. (Acceptance rate: 25%).

1.4 Thesis Structure

The thesis is structured in seven chapters, outlined as follows.

- *Chapter* 1 - *Introduction* prefaces the thesis covering the main research problem and challenges, motivation for the conducted work, research questions, and scientific contributions that address the research questions.

- *Chapter* 2 - *Background and Preliminaries* introduces the key concepts required to understand the work of this thesis. Initially, we provide additional background on the different levels of automation specified for autonomous vehicles. Next, we provide insights into HD maps. Finally, the preliminaries of knowledge representation technologies are described.

- *Chapter* 3 - *Related Work* examines current state-of-the-art approaches to provide the reader with a better comprehension of the work conducted in this thesis. First, ontology-based data integration approaches are investigated, followed by existing generic road environmental and map specific ontologies. Next, works regarding the ontology-based approach for reasoning and decision making are described. Finally, existing methods for spatial data quality are described.

- *Chapter* 4 - *Two-level Map Ontologies* describes an ontology-based approach for integrating low-level heterogeneous map data to a high-level generic map model and outlines a practical methodology to build the two-level ontologies.

- *Chapter* 5 - *Dynamic Map Processing* presents the utilisation of the developed two-level map ontologies and rules to process the map continuously for autonomous vehicles via incremental updates combined with a spatial sliding window. The deployment choices of the proposed solution are also discussed in this chapter.

- *Chapter* 6 - *Map Quality Assurance* describes a workflow using ontologies and rules to ensure map data quality; and also presents the rule template for violation detection and the violation tolerance based on RDF graph characteristics.

- *Chapter* 7 - *Conclusion and Outlook* finalises this thesis with a summary of the results and contributions to the problem of map modelling and processing for autonomous vehicles. The limitations of the presented approach are discussed, and an outlook on a possible direction for future research is provided.

Chapter 2

Background and Preliminaries

This chapter outlines the background of the work and the preliminaries of applied technologies conducted in this thesis. First, we introduce the different levels of automation as keyed by the German Association of the Automotive industry in Section 2.1. Secondly, we provide an insight into HD maps in Section 2.2 including a description of general HD map models. The comparison between two HD maps used in this thesis - an NDS map and a HERE HD Live Map is highlighted in this section, and a brief explanation of other Advanced Driver Assistance Systems (ADAS) related terms such as electronic Horizon. Map quality related background is provided at the end of this section. Finally, knowledge representation technologies to represent ontologies and reasoning, e.g., RDF, OWL, Datalog, SPARQL are described in Section 2.3.

2.1 Specification of Automation Levels

The Surface Vehicle Recommended Practice from the Society of Automotive Engineers (SAE) International provides a taxonomy describing the full range of driving automation levels for on-road motor vehicles [103]. It includes the terms and functional definitions for the advanced levels of driving automation. Figure 2.1 briefly summarises the levels of automation ranging from Level 0 (no automation) to Level 5 (full automation). At level 0, the drivers have to perform all the common driving tasks themselves. Whereas, at level 5, vehicles can complete an entire hand-off, driverless journey under any condition. The driver becomes a pure passenger, as neither driving ability nor a driving licence is required to use the vehicle. This level of automation creates opportunities for passengers who are unable to operate a vehicle independently (blind, disabled and older people). True level 5 automation, however, has not been reached until now (2021). Even Google's Waymo division, often considered as one of the most advanced researching groups in the field of self-driving cars, did not yet reach that point. Although Waymo's fully autonomous taxi service was opened to the public in October 2020 in a 50-square-mile area in Phoenix, the cars currently still have remotes that oversee the task of sending high-level instructions to help the vehicles react in complicated situations [117].

At level 1 (driver assistance), the systems provide adaptive cruise control, lane keeping, and emergency brake assistance to help with driving fatigue. These systems assist drivers and require the driver to control and keep a constant eye on the traffic. At level 2 (partial automation), the systems perform more complex manoeuvres. Several assistance systems are often combined to independently perform individual driving manoeuvres, such as braking automatically, accelerating and taking over the steering. The driver can hand over control of the vehicle during these

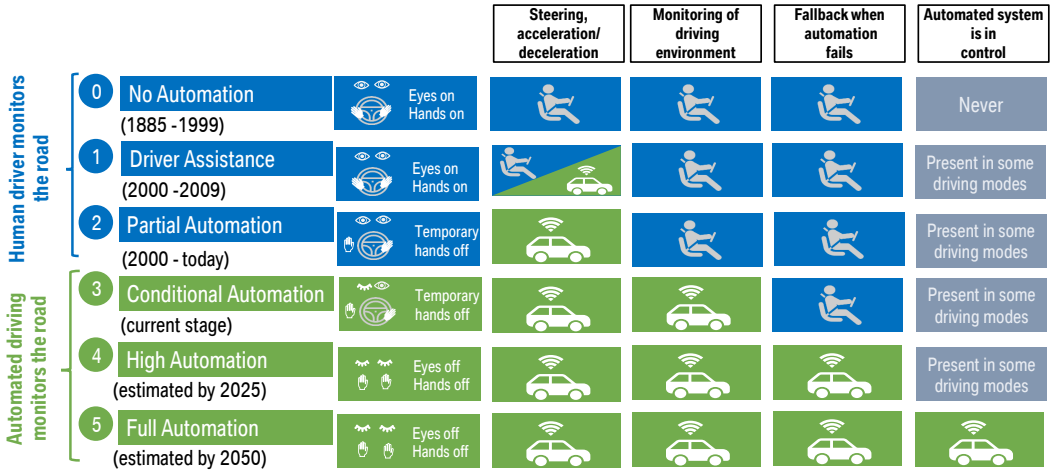


FIGURE 2.1: Overview of driving automation levels from Cavazza et al. (2019) adapted from SAE (2016).

manoeuvres, but must remain alert and be able to intervene at any time if something does not function as intended. Current vehicle models are often categorised as level 2. Examples of level 2 autonomous capabilities are Tesla Autopilot, Volvo Pilot Assist, Audi Traffic Jam Assist.

While level 2 is all about assistance systems, at level 3 (conditional automation), the car can drive autonomously under certain conditions (e.g., on highways). This includes the car's sensory equipment to work correctly and the road conditions ahead on the vehicle's track to be well known. In this case, the human driver does not have to be constantly supervising the car and the surrounding traffic. Instead, the car has to inform him when it identifies that safe driving conditions cannot be guaranteed any more. Then the driver is required to step in and take over the control of the wheel within seconds. In contrast to level 3, where the driver must take over the wheel again, level 4 (high automation) vehicles can complete entire journeys without human interaction. Google's Waymo division, ride-sharing companies like Uber and the car manufacturers, have reached or are currently aiming at the automation level 4. The driverless trucks by Einride also fall under this category since they do not have a cockpit, and a remote driver can engage and steer the truck from miles away.

To ensure the safety of automated driving, the onboard sensors of the vehicle (e.g., cameras, ultrasonic sensors, radar, and Lidar) may not be sufficient to detect all conceivable driving circumstances. There may be otherwise risky or unpleasant driving situations that may only be avoided adequately with foresight outside the sensor range of the car.

2.2 HD Maps

To address these issues, the autonomous vehicles also rely on an additional "virtual" sensor for their driving task, the so-called high-definition (HD) maps [161, 121].

This map provides detailed and critical information to the autonomous vehicles regarding their current surrounding traffic situation and environment. The provided information is highly accurate and geo-referenced up to the sub-meter level of precision. Current standard navigation map data (SD maps), on the other hand, only reaches a localisation accuracy of a few meters. As a result, the HD map may be seen as a highly precise virtual 3D representation of the real world [101]. That way, autonomous vehicles can compare their sensor readings with a virtual reference, facilitating the driving task.

2.2.1 HD Map Model

Ulbrich et al. (2017) proposed the functional system architecture of an automated driving system where HD maps are tightly associated with localisation functionality, interacting with the perception model and ultimately supporting the planning and control model (see Figure 2.2). The different levels of automated driving tasks require that the world is modelled at different levels of detail. In this figure, the terms *Road Level* relate to the road network topology, *Lane Level* to the semantic relationships among lanes and *Feature Level* to the metric properties used for localisation within a lane. Both HD maps and the vehicle's position are input to the perception module for world modelling, from which the planning and control module deduce actions and plans. Currently, GPS is one of the most widely used Global Navigation Satellite Systems (GNSS) for vehicle positioning, which consists of navigation satellites as radio signal sources, and a GPS receiver in the vehicle to receive the satellite signals.

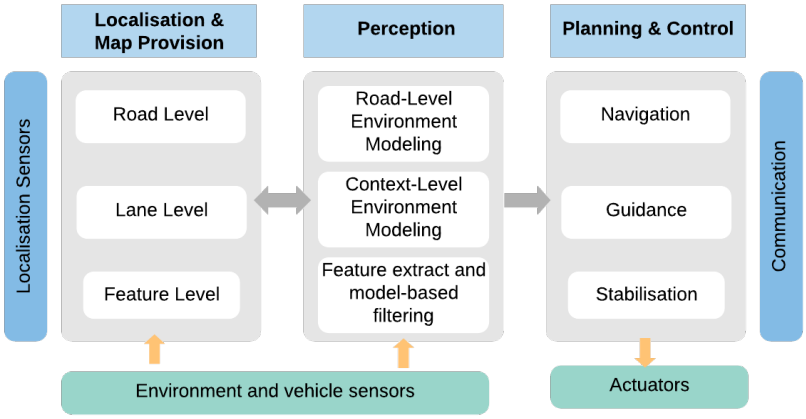


FIGURE 2.2: Functional system architecture of an automated driving system adopted from [175]

An HD map road model is used for strategic planning (navigation). It is modelled using an ordered sequence of shape points describing the geometry of a polyline that represents the course of a road [30]. Each road section has its start and end nodes, which belong to the intersection at the start and end of the road. When using shape points to define curved road geometry, an increase in the density of intermediate points can achieve better accuracy but requires storing a large volume of information.

The lane model is used for perception and tactical planning (guidance) considering the current road and traffic conditions. Lane models mainly include geometry

models, lane attributes and lane connectivity. The lane geometry model largely determines the accuracy, storage efficiency, and usability of the lane model [87], which covers the geometric structures of all lanes such as lane centre line, lane boundaries, and road markings and the underlying 3D road structure such as slope and overpass. Furthermore, it should be efficient for online calculations of coordinates, curvature, elevation, heading and distance. The lane attributes describe the lane type, road furniture and parking. Traffic regulations and relevant information may be embedded in the lane attributes, such as a lane type that may implicitly indicate the default speed limit. The lane connectivity describes the connection of the lanes or lane groups, which can be defined simply as a pair of predecessors and successors. Figure 2.3 shows the lane group connectivity using the lane connector IDs.

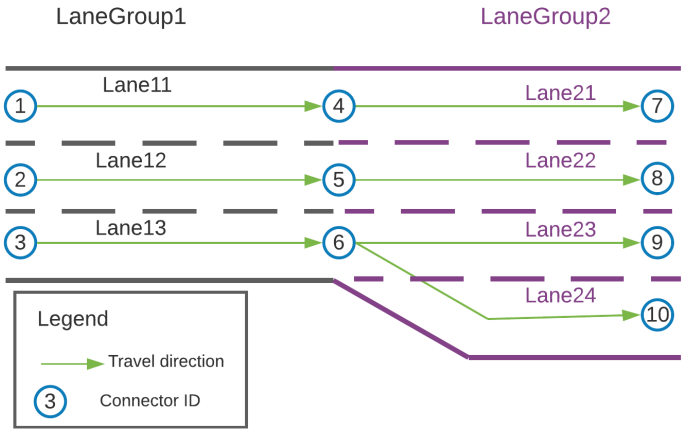


FIGURE 2.3: Example of lane group connectivity.

Navigation Data Standard and HERE HD Live Map

The *Navigation Data Standard* (NDS) is an HD map format developed by carmakers, mapping companies, research institutions, and universities [28, 165, 164]. The standard has been developed for size-efficient storage with performance requests on embedded devices. The HERE HD Live Map (HDLM), developed by the HERE company, is a cloud-based service to support connected ADAS and highly automated driving solutions provided by HERE. Although both maps are HD maps with *commonalities* at the high level in terms of the conceptual model, there are *differences* at the low level in the underlying logical and physical models regarding implementation.

Commonalities

At the conceptual level, NDS and HDLM share commonalities in the underlying models. The map data is partitioned into adjacent *tiles*. They form approximately rectangular territorial sections. The magnification *level* determines the edge length of a tile. *Nodes* within a map tile represent a point location on the surface of the Earth by a pair of longitude (y-coordinate) and latitude (x-coordinate) coordinates. *Links* represent a stretch of road between two nodes and are characterised by a line segment (corresponding to a straight section of the road) or a curve having a shape that is generally described by intermediate points called *shape points* along the link. Shape

points are represented by x-y coordinates as nodes, but shape points do not serve the purpose of connecting links, as do nodes. While link and road are synonyms, roads have the same meaning as in everyday language use, whereas links have attributes such as travel direction and types, such as highways. The shape points are ordered concerning the travel direction. The geometry of *Lanes* is described by shape points too. Lanes are connected via *lane connectors*. Each lane is represented by two *lane boundaries* with lane marking types (solid/dashed, single/double, etc.). Finally, lanes are organised into *lane groups* with link references. Table 2.1 summarises the commonalities between the models underlying the two map formats considering the topology, geometry, and attributes. Shape points, tiling and lane-road references are other aspects of shared commonalities.

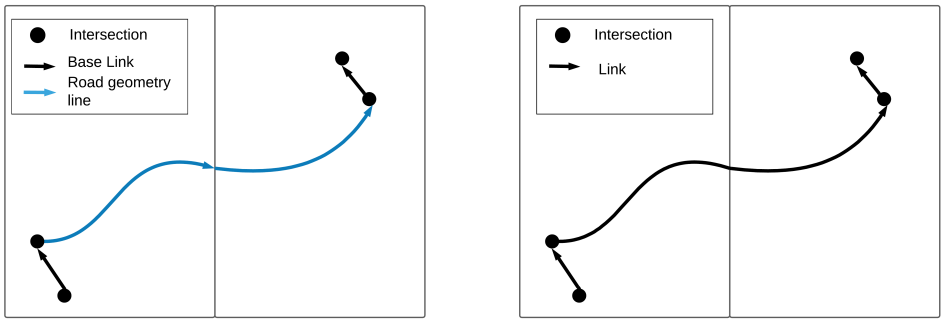
TABLE 2.1: The commonality between NDS and HERE Live Map

Category	Content	Description
Road Model	Road topology	It is a link and node network that models intersection-to-intersection connectivity.
	Road Geometry	A directed polyline connecting two nodes with optional intermediate shape points.
	Road attributes	They describe the link type, driving conditions, travel direction, speed limit etc.
Lane Model	Lane Topology	It is formed based on the lane connectors.
	Lane Geometry	A polyline with as a sequence of shape points representing a lane centre line. A lane is bounded with two-lane boundaries.
	Lane attributes	They describe the lane type, the lane number, the lane travel direction etc.
Other	Shape point	Each shape point has a coordinate with latitude and longitude values.
	Tiling	A geographical partition of map data.
	Lane-Road Reference	It provides references and range information between lanes and roads.

Differences
At the low level, however, there exists differences between NDS and HDLM in terms of *road model*, *lane model* and *formats*.

Road Model: In NDS, if a link crosses tiles, then it is represented as several *road geometry lines* (see Figure 2.4 (a)). Otherwise, the link is called *base link*. The reason is that the NDS provides self-contained data per tile, whereas, in HDLM, there is only the notion of the link (see Figure 2.4 (b)). Consequently, HDLM does not provide data in a tile-based self-contained fashion.

Lane Model: The way lanes are represented is one of the most significant differences between NDS and HDLM. Table 2.2 highlights the main differences between the two map formats. In general, lane connectivity is modelled with a lane connector. For NDS, each lane has a unique source lane connector and destination connector. Lane connection is formed by finding a lane whose source lane connector is equal to its destination lane connection. NDS assigns source and destination connectors in travel direction; the connectivity aligns with the travel direction (see Figure 2.5 (a)). However, lane connectivity is modelled differently in the HERE HD Live Map. First, each lane group has a unique start and end lane group connector. The lane connectivity can be formed based on lane group connectivity. Second of all, the connectivity of lane groups and lanes has to consider the travel direction. That is to



(a) NDS road representation (b) HERE Live Map road representation

FIGURE 2.4: The difference between the road representation of NDS and HDLM.

say if the travel direction is forward, then the orientation of the lane group is from start connector to end connector (see Figure 2.5 (b)). However, if the travel direction is backward, then the orientation of the lane group is from the end connector to the start connector (see Figure 2.5 (c)). The travel direction also affects the adjacent lane modelling.

TABLE 2.2: Lane model comparison of NDS and HERE HD Live Map

	NDS	HERE Live Map
Travel direction	No impact on lane model interpretation	Has impact on lane model interpretation
Lane connectivity	Lane connector source and destination	Lane group connector, lane connector
Left/right lane	Right to left, index 0 is the right most lane	Left to right, index 0 is the left most lane
Lane geometry	lane centre line	Lane Group Reference Geometry, lane geometry
Lane boundary	Parallel elements, sequential elements	Marking Element
Left/right lane boundary	Based on the lane number	Left/right lane boundary number
Lane type	Normal lane, entry lane, exit lane	Unknow, regular, drivable shoulder

For NDS, the numbering of lanes starts from right to left, and the rightmost lane is always the lane with the index of 0 (see Figure 2.5 (a)). For HDLM, however, the numbering of lanes starts from left to right regarding the lane orientation, and the interpretation of the rightmost lane depends on the lane travel direction. If the travel direction is forward, then the lane with the index of 0 is the left-most lane (see Figure 2.5 (b)). Otherwise, the lane with index 0 is the rightmost lane (see Figure 2.5 (c)).

Format: NDS uses SQLite as a file format and database engine. The data is stored in relational database tables with Binary Large Objects (BLOB) columns and is divided into building blocks as logical components; however, geographical partitioning by tiles is simultaneously applied. Its formal data description language is called Relational DataScript (RDS). Each building block addresses specific functional aspects.

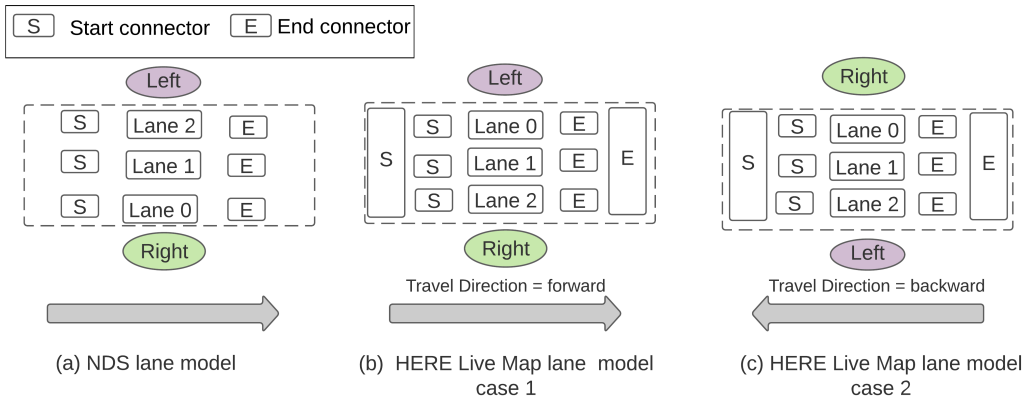


FIGURE 2.5: The difference between the lane model of NDS and HERE HD Live Map.

The HDLM is designed for providing a content service for AD applications [90, 131]. The schema for the HDLM map data model and the map data itself is published as files using the Protocol Buffer (ProtoBuf) data format. The data is composed of tiled map layers logically structured into Road Model and Lane Model.

TABLE 2.3: Format Comparison of NDS and HERE Live Map

	NDS	HERE Live Map
Version	2.5.4	3.0
Purpose	Size-efficient storage with performance requests on embedded Navigation systems	A content service providing information and services for AD applications.
Designed for	Embedded devices	Cloud
Storage format	BLOB	ProtoBuf
Schema format	Relational DataScript	ProtoBuf
Logical storage components	Building blocks	Layers
Structure	Routing Building Block, Lane Building Block	Road Model, Lane Model
Tools	SQLite, Zserio	HERE Content Platform APIs
Language	C++, Java, Python	Java, Scala

2.2.2 HD Maps with Advanced Driver Assistance Systems

Apart from HD maps, various other technical terminology have been used in the domain of map-based ADAS, such as the electronic Horizon (eHorizon) [51, 36] and the ADAS Horizon [153, 154]. All of those terms are often used synonymously. This makes it especially difficult to differentiate sharply between their conceptual emphasis. The key common component in all those systems is a map data source, facilitating the general driving task. This could be a normal navigation map designed for human drivers or HD maps specifically created for autonomous vehicles. The typical navigation map is also referred to as SD (Standard Definition) map. The electronic Horizon (eHorizon) and the ADAS Horizon are two different terms referring to the same general technical concept. The eHorizon emphasises the extraction and preparation of relevant traffic data out of a digital map specifically for the vehicle's

current route. This map data is further enhanced through the readings of the vehicles' onboard sensor, e.g., its current location and driving speed. The HD map can be used as such a map data source. The eHorizon is designed for providing the specific traffic information to a driver, and it cannot assume a route would be given beforehand. Instead, it utilises a so-called Most Probable Path (MPP) calculation to anticipate the most likely driving path [36, 154]. This is in contrast to autonomous vehicles, which always requires a given specific navigation route to perform their driving task.

The Local Dynamic Map (LDM) is an ADAS standardised by the European Telecommunications Standards Institute (ETSI) [69]. Unlike the eHorzion approach, which enhances the car's navigation data through additional onboard sensor readings, LDM enhances the data through direct ad hoc communication between the vehicles, allowing them to share information about the current traffic situation, such as their location, speed, and driving direction. LDM classifies traffic objects into four layers based on their degree of timely relevance: i) permanent static data, such as map data provided by a map supplier; ii) transient static data, such as speed limits; iii) transient dynamic data, such as weather conditions and traffic information; and iv) highly dynamic data, such as the speed and driving direction of surrounding vehicles. The data contained in layer iv) is not stored in the HD map as it has a very short time relevance and is only relevant for the specific vehicle. As a result, it is stored locally in the vehicles, thus the name of Local Dynamic Map.

2.2.3 HD Map Quality

HD maps are safety-critical features in self-driving vehicles that combine data from multiple sources to precisely locate the car's position. Creating an error-free HD map is very challenging because the map environment is complex and dynamic, and assuring the quality of large datasets from many sources is a complex task. There are many reasons for poor data quality. They can be broadly classified into data entry, data processing, data integration, data conversion and data age (loss of reliability/relevance over time) [57]. Typical faults are distinguished between a fault in the map topology (structure), geometry and attributes. Topological faults are related to the map object connectivity. Geometric faults are related to the shape or the geographic placement of these map entities. We next briefly explain each fault with some examples.

Topological Fault

The correct representation of road network and lane connectivity in the map is essential for optimal path planning. For example, a missing connection between a road may cause the path planner to choose a sub-optimal path. In turn, a connection between two roads in the navigation map that does not exist in reality may result in an impossible path. In this case, the driver may be misled by the navigation assistant and cause hazardous driving situations.

Geometric Fault

In the Geographic Information System (GIS) domain, absolute and relative accuracy are distinguished [34]. The former describes the accuracy of the geographic feature for a global reference coordinate system, whilst the latter describes accuracy relative to other features. Roads and lanes with low relative accuracy result in poor shape

definition. For example, the curve warning related ADAS is directly affected by the low relative accuracy of the road. In case of low absolute and high relative accuracy (road offset), the map-matching algorithm may not choose the right road candidate, especially in dense road network areas.

Attribute Fault

The speed limit can be displayed to the driver in an over-speed prevention (OSP) system or used to set the vehicle's curse speed in contextual ACC applications. The map stores the speed limit as an attribute for each road. This attribute originated from direct surveys or is inferred based on the road class, the number of lanes and the area (inside or outside built-up area). Smart cameras are nowadays capable of detecting speed signs in real-time, however, the challenge resides in determining whether the detected sign applies to the vehicle. Speed limits indicated by traffic signs may be dedicated to one particular lane (e.g., for a motorway exit), to a class of vehicles (e.g., trucks, vehicles towing caravans, buses) or to special weather conditions. Inappropriate estimation of the speed limit by the car would bother the driver and decrease the perceived vehicle's quality.

In navigation maps, roads are assumed to be drivable in both directions unless a dedicated attribute is associated with the road. Similarly, attributes are defined to establish some traffic restrictions (e.g., trucks, pedestrians and maximum height). The planner uses the attributes to exclude wrong-way roads and roads that do not comply with the vehicle type. Faults in these attributes may cause the path planner to ask the driver to take a forbidden route. This may have severe consequences, especially for large good vehicles that cannot manoeuvre easily.

2.3 Knowledge Representation

Knowledge representation (KR), as an area of artificial intelligence (AI), combines the multidisciplinary fields of ontologies to define and structure things, logic to formulate logical inference and knowledge and computation to allow computer systems to perform intelligent tasks [179]. Description logics (DLs) are a family of formal knowledge representation languages, which are the core of the representation language of the Semantic Web [21]. Datalog as a rule language originated from deductive databases [10], but nowadays, it is also used as a rule language in Semantic Web applications [157, 79]. The following sections explain aspects of the Semantic Web technologies and Datalog rules.

2.3.1 RDF and RDFS

The Resource Description Framework (RDF) is a generic data model for interchanging data on the Web recommended by the World Wide Web Consortium (W3C).[1] In RDF, data is represented as triples consisting of subjects, predicates, and objects, which can be combined to directed graphs composed of vertices representing subjects and objects and edges representing predicates. Formally, an RDF triple is defined as follows:

Definition 2.3.1 (RDF Triple [16])**.** Let I, L, and B be pairwise disjoint infinite sets of IRIs, literals, and blank nodes, respectively. A tuple $(s, p, o) \in I \cup B \times I \times (I \cup L \cup B)$ is called an RDF *triple*, where s is the *subject*, p is the *predicate*, and o is the *object*.

[1] https://www.w3.org/RDF/

Definition 2.3.2 (RDF Graph [50])**.** An RDF graph G is a finite set of RDF triples and induces a set of *vertices* $V = \{s \mid (s,p,o) \in G\} \cup \{o \mid (s,p,o) \in G\}$.

An example RDF graph representing the relationship between a lane and a shape is shown in Figure 2.6. The resource `lnds:lane_155` is of type lane. This is represented by the `rdf:type` property which connects, in this case, two resources, i.e., `lnds:lane_155` acting as a subject and the `lnds:Lane` acting as an object. Similarly, the resource `lnds:pos_155_0` is declared as type `lnds:ShapePoint` through the `rdf:type` property. In addition, this RDF graph represents that the `lnds:lane_155` has shape point `lnds:pos_155_0`.

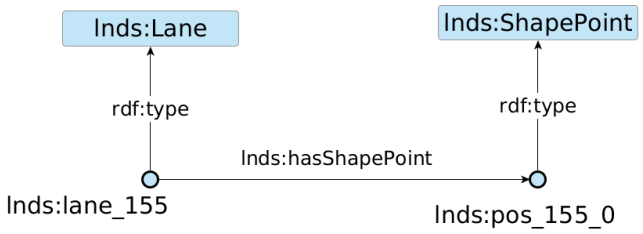


FIGURE 2.6: Example of an RDF graph representing the relationship between a lane and a shape point.

RDF can be serialized in different formats, such as RDF/XML[2], Turtle[3], RDFa[4] or JSON-LD.[5] Every serialisation has its pros and cons, depending on the use case. Throughout this document, the Turtle notation is used because it favours the readability of RDF documents.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix lnds:<http://www.bmw-carit.de/Foresight/Map/Ontologies/Low/NDS#>

lnds:lane_155 rdf:type lnds:Lane;
              lnds:hasShapePoint lnds:pos_155_0.
lnds:pos_155_0 rdf:type lnds:ShapePoint.
```

LISTING 2.1: Turtle serialisation of the RDF graph in Figure 2.6.

RDF has its limitations that it can not describe taxonomies of classes and properties. RDFS (RDF Schema) is an extension of the RDF vocabulary which provides the extra expressiveness [86]. It extends RDF by adding constructs such as `rdfs:Class` , `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain`, `rdfs:range` to mention the most important ones.

2.3.2 Ontologies and the Web Ontology Language

Ontologies serve as the backbone of the Semantic Web. A number of definitions of the term ontology exist within computer science. A widely used one is given by Studer et al. [168], where an ontology is defined as "a formal, explicit specification of a shared conceptualisation." It specifies a vocabulary used to describe a specific domain and a set of assumptions regarding the intended meaning of the words contained in it [85]. Typically, an ontology should capture a shared understanding of

[2]https://www.w3.org/TR/rdf-syntax-grammar/
[3]https://www.w3.org/TR/turtle/
[4]https://www.w3.org/TR/rdfa-syntax/
[5]https://www.w3.org/TR/json-ld/

a domain of interest and provide a machine interpretable formalisation [173, 105]. Hence, an ontology enables one to share and reuse knowledge across heterogeneous applications.

Ontologies are based on Description Logics (DLs) which is a formal language for knowledge representation [21]. A DL enables to model Concepts, Roles and Individuals. Concepts, also called classes, can be seen as sets of objects. Roles represent binary relations between individuals. Objects correspond to individuals. Typically, a DL knowledge base comprises the T(erminological)Box and the A(ssertional)Box. The TBox consists of a set of assertions on concepts and roles, while the ABox consists of assertions about individuals using the terms of the TBox. Note that DLs are under Open World Assumption (OWA), meaning that if something cannot be proven to be true, then it is not automatically assumed false.

DLs provide the semantics for the Web Ontology Language (OWL). The first working drafts of OWL were published in 2002, known as OWL 1, which became a formal W3C recommendation in 2004. Later in 2009, OWL 2 was released and became a W3C standard. In 2012, the specification of OWL 2 was issued [147]. The semantics of OWL 2 can be found in [180]. OWL 1 DL corresponds to the description logic $\mathcal{SHOIN}(\mathcal{D})$, while OWL 2 DL corresponds to description logic $\mathcal{SROIQ}(\mathcal{D})$ with some restrictions placed on ontology structures. For instance, in OWL 2 DL, number restrictions cannot be used with transitive properties [180]. OWL 1 and OWL 2 both have a number of profiles. As stated by Horrocks et al. [95], a profile of OWL 2 "is a trimmed down version of OWL 2 that trades some expressive power for the efficiency of reasoning". For OWL 1, there are three profiles, namely OWL Lite, OWL DL and OWL Full, while OWL DL is broadly used. The tractable profiles of OWL 2 include OWL 2 QL, OWL 2 EL, OWL 2 RL.

OWL 2 EL is ideal for reasoning over large TBoxes, while OWL 2 QL is suitable for query rewriting techniques [95]. OWL 2 RL profile corresponds to a naive intersection between the description logic $\mathcal{SROIQ}$ and Datalog and is very closely related to so-called Description Logic Programs (DLP) [83]. OWL 2 RL allows standard inference types to be implemented with polynomial-time algorithms using rule-based reasoning engines in a relatively straightforward way.

2.3.3 Datalog

Rules can be used to axiomatise the semantics of a particular Web ontology language, e.g., OWL 2 RL, via a static set of rules or, using Datalog rules, users can create custom rule sets. In order to define *Datalog rules*, we fix countable, disjoint sets of *constants* and *variables*. A *term* is a constant or a variable. An *atom* has the form $p(t_1, \ldots, t_k)$, where p is a k-ary predicate and each t_i, $1 \leq i \leq k$, is a term. We focus on unary and binary atoms only (i.e., $1 \leq k \leq 2$), which correspond to classes and properties of the ontology, respectively. An atom is *ground* if it does not contain variables. A *fact* is a ground atom and a *dataset* is a finite set of facts, e.g., as defined in an ontology. A Datalog *rule* is a logical implication of the form

$$H_1, \ldots, H_j \leftarrow B_1, \ldots, B_k, \tag{2.1}$$

where each H_i, $1 \leq i \leq j$, is a *head* atom, and each B_l, $1 \leq l \leq k$, is a *body* atom. A *negative* body atom has two forms: i) `NOT` B_i where $1 \leq i \leq k$; ii) `NOT EXISTS` $v_1, \ldots,$ v_j `IN` $(B_1, \ldots, B_i)$, where $v_1, \ldots, v_j$ are variables appear in $(B_1, \ldots, B_i)$ with $1 \leq i \leq k$. An *aggregate* is a function that takes a multiset of values as input and returns a single value as output. An aggregate atom has the form `AGGREGATE`$(B_1, \ldots, B_i$ `ON` $x_1, \ldots, x_j$

BIND $f_1(e_1)$ AS r_1 ... BIND $f_n(e_n)$ AS r_n), where $1 \leq i \leq k$, $j \geq 0$, $n \geq 0$, and $x_1, \ldots, x_j$ are variables that appears in $B_1, \ldots, B_i$. For $1 \leq v \leq n$, every f_v is an aggregate function (e.g, MIN or MAX), every e_v is an expression containing variables from $B_1, \ldots, B_i$, and every r_v is a variable that does not appear in $B_1, \ldots, B_i$. Aggregation is an extension of standard Datalog.

A rule r is *safe* if variables that appear in the head or in a negative body atom also appear in a positive body atom. A Datalog *program* is a finite set of safe rules. If all body atoms of a rule are true, the consequences (head) of the rule are true. The main computational problem in Datalog systems is answering queries over the facts that logically follow from the explicitly stated facts and a Datalog program. As contrary to DLs, the semantics of Datalog is under Closed World Assumption (CWA) meaning the only facts are those that the rules can imply. The process (and the result) of computing and storing all facts implied by a dataset and a Datalog program is called *materialisation*. The rules in a Datalog program can be stated in any order. Furthermore, the Datalog program is guaranteed to terminate with finite data. Datalog is P-complete for data complexity; that is to say, entailment can be computed in polynomial time with respect to the size of the input facts [81]. The reasoning of Datalog without negation and aggregation is monotonic, that is to say, that new facts can only produce additional knowledge. Datalog with negation [110] or aggregation, however, exploit nonmonotonic reasoning, meaning new facts will sometimes invalidate previously derived knowledge.

Additionally, stratification is required for a Datalog program with negation or aggregation to ensure that the semantics are well-defined [139, 70], which may be verified using a *dependency graph*. The dependency graph G_p of a Datalog program P is a directed graph that consists of *nodes* for each predicate in P; directed edges from node q to node p, if there is a rule with q in the head and p in the body [162]. An edge is *negative*, if p occurs in a negated or aggregate atom; otherwise, the edge is positive. A program is called *hierarchic*, if G_p does not contain cycles, otherwise it is called *recursive*. A program is called *stratified*, if cycles in the G_p only consist of positive edges [82]. As an example, consider a stratified and recursive program shown in Example 2.2 in which the cycle consists of only a positive edge in the dependency graph (see Figure 2.7). While Example in 2.3 demonstrates a recursive program that cannot be stratified due to the presence of a dependency graph cycle with negative edges (see Figure 2.8).

$$\begin{aligned} &\mathsf{goodPath}(x,y) \leftarrow \mathsf{path}(x,y),\ \texttt{NOT}\ \mathsf{trafficJam}(y). \\ &\mathsf{goodPath}(x,z) \leftarrow \mathsf{goodPath}(x,y),\ \mathsf{goodPath}(y,z). \end{aligned} \tag{2.2}$$

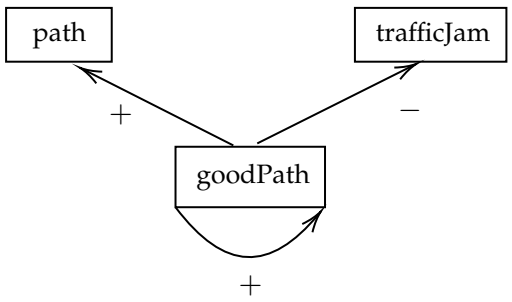


FIGURE 2.7: Dependency graph for stratified program.

$$\begin{aligned} \mathsf{hasLeft}(x,y) &\leftarrow \mathsf{hasNeighbouringLane}(x,y),\ \texttt{NOT}\ \ \mathsf{hasRight}(x,y). \\ \mathsf{hasRight}(x,y) &\leftarrow \mathsf{hasNeighbouringLane}(x,y),\ \texttt{NOT}\ \ \mathsf{hasLeft}(x,y). \end{aligned} \tag{2.3}$$

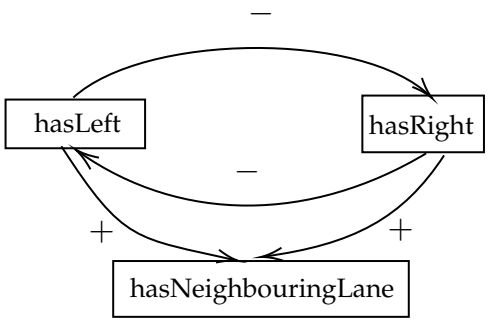


FIGURE 2.8: Dependency graph for non-stratified program.

2.3.4 SPARQL

SPARQL is a query language for RDF. A SPARQL query is syntactically represented by a block consisting of a query form (`ASK`, `SELECT`, `CONSTRUCT` or `DESCRIBE`), zero or more dataset clauses (`FROM` and `FROM NAMED`), a `WHERE` clause, and possibly solution modifiers (e.g., `DISTINCT`). The `WHERE` clause provides a graph pattern to match against the RDF dataset constructed from the dataset clauses.

The core idea of SPARQL is querying over simple graph patterns [174]. As explained by Seaborne and Prud'hommeaux [160], "Most forms of SPARQL queries contain a set of triple patterns called a basic graph pattern. Triple patterns are like RDF triples except that each subject, predicate and object may be a variable. A basic graph pattern matches a subgraph of the RDF data when RDF terms from that subgraph may be substituted for the variables." Hence, SPARQL query answering is generally about query pattern matching. Listing 2.2 shows an example of the use of SPARQL to query the lane type of which the current location is on and the travelled distance and remaining distance of a vehicle in that lane.

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix high: <http://www.bmw-carit.de/Foresight/Map/Ontologies/High#>

SELECT ?laneType ?td ?rd
WHERE {
        ?cl rdf:type high:CurrentLocation;
            high:isOnLane ?lane;
            high:travelledDistance ?td;
            high:remainingDistance ?rd.
        ?lane   rdf:type ?laneType.
}
```

LISTING 2.2: Example of a SPARQL query.

While some SPARQL features can be expressed in Datalog as shown in Table 2.4, others, such as `OPTIONAL`, subqueries, `LIMIT`, `OFFSET`, or `ORDER BY` are some SPARQL features that cannot. Additionally, although some SPARQL queries can be stated in Datalog, the semantics are different because Datalog assumes set semantics while SPARQL assumes multiset semantics.

TABLE 2.4: The correspondence between SPARQL and Datalog [13].

SPARQL	Datalog
Basic Graph Patterns	Conjunctions of atoms
Path expressions	Recursive rules
Union	Conjunction of several rules
Minus and Not Exists	Stratified negation
Filter	Filter
Bind	Bind
Aggregates	Stratified aggregates

2.3.5 Triple Store

"Triple Store" is the common name given to a database management system for RDF Data. The common techniques used in triple stores are *storage*, *indexing*, *join processing* and *query processing*. Data storage refers to how data are represented in memory. For example, a triple table is used to store an RDF graph as a single ternary relation, while the vertical partitioning approach [7] uses a binary relation for each property whose tuples encode subject–object pairs for that property. Indexing enables efficient lookup operations on RDF graphs such as triple indexes, entity-based indexes and property-based indexes. Natural joins are necessary for translating the query into "physical operators" that implement algorithms for efficient evaluation. Query processing concerns the features supported by SPARQL engines such as property paths. We refer the interested reader to the survey presented in [12], where a comprehensive review of triple stores with related techniques are provided.

Additionally, triple stores can manage background knowledge, which is typically represented using an OWL 2 Ontology amd may be complemented with SWRL [94] and Datalog. The background knowledge allows a triple store to answer queries with enriched results with inferred knowledge. Materialisation is a commonly used for answering queries over a Datalog program and a dataset (e.g., GraphDB [80] and Oracle's RDF store [146]). It is beneficial in situations where the performance of query answering is critical, because the consequence of the program and the dataset are precomputed and explicitly stored. Queries can then be evaluated without any further reference to the program.

RDFox is an in-memory triple store that supports Datalog reasoning [142]. The RDF graph is stored as a six-column triple table implemented as a linked list, which stores identifiers for subject, predicate and object for each triple, as well as three pointers in the list to the next triple with the same subject, predicate and object. The in-memory indexes [138] support efficient parallel updates, which is the key for fast materialisation. RDFox also supports incremental reasoning, meaning that changes to the input data do not require recomputing the materialisation from scratch. The incremental reasoning is suitable in scenarios where the input data changes frequently.

Chapter 3

Related Work

This chapter outlines the state-of-the-art for the work conducted in this thesis. Relevant approaches related to the research problem as well as to the research questions are investigated. Section 3.1 examines approaches related to semantic data integration and ontologies for road environmental modelling and maps. Next, in Section 3.2 we review existing works using ontology-based approaches for reasoning and decision making. Finally, a review of the current approaches for spatial data quality is carried out in Section 3.3.

3.1 Map Data Integration

Studies of ontology-based data integration approaches have been conducted and applied in multi-disciplinary engineering environments (MDEE), where a key challenge consists of obtaining a common view of the heterogeneous data sources [65]. To answer our research question R1, we first present the related work of Ontology-Based Data Integration (OBDI), then we analyse the existing ontologies related to the generic road environmental model and specific maps.

3.1.1 Ontology-Based Data Integration

Ontology-Based Data Integration (OBDI) is one of the most common techniques for semantic data integration since ontologies provide a semantic representation of the domain [52, 42, 38, 48]. In general, the OBDI approach comprises three components: i) the ontology for representing the knowledge about the domain; ii) the data source which typically contains the data of the domain; iii) the mappings between the two components [52]. Cruz et al. [49] discuss different views of the use of ontologies for semantic data integration: i) Single ontology approach. All sources are directly related to a shared global ontology; ii) Multiple ontology approach. Each data source is described by its local ontology separately; iii) Hybrid ontology approach. A combination of the single ontology approach for describing each data source in the domain with mappings to a shared ontology.

Global-as-View (GAV) was proposed by Ekaputra et al. [64] as an additional OBDI approach. The central concept of the GAV approach lies in the global ontology definition, which is similar to the hybrid OBDI. GAV OBDI, however, does not require re-development of existing local ontologies due to inter-ontology transformation definitions between the local and global ontologies. The advantages of this approach is, as pointed by Ekaputra et al. [64], that data sources can be easily added with moderate effort (i.e., mappings between the local ontology representing the new data source and the global ontology). Additionally, more complex relations beyond ontology representation capabilities are possible for defining mappings [114]. Dubinin et al. [58] utilized GAV OBDI for integrating information across data

sources in the automation domain. For instance mappings, they defined that two or more objects in different ontologies are considered the same if certain property values of these instances are the same. This method is referred to as *property value matching*. For the local ontologies to the global ontology transformation, they introduce the eSWRL transformation language as an extension of SWRL [94] for RDF-to-RDF transformation using a Prolog reasoner.

Ekaputra et al. [65] studied the use of Semantic Web technologies to support data change management within MDEE, where data changes in one engineering discipline need to be validated and propagated to other disciplines. GAV OBDI was used to represent the heterogeneous data as local and global ontologies and developed both local and global ontologies independently of each other. SPARQL `CONSTRUCT` queries were used to transform, validate and propagate changes between local ontologies and the global ontology. The Extract, Load, and Transform (ELT) method is used to convert data represented in spreadsheets into the developed local ontologies. All data are stored using an in-memory store, and accessed via custom APIs from software applications.

Imran and Young [102] demonstrate the potential of formal reference ontologies to support interoperability. Compared to the previously mentioned two approaches, they used a Common Logic-based Knowledge Frame Language framework (KFL) to define concepts in a multi-layered ontology instead of RDF and OWL. They argue that KFL is more expressive and has more powerful reasoning capabilities than OWL and SWRL rules. The ontology transformation, however, is achieved via arbitrary code.

Lin and Harding [120] used ontologies to support collaboration of engineers involved in a manufacturing system engineering process. GAV OBDI is used to develop independent local ontologies and map them to the global ontology. The mapping between local ontologies and the global ontology is done via RDF property mapping where classes, properties and instances of different ontologies are linked via RDF properties, e.g., `owl:sameAs`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, and `owl:equivalentClass`. The transformation is also done via arbitrary transformation code.

Table 3.1 summarises the overall technology options used in OBDI approaches, including the mentioned approaches above. The investigated approaches have shown the feasibility of using GAV OBDI to solve the integration problem. We notice, however, some limitations and drawbacks. First, the development of the local ontologies and the global ontology is isolated in their work. There is no general methodology for guiding the development process of building the local ontologies and the global ontology. Second, we noted that the mentioned approaches were applied for static input data. However, it is not the case in the autonomous driving domain, as the input data is dynamic during run time. Additionally, the local ontologies enrichment and RDF data aggregation from local ontologies to the global ontology are not considered during the transformation. Therefore, a novel approach for integrating map data to provide a generic map model for autonomous vehicles has to be developed.

3.1.2 The High-level Road Environment Model

Generic road environment modelling has been studied both in the Intelligent Transportation Systems (ITS) community [109] and automotive domains [107]. We present the related work based on these two categories.

TABLE 3.1: Technical realization of OBDI adapted from [64]

OBDI Element	Technical Options
Language Framework	RDF, OWL2, Topic Maps, Common Logic (CL)
Data Acquistion	ETL, ELT, OBDA, Manual
Mapping	RDF Property, URI/GUID Matching, Property Value Matching
Transformation	SILK, SPARQL Construct, Code, Reasoner/Rule Engine
Data Storage	Triplestore, In-memoery/file-based, RDBMS, Others
Data Access	SPARQL Endpoints, Custom APIs, Custom GUIs, Stream Data Engine

In the ITS community, one of the early works that used ontologies to model the road network was done by Lorenz et al. [125], where the Ontology for Transportation Networks (OTN) was presented. The OTN contains a road network model, the set of road segments between the road junctions, attributes such as driving direction, the number of lanes etc. However, the OTN ontology has not been updated since 2005, the date it has been published. The km4City ontology [29] and iCity ontology [134] are both parts of projects related to smart cities, where the transportation network is modelled. The presented transportation network ontologies are based on the OTN ontology.

In automotive domains, the road environment model act as a generic abstraction layer for autonomous driving functions [159]. Hummel et al. [100] proposed a road network ontology for vehicle situation understanding. The proposed ontology introduces the concepts of road networks (roads, lanes, dividers, road markings and junctions) and are used to complement vision sensors and digital maps to retrieve relevant information about intersections.

The representation of road intersection networks through ontologies was introduced by Regele [151]. It was used to solve the traffic coordination problem of autonomous vehicles. The approach uses a hierarchical world model and distinguishes between a low-level model containing detailed geometry data and a high-level model for driving functions at the high level. The proposed abstracted traffic model consists of a graph-like network of connected lanes, with vehicles and objects attached to specific positions on the lane.

To build a knowledge base for smart vehicles and implement different types of ADAS, Zhao et al. [188] proposed three ontologies: a map ontology, a control ontology and a car ontology. The map ontology is used to describe road networks such as road, intersection, lane, and traffic light information, etc. The control ontology is used to represent driving actions and paths of autonomous vehicles. The car ontology describes the types of vehicles and devices installed in a car.

To facilitate the automatic use case generation for the automated vehicle in highway scenario, a highway ontology was proposed in the work of [43]. The ontology involves four main concepts: RoadPart, RoadWay, Zone and Equipment. Concepts related to lanes such as MainLane, EntranceLane and ExitLane are defined under Roadway.

We value the existing road environmental models as a good domain knowledge input for our work. The presented models provide background knowledge of the road-level model, however, the detailed lane-level information is missing, which is particularly required to support autonomous driving functions. We considered this limitation and proposed an ontology to represent the generic high-level road model.

3.1.3 The Low-level HD Map Model

As for low-level map models, there are existing works representing fundamental geospatial data in an ontology and the specific map-related ontologies. A geospatial ontology was created in the work of [23] from geographic data stored in spatial databases, focusing on representing geographic location and geometry. The ontology contains geometric concepts such as longitude and latitude coordinates, point, line, and polygon.

The Open Geospatial Consortium (OGC) has defined the GeoSPARQL standard [166] to represent and query geospatial data in the Semantic Web. This standard defines a small topological ontology in RDFS/OWL to represent geospatial data, which uses three top-level classes to represent spatial objects' features and geometries. A geographic query language is also proposed to support queries related to geometries of the spatial objects, e.g., the calculation of distance between two spatial objects.

OpenStreetMap (OSM) is a collaborative project for creating and distributing free geographic data about the world [3]. In this project, maps are stored in OSM files, which consist of elements and tags. Elements like node, way etc., represent geographic entities like buildings, roads etc. Characteristics of these elements are represented using tags like street-type, speed-limit, amenities etc. An ontology called OSMonto [45] is proposed to model the hierarchy of OSM tags in order to enable the integration of OpenStreetMap into the Semantic Web. A rule-based framework for creating instance data from OSM was proposed by Eiter et al. [62], where Datalog rules are used to extract instance data from an OSM database.

Local Dynamic Map (LDM) is used in cooperative ITS systems. Eiter et al. [63] proposed a Local Dynamic Map (LDM) ontology for integrating data from other related domains like weather. The LDM ontology contains classes such as GeoFeature for representing the GIS aspects of the LDM including point of interests (POIs) and the road networks, Geometry for geometrical representation of features. OWL2 QL [39] is used to represent the LDM ontology for ontology-based data access (OBDA) [156] combined with the OWLGRES 0.1 reasoner [167].

The existing ontologies do not contain HD map-related information, particularly lane models and the relationship between roads and lanes. Nevertheless, the presented ontologies were a source of valuable inputs for designing the low-level HD map ontologies in this thesis.

3.2 Ontology-Based Approaches for Reasoning and Decision Making

A knowledge-enabled robot programming paradigm is used in robotics to separate knowledge from the program and modularise it into small broadly applicable chunks [27], such as *there is an upcoming exit*, *current lane is the left most lane*, *change lanes to the right*, etc. For example, the program would ask: *How should the vehicle reach the exit lane?* A reasoning mechanism would then collect the relevant knowledge units and combine them in order to propose the appropriate steps. The advantage of knowledge-based approaches to robot programming is that these knowledge pieces apply to many applications and can dramatically accelerate the realization of new robot applications [145].

One of the keys to the success of knowledge-based approaches in autonomous robotics is the use of ontologies as *enabler* to help the robot to understand and reason about its environment when executing tasks. There are nine functional capabilities

that every autonomous robot should exhibit [115]: *recognition and categorization*, *decision making and choice*, *perception and situation assessment*, *prediction and monitoring*, *problem-solving and planning*, *reasoning and belief maintenance*, *execution and action*, *interaction and communication*, *remembering*, *reflection, and learning*.

Reasoning essentially allows a robot to expand its knowledge state, drawing conclusions from other beliefs or assumptions the robot already maintains. This requires the existence of a representation of beliefs and the relationships among them. A common formalism used to encode such knowledge is first-order logic (FOL). Ontologies are often written in languages based on less expressive formalisms than FOL (e.g., DLs) in order to reduce the computational cost of inference. These formalisms allow the use of different sorts of reasoning, such as: deductive or inductive. Note that reasoning is not only relevant to infer new beliefs (monotonic) but also to decide whether to hold existing ones (belief maintenance, non-monotonic). Such belief maintenance is essential for dynamic environments in which the situation may change. We refer the interested reader to the surveys done by Gayathri and Uma [76] and Olivares-Alarcos et al. [145] for an in-depth understanding.

In this section, we first describe the state of the art concerning ontology-based reasoning and decision making in the autonomous vehicle domain. Then, we review the current status and limitations of streaming reasoning.

3.2.1 Ontology-based Approaches for Autonomous Vehicles

Ontology-based approaches have been applied to enable autonomous vehicles' situation awareness and decision makings. We are particularly interested in the interaction between the road environment model and decision making via reasoning and the treatment of environmental dynamics. Most of the approaches used ontology to model and understand the road networks from the point of view of the subject vehicle. Later, logic axioms or rules are used to understand and reason over the interaction between the road environmental and dynamic entities.

One of the first works exploiting DL for road network modelling was done by Hummel et al. [100]. They developed a Road Network Knowledge Base interpreting data from onboard vehicle sensors. The TBox describes general knowledge about road networks. At the same time, the ABox captures partial information about a particular road or intersection acquired from a video camera, a digital map and positioning devices. The reasoning is achieved over the knowledge base via logic axioms. Their studies argue that deductive reasoning, which is monotonic, should be combined with non-monotonic reasoning to deal with real-world scene interpretation.

Hülsen et al. have proposed an ontology to describe traffic intersection situations for ADAS [99]. Combined with traffic rules, infrastructure elements like roads, lanes, traffic lights etc., are modelled in this ontology. Logic axioms in the TBox and rules are used for infrastructure reasoning, inferring relations and concept memberships and primarily deriving logical conflicts in driving paths and right-of-way dependencies. A real-time implementation of the proposed approach in simulated environments was successfully performed [98], in which the RacerPro reasoner [88] was used to perform reasoning tasks. The ontology was represented using DL, and the rules are modelled using the RacerPro rule language (based on SWRL syntax).

Inspired by the previous works, Armand et al. [17] also presented an ontology-based approach for providing context awareness to driving assistant systems such that the vehicle can perform human-like reasoning about the driving environment using the information perceived through map data and sensors. The developed

ontology considered the static entities (e.g., entities in the map data) and the dynamic entities (e.g., mobile entities perceived in real-time during driving). They used SWRL rules which allow defining complex and expressive rules because they argue that basic DL axioms are not expressive enough and only enables the definition of basic class equivalences. The Pellet reasoner was used for inferences and reasoning is carried out at every 0.5 s. Care was taken to keep the ontology simple with a reasonable amount of rules for short reasoning times.

Morignot and Nashashibi [136] proposed an ontology-based approach to relax traffic regulation for autonomous vehicle assistance. The proposed ontology represents the vehicle, the road infrastructure and the traffic regulations. The rules are used to encode the traffic relation relaxation behaviour of a vehicle in a situation where crossing a solid lane marking is necessary, for example, to avoid being stuck in a deadlock behind an illegally parked vehicle. The approach demonstrated how the SWRL rules with the Pellet reasoner could be used for decision making. The experiment is carried out, however, over static data.

Buechel et al. [35] present a modular framework of ontology-based traffic scene modelling with decision-making rules derived from traffic regulations. The ontology of the road network contains the lane network (lateral and longitudinal relationship) modelling, as well as the relationships between lanes and lane markings. The ontology is formalised in OWL 2 DL which corresponds to the DL $\mathcal{SROIQ}(\mathcal{D})$, the Pellet reasoner is used to infer knowledge from logic axioms and rules encoded in SWRL. The performance of the proposed approach, however, is not provided.

The approach proposed by Zhao et al. [189], however, is different from the other mentioned approaches as the sensor data is treated as RDF streams. With the developed ontologies (e.g., a map ontology, a control ontology and a car ontology) and rules, a knowledge base was created to decide the driving strategy for an autonomous car at intersections. The RDF streams are processed continuously using the C-SPARQL query engine, and the rules are modelled using SWRL.

So far, the presented approach has relied on static map data. That is to say, the map does not change in the knowledge base during the proposed scenario. The approach proposed by Suryawanshi et al. [172], however, was conducted over dynamic map data. In addition, they proposed layered ontologies to separate the underlying map data structures from the represented knowledge for modelling SD map data used in vehicles, which we took as an inspiration for this thesis.

Asmar et al. [66] propose a knowledge-enabled framework for robots' situational awareness to support autonomous logistics vehicles operating in automobile manufacturing plants. The framework comprises an OWL 2 ontology representing robot observations, and a set of behaviour rules written in Prolog [32], and a reasoner implemented using SWI-Prolog [4]. Their knowledge base (KB) contains time-invariant instances and time-variant instances. In particular, time-variant instances are characterised by a timestamp. The KB is updated every minute, the observations and inferences are stored over a time window for smooth decision making.

Table 3.2 provides an overview of the examined methods used in existing approaches that represent the vehicle environment using ontologies and rules to provide an explicit and implicit specification of vehicle context information. The majority of the existing approaches used OWL and SWRL with the Pellet reasoner. Among these, some considered applications with a knowledge base updated at run-time. The reasoning, however, is performed from scratch whenever the data is changed. We considered this limitation and proposed using incremental and non-monotonic reasoning supported by RDFox [142] for dealing with changing data.

TABLE 3.2: Technology realization of ontology-based approach for Autonomous Vehicles

Technical aspect	Technical options	Related work
Ontology Language	OWL DL	Hummel et al. [100], Hülsen et al. [99], Morignot and Nashashibi [136],Armand et al. [17]
	OWL 2 DL	Buechel et al. [35], El Asmar et al. [66],Kohlhaas et al. [112],Zhao et al. [189]
Rule Language	RacePro rule	Hülsen et al. [99]
	SWRL	Morignot and Nashashibi [136],Armand et al. [17],Buechel et al. [35],Zhao et al. [189], Suryawanshi et al. [172]
	Prolog	El Asmar et al. [66]
Reasoner	RacerPro	Hülsen et al. [99]
	Pellet	Morignot and Nashashibi [136],Armand et al. [17],Buechel et al. [35], Zhao et al. [189], Suryawanshi et al. [172]
	SWI-Prolog related	El Asmar et al. [66]

3.2.2 Reasoning Over Dynamic Data

Autonomous driving functions operate over dynamic and big volume data. Currently, streaming reasoning has been studied to infer knowledge over frequently changing data.

Stream reasoning [55] emerged in the last few years as a new research area that focuses on the adoption of semantic reasoning techniques for highly dynamic data. A data stream is defined as a collection of time-annotated items that are arranged according to a set of temporal parameters. RDF stream is referred as a stream where items are represented according to RDF [25].

The first generation of continuous query answering systems for RDF data streams mainly focus on stream processing. Thus, they are often called RDF Stream Processing (RSP) engines [169]. A time-based window is placed on top of a continuous data stream because the data stream has no defined ending. A continuous SPARQL query is registered once, and continuously delivers responses as the streaming data flows through the window. As such, these RSP engines can filter and query a continuous flow of RDF data and can provide real-time answers. The most well-known examples of RSP engines are C-SPARQL [26] and CQELS-QL [116], but others also exist, such as EP-SPARQL [14] and $\text{SPARQL}_{\text{stream}}$ [37]. These engines verify a SPARQL query graph pattern against an input graph with different semantics and are designed for different use cases. Other solutions, e.g., Sparkwave [113] and INSTANS [155], use extensions of the RETE algorithm [73] for pattern matching. StreamRule [77] is a two-layered approach, combining stream processing with rule-based non-monotonic Answer Set Programming (ASP) to enable reasoning over data streams. However, this approach does not support the integration of background knowledge when handling the data streams.

Stream reasoning is most applied in the domain where data streams are time-based. Insights are extracted for the data streams within a time range. In contrast, HD maps used for autonomous driving functions are spatial data and knowledge is derived within a spatial range (e.g., distance). Furthermore, time-based data expires only when it becomes old in the system. Spatial data streams, however, may expire because of the change of an object's location. For example, a road object may be

expired because it is far behind the location of the subject vehicle. Second, the reasoning capabilities of existing approaches are limited to RDFS as shown in Table 3.3. Negation, which is an important construct for reasoning is still *expensive* [133], and updating the knowledge base is not supported [26].

TABLE 3.3: Reasoning support in state-of-the-art RDF Stream Processing (RSP) engines

	Background Knowledge	Reasoning Capabilities
C-SPARQL	Yes	RDFS
EP-SPARQL	Yes	RDFS (in Prolog)
CQELS	Yes	None
SPARQL Stream	Yes	None
Sparkwave	Yes	RDFS subset
INSTANS	No	None
StreamRule	No	ASP

We considered all the mentioned limitations of current stream reasoning approaches and looked into other reasoning techniques for an alternative. The closest to the stream reasoning idea is incremental reasoning [55]. Therefore, a novel approach using incremental reasoning can be considered to provide reasoning and decision making support for autonomous driving functions in order to address RQ2.

3.3 Quality assurance for Spatial Data

In this section, we give an overview of the studies and the usage of ontology-based approaches for checking spatial data quality.

Furber and Hepp [74] propose a general representation of data quality constraints. Eine et al. [61] argue that an ontology-based approach can improve data quality in information systems, and several approaches for supporting data quality measurements exist in use cases such as sensor networks [68].

The consistency of spatial data can be validated using spatial integrity constraints [178]. Such constraints categorise the spatial relations into metrical (e.g., distance and direction) and topological relations (e.g., overlapping or disjoint). Furthermore, Mäs [130] has investigated reasoning algorithms for checking the internal consistency of a set of spatial semantic integrity constraints. Bravo and Rodriguez [33] formalise a set of spatial integrity constraints and study the satisfiability of these constraints.

One of the existing software applications for spatial data quality evaluation is the 1Validate service provided by 1Spatial [152]. It validates spatial data against different kinds of standard rules including, geometric, polygon and network. Another software is ArcGIS Data Reviewer developed by a GIS software company called ESRI [15]. It allows for feature integrity checks, spatial relationship checks, and attribute checks.

Spatial data quality can be assessed with ontology-based approaches. Mostafavi et al. [137] propose an ontology-based approach for quality assessment of spatial databases. The ontology is encoded in Prolog, and queries are used to determine the existence of inconsistencies. Wang et al. [183] investigate the feasibility of applying rule-based spatial data quality checks over mobile data using the Semantic Web Rule Language (SWRL). The authors show that the system can warn the data collector if

any inconsistent data is gathered in the field. Yilmaz et al. [186] created an ontology associated with spatial concepts from the Open Geospatial Consortium and rules implemented as GeoSPARQL queries for detecting inconsistencies. Yilmaz et al. also developed the Spatial Data Quality Ontology together with SWRL rules for performing quality assessment [187]. Huang et al. [96] investigate the feasibility of combining ontologies and semantic constraints modelled in the Shapes Constraint Language (SHACL) for ensuring the semantic correctness of geospatial data from different levels of detail. A number of RDF stores also support geospatial queries and integrity constraints, e.g., Stardog,[1] Virtuoso,[2] and GraphDB.[3]

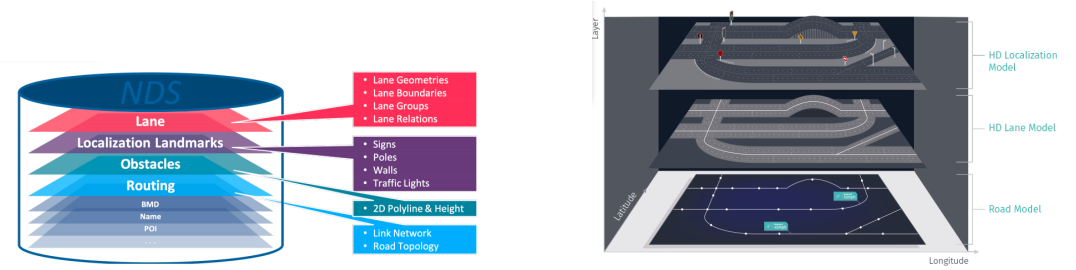
The existing ontology-based approaches, however, focus on general spatial data. Map-related concepts and relationships, such as the relationships among coordinate points, lanes, and roads, are not studied. While SHACL is designed for RDF validation, by checking nodes w.r.t. class axioms or paths w.r.t. property axioms, it cannot describe complex (spatial) relationship constraints, which is crucial for map data. Although SHACL provides validation reports, it does not provide a mechanism (e.g., vocabulary) for fixing errors, while we aim at supporting violation detection and handling in a closed loop. Therefore, a novel approach for addressing map data quality with respect to spatial knowledge is needed as defined by RQ3.

[1] https://www.stardog.com/
[2] https://virtuoso.openlinksw.com/
[3] https://graphdb.ontotext.com/

Chapter 4

Two-level Map Ontologies

High-definition (HD) maps provide detailed *lane-level* information to support vehicle perception and localisation [84]. Based on the functional system architecture of an automated driving system by Ulbrich et al. [176], HD maps with localisation functionality provide inputs to the perception module for world modelling and eventually support the decision making of the planning and control module. Almost all players in the area of highly automated driving, e.g., Google, HERE, TomTom, Baidu, BMW, or Toyota, rely on HD maps to further improve the driving capabilities of their autonomous vehicles. Recently, collective efforts have been made towards standardising HD maps, such as Geographic Data File (GDF) 5.1 at the international level [104] or Navigation Data Standard (NDS) and the ADASIS protocol V3 at industrial level [141]. However, heterogeneous map data poses major challenges for integration in practice. Figure 4.1 shows two HD maps with different logical map models.



(A) NDS building blocks [140]

(B) HERE HD Live Map [90]

FIGURE 4.1: Two HD maps with different logical map models.

Problem statement. In this chapter, we investigate the integration problem among different map formats. The following research question is investigated:

> RQ1: Can an ontology-based approach solve the map data integration problem and provide a generic and unified map model?

Proposed solution. We tackle the problem of map data integration and devise an ontology-based approach that allows for integrating different low-level map data formats into a generic high-level map model. To meet the need for map data integration, we adopt the Global as View (GAV) architecture proposed by Wache et al. [181] with ontologies and rules (see Figure 4.2). We propose a practical methodology to develop two-level ontologies and associated rules. The methodology is used to build map specific low-level ontologies, the generic high-level map ontology and the transformation from low-level map ontologies to the unified high-level map ontology. Finally, we evaluate the methodology via three use cases.

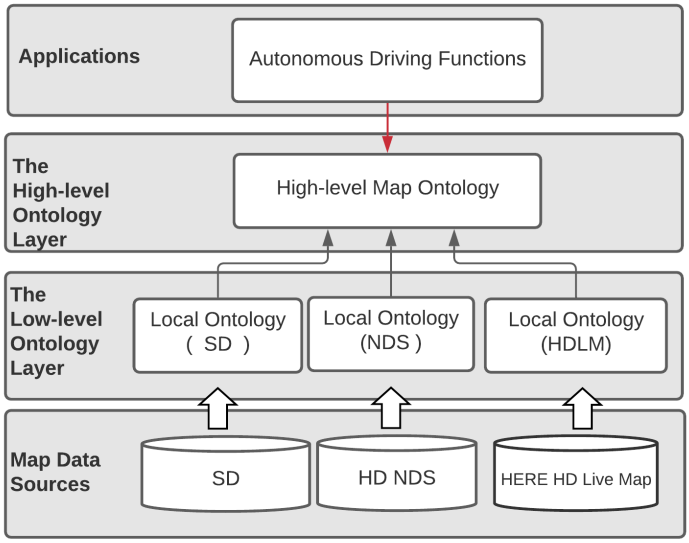


FIGURE 4.2: Ontology-Based Map Data Integration Global-as-View (GAV): Black arrows represent rule-based transformation, the red arrow indicates SPARQL query access from an application layer to the high-level ontology.

Particularly, we present the following contributions in response to RQ1:

- A practical methodology for building the generic high-level ontologies and low-level ontologies.
- A categorisation of the types of rules that are needed for knowledge abstraction and spatial reasoning.
- The generic high-level map (HLM) ontology.
- The low-level map specific ontologies: LNDS ontology based on NDS map format and LHERE ontology based on HERE HD Live Map.
- Use case evaluations showing the applicability and benefits of two-level ontologies.

The remainder of this chapter is structured as follows. Section 4.1 outlines the methodology used for developing the two-level ontologies. The development of the HLM ontology following the proposed methodology is presented in Section 4.2. Section 4.3 and Section 4.4 describe the development of the two low-level map ontologies, namely LNDS and LHERE. In Section 4.5, we highlight the instance transformation from the low-level ontologies to the HLM ontology. Section 4.6 shows the evaluation of the resulted HLM ontology based on three use cases. Finally, the concluding remarks for this chapter are presented in Section 4.7.

4.1 Methodology

This section presents a methodology for creating the low-level ontologies, the high-level ontology and the transformation from low-level ontologies to the high-level ontology (see Figure 4.3). This methodology is composed of four steps:

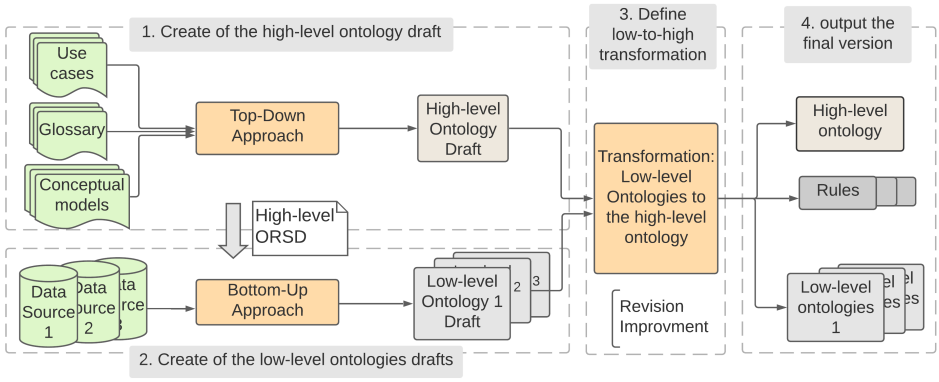


FIGURE 4.3: The methodology of creating the high-level ontology and the low-level ontologies.

1. Create the high-level ontology following a top-down approach:
   - analyse the high-level ontology requirements and produce the high-level ontology requirement specification document (ORSD);
   - develop a draft version of the high-level ontology.
2. Create the low-level ontologies following a bottom-up approach:
   - based on the high-level ORSD, analyse the specific low-level ontology formats and derive low-level ORSDs;
   - develop a draft version of each low-level ontology;
   - populate each low-level ontology using corresponding data source.
3. Transform the low-level ontologies to the high-level ontology:
   - define the mapping for transforming each low-level ontology to the high-level ontology;
   - revise and improve the drafts of the high-level ontology and the developed low-level ontologies.
4. Output the developed ontologies and related rules:
   - finalize the high-level ontology and the low-level ontologies;
   - modularise the rules based on the ontology type and inference purpose.

4.1.1 Top-down Approach

For the top-down approach, we propose the use of METHONTOLOGY [123], one of the most widely used ontology engineering methodologies [6]. It emphasises the reuse of existing domain and upper-level ontologies and proposes to use formalisation purposes, a set of intermediate representations that can later be transformed automatically into different formal languages. Therefore this methodology is suitable for developing ontologies at the knowledge level. Moreover, it considers the main activities identified by the IEEE software development process [5] and other knowledge engineering methodologies.

Different groups have used METHONTOLOGY to build ontologies in different knowledge domains, such as Chemistry [124], Science [158], Knowledge Management, e-Commerce, etc. [72]. Other methodologies, like DILIGENT [150] and NeOn

[170], were considered before starting the construction of the high-level map ontology. However, the characteristics of such methodologies, like the emphasis on decentralised engineering, did not fit our scenario well.

METHONTOLOGY is composed by seven stages: *Specification*, *Knowledge Acquisition*, *Conceptualisation*, *Integration*, *Implementation*, *Evaluation*, and *Documentation* (see Figure 4.4). The overall process involved two domain experts from BMW. The main output of the *Specification* stage is the high-level ORSD written in natural language followed by the guidelines proposed by Suárez-Figueroa et al. [171]. Importantly, this high-level ORSD serves as a guiding document for the development of low-level ontologies.

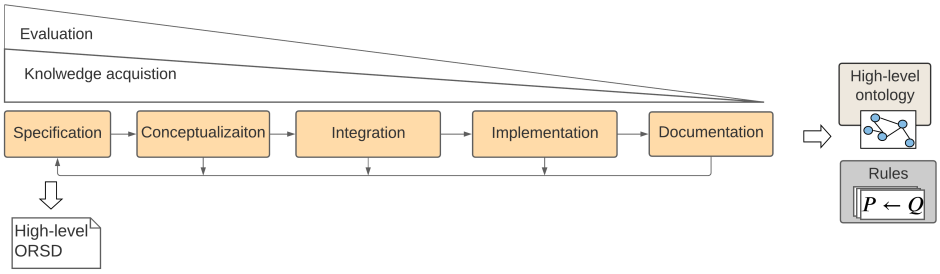


FIGURE 4.4: Ontology development process inspired by the METHONTOLOGY lifecycle [47].

4.1.2 Bottom-up Approach

Besides a top-down approach, another option for developing an ontology without starting from a blank state is to use bottom-up approaches. Bottom-up approaches reuse existing data, information, or knowledge and develop an ontology from these non-ontological resources. Practically, it means that the bottom-up approach takes some non-ontological material and converts it into an ontology with some manual pre- and/or post-processing.

For the bottom-up approach, we adapted the methodology proposed by Uschold et al. [177] to model low-level ontologies. The methodology identifies a five-stage process for ontology creation:

1. **Definition of purpose and scope.** In this stage, with the guidance of the high-level ORSD, the low-level ontologies' domain, purpose of the ontology, and ontology scope are captured in the ORSD.
2. **Capture the domain knowledge.** The process requires cooperation between the ontology engineer and the domain experts. Furthermore, investigation and analysis of existing ontologies (if any) are required. If no ontologies exist, the domain knowledge is typically extracted from documents and data schema.
3. **Develop the ontology.** Develop the ontologies by focusing on the concepts and relationships needed for the high-level ontology. Rules are used to process low-level knowledge and derive implicit knowledge. Consider best practices for developing ontologies concerning reuse, documentation and naming conventions.
4. **Populate the ontology using the data sources.** Existing data sources are used to populate the low-level ontology via mapping between the data schema with the low-level ontology TBox.

5. **Evaluate the ontology.** Evaluate the applicability of the ontology with competency questions.

4.1.3 Ontology Design Patterns and Rules

Ontology design patterns (ODPs) are reusable modelling building blocks providing solutions to recurrent domain modelling problems [31]. ODPs are an important means to improve the quality of an ontology design as they represent best practices in ontology modelling frequently used by ontology developers. In the work of this thesis, we are interested in finding suitable ODPs in the category of *Content ODP* which solves the modelling issues regarding ontology content [89].

We exhaustively use rules for knowledge processing and reasoning. There are three main tasks designated to rules, namely, *Semantic Enrichment*, *Low-to-high knowledge process* and *Reasoning* (see Figure 4.5). Semantic Enrichment is a process of deriving primitive attributes and relationships among instances of a low-level ontology by rules. The rules used in this step enrich instances with one-step inferences, and their results serve as the input for the Low-to-high knowledge process. The Low-to-high knowledge process refers to the transfer of the low-level map ontology to the generic high-level ontology. The rules used in this step transfers the enriched low-level ontology to the high-level ontology based on defined mappings. Reasoning refers to the process of inferring the implicit knowledge using a set of rules combined with the high-level ontology.

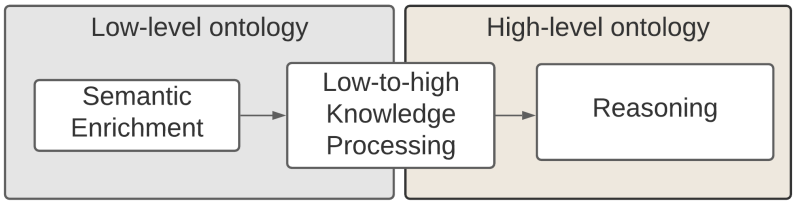


FIGURE 4.5: Rule-based knowledge process and reasoning.

4.1.4 Nomenclature

Graffoo (short for Graphical Framework for OWL Ontologies) is used for OWL oriented graph representation [71]. There are two kinds of graphical elements, i.e., *blocks* (or nodes) and *arcs*. Rectangles with solid borders represent classes. Individuals are presented with pink circles with a solid black border. Arcs are represented by black lines with a solid arrow at the end (see Figure 4.6). Falco et al. [71] compared Graffoo with the Manchester Syntax, Protégé and E/R in terms of usability according to SUS (System Usability Scale) [118] showing that Graffoo has the highest SUS score.

According to Noy and McGuinness [144], it is beneficial to define the naming conventions for the ontology modelling and strictly adhere to them. The conventions for developed ontologies are defined as follows:

- When entities are mentioned within text, `typewriter` is used;
- CamelCase is used for concepts and instances names, such as concept `RoadPart`;
- Concepts names and domain specific instance names start with an upper case;

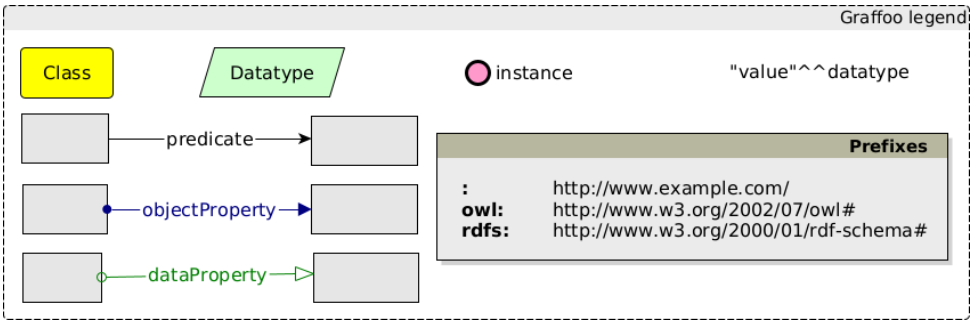


FIGURE 4.6: The graphical elements of Graffoo [71].

- Instance names start with a lower case initial letter;
- Class names are used in the singular ;
- Property names are CamelCase with a lower case initial letter, e.g., `hasLane`.

Table 4.1 lists all the prefixes used in the thesis. For the high-level map ontology, all entities are prefixed with *high*. For low-level map specific ontologies, the prefix is constructed from `http://www.bmw-carit.de/Foresight/Map/Ontologies/Low/` and the map name. For example, the prefix of the NDS ontology is `http://www.bmw-carit.de/Foresight/Map/Ontologies/Low/NDS#`.

TABLE 4.1: Prefixes used in ontologies

Prefix	URI
high	`http://www.bmw-carit.de/Foresight/Map/Ontologies/High#`
lnds	`http://www.bmw-carit.de/Foresight/Map/Ontologies/Low/NDS#`
lhere	`http://www.bmw-carit.de/Foresight/Map/Ontologies/Low/HERE#`
lsd	`http://www.bmw-carit.de/Foresight/Map/Ontologies/Low/SD#`

4.1.5 Documentation

Providing a user-friendly view of vocabularies for non-experts is crucial for integrating Semantic Web technologies [149]. It facilitates the contribution of domain experts during the development process. In addition, it helps other interested parties for easy use of the ontology in later phases as well. There exist different tools for documentation generation. The Live OWL Documentation Environment (LODE) is an online service that automatically generates a human-readable description of an OWL ontology (or, more generally, an RDF vocabulary), considering both ontological axioms and annotations and ordering these with the appearance and functionality of a W3C Recommendations document [148].

4.2 High-level Map Ontology

In this section, we represent the high-level map (HLM) Ontology. This ontology models the high-level generic map concepts and relations, representing real-world entities like road parts, lanes, lane boundaries, etc. The HLM ontology needs to provide a generic view over different map formats. At the moment, there are three

low-level maps: an SD map based on the Advanced Driver Assistance System Interface Specification (ADASIS) protocol V2, an HD map based on NDS standard and another HD map from HDLM (see Figure 4.7). An ontology-based representation of the high-level map model permits the following improvements:

- flexible schema refinement and heterogeneous data linking and integration;
- using the semantic technology stack (e.g., rules and queries) to enhance the mapping process and facilitate the decision-making process;
- detecting knowledge inconsistency by connecting to other domains of knowledge that already has semantic representations, e.g., the Vehicle Ontology [1] and the SSN (Semantic Sensor Network) Ontology [2] to name just a few.

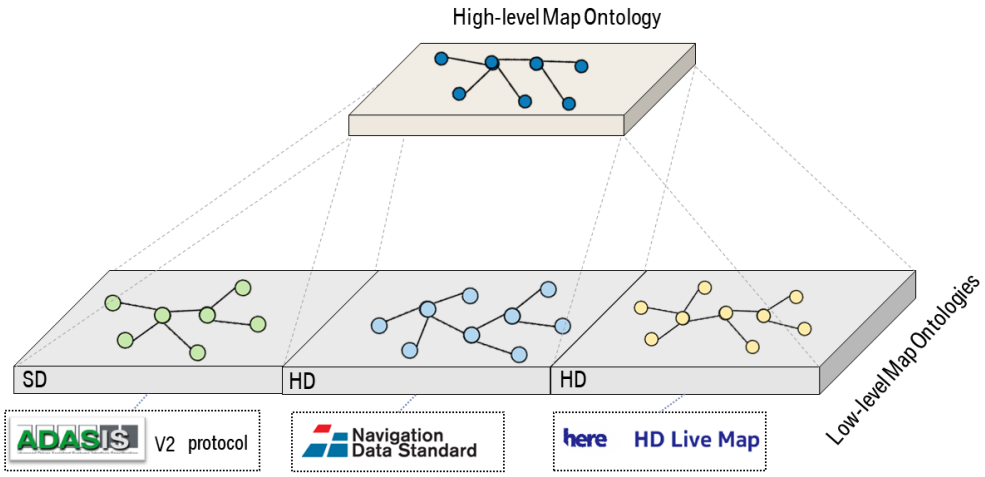


FIGURE 4.7: The high-level map ontology is a generic representation of different low-level map ontologies.

In the following, we describe the process of following METHONTOLOGY for creating the HLM ontology.

4.2.1 Specification

In the stage of specification, domain analysis and knowledge acquisition are carried out. The requirements of the overall HLM ontology needs to be clarified. The main output at this step is HLM ORSD (see Table 4.2). It was written in natural language followed by the guidelines proposed by Suárez-Figueroa et al. [171]. In general, the purpose of the HLM ontology is : i) to represent the generic road knowledge for the AD (Autonomous Driving) vehicles, ii) to provide road environmental context for location-awareness and lane change via spatial reasoning, and iii) to support continuous dynamic map processes along a route. The details of purpose ii) and iii) are further discussed in Chapter 5.

[1] https://enterpriseintegrationlab.github.io/icity/Vehicle/doc/index-en.html
[2] https://www.w3.org/TR/vocab-ssn/

<table>
<tr><th colspan="2">HLM Ontology Requirements Specification Document</th></tr>
<tr><th colspan="2">1. Purpose</th></tr>
<tr><td colspan="2">The purpose of the high-level map (HLM) ontology are i) to represent the generic road knowledge for the AD vehicles, ii) to provide location-awareness and decision making (e.g., lane change) via spatial reasoning, and iii) to support continuous dynamic map process.</td></tr>
<tr><th colspan="2">2. Scope</th></tr>
<tr><td colspan="2">The ontology has to cover a set of core concepts in terms of generic map knowledge, vehicle location awareness, lane change notification and dynamic map processing:
Map
-Road part: represents a part of a road described by geometry points and attributes
-Lane: represents a part of a road that is designated to be used by a single line of vehicles.
-Lane divider: represents the borderline of the lane that is determined by a visible lane marking.
-Point: represents a geo-point with latitude and longitude.

Location awareness
-Vehicle: represents a real-world vehicle concept.
-Current location: represents a place where a vehicle is located at a time. It can be specified by a GPS position.
-Current position: represents the coordinate of the current location of a vehicle.
-Current lane: represents the lane where the vehicle is localised at a time.
-Current road part: represents the road part where the vehicle is localised at a time.
Planning
-Route: represents a list of road parts (lanes) including start and destination points.
-Change lane activity: represents an activity of lateral lane change with the guidance.
Dynamic map processing
-Spatial window: represents a fixed width or region in terms of a geographic element shift (slide) over a path line.</td></tr>
<tr><th colspan="2">3. Implementation Language</th></tr>
<tr><td colspan="2">OWL RL, Datalog and SPARQL</td></tr>
<tr><th colspan="2">4. Intended End-Users</th></tr>
<tr><td colspan="2">User 1. Map companies who want to exchange map data
User 2. Navigation systems that need to integrate heterogeneous map data.
User 3. Connected vehicles that want to share their maps.
User 4. Collaboration between private map companies and open map data communities such as OpenStreetMap [2].</td></tr>
<tr><th colspan="2">5. Intended Uses</th></tr>
<tr><td colspan="2">Use 1. Provide the road and lane information to the users
Use 2. Enable location-awareness of a moving object in a dynamic environment;
Use 3. Support path planning of a moving object in a dynamic environment</td></tr>
<tr><th colspan="2">6. Ontology Requirements</th></tr>
<tr><th colspan="2">a. Non-Functional Requirements</th></tr>
<tr><td colspan="2">NFR 1. The ontology must follow the naming conventions;
NFR 2. The ontology must re-use existing ontologies wherever possible.</td></tr>
<tr><th colspan="2">b. Functional Requirements: Groups of Competency Questions</th></tr>
<tr><td>CQG1. Lane</td><td>CQG2. Road Part</td></tr>
<tr><td>CQ1. What are the predecessors/successors of a lane?
CQ2. What are the points representing the geometry of a lane?
CQ3. What is the latitude and longitude of a point?
CQ4. What is the left/right lane divider of a lane?
CQ5. What are the left/right lanes of a lane?
CQ6. What are the neighbouring lanes of a lane?
CQ7. What is the type of lane?
CQ8. What is the length of a lane?
CQ9. Is the lane diver open to the left/right?</td><td>CQ1. What are the predecessors/successors of a road part?
CQ2. What is the type of road part?
CQ3. What are the points representing geometry of a road part?
CQ4. What is the length of a road?</td></tr>
<tr><td>CQG3. Location awareness</td><td>CQG4. Route planning and dynamic processing</td></tr>
<tr><td>CQ1. What is the position of the current location?
CQ2. What is the current lane?
CQ3. What is the travelled/remaining distance in the current lane?
CQ4. What is the current road part?
CQ5. What is the travelled/remaining distance in the current road part?</td><td>CQ1. Is there a route?
CQ2. What are the route segments?
CQ3. Does the vehicle need to change lanes based on the route and the foresight parameter?
CQ4. What are the forward and backward parameters of a spatial window?
CQ5. Does the vehicle need to pre-fetch map data based on the route and forward parameter of the spatial window?
CQ6. Does the vehicle need to delete expired map objects (road parts, lanes and points) based on the spatial window?</td></tr>
<tr><th colspan="2">7. Pre-Glossary of Terms</th></tr>
<tr><th colspan="2">a. Terms from Competency Questions</th></tr>
<tr><td colspan="2">Lane, Predecessor, Successor, Point, Latitude, Longitude, Lane Divider, Left, Right, Neighbouring, Lane Type, Length, Open-to-left, open-to-right, Road Part, Road Part Type, Road Part Length, Current Location(Position, Lane, Road Part), Travel Distance, Remaining Distance, Route, Route Segment, Change Lane Activity, Foresight Parameter, Spatial Window, Pre-fetch, Expired</td></tr>
</table>

TABLE 4.2: HLM Ontology Requirements Specification Document

4.2.2 Knowledge Acquisition

The knowledge necessary for building the HLM ontology has been acquired in three steps: i) analyse the low-level map formats and extract common and generic high-level of interest in the map and AD functions, ii) discussions with domain experts for deciding how to model generic classes, individuals, and properties exploited to support reasoning to provide the AD functions; iii) search for existing reusable ontologies describing the road environment.

TABLE 4.3: The low-level SD and HD map formats information resources.

Map Type	Document	Description
SD	ADASIS v2 Protocol [59]	a specification to describe the road geometry with its related attributes ahead of a vehicle based on the vehicle's position and a digital map (ADAS Horizon).
HD	Navigation Data Standard (NDS) Format Specification [164]	the specification of Navigation Data Standard (NDS), a standardized physical storage format for navigation systems. This document describes the general concepts and structure of the database format.
	HERE HD Live Map (HDLM) Developer Guide [90]	describes the logical data model and publication format of the HERE HD Live Map Content service containing Logical Data Model and publication format for HD Live Map data.

In the first step, we collect information resources regarding the domain of interest. The primary unstructured information resources concerning low-level maps are the SD ADASIS v2 Protocol, the HD NDS (Navigation Data Standard) Format Specification and the HDLM (HERE HD Live Map) Developer Guide (see Table 4.3). We also performed a qualitative comparison between NDS and HDLM in terms of commonalities and differences described in Section 2.2.1.

The second step consisted of defining the proper entities, enabling the road knowledge's reasoning at a high level. Based on the discussions with domain experts, the static and dynamic aspects of road knowledge require careful consideration. Static knowledge such as roads and lanes is related to the map. Dynamic knowledge is related to vehicle location awareness, lane change planning and map processing (preloading and deletion). Hence, we define two main modules: *Static Road Knowledge* and *Dynamic Vehicle Knowledge* as following:

- The *static road environment* refers to the road entity that does not change during a scenario. This includes geo-spatially stationary elements, such as the road network, the type of road (lane) and geometry shapes.
- The *dynamic vehicle environment* opposed to the static road environment, and it refers to the part of a scenario that changes during the time frame of a scenario. The dynamic environment in our work refers to the change of the vehicle position and the road knowledge update. The road knowledge update is caused by dynamic map processing, which consists of prefetching and deleting map

data. The basic idea behind prefetching is to predict which map tiles the vehicle may wish to use "next" or "soon", and request them from the map database ahead of time [106].

For the third step, we analysed some existing works to represent the semantics of the road environment employing ontologies [100, 151, 98, 189, 18, 172], which benefit our understanding of the domain knowledge. However, existing works present the following drawbacks: i) the ontology covers road or lanes typologies, but not the geometries or lane dividers; ii) the location awareness based on the position of the vehicle and lane change activities are not considered; iii) continuous map processing (preloading and deletion) has not been studied.

4.2.3 Conceptualisation

As proposed in the *METHONTOLOGY* [123], in the activity of *Conceptualisation*, the domain knowledge is conceptualised into a model that describes the ontology requirements in terms of the domain vocabulary identified in the *Specification* activity (see Section 4.2.1). It is worth mentioning that there is no single correct way to model the domain of interest. There are always alternative designing choices [144].

HLM Ontology Overview

Figure 4.8 shows the overall design of the HLM ontology. In general, the ontology covers the *Static Road Knowledge* and *Dynamic Vehicle Knowledge*. The static road knowledge contains generic concepts, relationships, and properties of map representations, which are independent of the specific low-level map formats. The dynamic vehicle knowledge includes dynamic concepts, relationships, and entities related to the continuously updated vehicle position. The spatial reasoning in the context of dynamic vehicle knowledge is described in detail in Chapter 5.

Static Road Knowledge The `high:RoadPart` class represents a part of a road. The instances of `high:RoadPart` in sibling relationship are linked to each other via the object property `high:hasSibling`. The longitude relationship of `high:RoadPart` instances is represented via the transitive object property `high:hasNext`. `high:Lane` represents a part of a road that is designated to be used by a single line of vehicles. The instances from the classes `high:RoadPart` and `high:Lane` are linked via object property `high:hasLane`. Furthermore, the subclasses of `high:RoadPart` and `high:Lane` are shown in Figure 4.9. The generic lateral relationships among the instances of `high:Lane` are described via the object property `high:hasNeighbouringLane`, and the longitudinal relationship is described via the object property`high:hasNextLane`. The `high:LaneDivider` class models the lane boundary of `high:Lane` instances with traversal attributes represented by the `high:openToLeft` and `high:openToRight` data properties. The geometry of `high:RoadPart` and `high:Lane` is described by the instances of the `high:Point` class. The data properties `high:x` and `high:y` of `high:Point` represent latitude and longitude respectively. The class `high:Route` consists of a list of `high:RoadPart` instances or `high:Lane` instances via the object properties `high:hasRoadPartRouteSegment` or `high:hasLaneRouteSegment`, respectively.

Dynamic Vehicle Knowledge Vehicles are represented by instances of the class `icity-vehicle:Vehicle` reused from the *Vehicle Ontology* which specifies concepts related to vehicles such as brand, model, and type [108]. Each `icity-vehicle:Vehicle` instance is linked to a set of instances from `high:Position` via the object property `high:hasTracePosition`. `high:Position` instances are described by the data

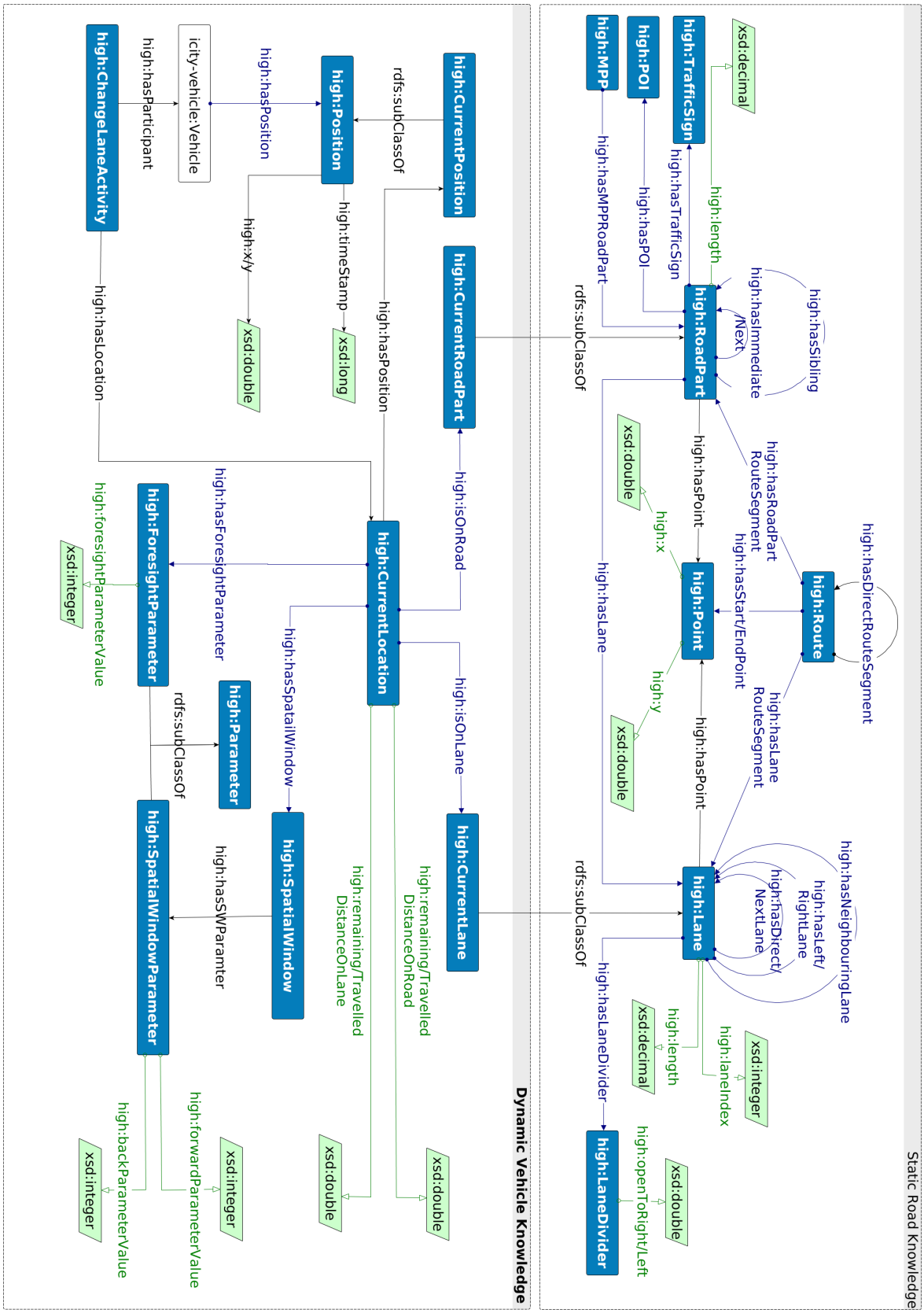


FIGURE 4.8: Overview of the HLM ontology.

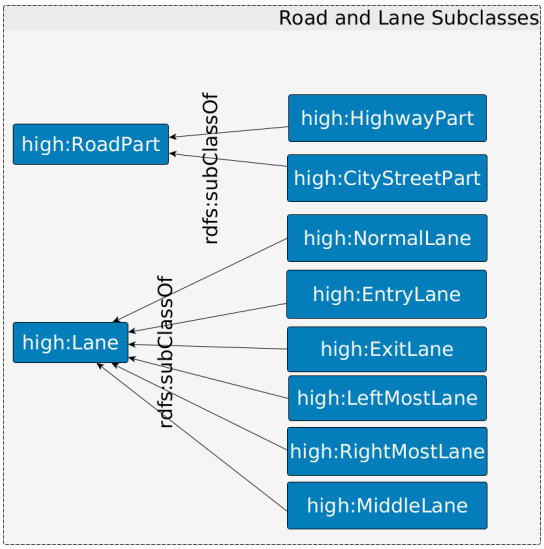


FIGURE 4.9: The subclasses of `high:RoadPart` and `high:Lane`.

properties `high:x`, `high:y` and `high:timestamp`. The current position of the vehicle is modelled as an instance of the class `high:CurrentPosition`, a subclass of `high:Position`. The class `high:CurrentLocation` represents the current location of a car. It is linked to the instance from the class `high:CurrentPosition` via the object property `high:hasPosition`. The class `high:CurrentRoadPart` is modelled as the subclass of `high:RoadPart`. Similarly, the class `high:CurrentLane` is modelled as the subclass of `high:Lane`. Importantly, the class `high:SpatialWindow` represents the spatial sliding window concept. Each instance of `high:SpatialWindow` links to an instance of `high:SpatialWindowParameter` described by the data property `high:forwardParameterValue` and `high:backParameterValue`. The class `high:ChangeLaneActivity` describes the lateral lane change activity. It is connected to the class `high:CurrentLocation` via object property `high:hasLocation`. The participant of an instance of `high:ChangeLaneActivity` is described by an instance of the class `icity-vehicle:Vehicle` via object property `high:hasParticipant`.

Rules

Rules play a key role in spatial reasoning and decision making. The main challenges for rule modelling using HLM are to derive spatial relations for spatial reasoning and capture decision-making algorithms (in rules). Based on the challenges, we classify rules into (1) *spatial* and (2) *algorithmic rules*, and we describe them using sub-categories, examples and more formal rule patterns below. In the formal patterns, we use (possibly with subscripts) C for classes, op for object properties, and dp for data properties of the ontology over which the rules are to be executed. The details of the utilisation of rules are described in Chapter 5 Dynamic Map Processing.

(1) Spatial Rules These rules infer important road environmental knowledge for vehicle navigation [76]. We further subdivide them into *bounding rules*, *topological rules* and *distance rules*.

(a) Bounding rules infer the boundaries of an area or the range of a line, such as a start/end point or the left or right-most lane. Aggregation functions (e.g., `MIN` or `MAX`) can be used to identify an individual with a minimal or maximal bounding

value. As a concrete example, consider:

$$\begin{aligned}\mathsf{LeftMostLane}(z) \leftarrow\ & \mathsf{Lane}(p),\\ & \texttt{AGGREGATE}(\mathsf{hasIdx}(p,\ idx)\\ & \texttt{ON}\ p\ \texttt{BIND}\ \texttt{MAX}(idx)\ AS\ m),\\ & \mathsf{Lane}(z),\ \mathsf{hasIdx}(z,\ m).\end{aligned}$$

More generally, such rules have the form

$$\begin{aligned}\mathsf{C}_2(z) \leftarrow\ & \mathsf{C}_1(x),\ \texttt{AGGREGATE}(\mathsf{dp}_1(x,v)\ \texttt{ON}\ x\\ & \texttt{BIND}\ \texttt{MAX}(v)\ \texttt{AS}\ m),\ \mathsf{C}_1(z),\ \mathsf{dp}_1(z,m).\end{aligned}$$

Such rules might also use (stratified) negation to identify individuals without some properties:

$$\begin{aligned}\mathsf{EndLane}(x) \leftarrow\ & \mathsf{Lane}(x),\ \texttt{NOT EXISTS}\ y\ \texttt{IN}\\ & (\mathsf{Lane}(y),\ \mathsf{hasNext}(x,\ y))\end{aligned}$$

More generally, such rules have the form

$$\mathsf{C}_2(x) \leftarrow \mathsf{C}_1(x),\ \texttt{NOT EXISTS}\ y\ \texttt{IN}\ (\mathsf{C}_1(y),\ \mathsf{op}_1(x,\ y)).$$

(b) Topological rules refer to topological relations, more specifically, lateral (left/right) and longitudinal (predecessor/successor) relations. Reachability can naturally be expressed using recursive rules.

$$\begin{aligned}&\mathsf{hasLeft}(x,y) \leftarrow \mathsf{hasDirectLeft}(x,y)\\ &\mathsf{hasLeft}(x,z) \leftarrow \mathsf{hasDirectLeft}(x,y),\ \mathsf{hasLeft}(y,z).\end{aligned}$$

More generally, such rules have the form

$$\begin{aligned}&\mathsf{op}_1(x,y) \leftarrow \mathsf{op}_2(x,y)\\ &\mathsf{op}_1(x,z) \leftarrow \mathsf{op}_2(x,y),\ \mathsf{op}_1(y,z)\end{aligned}$$

(c) Distance rules refer to the spatial arrangement of objects, such as the distance to the point of interest. There are two types of distance relations: *coordinate distance* and *length distance*.

Coordinate distance rules indicate the distance between two points using coordinates. An auxiliary concept (CoordinateDistance) represents the ternary relation that connects the source point to the target point via two object properties hasSource and hasTarget and the calculated distance value via the data property distance:

$$\begin{aligned}\mathsf{CoordinateDistance}(d),\ & \mathsf{hasSource}(d,s),\\ & \mathsf{hasTarget}(d,t),\ \mathsf{distance}(d,z) \leftarrow\\ & \mathsf{Point}(s),\ \mathsf{x}(s,x_s),\ \mathsf{y}(s,y_s),\\ & \mathsf{Point}(t),\ \mathsf{x}(t,x_t),\ \mathsf{y}(t,y_t),\\ & \texttt{BIND}(\texttt{sqrt}((x_s-x_t)^2+(y_s-y_t)^2)\ \texttt{AS}\ z),\\ & \texttt{BIND}(\texttt{SKOLEM}(\texttt{"}d\texttt{"},\ s,\ t)\ \texttt{AS}\ d).\end{aligned}$$

More generally, such rules have the following form, where A represents the auxiliary concept:

$$\begin{aligned}\mathsf{A}(d),\ & \mathsf{hasSource}(d,s),\\ & \mathsf{hasTarget}(d,t),\ \mathsf{distance}(d,z) \leftarrow\\ & \mathsf{C}_1(s),\ \mathsf{x}(s,x_s),\ \mathsf{y}(s,y_s),\\ & \mathsf{C}_1(t),\ \mathsf{x}(t,x_t),\ \mathsf{y}(t,y_t),\\ & \texttt{BIND}(\texttt{sqrt}((x_s-x_t)^2+(y_s-y_t)^2)\ \texttt{AS}\ z),\\ & \texttt{BIND}(\texttt{SKOLEM}(\texttt{"}d\texttt{"},\ s,\ t)\ \texttt{AS}\ d).\end{aligned}$$

Length distance rules are similar to coordinate distance rules, but the distance is calculated by aggregating the length of intermediate path elements between two points. As a concrete example consider:

$$\begin{aligned}&\mathsf{LengthDistance}(d),\ \mathsf{hasSource}(d,s),\\ &\qquad \mathsf{hasTarget}(d,t),\ \mathsf{distance}(d,z) \leftarrow\\ &\qquad \mathsf{Lane}(s),\ \mathsf{length}(s,v),\\ &\qquad \texttt{AGGREGATE}(\mathsf{hasNext}(s,p),\ \mathsf{hasNext}(p,t)),\\ &\qquad \mathsf{length}(p,l)\ \texttt{ON}\ s\ \texttt{BIND SUM}(l)\ \texttt{AS}\ u),\\ &\qquad \texttt{BIND}((v+u)\ \texttt{AS}\ z),\\ &\qquad \texttt{BIND}(\texttt{SKOLEM}(\texttt{"}f\texttt{"},\ s,\ t)\ \texttt{AS}\ d).\end{aligned}$$

More generally, such rules have the form

$$\begin{aligned}&\mathsf{A}(d),\ \mathsf{hasSource}(d,s),\\ &\qquad \mathsf{hasTarget}(d,t),\ \mathsf{distance}(d,z) \leftarrow\\ &\qquad \mathsf{C}_1(s),\ \mathsf{length}(s,v),\\ &\qquad \texttt{AGGREGATE}(\mathsf{op}_1(s,p),\ \mathsf{op}_1(p,t)),\\ &\qquad \mathsf{length}(p,l)\ \texttt{ON}\ s\ \texttt{BIND SUM}(l)\ \texttt{AS}\ u),\\ &\qquad \texttt{BIND}((v+u)\ \texttt{AS}\ z),\\ &\qquad \texttt{BIND}(\texttt{SKOLEM}(\texttt{"}f\texttt{"},\ s,\ t)\ \texttt{AS}\ d).\end{aligned}$$

(2) Algorithmic Rules These rules are used for decision making and vehicle motion planning, e.g., determining the road's characteristic change points (e.g., speed limit), the most probable path a vehicle will take, providing lane-change notifications and prefetching map tiles. The rules are modelled as a combination of rules of a different type, possibly requiring a specific rule order.

(a) Change-point rules refer to the rules that detect whether any attribute-value change points will occur on the road ahead of the vehicles, such as a change of the speed limit, slope etc., for the vehicles to make manoeuvre decisions. The change point is the point where the ahead attribute value starts to change. In order to capture the key notions of "ahead" and "starting", we use the property chain of hasNext ∘ hasImmediatePrevious. Figure 4.10 shows an example where the current road part is rp1, and it connects to all roads ahead with the property hasNext. The speed limit change points are on rp2 (80 kmh → 90 kmh) and rp6 (90 kmh → 80 kmh) along the path. For rp2, it connects to rp1 via hasNext, in particular, hasImmediateNext. Hence, it is the road part ahead of rp1. Moreover, it connects to rp1 via hasImmediatePrevious, and rp2's speed limit is different from rp1's. Hence, it is the starting point of the change. The reasoning for rp6 as a change point is applied in similar fashion.

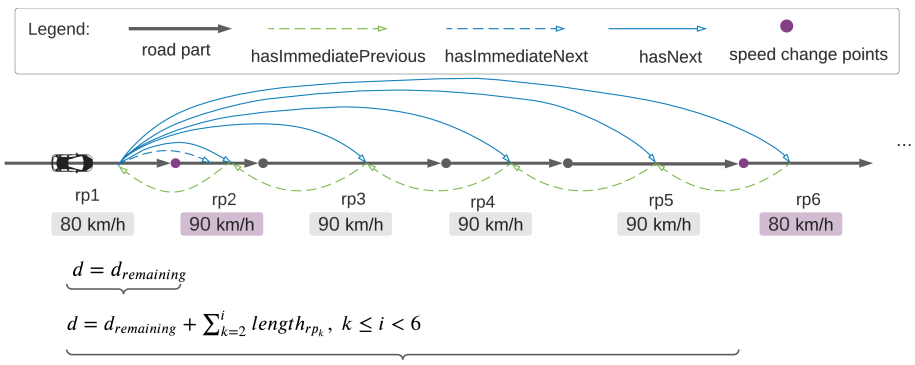


FIGURE 4.10: Illustration of the speed change points.

In general, the change-point detection for a road path $rp_1, \ldots, rp_n$ and a given a road part rp_j, where $1 < j \leq n$ consists of two steps:

1. identify the road parts R ahead where the attribute-value change point appears via the property chain hasNext $\circ$ hasImmediatePrevious;
2. calculate the distance d from rp_j to $r_i \in R$:
   (a) if r_i is rp_j's immediate next road part, then $d = d_{remaining}$
   (b) otherwise, $d = d_{remaining} + \sum_{k=j+1}^{i-1} length_{rp_k}$.

As a concrete example, consider the following rules. Note that the *Topological rules* and *Distance rules* pattern defined in the previous section are reused.

(1) $\mathsf{hasSpeedLimitChangeAt}(cp, x_1) \leftarrow$
$\mathsf{CurrentRoadPart}(cp)$,
$\mathsf{hasNext}(cp, x_1)$, $\mathsf{hasImmediatePrevious}(x_1, x_2)$,
$\mathsf{speedLimit}(x_1, s_1)$, $\mathsf{speedLimit}(x_2, s_2)$,
$\mathtt{FILTER}(s_1 \neq s_2 2)$.

(2a) $\mathsf{distanceToChangePoint}(cp, d) \leftarrow$
$\mathsf{hasSpeedLimitChangeAt}(cp, x_1)$, $\mathsf{hasImmediateNext}(cp, x_1)$,
$\mathsf{remaingingDistance}(cp, d)$.

(2b) $\mathsf{distanceToChangePoint}(cp, d) \leftarrow$
$\mathsf{hasSpeedLimitChangeAt}(cp, x_1)$, $\mathsf{hasNext}(cp, x_1)$,
$\mathsf{remaingingDistance}(r)$, $\mathsf{LengthDistance}(l)$, $\mathsf{hasSource}(l, cp)$,
$\mathsf{hasTarget}(l, x_1)$, $\mathsf{distance}(l, z)$,
$\mathtt{BIND}((r + z)\ \mathtt{AS}\ d)$.

More generally, such rules have the form

(1) $\mathsf{hasChangeAt}(c, p) \leftarrow$
$\mathsf{C}_1(c)$, $\mathsf{hasNext}(c, n)$, $\mathsf{hasImmediatePrevious}(n, p)$,
$\mathsf{dp}_1(n, a_1))$, $\mathsf{dp}_1(p, a_2)$,
$\mathtt{FILTER}(a_1 \neq a_2)$.

(2a) $\mathsf{distanceToChange}(c, d) \leftarrow$
$\mathsf{hasChangeAt}(c, p)$, $\mathsf{hasImmediateNext}(c, p)$,
$\mathsf{distance}(c, d)$.

(2b) $\mathsf{distanceToChange}(c, d) \leftarrow$
$\mathsf{hasChangeAt}(c, p)$, $\mathsf{hasNext}(c, p)$, $\mathsf{distance}(c, r)$,
$\mathsf{LengthDistance}(l)$, $\mathsf{hasSource}(l, c)$,
$\mathsf{hasTarget}(l, p)$, $\mathsf{distance}(l, z)$,
$\mathtt{BIND}((r + z)\ \mathtt{AS}\ d)$.

(b) Path rules refer to a set of rules that iteratively find a path satisfying a set of conditions. A "path" may comprise at least a portion of one or more road parts or lanes. It represents a trajectory that may be taken by a vehicle through the road network, such as the most probable path (MPP) at the road level in ADAS horizon [1]. Figure 4.11 shows the path searching process. The process starts with certain map objects, such as the current road part, then adds the immediate next object satisfying *condition 1* to the path. Otherwise, the process checks *condition 2*, if necessary, until *condition n*. Based on the newly added path part, the process continues further until no map object is found. The iterative building process mainly takes advantage of recursive rule structures, and there is a set of recursive rules for each condition.

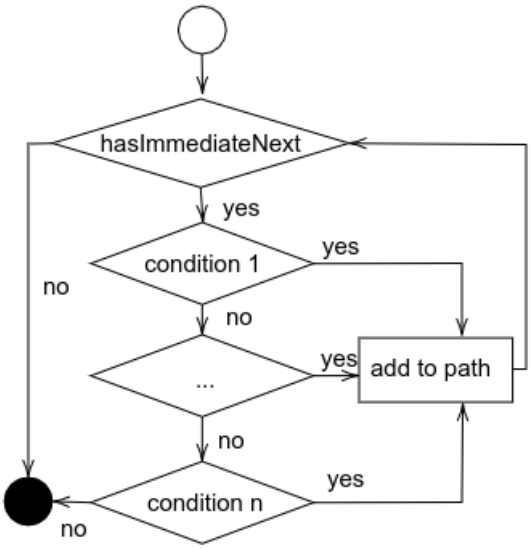


FIGURE 4.11: Illustration of the path finding process.

As a concrete example, consider the following rules (partial) for finding the MPP.

$\mathsf{isPartOfMpp}(cp, m) \leftarrow$
$\quad \mathsf{MPP}(m),\ \mathsf{CurrentRoadPart}(cp)$

$\mathsf{isPartOfMpp}(rp_1, m) \leftarrow$
$\quad \mathsf{isPartOfMpp}(rp, m),\ \mathsf{hasImmediateNext}(rp, rp_1),$
$\quad \mathsf{isPlannedRoadPart}(rp_1, v),\ \mathtt{FILTER}(v = \top).$

...
$\mathsf{isPartOfMpp}(rp_1, m) \leftarrow$
$\quad \mathsf{isPartOfMpp}(rp, m), \mathsf{hasImmediateNext}(rp, rp_1),$
$\quad \mathsf{isPlannedRoadPart}(rp_1, v1),\ \mathsf{isMaxProbabilityRoadpart}(rp_1, v2),$
$\quad \mathtt{FILTER}(v1 \neq \top \wedge v2 = \top).$

More generally, such rules have the following form considering the conditions of cd_i, where $2 \leq i \leq n$

$\mathsf{isPartOfPath}(c, p) \leftarrow$
$\quad \mathsf{Path}(p),\ \mathsf{C}_1(c).$

$\mathsf{isPartOfPath}(rp_1, p) \leftarrow$
$\quad \mathsf{isPartOfPath}(rp, p),\ \mathsf{hasImmediateNext}(rp, rp_1),$
$\quad \mathsf{cd}_1(rp_1, v),\ \mathtt{FILTER}(v = \top).$

$\mathsf{isPartOfPath}(rp_1, p) \leftarrow$
$\quad \mathsf{isPartOfPath}(rp, p),\ \mathsf{hasImmediateNext}(rp, rp_1),$
$\quad \mathsf{cd}_{i-1}(rp_1, v_{i-1}),\ \mathsf{cd}_i(rp_1, v_i),$
$\quad \mathtt{FILTER}(v_{i-1} \neq \top \wedge v_i = \top).$

4.2.4 Integration

In this section, we first describe the related ontologies for the HLM ontology design. Then we present the identified ODPs that can be applied to the modelling of the HLM ontology.

Related Ontologies

According to the HLM ORSD, three areas have been identified where existing ontologies may be used. These areas are static road information, dynamic vehicle environment and vehicle information. After reviewing possible ontologies, some of them are selected for reuse and described next.

- The Core Ontology [172] is used for specifying a standard (road-level) map model. The namespace and naming convention of the *core ontology* is changed and adapted to follow the best practice of ontology development. For example, `core:CURRENT_LOCATION` is changed to `high:CurrentLocation` to follow CamelCase naming convention.
- The Vehicle Ontology [108] specifies concepts related to vehicles such as brand, model, and type. The concept of `icity-vehicle:Vehicle` is reused in our ontology, which serves as the bridging concept between two ontologies, namely the HLM ontology and the Vehicle ontology.

Reusing Ontology Design Patterns

Hereby, we discuss the existing ODPs and how they are applied to support the modelling of the HLM ontology. However, in some cases, we renamed some properties upon the need for applied domain knowledge. *Sequence ODP*[3] represents the "path" is employed to model the directed connectivity of road parts (lanes). For example, the successors of a road part are connected via `high:hasNext`, and the predecessors are connected via `high:hasPrevious`. For modelling the route, we used *Trajectory ODP*[4] and Sequence ODP to represent a route with a start point and an end point which are connected by a sequence of road parts or lanes. Figure 4.12 illustrates a vehicle's route formed by road parts, and Figure 4.13 shows the corresponding RDF graph of the example route.

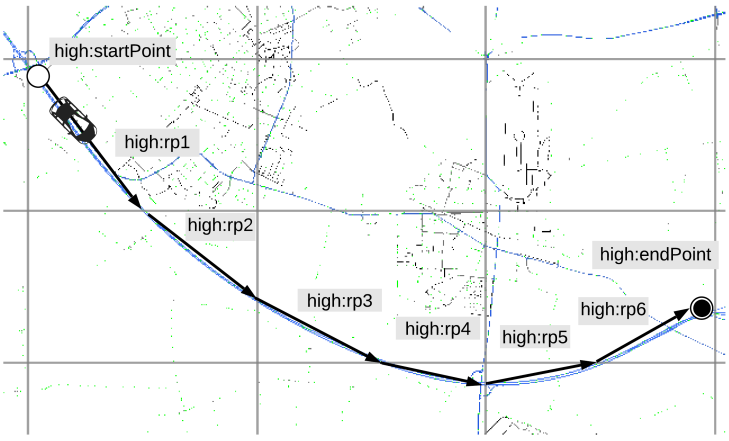


FIGURE 4.12: An example of a vehicle route.

We used the *N–Ary Relation ODP*[5] to model the distance between two points. Figure 4.14 shows an example of the distance relationship between the current position

[3] http://ontologydesignpatterns.org/wiki/Submissions:Sequence
[4] http://ontologydesignpatterns.org/wiki/Submissions:Trajectory
[5] `http://ontologydesignpatterns.org/wiki/Submissions:N-Ary_Relation_Pattern_(OWL_2)`

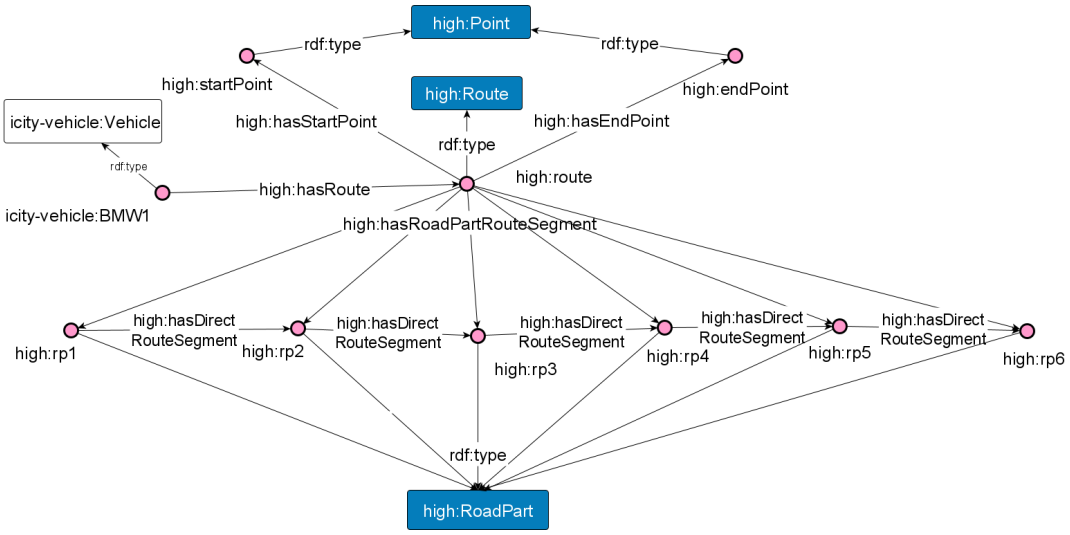


FIGURE 4.13: An RDF graph representation of the route.

of the vehicle and a point on the road. Clearly, we need to qualify the distance relationship with a value. Hence, we introduce a class called `high:CoordinateDistance` whose instance is connected to the instance of class `high:CurrentPosition` and an instance of `high:Point` via the `high:hasSource` and `high:hasTarget` object properties respectively, and the distance is captured via the data property `high:distance` (see Figure 4.15).

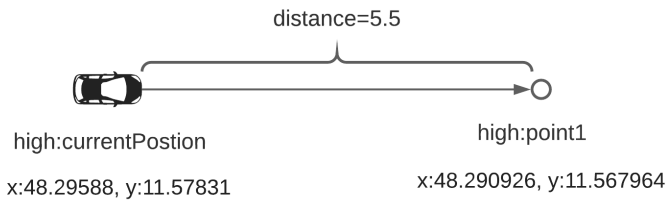


FIGURE 4.14: An example of the coordinate distance between the vehicle's position and a point on the road.

The parameters are modelled using *Parameter ODP*[6]. The foresight parameter is modelled as the class `high:ForesightParameter` with data property `high:foresightParameterValue`. The forward and backward parameters of the spatial window is modelled as data properties `high:forwardParameterValue` and `high:backwardParameterValue` of class `high:SpatialWindowParameter`. The classes `high:ForesightParameter` and `high:SpatialWindowParameter` are subclasses of `high:Parameter`.

[6]http://ontologydesignpatterns.org/wiki/Submissions:Parameter

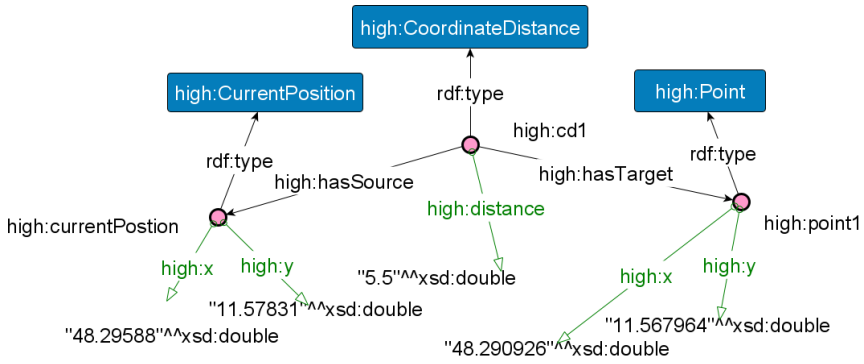


FIGURE 4.15: An RDF graph representing the coordinate distance between the current position and a point on the road.

4.2.5 Summary of Ontology Characteristics

Table 4.4 reports the summary of characteristics for the HLM ontology. The reused ontologies and ODPs are included in the design process of the ontology.

TABLE 4.4: Summary of the HLM Ontology Characteristics

Name	**HLM ontology**
Size	31 classes, 35 object properties, 20 data properties, 3 individuals, 209 axioms
DL Expressivity	ALEHI (D)
Reused Ontologies	Core, Vehicle Ontology
Reused ODP	Trajectory, N-nary, Parameter, Sequence
Naming Conventions	CamelCase notation
Methodology	METHONTOLOGY [123]

4.3 Low-level LNDS Ontology

In this section, we describe the modelling process of the LNDS ontology following the bottom-up approach (see Section 4.1.2). The LNDS ontology is built based on the NDS specification considering the HLM ORSD (see Section 4.2.1). To the best of our knowledge, there is no previous work on this since NDS is a relatively new standard, and it is not a public resource.

4.3.1 Purpose and Scope

The purpose of the LNDS ontology is to provide a semantic representation of the NDS format and populate the HLM ontology by knowledge transferring rules. Table 4.5 provides the requirements' specification of the LNDS ontology.

4.3.2 Capture

The capture of the domain of interest, i.e., the NDS format, is achieved in two ways. First, we used the text document and data visualisation tools to investigate the domain of interest. The text document we used is the NDS Format Specification 2.5.4 that comprises 1114 pages. This document contains terminological definitions, which represent a major source for studying the domain of interest. The visualisation tools we used are the NDS map data viewer and the SQLLite-based data

<table>
<tr><th colspan="2">LNDS Ontology Requirements Specification Document</th></tr>
<tr><th colspan="2">1. Purpose</th></tr>
<tr><td colspan="2">The purpose of the low-level NDS map ontology is to provide a semantic representation of the NDS standard and sufficient information to populate the HLM ontology.</td></tr>
<tr><th colspan="2">2. Scope</th></tr>
<tr><td colspan="2">The ontology has to cover a set of core concepts from routing building blocks (e.g., links, shape points), and lane building block (e.g., lanes, lane boundaries) :

-Link: represents a directed polyline connecting two nodes with optional intermediate shape points. Both nodes are located in the same tile.
-Road Geometry Line: represents a course of a road using a sequence of shape points describing a polyline.
-Shape Point: represents a geo-point with latitude and longitude providing a generalized road centreline geometry.
-Lane Group: represents a set of one or more lanes.
-Lane: A lane is a part of a road that is designated to be used by a single line of vehicles. The place at which lanes begin or end is called a lane connector.
-Lane boundary: represents the borderline of the lane that is determined by a visible lane marking.</td></tr>
<tr><th colspan="2">3. Implementation Language</th></tr>
<tr><td colspan="2">OWL RL, Datalog and SPARQL</td></tr>
<tr><th colspan="2">4. Intended End-Users</th></tr>
<tr><td colspan="2">User 1. Map companies want to exchange map data based on the NDS standard.
User 2. Navigation systems that need to integrate heterogeneous NDS map data with the other domain of knowledge, e.g., weather or sensor data.
User 3. Connected vehicles that want to share their NDS maps.</td></tr>
<tr><th colspan="2">5. Intended Uses</th></tr>
<tr><td colspan="2">Use 1. Provide semantic representation of NDS data to the users.
Use 2. Support refinement of map data via spatial reasoning.
Use 3. Facilitate Validation process of map data using spatial relationships.</td></tr>
<tr><th colspan="2">6. Ontology Requirements</th></tr>
<tr><th colspan="2">a. Non-Functional Requirements</th></tr>
<tr><td colspan="2">NFR 1. The ontology must follow the naming conventions;
NFR 2. The ontology must re-use existing ontologies wherever possible.</td></tr>
<tr><th colspan="2">b. Functional Requirements: Groups of Competency Questions</th></tr>
<tr><td>CQG1. Lane</td><td>CQG2. Feature</td></tr>
<tr><td>CQ1. What are the lanes/lane boundaries in a lane group?
CQ2. What are the start/end lane point?
CQ3. What is the index/type/length of a lane?
CQ4. What is the direct successor of a lane?
CQ5. What are the shape points of a lane?
CQ6. What is the latitude and longitude of a point?
CQ7. What is the left/right lane boundary of a lane?
CQ8. What are the parallel elements of a lane boundary?
CQ9. What are the sequential elements of a parallel element?
CQ10. What is the type of a sequential element?</td><td>CQ1. What are the features to which a lane group refers?
CQ2. What is the start/end point of a feature?
CQ3. What is the type of feature?
CQ4. What is the travel direction of a feature?
CQ5. What are the points representing the geometry of a feature?
CQ6. What is the length of a feature?
CQ7. Is the feature controlled access?
CQ8. Is the feature a motorway/urban street?</td></tr>
<tr><th colspan="2">7. Pre-Glossary of Terms</th></tr>
<tr><th colspan="2">a. Terms from Competency Questions</th></tr>
<tr><td colspan="2">Lane Group, Lane, Direct Successor, Shape Point, Latitude, Longitude, Lane Boundary, Parallel Element, Sequential Element, Left, Right, Lane Type, Length, Feature, Length, Travel Direction, Controlled Access, Motor Way, Urban</td></tr>
</table>

TABLE 4.5: LNDS Ontology Requirements Specification Document

inspector for exploring the NDS map database. Second, a domain expert with deep knowledge in the utility of maps in vehicles supported the process of capturing the knowledge of the domain.

4.3.3 Development

A low-level ontology, LNDS, representing the NDS format with core concepts is provided in this section. The namespace we used for this ontology is *lnds*. We first provide an *overview* of LNDS, and then we present the *ODPs* used during the development process. At last, we show the utilisation of the *rule patterns* for semantic enrichment.

Ontology Overview

In this section, we describe the main classes of the LNDS ontology (see Figure 4.16).

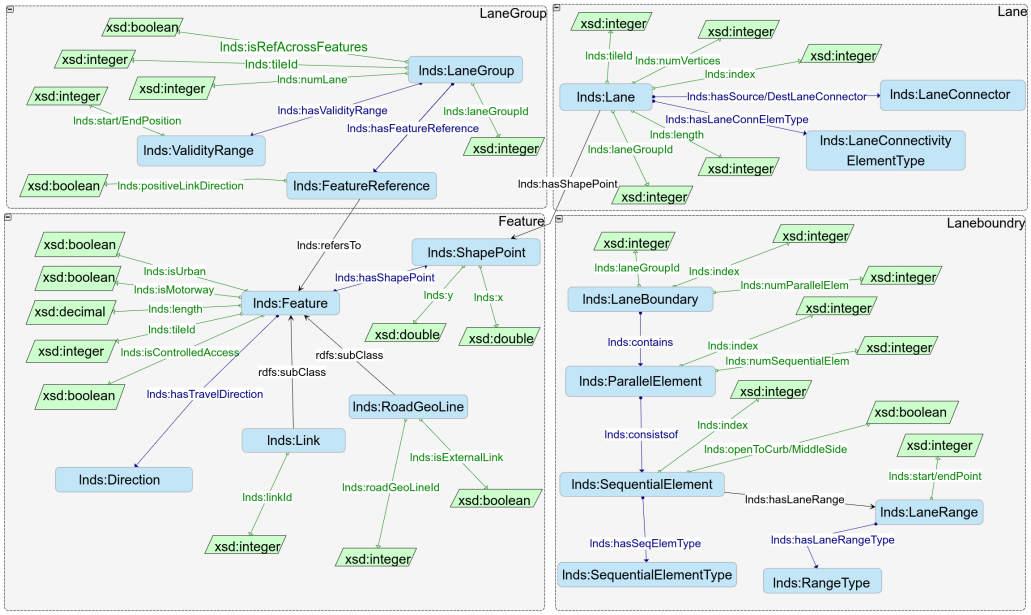


FIGURE 4.16: The low-level NDS map ontology.

lnds:LaneGroup represents the lane group concept and its properties. It provides context for associated lanes with lane boundaries and lanes with features. All lanes and lane boundaries in a lane group have the same lane group ID. The lane group ID is modelled via the data property `lnds:laneGroupId`.

lnds:Lane represents the lane concept and is the core element of the ontology. The class `lnds:LaneConnector` is connected to the `lnds:Lane` to describe the connectivity of lanes via `lnds:hasSourceLaneConnector` and `lnds:hasDestLaneConnector` object properties. The length of each instance of `lnds:Lane` is described via the data property `lnds:length`. The geometry of `lnds:Lane` instances is modelled using a set of instances of `lnds:ShapePoint` via the `lnds:hasShapePoint` object property.

lnds:LaneConnectivityElementType represents the different lane types. Lane types are modelled as instances of `lnds:LaneConnectivityElementTyp`. `lnds:Normal-Lane`, for example, is declared as an instance of this class to represent normal lane type.

lnds:LaneBoundary represents the lane boundary concept in LNDS. An instance of `lnds:LaneBoundary` is described by instances of `lnds:ParallelElement` via the object property `lnds:contains`. An instance of `lnds:ParallelElement` is modelled using instances from `lnds:SequentialElement` via the object property `lnds:consistsOf`.

lnds:SequentialElementType represents the different sequential element types. The declared instances of `lnds:SequentialElementType` are used to describe different sequential element types. For example, `lnds:SingleSolidLine` is declared as an instance to describe the single solid line type.

lnds:FeatureReference represents the feature reference in LNDS. It describes the reference between `lnds:LaneGroup` and `lnds:Feature` via the object property `lnds:has FeatureReference`.

lnds:Feature represents the feature concept in LNDS. The instance of `lnds:Feature` is described by `lnds:Direction` via the object property `lnds:hasTravelDirection`. The instance of `lnds:Feature` has a length value described by the `lnds:length` data property. An `lnds:Feature` instance may be a controlled access road described via the `lnds:isControlledAccess` data property. The geometry of `lnds:Feature` is modelled using instances of `lnds:ShapePoint` via the `lnds:hasShapePoint` object property. The `lnds:isMotorway` data property indicates whether the feature is part of a motorway. The `lnds:isUrban` data property indicates whether the feature is located in an urban environment.

lnds:Direction represents the travel direction concept in LNDS. Travel directions are modelled as instances of `lnds:Direction`. For example, `lnds:PositiveDirection` indicates that the direction is from the start point to the endpoint.

lnds:Link models the base link concept in LNDS. A base link is a link that does not cross several tiles. The class `lnds:Link` is modelled as a subclass of `lnds:Feature`.

lnds:RoadGeoLine models the road geometry line concept in LNDS. It is a subclass of `lnds:Feature`.

lnds:ShapePoint models the shape point concept in LNDS. Each point has an x-y coordinate. This is defined via the data properties `lnds:x` and `lnds:y`.

Ontology Design Patterns

We analyse existing ODPs that can be applied to the modelling of LNDS. A pattern to be included is the *Sequence ODP*. The objective is to model the sequence of map objects, such as the sequential elements of a parallel element in a lane boundary. The *Constituency ODP*[7] is a pattern of interest in the LNDS domain as well. In LNDS, a parallel element in a lane boundary consists of several sequential elements. Therefore, we use the Constituency ODP to model the constituents of map objects, such as the relationship between sequential elements and parallel elements. Moreover, the *N–Ary ODP* is used to model the reference mechanism in LNDS, such as lane groups with a feature reference.

Rule Patterns

The purpose of using rule patterns is to enrich the semantic of the data via rules. We use *Primitive Rules* to form the fundamental relationships among entities of LNDS ontology. These rules enrich instances with one-step inferences, and their results

[7] http://ontologydesignpatterns.org/wiki/Submissions:Constituency

serve as input for all other rules. Primitive rules use the identifiers to infer the relationships or attributes, and are divided into *primitive relationship rules* and *primitive attribute rules*.

(a) Primitive relationship rules infer relationships between individuals. For a concrete example consider:

$$\mathsf{hasLane}(x,y) \leftarrow \mathsf{LaneGroup}(x),\ \mathsf{laneGroupId}(x,i),\ ,\mathsf{Lane}(y),\ \mathsf{laneGroupId}(y,i).$$

More generally, such rules have the form

$$\mathsf{op}(x,y) \leftarrow \mathsf{C}_1(x),\ \mathsf{dp}(x,z),\ \mathsf{C}_2(y),\ \mathsf{dp}(y,z)$$

(b) Primitive attribute rules infer an attribute of an individual. For a concrete example consider:

$$\mathsf{speed}(x,v) \leftarrow \mathsf{Node}(x),\ \mathsf{hasIdx}(x,i),\ \mathsf{speedVal}(y,v),\ \mathsf{hasIdx}(y,i).$$

More generally, such rules have the form

$$\mathsf{dp}(x,v) \leftarrow \mathsf{C}(x),\ \mathsf{dp}_1(x,z),\ \mathsf{dp}_2(y,v),\ \mathsf{dp}_1(y,z).$$

In addition, *Spatial Rule* patterns described in Section 4.2.3 are used to infer topological relationships and distance relationships for `lnds:Lane` and `lnds:Feature`.

4.3.4 The Ontology Population

In NDS, tiles represent a unit for storing data in the database. The features are compactly encoded in RDS (Relational DataScript) and stored as the BLOBs (Binary Large Objects). We use SQLite, and Zserio[8] generated Python NDS APIs to extract data from the NDS database and construct RDF triples based on the LNDS ontology. String concatenation is used to generate triples instead of existing RDF APIs written in Java or Python for deployment environment independence. Table 4.6 shows an excerpt of the mapping between the ontology and the NDS schema. Figure 4.17 shows the RDF graph of a lane instance with related relationships.

TABLE 4.6: Mapping NDS schema with LNDS properties. The attributes of NDS schema are mapped to the concepts and properties of the LNDS ontology.

Attribute from NDS DB	**Concept**	**Property**
laneGroupId	lnds:LaneGroup	lnds:laneGroupId
laneConnectivityElements[index]	lnds:Lane	lnds:laneIndex
sourceLaneConnecorId	lnds:LaneConnector	lnds:hasSourceLaneConnector
...	...	...
linkId	lnds:Link	lnds:linkId
roadGeoLineId	lnds:RoadGeoLine	lnds:roadGeoLine_id
latitude	lnds:ShapePoint	lnds:x
longtitude	lnds:ShapePoint	lnds:y
tileId	lnds:Tile	lnds:tileId

4.3.5 Summary of Ontology Characteristics

Table 4.7 reports the summary of characteristics for the LNDS ontology. The reuse of ODPs is included in the design process of the ontology.

[8] https://github.com/ndsev/zserio

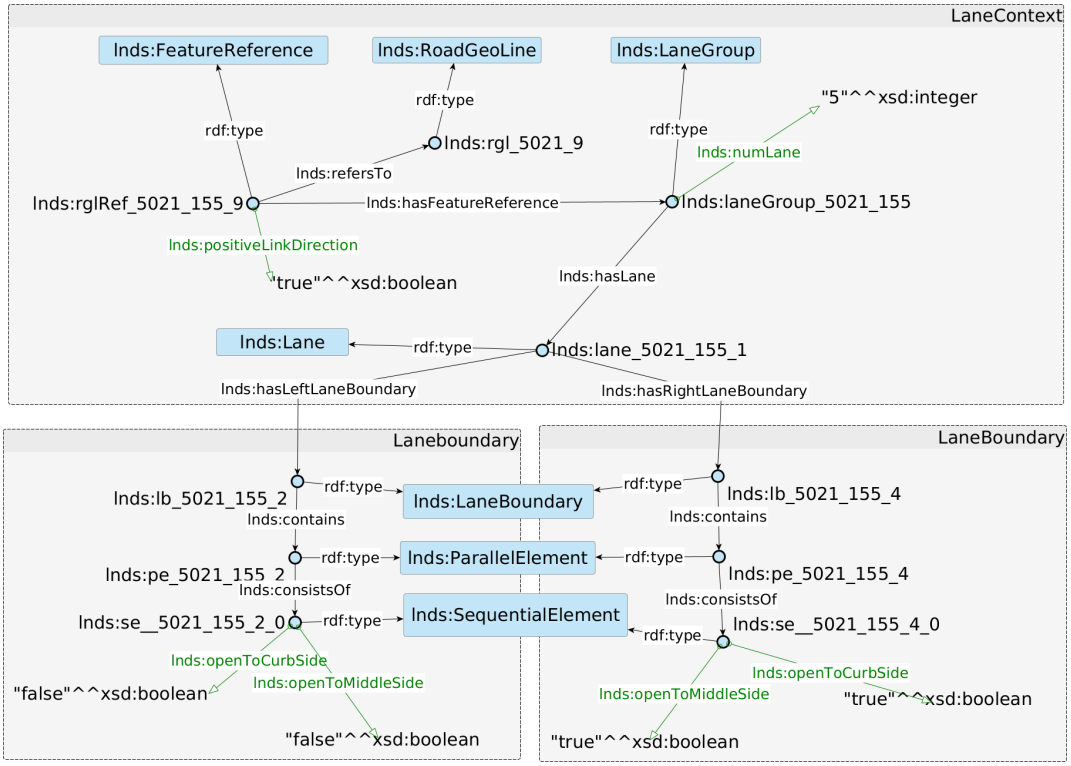


FIGURE 4.17: A RDF graph representation of a lane instance with related relationships.

TABLE 4.7: Summary of the LNDS Ontology Characteristics

Name	**LNDS ontology**
size	32 classes, 33 object properties, 23 data properties, 32 individuals, 275 axioms
DL Expressivity	ALEHI (D)
Reuse ODP	Sequence, Constituency, N-nary
Naming Conventions	CamelCase notation
Methodology	Uschold and Gruninger [177]

4.3.6 Evaluation

In this section, we present the evaluation of the LNDS ontology. First, we present the experimental results of building the LNDS ontology. Second, we evaluate the LNDS ontology's adequacy for the use cases of lane identification.

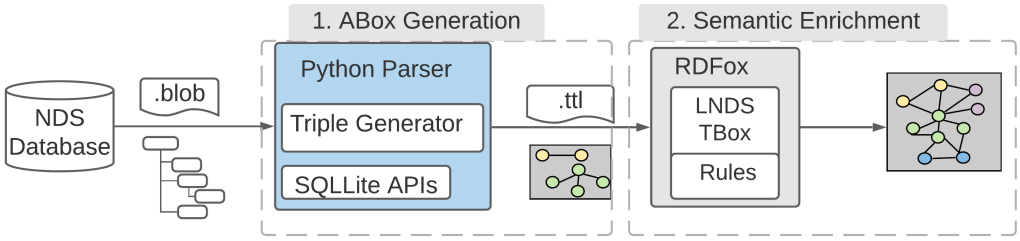


FIGURE 4.18: The low-level NDS map knowledge process

Performance

Figure 4.18 shows the knowledge process of the low-level NDS map, which consists of two steps: i) LNDS ABox Generation; ii) Semantic Enrichment. The LNDS ABox generation is described in Section 4.3.4. For the semantic enrichment process, we developed an application called *SmartMapApp* using the RDFox 4.0.0 triple store and incorporated the process into the application. All evaluations were performed on a 64-bit Ubuntu virtual machine with 4 Intel(R) Core(TM) i7-6820HQ CPUs @ 2.70GHz with 15 GB memory.

Table 4.8 shows the computation time for the ABox Generation and the Semantic Enrichment stage, respectively, the number of generated triples, and the total number of triples after enrichment. The average ABox generation time is 1.96 s, and the average semantic enrichment time is 1.47 s. In general, the more data a tile has, the longer it takes to generate ABoxes and enrich semantics. The average number of triples generated is 137,209, while the average number of triples after semantic enrichment is 296,832, almost twice as many.

TABLE 4.8: The low-level NDS map process performance.

	Time (s)		#Triple	
	ABox Generation	Semantic Enrichment	Generated	Enriched
5034	1.3	0.145	28,714	60,042
5059	1.2	0.166	32,730	67,590
5021	1.1	0.196	38,515	79,694
5035	2.6	0.694	82,811	171,574
5056	1.4	0.506	62,744	129,558
5057	2.6	1.834	184,995	389,020
5032	1.9	1.478	154,529	322,284
5033	2.4	1.958	199,550	420,490
5062	2.7	3.613	284,908	596,946
5020	2.3	4.092	302,596	731,124
AVG	1.96	1.4682	137,209	296,832

Lane Identification

Listing 4.1 shows the nested SPARQL query for retrieving the relevant information about a lane based on a GPS position with latitude and longitude coordinate. First, the inner query identifies the lane where the given position is located. This is

achieved by calculating the distance between the given position with all lane shape points, then retrieve the lane whose shape point has the minimum distance to the given position. Second, based on the result of the inner query, the outer query retrieves the related lane knowledge inferred by rules, such as the left/right lane boundary and the successor lane.

```
SELECT   ?fe  ?lgId ?laneIdx ?lane  ?laneType ?length ?succrLane
        ?llb  ?openToMiddleSide  ?openToCurbSide
WHERE{
?lane lnds:hasLaneConnElemType ?laneType;lnds:laneGroupId ?lgId;
     lnds:hasLeftLaneBoundary ?llb; lnds:hasRightLaneBoundary ?rlb;
     lnds:index ?laneIdx; lnds:length ?length;
     lnds:hasDirectSuccrLane ?succrLane;
     lnds:hasFeature ?fe.
?llb lnds:contains ?ple.
    ...
    lnds:openToMiddleSide ?openToMiddleSide.
    lnds:openToCurbSide ?openToCurbSide.
    {SELECT ?lane
    WHERE {
        ?lane a lnds:Lane;lnds:hasShapePoint ?sp.
        ?sp lnds:x ?x; lnds:y ?y.
         BIND(sqrt((48.251881-?x)*(48.251881-?x)+
         (11.532547-?y)*(11.532547-?y)) AS ?distance ).
    }  ORDER BY ?distance LIMIT 1
}
```

LISTING 4.1: SPARQL query for retrieving LNDS lane information using a given position (48.251881 11.532547).

The retrieved lane information based on a position coordinate in *SmartMapApp* is shown in Figure 4.19a. The corresponding map visualisation is depicted in Figure 4.19b with the travel direction from top to down.

(A) The LNDS lane information based on a position with coordinate (48.251881,11.532547).

(B) Google map visualisation of a position with coordinate (48.251881,11.532547)

FIGURE 4.19: Lane identification result.

4.4 Low-level LHERE Ontology

In this section, we present the LHERE ontology based on the provided HDLM map format document. The modelling process follows the bottom-up approach described in Section 4.1.2. To the best of our knowledge, there is no previous work on this topic

since HDLM is a relatively new map format, and it is privately owned by the HERE company.

4.4.1 Purpose and Scope

The purpose of the LHERE ontology is to provide a semantic representation of HDLM and populate the HLM ABox with sufficient information by knowledge transferring rules. Table 4.9 provides the detailed requirement specification of the LHERE ontology.

<table>
<tr><th colspan="2">LHERE Ontology Requirements Specification Document</th></tr>
<tr><td colspan="2">1. Purpose</td></tr>
<tr><td colspan="2">The purpose of the low-level HERE map ontology is to provide a semantic representation of the HLDM and sufficient information to populate the HLM ontology.</td></tr>
<tr><td colspan="2">2. Scope</td></tr>
<tr><td colspan="2">The ontology has to cover a set of core concepts from the TopologyGeometryLayer (e.g., links, shape points), the RoutingAttributeLayer (e.g., link type), the LaneTopologyLayer (e.g., lane connectivity), the LaneGeometryLayer, the LaneAttributeLayer (e.g., lane and lane boundary type), the LaneRoadReferenceLayer :
-Link: represents a directed polyline connecting two nodes with optional intermediate shape points.
-Shape Point: represents a geo-point with latitude and longitude providing generalized road centreline geometry.
-Lane Group: represents a set of one or more lanes, which begin and end at a consistent location across the road. The place at which lanes begin or end is called a lane group connector.
-Lane: represents a part of a road that is designated to be used by a single line of vehicles.
-Lane boundary: represents the borderline of the lane that is determined by a visible lane marking.</td></tr>
<tr><td colspan="2">3. Implementation Language</td></tr>
<tr><td colspan="2">OWL RL, Datalog and SPARQL</td></tr>
<tr><td colspan="2">4. Intended End-Users</td></tr>
<tr><td colspan="2">User 1. Map companies want to exchange map data based on HLDM.
User 2. Autonomous driving systems need to integrate HLDM data with other domain of knowledge, e.g., weather or sensor data.
User 3. Smart City systems where HLDM is used for urban planning and management.</td></tr>
<tr><td colspan="2">5. Intended Uses</td></tr>
<tr><td colspan="2">Use 1. provide a semantic representation of HLDM to the users.
Use 2. integrate with other domains of data.
Use 3. refinement of the map model via graph representation.</td></tr>
<tr><td colspan="2">6. Ontology Requirements</td></tr>
<tr><td colspan="2">a. Non-Functional Requirements</td></tr>
<tr><td colspan="2">NFR 1. The ontology must follow the naming conventions;
NFR 2. The ontology must re-use existing ontologies wherever possible.</td></tr>
<tr><td colspan="2">b. Functional Requirements: Groups of Competency Questions</td></tr>
<tr><td>CQG1. Lane</td><td>CQG2. Link</td></tr>
<tr><td>CQ1. what are the lanes/lane boundaries in a lane group?
CQ2. what are the start/end lane point?
CQ3. what is the index/type/length of a lane?
CQ4. what is the direct successor of a lane?
CQ5. what are the shape points of a lane?
CQ6. what is the latitude and longitude of a point?
CQ7. what is the left/right lane boundary of a lane?
CQ8. what are the elements of a lane boundary?
CQ9. what is the type of an element?</td><td>CQ1. what are the links a lane group refers to?
CQ2. what are the start/end point of a link?
CQ3. what is the type of a link?
CQ4. what is the travel direction of a link?
CQ5. what are the points representing the geometry of a link?
CQ6. what is the length of a link?
CQ7. is the link controlled access?
CQ8. is the link a motorway/urban street?</td></tr>
<tr><td colspan="2">7. Pre-Glossary of Terms</td></tr>
<tr><td colspan="2">a. Terms from Competency Questions</td></tr>
<tr><td colspan="2">Lane Group, Lane, Direct Successor, Shape Point, Latitude, Longitude, Lane Boundary, Element, Left, Right, Lane Type, Length, Link, Length, Travel Direction, Controlled Access, MotorWay, Urban</td></tr>
</table>

TABLE 4.9: LHERE Ontology Requirements Specification Document

4.4.2 Capture

The capturing of the domain of interest, i.e., HDLM, consists of two steps. First, we used the text document, visualisation tool, and data inspector to investigate the domain of interest. The text document we used is the HDLM Developer Guide document that comprises 170 pages. This document contains terminological definitions,

which represent a major source for studying the domain of interest. The visualisation tools we used are HDLM Map Viewer and map data catalog inspector. Second, a domain expert supported the process of capturing the knowledge of the domain.

4.4.3 Development

A low-level ontology LHERE representing HDLM map with core concepts is provided in this work; the namespace *lhere* is used for this ontology. We first provide an overview of LHERE and present the ODPs used during the development process. At last, we show the utilisation of the rule patterns for semantic enrichment.

Ontology Overview

In this section, we describe the main classes of the LHERE ontology (see Figure 4.20).

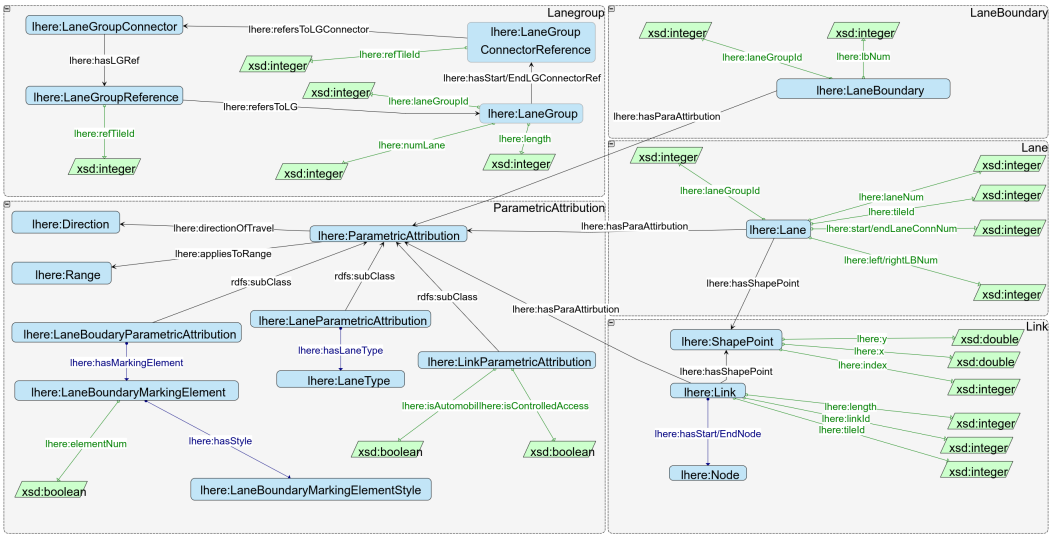


FIGURE 4.20: The low-level LHERE ontology representing the HDLM map.

lhere:LaneGroup represents the lane group concept and its properties. It provides context for associated lanes with lane boundaries and lanes with links. All lanes and lane boundaries in a lane group have the same lane group ID. The lane group ID is modelled via the data property `lhere:laneGroupId`. The lane group connectivity is modelled via `lhere:hasStartLGConnectorRef` and `lhere:hasEndLGConnectorRef` object properties linking to the class `lhere:LaneGroupConnectorReference`.

lhere:LaneGroupConnectorReference represents the association between lane group connector and the located tile ID. It links to `lhere:LaneGroupConnector` via the object property `lhere:refersToLGConnector` and to the located tile ID via the data property `lhere:refTileId`. The length of each instance of `lhere:LaneGroup` is described via the data property `lhere:length`.

lhere:Lane represents the lane concept and is the core element of the ontology. The data properties `lhere:endLaneConnNum` and `lhere:endLaneConnNum` are used to model the lane connectivity based the lane group connectivity. The geometry of `lhere:Lane` instances is modelled using a set of instances of `lhere:ShapePoint` via the `lhere:hasShapePoint` object property. An instance of `lhere:Lane` is described by instances of

lhere:LaneParametricAttribution via the lhere:hasParaAttirbution object property. Each instance of lhere:LaneParametricAttribution is described by an instance of lhere:LaneType via the object property lhere:hasLaneType.

lhere:LaneBoundary represents the lane boundary concept. Its instance is described by instances of lhere:LaneBoundaryParametricAttribution via the lhere:hasPara-Attribution object property. lhere:LaneBoundaryParametricAttribution is further described by the class lhere:LaneBoundaryMarkingElement. Each instance of lhere:LaneBoundaryMarkingElement is described by an instance of lhere:Lane-BoundaryMarkingElementStyle via the property lhere:hasStyle.

lhere:ParametricAttribution models the parametric attribute with a defined range in a direction. This is modelled via connecting to the class lhere:Direction through the lhere:directionOfTravel object property and to the class lhere:Range through the lhere:appliesToRange object property. The specific feature attributions such as lhere:LinkParametricAttribution is modelled as the subclass of lhere:Parametric-Attribution.

lhere:Link represents the link concept. Each instance of lhere:Link is described by two instances of the lhere:Node class. The geometry of lhere:Link instances is modelled using a set of instances of lhere:ShapePoint via the lhere:hasShapePoint object property. Each instance of lhere:Link has a length value described by the lhere:length data property. The attribute of an instance of lhere:Link is described by lhere:LinkParametricAttribution via lhere:hasParaAttribution. Each instance of lhere:LinkParametricAttribution is described by the data properties lhere:isControlledAccess and lhere:isAutomobiles.

lhere:Direction represents the travel direction concept. Travel directions are modelled as instances of lhere:Direction. For example, lhere:Forward is an instance of lhere:Direction, and it indicates that the direction is from the start point to the endpoint. Another instance is lhere:Backward indicating that the direction is from the endpoint to the start point.

lhere:ShapePoint models the shape point concept. Each point has an x-y coordinate. This is defined via the data properties lhere:x and lhere:y.

Ontology Design Patterns

We analyse existing ODPs that can be applied to the modelling of LHERE. The *Sequence ODP* is used to model the sequence of map objects, such as the sequence of lane groups and links. In LHERE, a reference mechanism is widely used to model attribute reference. Therefore, we use the *N–Nary ODP* to model the map objects attribute references, such as the parametric attribute references.

Rule Patterns

Similar to LNDS ontology, rules are also used for the *Semantic Enrichment*. We use *Primitive Rules* described in Section 4.3.3 to infer the fundamental relationships among entities of lhere:LaneGroup, lhere:Lane and lhere:LaneBoundary as well as their descriptive attribute relationships. *Spatial Rule* described in Section 4.2.3 are used to infer topological relationships and distance relationships for lhere:Lane and lhere:Link.

4.4.4 The Ontology Population

HDLM is a cloud-based web service that enables access to continuously updated map data [90]. The data is composed of tiled map layers containing information

such as the topology, geometry, and attributes of roads and lanes. The data is stored in a series of map catalogs that correspond to geographic regions. We use the provided Java map API to extract map data and construct RDF triples based on the LHERE ontology. Similar to LNDS, the triple are created based on string concatenation instead of existing RDF APIs. Because existing RDF APIs are based on Java or Python, however, we aim for deployment environment independence. Table 4.6 shows an excerpt of the mapping between the ontology and the HDLM data schema.

TABLE 4.10: Mapping schema of the HDLM map with LHERE properties. The attributes in the HDLM are mapped to the concepts and properties of the LHERE ontology.

Attribute from HERE catalog	Concept	Property
lane_group_id	lhere:LaneGroup	lhere:LaneGroupId
start_lane_group_connector_ref	lhere:Lane-GroupConnector-Reference	lhere:hasStart-LGConnectorRef
lanes[index]	lhere:Lane	lhere:index
start_lane_connector_number	-	lhere:startLaneConnNum
...	...	...
link_id	lhere:Link	lhere:linkId
link_length_meters	-	lhere:length
here_tile_id	lhere:Tile	lhere:tileId

4.4.5 Summary of Ontology Characteristics

Table 4.11 reports the summary of characteristics for the LHERE ontology. The reused of ODPs is included in the design process of the ontology.

TABLE 4.11: Summary of the LHERE Ontology Characteristics

Name	**LHERE ontology**
size	32 classes, 32 object properties, 26 data properties, 10 individuals, 213 axioms
DL Expressivity	ALEHI (D)
Reused ODP	Sequence, N-nary
Naming Conventions	CamelCase notation
Methodology	Uschold and Gruninger [177]

4.4.6 Evaluation

In this section, we present the evaluation of the LHERE ontology. First, we present the experimental results of building LHERE ontology. Second, we evaluate the LHERE ontology's adequacy based on the use case of lane identification.

Performance

Figure 4.21 shows the knowledge process of the HDLM map, which consists of two steps: i) LHERE ABox generation; ii) Semantic Enrichment. We extended the *SmartMapApp* application to work with LHERE ontology.

Table 4.12 shows the computation time of the ABox generation and semantic enrichment stage, respectively, and the number of triples generated and the total number of triples after enrichment. The average ABox generation time is 0.1378 s, and

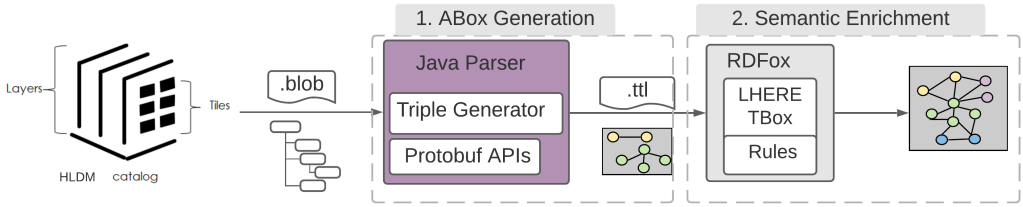


FIGURE 4.21: The low-level HDLM map knowledge process.

the average semantic enrichment time is 0.809 s, 0.9 s in total. In general, the more data one tile contains, the more time is needed for ABox generation and semantic enrichment. The average number of triples generated is 89,656, which is the input triple number of semantic enrichment. The average number of triples after semantic enrichment is 219,001, almost 2.5 times the size of the input triples.

TABLE 4.12: The low-level HDLM map process performance.

	Time (s)		#Triple	
	ABox Generation	Semantic Enrichment	Generated	Enriched
2874	0.024	0.061	13,644	31,765
2899	0.037	0.221	15,303	37,060
2861	0.02	1.556	21,129	48,953
2875	0.056	0.14	50,642	120,261
2896	0.101	0.302	39,651	96,198
2897	0.201	0.286	106,930	263,865
2872	0.089	1.571	87,537	214,613
2873	0.157	1.845	121,597	297,660
2902	0.468	0.532	223,636	551,243
2860	0.234	1.572	216,491	528,392
AVG	0.1387	0.809	89,656	219,001

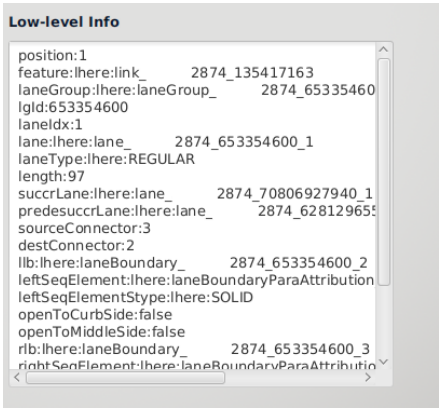
Lane Identification

The SPARQL query we used for LHERE lane identification is shown in Listing 4.2 with nested queries. We also used the same given position (48.251881,11.532547) for lane identification. The result shown in Figure 4.22 is similar to the LNDS lane identification presented in Figure 4.19. This indicates the semantic commonalities between LHERE and LNDS.

```
SELECT  ?laneGroup ?lgId ?laneIdx ?lane
        ?laneType ?length ?succrLane  ?llb    ?rlb
WHERE{
?laneGroup lhere:hasLane ?lane.
?lane lhere:hasLaneType ?laneType;
      lhere:laneGroupId ?lgId;
      lhere:index ?laneIdx;
      lhere:length ?length;
      lhere:hasDirectSuccrLane ?succrLane.
       ...
OPTIONAL {?lane      lhere:hasLeftLaneBoundary ?llb.   ...}
OPTIONAL {?lane    lhere:hasRightLaneBoundary ?rlb.   ...}
    {SELECT ?lane
        WHERE {
        ?lane a lhere:Lane;lhere:hasShapePoint ?sp.
        ?sp lhere:x ?x; lhere:y ?y.
        BIND(sqrt((48.251881-?x)*(48.251881,-?x)+
         (11.532547-?y)*(11.532547-?y)) AS ?distance ).
```

```
        } ORDER BY ?distance LIMIT 1
    }
}
```

LISTING 4.2: SPARQL query for retrieving LHERE lane information using a given position (48.251881 11.532547).



(A) The LHERE lane information of a position with coordinate (48.251881,11.532547).

(B) Google map visualisation of a position with coordinate (48.251881,11.532547)

FIGURE 4.22: Lane identification result.

4.5 The Low-to-high Transformation

In this section, we explain the rule-based RDF graph transformation from the low-level map ontologies $\mathcal{O}_1$ (e.g., LNDS and LHERE) to the HLM ontology $\mathcal{O}_2$. We use rules to create a new individual in the $\mathcal{O}_2$ based on one or more individuals in $\mathcal{O}_1$. `SKOLEM` functions and string manipulations are used to dynamically generate named (not anonymous) individuals [163]. We distinguish two different RDF graph transformations: *simple* (1:1) and *complex* (n:1) transformations. The transformation comprises two steps:

1. Transferring the instances of classes with data property values in $\mathcal{O}_1$ to the $\mathcal{O}_2$;
2. Inferring the object properties of the new instances in the $\mathcal{O}_2$ based on the corresponding newly created instances' object properties in the $\mathcal{O}_1$.

4.5.1 Simple Transformation

Simple transformation rules create, for each individual of a certain type in a $\mathcal{O}_1$, a new individual of a certain type in the $\mathcal{O}_2$. The data property values are assigned during the individual creation process. The temporary object property hasLow is used to mark the correspondence between the individual in $\mathcal{O}_1$ and $\mathcal{O}_2$ such that object properties can be inferred between newly created individuals in the $\mathcal{O}_2$. For a concrete example, the rules below, which create an individual of RoadPart in $\mathcal{O}_2$ from the individual of Link in $\mathcal{O}_1$ with the value of data property length. The individual of Point in $\mathcal{O}_2$ with associated data properties is created in similar fashion.

$$\mathsf{RoadPart}(r),\ \mathsf{length}(r,l),\ \mathsf{hasLow}(r,k) \leftarrow$$
$$\mathsf{Link}(k),\ \mathsf{length}(k,l),\ \mathtt{BIND}(\mathtt{SKOLEM}(\mathtt{"}rp\mathtt{"},k)\ \mathtt{AS}\ r).$$

$\mathsf{Point}(p)$, $\mathsf{x}(p,x)$, $\mathsf{y}(p,y)$, $\mathsf{hasLow}(p,sp) \leftarrow$
$\mathsf{ShapePoint}(sp)$, $\mathsf{x}(sp,x)$, $\mathsf{y}(sp,y)$, $\texttt{BIND}(\texttt{SKOLEM}(\texttt{"}p\texttt{"},sp)\ \texttt{AS}\ p)$.

The rule below infers the object property `hasPoint` between the individuals of `RoadPart` and `Point` in $\mathcal{O}_2$ from the object property `hasShapePoint` in $\mathcal{O}_1$.

$\mathsf{hasPoint}(r,p) \leftarrow$
$\mathsf{RoadPart}(r)$, $\mathsf{hasLow}(r,k)$, $\mathsf{Point}(p)$, $\mathsf{hasLow}(p,sp)$, $\mathsf{hasShapePoint}(k,sp)$.

More generally, simple transformation rules have the following form, where `hl` stands for `hasLow`, C_1, dp_1 and op_1 are from a $\mathcal{O}_1$ and C_2, C_3, dp_2 and op_2 are from the $\mathcal{O}_2$.

$\mathsf{C}_2(n)$, $\mathsf{dp}_2(n,v)$, $\mathsf{hl}(n,x) \leftarrow$
$\mathsf{C}_1(x)$, $\mathsf{dp}_1(x,v)$, $\texttt{BIND}(\texttt{SKOLEM}(\texttt{"}c\texttt{"},\ x)\ \texttt{AS}\ n)$.

$\mathsf{op}_2(n,v) \leftarrow$
$\mathsf{C}_2(n)$, $\mathsf{hl}(n,m)$, $\mathsf{C}_3(v)$, $\mathsf{hl}(v,u)$, $\mathsf{op}_1(m,u)$.

4.5.2 Complex Transformation

Complex transformation rules create a new individual in $\mathcal{O}_2$ based on several individuals in the $\mathcal{O}_1$, hence, perform an aggregation process. As a concrete example, consider the rule below, which creates a new individual of `Lane` in $\mathcal{O}_2$ and assigns data property value `laneNum` in $\mathcal{O}_2$.

$\mathsf{Lane}(l)$, $\mathsf{laneNum}(l,n)$, $\mathsf{hasLow}(l,l_1)$, $\mathsf{hasLow}(l,l_2), \leftarrow$
$\mathsf{Lane}(l_1)$, $\mathsf{Lane}(l_2)$, $\mathsf{laneNum}(l_1,n)$, $\mathsf{laneNum}(l_2,n)$,
$\mathsf{hasConn}(l_1,c)$, $\mathsf{hasConn}(l_2,c)$, $\texttt{BIND}(\texttt{SKOLEM}(\texttt{"}l\texttt{"},\ c)\ \texttt{AS}\ l)$.

The data property `laneNum` is 1:1 value transfer. There is also a need for aggregated value transfer, such as the length of a new individual of `Lane` in $\mathcal{O}_2$.

$\mathsf{length}(l,y) \leftarrow$
$\mathsf{Lane}(l)$, $\mathsf{hasLow}(l,k)$,
$\texttt{AGGREGATION}(\mathsf{length}(k,n), \texttt{BIND}(\texttt{SUM}(n)\ \texttt{AS}\ y)$.

More generally, complex transformation rules have the form

$\mathsf{C}_2(w)$, $\mathsf{dp}_2(w,v)$, $\mathsf{hl}(w,x)$, $\mathsf{hl}(w,y) \leftarrow$
$\mathsf{C}_1(x)$, $\mathsf{C}_1(y)$, $\mathsf{dp}_1(x,v)$, $\mathsf{dp}_1(y,v)$,
$\mathsf{op}_1(x,z)$, $\mathsf{op}_1(y,z)$, $\texttt{BIND}(\texttt{SKOLEM}(\texttt{"}c\texttt{"},\ z)\ \texttt{AS}\ w)$.

$\mathsf{dp}_2(x,v) \leftarrow$
$\mathsf{C}_2(x)$, $\mathsf{hl}(x,k)$,
$\texttt{AGGREGATION}(\mathsf{dp}_1(k,n), \texttt{BIND}(\texttt{SUM}(n)\ \texttt{AS}\ v)$.

The object properties between individuals created in complex transformation are inferred using the same rule pattern as illustrated in simple transformation (see Section 4.5.1).

Let us consider an example of lane aggregation from LNDS to HLM (see Figure 4.23). LaneGroup1 and LaneGroup2 both contain two lanes, and the same side of lanes in each lane group is contiguous and longitudinally connected. For instance, Lane22 is the contiguous successor lane of Lane12, and they are longitudinally connected. The difference of LaneGroup1 and LaneGroup2 is that Lane12 has a guardrail as lane boundary, while Lane22 does not. In this case, human beings perceive Lane12 and Lane22 as one lane regardless of the presence of guardrails. We followed the human perception for our modelling by applying the aggregation rules, reducing

the amount of data. After the complex transformation, Lane12 and Lane22 are aggregated into the high-level entity Lane2 and Lane11 and Lane21 into Lane2.

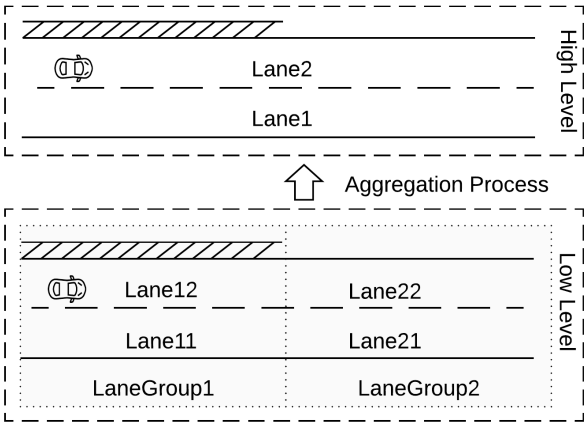


FIGURE 4.23: Aggregation of low-level lanes into high-level lanes.

The aggregation process is as following: First, the instances of `lnds:Lane` in LNDS that will be aggregated are assigned with a value generated from the identified properties such as tile ID, the number of lanes, the feature ID, the feature type, the travel direction, the lane index, and the lane type (see Listing 4.3).

```
lnds:laneValue[?l, ?v]:-
lnds:LaneGroup[?lg], lnds:hasLane[?lg,?l], lnds:tileId[?lg, ?tileId],
lnds:numLane[?lg, ?numLane], lnds:hasFeatureReference[?lg, ?fr],
lnds:refersTo[?fr,?f], lnds:directionRefValue[?fr,?dv],
lnds:featureId[?f,?fid], lnds:featureTypeValue[?f,?ftv],
lnds:index[?l,?idx], lnds:laneTypeValue[?l,?ltv],
BIND(CONCAT(STR(?tileId),STR(?numLane),STR(?fid),
STR(?ftv),STR(?dv),STR(?idx),STR(?ltv)) AS ?v).
```

LISTING 4.3: Lane value generation for aggregation.

Second, for all the instances of `lnds:Lane` with the same value of `lnds:laneValue`, the rules generate one instance of `high:Lane` and assign values to data property `high:laneNum` and `high:length` of newly created instances (see Listing 4.4). We omit the value generation of data property `lnds:LBValue` for `lnds:LaneBoudnary` here, since it is achieved similarly as the data property `lnds:laneValue` of `lnds:Lane`.

```
high:Lane[?h], high:laneNum[?h,?idx],
high:hasLow[?h,?l1], high:hasLow[?h,?l2]:-
lnds:Lane[?l1], lnds:laneValue[?l1, ?v],lnds:index[?l1,?idx],
lnds:hasLeftLB[?l1,?lb1],lnds:hasRightLB[?l1,?lb2],
lnds:LBValue[?lb1, ?v1], lnds:LBValue[?lb2, ?v2],
lnds:Lane[?l2], lnds:laneValue[?l2, ?v],lnds:index[?l2,?idx],
lnds:hasLeftLB[?l2,?lb11], lnds:hasRightLB[?l2,?lb22],
lnds:LBValue[?lb11, ?v1], lnds:LBValue[?lb22, ?v2],
BIND(SKOLEM("highLane", ?v) AS ?z),
BIND(IRI(CONCAT("http://.../Map/Ontologies/High#", str(?z))) AS ?h).

high:length[?h, ?d]:-
high:Lane[?h], high:hasLow[?h, ?l],
AGGREGATE(lnds:length[?l,?len] BIND(SUM(?len) AS ?d).
```

LISTING 4.4: Aggregate the lanes with the same lane value.

4.6 Evaluation

In this section, three representative use cases supported by two-level map ontologies are presented. The use cases show how the ontology and rules can achieve the following advantages:

- the *interoperability* of a semantic representation of the map. Logical inconsistency can be detected using rules by combining the high-level map ontology with sensor observations (see Section 4.6.1);
- *reducing data size* via layered abstraction. The number of map objects can be reduced, in particular, the number of lanes and lane boundaries in highways by the two-level knowledge abstraction (see Section 4.6.2);
- providing a *unified view* over different map formats. By lane aggregation, two different map formats, namely NDS and HDLM, result in a unified high-level lane representation (see Section 4.6.3).

4.6.1 Use Case: Inconsistency Detection with Sensor Observations

Fusing the data from vehicle sensors and the map can help to form a consistent environmental view of the surroundings of the vehicle. The AD system reasons on the generated environmental model to decide which actions are appropriate for specific driving situations. This use case shows that the proposed high-level ontology enables us to detect logical inconsistencies by reasoning on the environmental model based on the vehicle sensor data and the map. We illustrate this by a false emergency break scenario, which might result from mislocalisation of the car's position due to, for example, noisy or non-available sensor data.

As shown in Figure 4.24, the vehicle E is driving in Lane2 following vehicle C2, and vehicle C1 is driving in Lane1. Due to the non-availability of some sensor data, E (in grey) is mislocalised in Lane1 instead of Lane2. The object tracking system (OTS) of E provides a list of positions of tracked objects (C1 and C2) around E. The provided positions are relative to the position of E as E is the origin of the vehicle coordinate system. Because E is mislocalised in Lane1 and the calculated distance of E to C1 is less than the distance threshold of the autonomous emergency braking system (e.g., 80 m), the braking system of E would initiate partial braking to reduce the speed and give the driver valuable time to react in order to avoid a collision with C1 "in front", although, in reality, E is driving in Lane2 with a safe distance to C2.

This false emergency break could be avoided by creating an ontology for sensor data fused with the HLM ontology as shown in Figure 4.24. The isInRightFrontOf, isInFrontOf and isIn relations are derived from sensor data and represent, respectively, that: i) C1 is in the right front lane of E; ii) C2 is in front of E on the same lane; iii) E is localised in Lane1. According to the map, the lane (Lane1) where E is localised is the rightmost lane. This contradicts the derived knowledge of isInRightFrontOf, which means that a car is in the right front lane of E because it is impossible to have a lane at the right-hand side of the rightmost lane. Listing 4.5 shows the rule classifying this inconsistent knowledge as a fault.

4.6.2 Use Case: Lane Aggregation for LNDS

Figure 4.25 shows the result of the aggregation rules over map data covering 63.75 km^2 in Germany, considering motorway and urban scenarios. The number of high-level

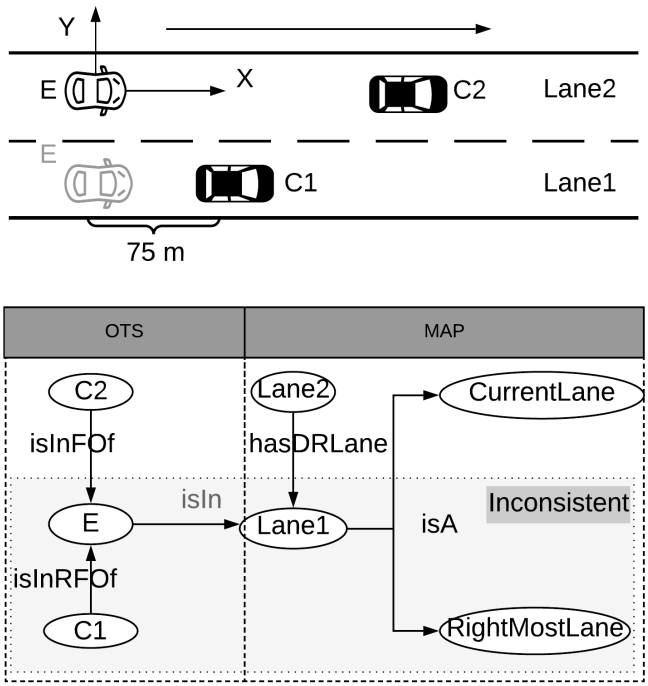


FIGURE 4.24: Illustration of the environmental model of the ego vehicle based on the raw data provided by the Object Tracking System (OTS). The main relations of this model are isInFOf (isInFrontOf), isInRFOf (isInRightFrontOf), isIn, hasDRLane (hasDirectRightLane) and isA

```
high:Fault[?fault] :-
icity-vehicle:Vehicle[?v1], high:isIn[?v1, ?lane],
high:RightMostLane[?lane],
icity-vehicle:Vehicle[?v2], high:isInRightFrontOf[?v2, ?v1],
BIND(SKOLEM("fault", ?v1, ?v2, ?lane) AS ?z),
BIND(IRI(CONCAT("http://.../Map/Ontologies/High#", str(?z))) AS ?fault)
    .
```

LISTING 4.5: The rule for fault detection

Lane entities decreases after the aggregation process in both scenarios. For motorways, the reduction rate of the Lane and LaneBoundary entities can reach 50%, which helps decrease the amount of stored data and increases processing efficiency. In the urban scenario, the ratio is less visible due to the complex nature of the topology and geometry of the roads inside the city.

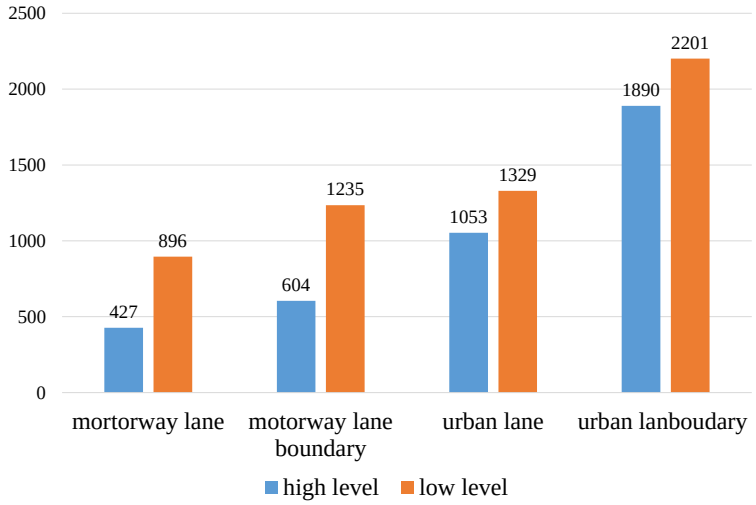
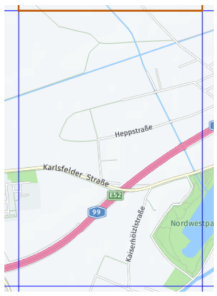


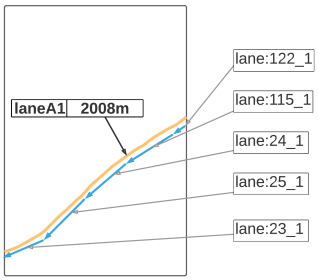
FIGURE 4.25: Aggregation over lanes and lane boundaries in motorway and urban scenarios

4.6.3 Use Case: Unified Lane View for LNDS and LHERE

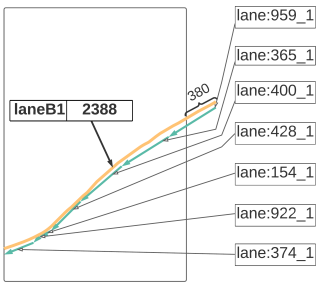
The high-level HLM ontology can provide a unified view over different map formats via ontology transformation. This use case shows that the lanes in the HLM ontology which are resulted from different low-level map ontologies have the similar semantics. Figure 4.26a shows the Federal Highway A99 in Germany partially in reality. The lanes covering this part of the road are modelled differently in a NDS map (see Figure 4.26b) and a HDLM map (see Figure 4.26c).



(A) The reality of a part of Federal Highway A99 in Germany.



(B) The lanes in a LNDS (blue) aggregated in a lane in HLM (yellow).



(C) The lanes in a LHERE (green) aggregated in a lane in HLM (yellow).

FIGURE 4.26: An example of lane aggregation over LNDS and LHERE resulted in a united lane view in HLM.

The presented road is covered with five lane groups in the NDS map, and the lane with index 1 are illustrated as blue arrow lines. After lane aggregation, the lanes represented in the LNDS ontology are aggregated into one single high-level lane laneA1 (yellow line) in the HLM ontology with length 2008 m. As for the HDLM

map, the same road is covered with seven lane groups and the lanes represented in the LHERE ontology is aggregated into a high-level lane laneB1 (yellow line) with length 2388 m. The 380 m length difference from laneA1 is caused by the fact that HDLM models the lane groups at the tile border differently from NDS. However, the part of the laneB1 within the map tile has the same length as laneA1, which shows that the HLM ontology provides the generic lane representation over the LNDS ontology and the LHERE ontology. The detailed information about lane groups with the lane of the index 1 for both maps are presented in Table 4.13. For detailed RDF graph representations of related lanes in LNDS and LHERE respectively see Section A.1 in the Appendix. For the commonality and differences between NDS and HDLM, please refer to Section 2.2.1.

TABLE 4.13: Lane information in NDS and HDLM wit Lane Group (LG) ID, number of Lanes (#Lane), the lane index, and the lane length.

(A) Lane information in NDS

LG ID	#lane	index	length (m)
122	5	1	131
155	5	1	520
24	5	1	519
25	5	1	518
23	5	1	320
Total			2008

(B) Lane information in HDLM with last four digits lane group ID

LG ID	#lane	index	length (m)
8959	5	1	511
8365	5	1	520
9400	5	1	519
7428	5	1	354
7154	5	1	133
8922	5	1	33
0374	5	1	318
Total			2388

4.7 Concluding Remarks

In this chapter, we investigated the development and advantages of two-level map ontologies for Autonomous Driving scenarios. To this end, a practical methodology for building the unified high-level ontology and for specific low-level ontologies is presented. This methodology combines a top-down approach and a bottom-up approach. The top-down approach is based on METHONTOLOGY, and the bottom-up approach is adapted from Uschold et al. We present concrete practices being applied in this kind of setting. Additionally, relevant ontology design patterns are presented, as well as derived rule patterns for knowledge processing and spatial reasoning. Furthermore, the generic road representation is investigated, and two main map formats in the HD map area are researched, i.e., the NDS map and the HERE Live HD map (HDLM). We have applied the proposed methodology to build the high-level map ontology HLM and two low-level map ontologies.

Firstly, the HLM ontology has been semantically described. The ontology covers static road knowledge and dynamic vehicle information. The generic concepts and spatial relationships of roads and lanes are modelled in this ontology. This permits to provide a common representation of the road knowledge over different types of map data in various formats.

Second, we developed two low-level map ontologies, namely LNDS and LHERE, converging NDS and HDLM maps, respectively. The LNDS ontology provides the main concepts and relationships for capturing the NDS format to support the knowledge abstraction process and eventually populate the HLM ontology. The LHERE ontology is developed similarly. We evaluated both ontologies in terms of ABox

generation and semantic enrichment stage performance and showed that both ontologies can answer real-world queries, such as lane identification.

Finally, we evaluated the two-level ontology approach by three use cases. The first use case demonstrated the interoperability with other domains of knowledge by combining the HLM ontology with sensor observations to detect logical inconsistencies using rules. The second use case shows the ability to reduce data size via levelled abstraction by the example of lane aggregation. The last use case shows the ability to provide a unified high-level road view over different low-level map formats, namely LNDS and LHERE.

To conclude, we demonstrate the benefits of the two-level ontological maps models and rule-based knowledge processing: 1) interoperability of a semantic representation of map; 2) reducing data size via layered abstraction, and 3) the unified view over different map formats via integration. The flexible schema representation and query language are additional advantages that we observed in the proposed approach.

Chapter 5

Dynamic Map Processing

HD maps are used for AD systems as a virtual sensor to anticipate the road ahead, understand and navigate the vehicles' environment and make decisions [132]. Due to their high precision and high dynamic information content about the traffic situations, the HD maps get outdated after a short period and therefore profits from regular updates. Furthermore, HD maps are extremely detailed and require significantly more processing power and computational resources than SD maps. It is largely impossible to store a completely detailed map within a vehicle's navigation system. Hence, the navigation system needs to constantly process and update the map data while the car is progressing along a route, and care has to be taken to provide any relevant information in due time. The characteristics of HD maps require a novel approach that allows for a light-weighted generic representation of the road environment to support vehicle decision-making and a continuous map process mechanism to provide sufficient road knowledge ahead. In order to meet this need, we propose using two-level map ontologies and rules with a spatial-sliding window to perform incremental spatial reasoning while map data is continuously requested and processed.

Problem statement. In this chapter, we investigate the problem of efficient spatial reasoning while the input map data is changing. This chapter addresses the second level in the general proposed contributions presented in this thesis, i.e., an efficient dynamic map process strategy to provide continuous road knowledge ahead.

The following research question is investigated in this chapter:

> RQ2: Can an ontology-based approach perform efficient knowledge processing and spatial reasoning while the knowledge base is continuously changing?

Proposed solution. We tackle the problem of incremental knowledge processing and spatial reasoning by applying an ontology-based approach. The map data are requested and processed by applying the two-level map ontologies described in the previous chapter. Spatial sliding window and *two levels of datastores* are used to achieve efficient knowledge processing and spatial reasoning. The low-level datastores are dedicated to the low-to-high map knowledge process and provide high-level map knowledge to the high-level datastore. The spatial and decision rules are embedded in the high-level datastore and utilise the available high-level map knowledge. Furthermore, the vehicle *spatial window* is operated on the high-level datastore, responsible for prefetching future map tiles and determining expired map objects.

The contributions of this chapter are outlined as follows:

- A knowledge-spatial architecture realised in a practical solution for dynamic map processing.
- The utilisation of a spatial-sliding window concept for continuous knowledge processing in terms of prefetching and deletion.
- An empirical evaluation of the approach for scalability in a highway scenario within our prototype called *SmartMapApp*.
- A discussion about two alternative deployment choices, namely, *client* deployment and *client-server* deployment.

The rest of this chapter is structured as follows. In Section 5.1, the context and requirements of the motivating scenario are described. The core contributions, i.e., the ontology-based approach and its implementation are presented in Section 5.2 and Section 5.3. In Section 5.4, the empirical results demonstrate the feasibility of the proposed approach, and two deployment choices are discussed in Section 5.5. Finally, concluding remarks for this chapter are presented in Section 5.6.

5.1 Motivation

Figure 5.1 illustrates the motivating scenario where a vehicle is progressing along a route and is monitoring the situation. First, based on the received GPS position and the road environmental knowledge, it localises itself in a lane and stores it as the current lane. Subsequently, it understands the environmental knowledge around the current lane. For instance, it knows that the current lane is the left-most lane and derives that it cannot change the lane to the left. Further, based on the lane-road relation, it also knows that the road is a highway, and there are ramps (exit/entry) along the way.

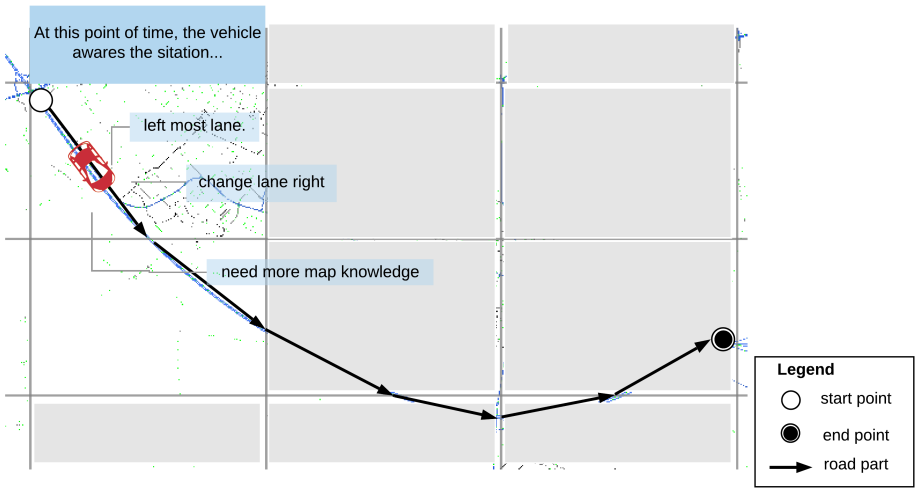


FIGURE 5.1: A vehicle progressing along a route with situation awareness.

As the vehicle progresses, it needs to change lanes to the right to exit the highway at some point in time according to the route. Since the vehicle is equipped with a limited range of foresight, it will proactively request more map data to ensure

sufficient road knowledge ahead periodically and remove the long past (expired) road knowledge.

As illustrated in the above scenario, the key for proactive actions is that the vehicle can understand the situation concerning time or space and predicate the future status [22]. This is known as situation awareness, as defined in the work of Endsley [67]. It is a major function before decision-making and the lateral/longitudinal motion control of the vehicle. To achieve situation awareness, it is necessary for the vehicle to understand the Spatio-temporal relationship between itself and the perceived road entities in the context provided by HD maps. The ontology-based approach for autonomous vehicle situation awareness using maps has already been studied [18, 189]. However, less work has been done towards using HD maps and the vehicle position to provide road environmental context for vehicle situation awareness.

5.2 Design

In this section, we first describe the conceptual architecture and the related framework. Then we illustrate the design of ontological modules of location awareness, lane change and continuous map processing.

5.2.1 Conceptual Architecture

With the aim of providing efficient situation awareness and continuous map processing, we design the knowledge-spatial architecture illustrated in Figure 5.2. The proposed architecture is extensible and able to accommodate additional components for accessing other types of data sources. In the following, the two-dimension of the architecture are described.

The *knowledge dimension* (vertical axis) illustrates a knowledge abstraction process from the format-specific and detailed low-level ontologies to the generic high-level ontology. The horizontal (time) axis represents road knowledge acquisition events, triggering the knowledge abstraction process via spatial reasoning. Spatial reasoning considers updated vehicle motion events determined in the knowledge abstraction process and searches for spatial patterns to derive the relevant consequences of what is happening on the road. Different low-level ontologies (e.g., LNDS and LHERE) for the various map formats can be used to populate the high-level ontology (e.g., HLM ontology). Application-oriented queries (e.g., AD functions), thus, can be posed over the high-level ontology, resulting in a flexible architecture for realising high-level driving functions using different map formats.

The *spatial dimension* is orthogonal to the knowledge dimension and correlates facts that are true within a certain space and time at the high-level ontology. It describes the continuous spatial reasoning process regarding the updated vehicle position and dynamic road environmental knowledge. In this dimension, the vehicle can perform situation awareness and decision-making. We adopt the notion of a *spatial window* with a fixed width or region in terms of geographic elements shifting (sliding) over a path line. Within the spatial window, the prefetching mechanism incrementally updates the high-level ontology to ensure sufficient road knowledge ahead. Inspired by Mokbel et al. [135], we use the notion of *spatial expiration* depending on the spatial location of a moving object, e.g., the ego vehicle, and stored data expires only when the object leaves the spatial window.

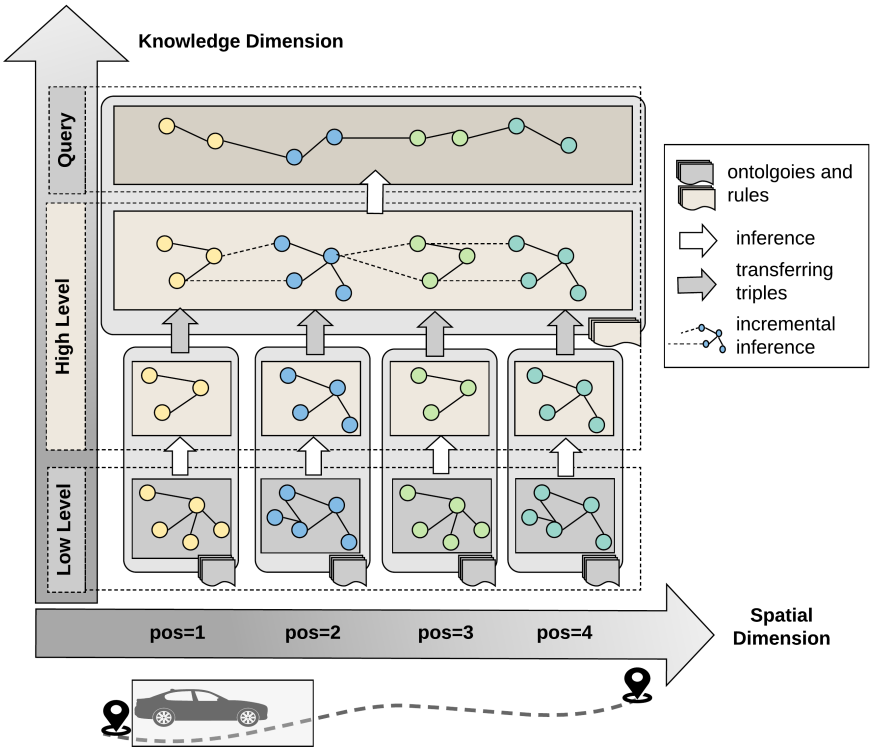


FIGURE 5.2: Overview of the knowledge-spatial conceptual architecture.

Inspired by the work of Armand et al. [19], we design the workflow for using the HLM ontology described in Section 4.2 and a set of rules for vehicle situation awareness and decision-making in Figure 5.3. The low-level map ontologies and the vehicle position constitute the perception of the vehicle, whose meaning needs to be understood by situation awareness. Since the vehicle position is Spatio-temporal data describing when and where the vehicle is driving, the current situation of the vehicle can be derived by location awareness rules for each received position via non-monotonic reasoning. Consequently, various decisions can be made based on the newly derived current location using decision rules, such as lane change and prefetching map tiles.

Situation awareness is based on the HLM ontology (TBox and ABox) and rules. The TBox and situational rules consist of a conceptual description of the road entities in a situation. The TBox and rules are permanent. The Abox is the high-level abstraction of the low-level perception. The ABox is the changing part of the HLM ontology, which forms the dynamic vehicle environment, and it is maintained by the reasoner. The dynamics of the HLM ABox is caused by two factors:

- Vehicle position. The vehicle position is received each one second, the deduced current situation of the vehicle is updated accordingly by non-monotonic reasoning. This means that the current situation is a temporary conclusion drawn by the reasoner and will be retracted based on the future received position.
- Spatial sliding window. The road knowledge of the vehicle is updated with future road knowledge, and the expired map objects are deleted based on the spatial sliding window of the vehicle. The process of update road knowledge is achieved by incremental reasoning.

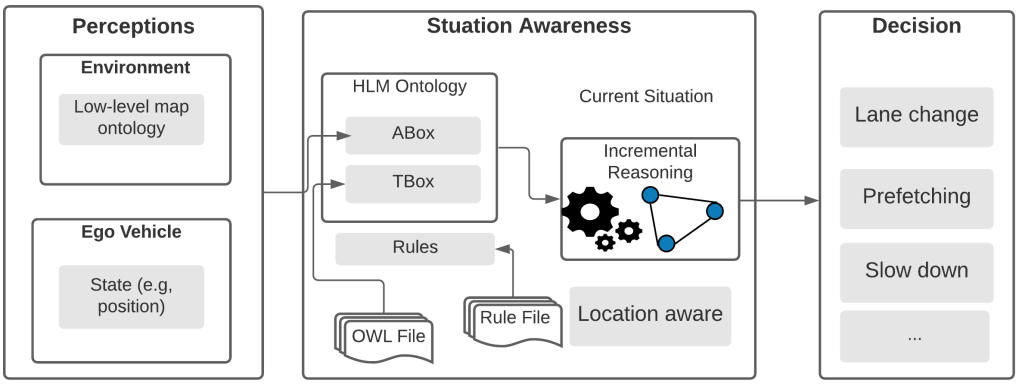


FIGURE 5.3: The exploitation of the HLM ontology for vehicle situation awareness.

5.2.2 Location Awareness

In a nutshell, location awareness is the process of determining the vehicle's location on a given map. It is done through map matching using the received GPS position, a Spatio-temporal event happening at a particular time and place. The spatial aspect of the position can be used to match the corresponding position of the road (lane). With the map knowledge, the type of localised lane (e.g., left/righ most lane or middle lane) where the vehicle is in can be inferred to facilitate the decision making. The time aspect of the position provides temporal information to infer temporal knowledge of the car, such as the current and previous lanes of the vehicle.

First, we describe the ontology module of location awareness, and then we illustrate the spatial and temporal aspects of location awareness in detail. The algorithm and rules are presented at last.

Ontology Module

Figure 5.4 shows the relevant classes and properties of the HLM ontology regarding the location-awareness module. It focuses on capturing the notion of "current" location, which is represented by the class `high:CurrentLocation`. The instance of `high:CurrentLocation` is predefined and will not change over the time. However, the instances of the classes `high:CurrentPosition`, `high:CurrentLane`, `high:PreviousCurrentLane` and `high:CurrentRoardPart` are dynamic and inferred for each newly added instance of `high:Position`. The data property `high:timestamp` of `high:Position` is used to infer the instance of `high:CurrentPosition`. The class `high:CurrentLane` is defined as a subclass of `high:Lane`. With the defined subclasses of `high:Lane`, the specific type of lane can be inferred, such as `high:LeftMostLane`. Similar class reasoning applies to `high:CurrentRoadPart` as well.

Localisation

The vehicle must localise itself at both lane level and road level for each received GPS position. Localisation on road level is required for function activation, e.g., if the function is limited to a certain set of roads, such as highways, to determine the route from the current position to the desired destination. Map matching algorithms at the road level are actively studied for vehicle localisation [97]. The research was

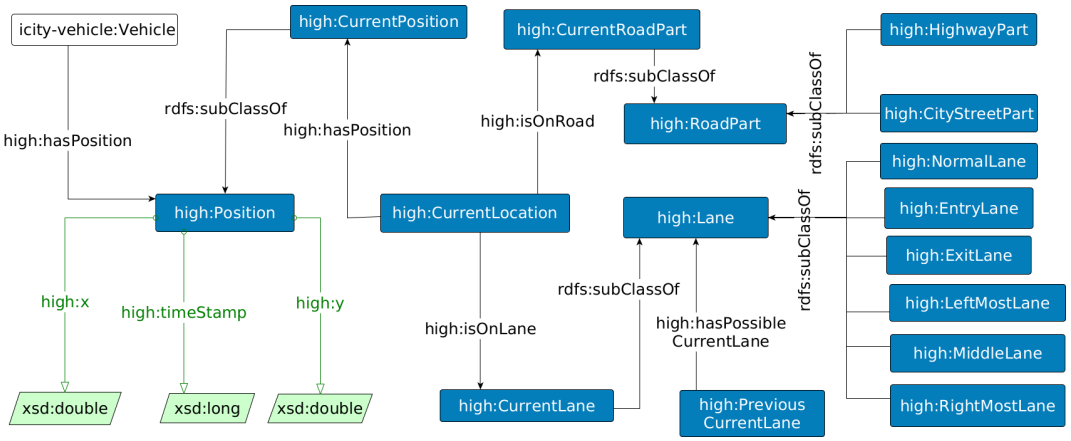


FIGURE 5.4: Vehicle location awareness modelling based on the vehicle position and map knowledge.

done by Lou et al. [127], who proposed a Spatio-temporal matching algorithm that considers the spatio and temporal factors to improve the accuracy and efficiency of map matching at the road level.

As for lane-level localisation, however, the temporal aspect is not yet studied. Our work adopts a map-based lane localisation approach using GPS position, considering the spatio and temporal factors. The spatial aspect is used to find the closest lane shape point in the HD map using GPS coordinates. The temporal aspect is used to narrow down the search scope of the shape points by identifying the possible current lanes. Before we describe the lane localisation in detail, let's consider the notion of *GPS trajectory*.

Definition 5.2.1 (GPS Trajectory [126])**.** A GPS Trajectory is a sequence of GPS points $T = \{p_1, p_2, \ldots, p_n\}$. Each GPS point $p_i \in T$ is represented as a tuple $p_i(x_i, y_i, t_i)$ (x_i and y_i are the x and y coordinates of the GPS point, and t_i denotes the timestamp).

Figure 5.5 shows an example of a vehicle GPS trajectory. The timestamp t of the GPS point p provide temporal aspect to infer the "current" status of the vehicle. Hence, we define *current position* as:

Definition 5.2.2 (Current Position)**.** A current position cp of the vehicle for a vehicle trajectory $T = \{p_1, p_2, \ldots, p_n\}$ at any time is the point $p_i \in T$ with maximum t_i value.

Based on the current position, the *current lane* can be identified by comparing the distances between the current position and all other lane shape points.

Definition 5.2.3 (Current Lane)**.** The current lane for current position cp is the lane l with the closest point c such that $c = \arg\min_{\forall c_i \in l} dis(c_i, cp)$, where $dis(c_i, cp)$ is the distance between cp and any point c_i on l. Figure 5.6 shows an example of identified current lane l_1 where the closest point c is located.

Possible Current Lanes

The temporal aspect of the received GPS point can be used to refine the search scope of current lanes to reduce the amount of distance calculation. To achieve this goal, we rely on the spatial relationship (e.g., neighbouring and successor relationships) associated with previously identified current lanes to infer possible current lanes.

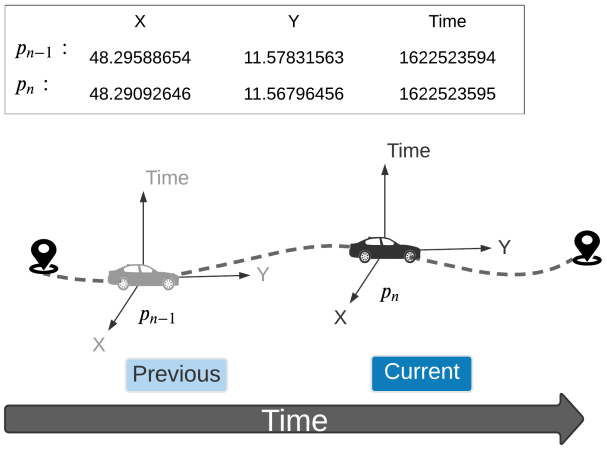


FIGURE 5.5: Vehicle position with coordinate and timestamp.

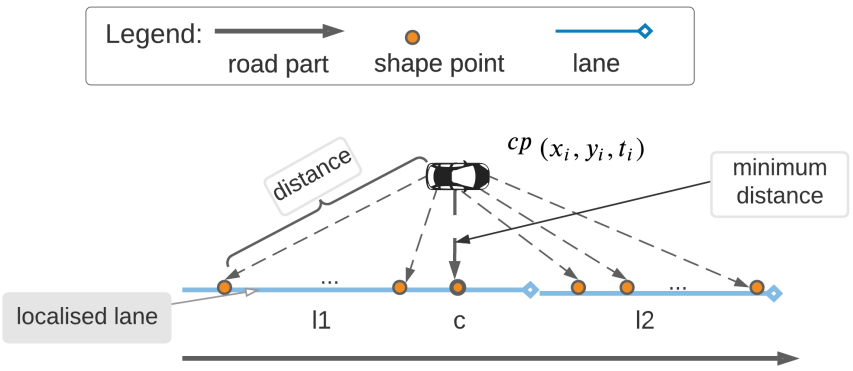


FIGURE 5.6: Vehicle lane localisation is based on the vehicle position and the lane shape points.

Depending on whether the vehicle has entered the next part lanes on the road, two cases are identified. The first case is when the current vehicle position is still located on the same part of the lanes for the previous lane. The second case is when the vehicle has crossed the previous lane, and entered the next part of lanes on the road. It must be noted the premise of the approach is that vehicles will progress ahead on the road as time does.

Figure 5.7 illustrate the first case where the vehicle's current position is possibly located in the same lane as the previous position did. In this case, the possible current lanes are the previously identified lane l and its neighbouring lanes nl. We use the hasNeighbouringLane relationship to retrieve nl and infer l and nl as possible current lanes. Figure 5.8 shows the second case where the vehicle's current position is possibly located in the next part of lanes on the road. That is, to day, the possible current lanes are the immediate next lane l of the previous identified lane pl and the neighbouring lanes nl of that immediate next lane$l$. We use hasDirectNextLane relationship to retrieve l and the combination of hasDirectNextLane$\circ$hasNeighbouringLane relationship to retrieve nl, and infer l and pl as possible current lanes.

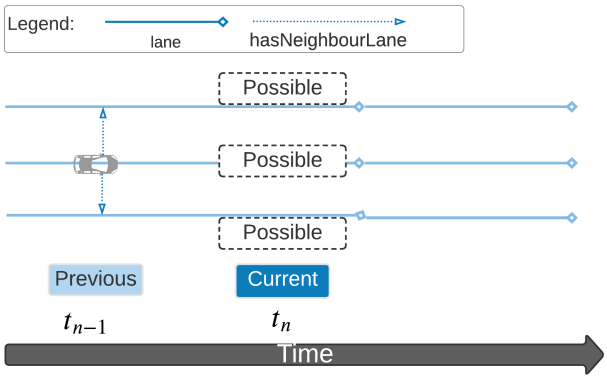


FIGURE 5.7: Possible current lanes are the previous current lane and its neighbouring lanes.

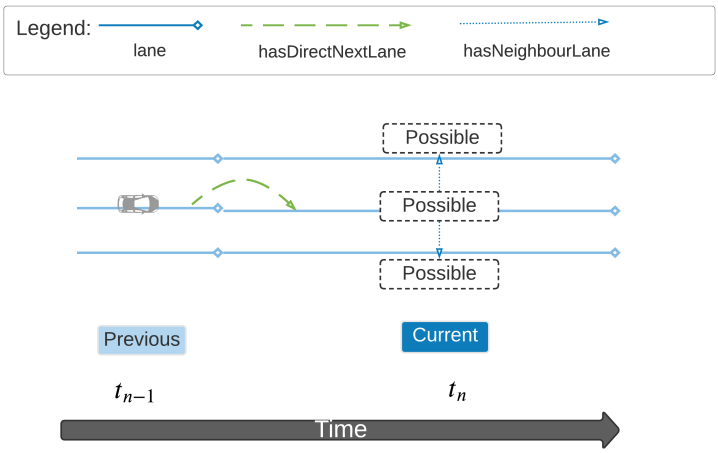


FIGURE 5.8: Possible current lanes are the direct next lane and related neighbouring lanes.

Algorithm and Rules

The specific algorithm of location awareness in terms of current lane identification is shown in Figure 5.9. The algorithm starts by receiving a GPS point, and calculating the distance between the received GPS point and all the points of lanes. Then the current lane is identified by selecting the lane having a point with minimum distance to the received GPS point. For any further GPS point, the possible current lanes are derived based on the previously identified current lane. It follows that the current lane is identified based on the minimum distance between the points of the possible current lanes and the received GPS point. The algorithm is continuous as long as the vehicle receives a GPS point. We then describe how the rules are used to model the algorithm starting from the second received GPS point onwards.

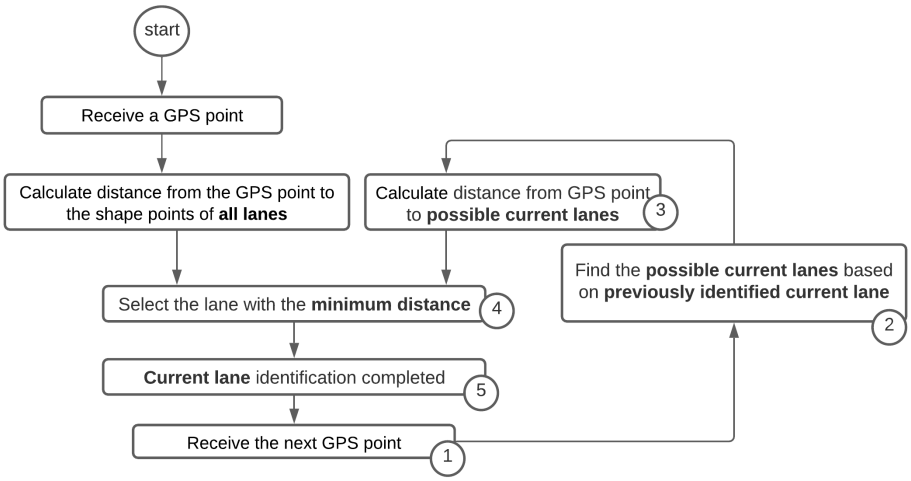


FIGURE 5.9: Current lane identification algorithm. Key steps are marked with numbers.

(1) Receive a GPS Point. A `INSERT` SPARQL query is used to add a position instance with the x-y coordinate values and timestamp as an explicit fact for each received GPS point. The timestamp value is assigned by using the `NOW` function, which returns the date-time value of the current moment (see Listing 5.1). As time goes by, all the inserted GPS points form a historical vehicle trajectory. To find the *current* position in the trajectory, we use `AGGREGATION` with `MAX` function to infer the position with the maximum timestamp value as the current position (see Listing 5.2).

```
INSERT { ?position a high:Position ;
         high:timeStamp ?timeStamp ;
         high:x "+lat+";
         high:y "+lon+". }
WHERE {
    BIND(xsd:dateTime(NOW()) AS ?timeStamp)
    BIND(IRI(CONCAT("http://.../High#position", ?timeStamp)) AS ?position)
}
```

LISTING 5.1: SPARQL query to insert vehicle position.

```
CurrentPosition[?pos1]:-
AGGREGATE(timeStamp[?pos, ?ts] BIND MAX(?ts) AS ?mts),
timeStamp[?pos1, ?mts].
```

LISTING 5.2: Current poisiton inference.

(2) Possible current lanes. To find the possible current lane, we first identify the previous current lane where the vehicle is on. This is achieved by identifying the lane with the maximum timestamp value, which is similar to how the current position is determined. Subsequently, then the possible current lanes can be inferred based on that identified previous, current lane (see Listing 5.3).

```
PreviousCurrentLane[?l1]:-
AGGREGATE(laneTimeStamp[?l, ?ts]  BIND MAX(?ts) AS ?mts),
laneTimeStamp[?l1, ?mts].

hasPossibleCurrentLane[?l,?l],hasPossibleCurrentLane[?l,?nbl]:-
PreviousCurrentLane[?l], hasNeighbourLane[?l, ?nbl].

hasPossibleCurrentLane[?l,?dnl],hasPossibleCurrentLane[?l,?nbl]:-
PreviousCurrentLane[?l], hasDirectNextLane[?l, ?dnl],
hasNeighbourLane[?dnl, ?nbl].
```

LISTING 5.3: Possible current lanes inferences based on identified previous current lane.

(3) Distance calculation. The distance between the current position and all points associated with the identified possible current lanes are calculated (see Listing 5.4).

```
CoordinateDistance[?cd], hasSource[?cd,?n],
hasTarget[?cd,?o], distance[?cd,?d]:-
hasPossibleCurrentLane[?l,?m], hasPoint[?m,?o], x[?o,?x], y[?o,?y],
CurrentPosition[?n], x[?n,?cx], x[?n,?cy],
BIND(sqrt((?x-?cx)*(?x-?cx)+(?y-?cy)*(?y-?cy)) AS ?d),
SKOLEM("cd",?n,?o, ?d, ?cd).
```

LISTING 5.4: Distance cacluation.

(4) Minimum distance inference. The minimum distance is inferred using `AGGREGATION` with `MIN` based on a set of distances calculated in the previous step (see Listing 5.5).

```
MinCoordinateDistance[?cd1]:-
AGGREGATE(distance[?cd,?d] BIND MIN(?d) AS ?md),
distance[?cd1,?md].
```

LISTING 5.5: Minimum disntacne inference.

(5) Current lane identification. The current lane is inferred based on the identified minimum distance. Note that the timestamp value of the current position is added to the current lane such that it can be used for the upcoming, current lane identification process using the next received GPS point (see Listing 5.6).

```
CurrentLane[?l]:-
MinCoordinateDistance[?cd], hasTarget[?cd,?pos], hasPoint[?l,?pos].

laneTimeStamp[?l,?ts]:-
CurrentLane[?l], CurrentPosition[?pos], timeStamp[?pos,?ts].
```

LISTING 5.6: Current lane inference and timestamp association

Datalog program containing nonmonotonic rules (e.g., negation and aggregation) must guarantee stratification to provide well-defined semantics and thus reach a fixpoint. The predicate dependency can determine if a Datalog program is stratifiable. The program is stratifiable if the predicate graph has no directed cycles with a negative edge. This is not the case for the rules presented in Listings 5.3, 5.4, 5.5 and 5.6 as the rules cannot be stratified due to a negative predicate dependency cycle (see Figure 5.10).

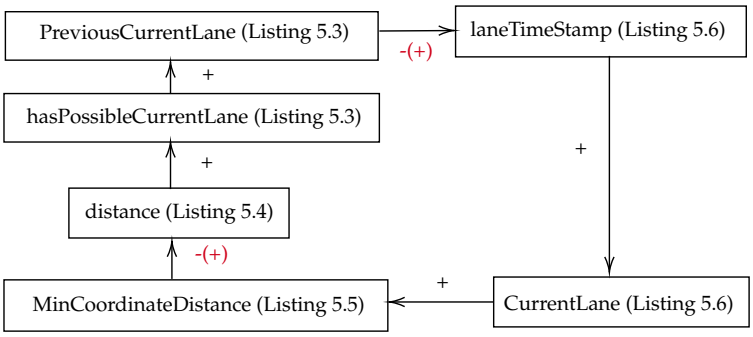


FIGURE 5.10: The predicate dependency graph with a negative dependency cycle, excluding the irrelevant predicates. The arrow shows the dependence between the head predicate and body predicate of the rule. Positive (negative) dependencies are labelled by + (–).

To break the negative dependency cycle, the rule of inferring the `laneTimeStamp` relationship in Listing 5.6 is changed to a SPARQL INSERT query (see Listing 5.7).

```
INSERT { ?l :laneTimeStamp ?ts. }
WHERE {
     ?l a :CurrentLane.  ?pos1 a :CurrentPosition; :timeStamp ?ts. }
```

LISTING 5.7: SPARQL query for adding timestamp to current lane.

Rules and Queries Execution Sequence

Since there are dependencies between SPARQL INSERT queries and the rule evaluation, the execution sequence of the queries and rules affects the computed result. The current lane identification implementation involves four major steps: (1) SPARQL INSERT a GPS point; (2) the incremental updates starting from the `CurrentPosition` inference; (3) SPARQL INSERT `laneTimeStamp` to the identified `CurrentLane` instance; (4) the incremental updates starting from the `PreviousCurrentLane` inference. Figure 5.11 shows the execution sequence, and details of each step are explained as follows:

(1) SPARQL INSERT a GPS point with a coordinate and a timestamp.

(2) The incremental updates are triggered by the insertion, and the instance of `CurrentPosition` is determined by nonmonotonic reasoning.That is, there is only one instance inferred as `CurrentPosition` at any one time, and all previous `CurrentPosition` derivations are retracted. Thus, the number of distance calculations required after `CurrentPosition` derivation for `CurrentLane` inference is limited to the number of points in the possible current lanes. The derivation retraction applies to any other rules that are dependent on the derivation of `CurrentPosition`.

(3) Once the rule evaluation in step (2) reaches a fixpoint, a INSERT query is executed to add a lane timestamp to the derived instance of, `CurrentLane` preparing for the next received GPS point.

(4) Following that, incremental updates are triggered once again, and nonmonotonic reasoning derives the new instance of `PreviousCurrentLane`. As with the inserted GPS points, nonmonotonic reasoning guarantees one derivation of `PreviousCurrentLane` over stored `laneTimeStamp` data. Although the new

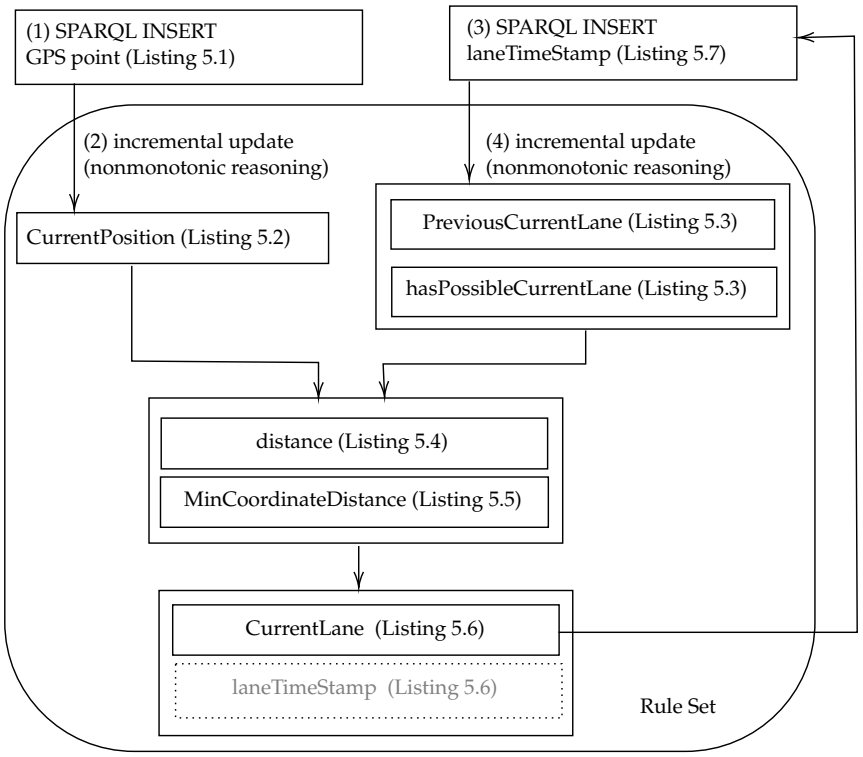


FIGURE 5.11: The execution sequence of the rule set and the SPARQL INSERT query implementing the current lane identification algorithm are shown in Figure 5.9.

derivation of `PreviousCurrentLane` triggers the inference chain, the current rule evaluation cycle's derivations of `MinCoordinateDistance` and `CurrentLane` remain unchanged, because the next GPS point is not received yet. This represents the fact that the vehicle's current state changes only when a new GPS point is received. As a result, once the next GPS point is received and inserted in step (1), the entire rule evaluation cycle starts again.

In general, the current lane identification process involves two types of facts: (1) explicit facts such as GPS points and lane timestamps that accumulates in the data over time; and (2) implicit facts such as the instances of `CurrentPosition`, `MinCoordinateDistance` and `CurrentLane`, that are derived by the rule evaluation is part of the incremental maintenance via nonmonotonic reasoning.

5.2.3 Lane Change Activity

In general, a lane change is defined as a driving manoeuvre that shifts a vehicle from one lane to another where both lanes have the same travel direction. AD systems need to, first, understand the situation and recognize the necessity to change lanes. Afterwards, the lane change manoeuvres have to be planned to reach the navigation goal. Lane change has gained increasing attention in the AD field in the last year [184, 182, 122]. Ontology-based automated vehicle guidance focusing on road situations has gained increasing attention in the last years [78, 24, 43, 119]. In this thesis, we show the utilization of a generic high-level map (HLM) ontology and rules to achieve lane change functionality with selected use cases.

Consider the two scenarios shown in Figure 5.12 in which a vehicle is driving on a highway. In both cases, there is a lane (green) that is part of the route that is not

longitudinally reachable by the current lane of the vehicle. We refer to such lanes as the target lanes. To reach the target lanes, the vehicle must perform a certain number of lateral lane changes. In such situations, the AD system generates a lane change notification if the remaining distance reaches a threshold (e.g., 3 km). Subsequently, a corresponding left or right lane change plan is generated.

(A) Lane Lane3 is not longitudinally reachable.

(B) Lane5 is not longitudinally reachable.

FIGURE 5.12: Scenarios for a lane change.

In this section, we begin by describing the ontology module of lane change activity. Then we illustrate the lane change detection and lane change guidance and the related algorithm and rules, respectively.

Ontology Module

Figure 5.13 shows the relevant classes and properties from the HLM ontology regarding the lane change activity module. The modelling of lane change activity is inspired from the *Activity Ontology* developed in the work of Abdalla et al. [8].

The main class is `high:ChangeLaneActivity` in Lane Change Guidance Box (see Figure 5.13). It describes the occurrence of lateral lane change activity at some point in time and place. This is designed by the data property `high:timeStamp` and linking to class `high:CurrentLocation` via property `high:hasLocation`. The start lane of the planned lane activity is modelled by connecting the instance of `high:ChangeLaneActivity` to the instance of `high:CurrentLane` via the property `high:fromLane` and the destination lane is modelled by connecting the instance of `high:ChangeLaneActivity` to the instance of `high:Lane` via the property `high:toLane`. The `high:ChangeLaneActivity` is further divided into `high:ChangeLaneLeft` and `high:ChangeLaneRight`. The data properties `high:changeToRightNumOfLane` and `high:changeToLeftNumOfLane` specifies the required number of lane changes to the right and the left respectively.

The detection of lane change is modelled in the Lane Change Detection Box. The connection between `high:CurrentLocation` and `high:RouteSegment` via the object property `high:hasNotLongReachable` describes the longitudinal non-reachable relationship between the current location and some part of the route. The final target lane, based on the route, is modelled via the object property `high:hasTargetLane`.

Lane Change Detection

Since vehicles travel longitudinally in a single lane and move laterally when changing lanes, we classify the lane relations into two categories, namely lateral and longitudinal relations. With these two types of relations, we can represent a graph-based environmental model using lanes.

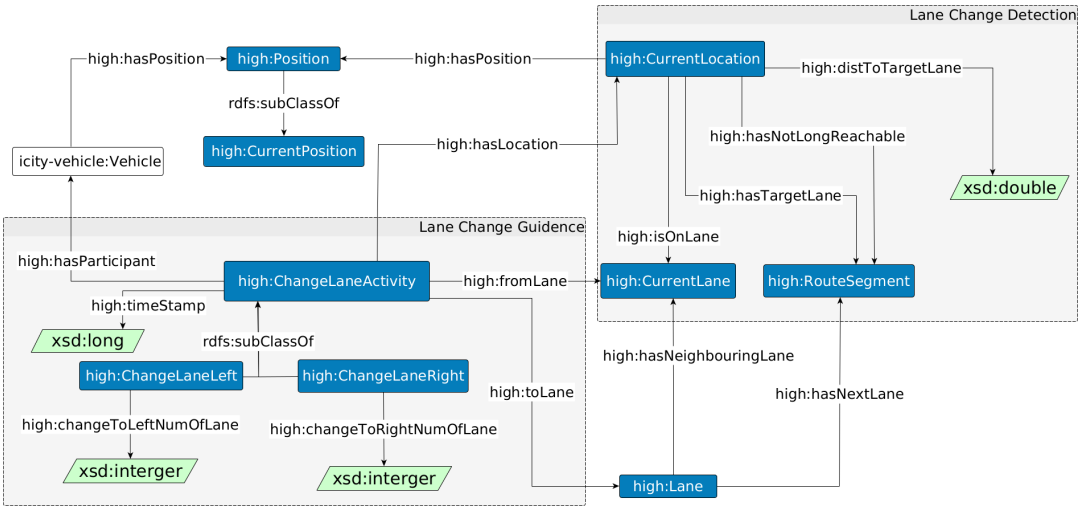
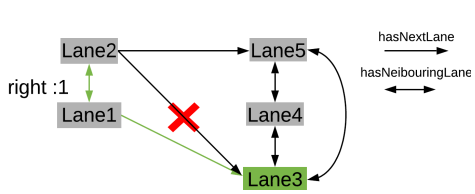
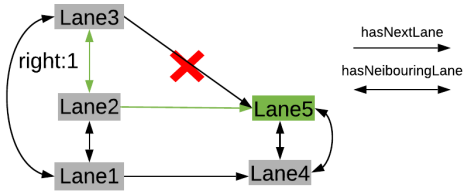


FIGURE 5.13: Lane change modelling based on the vehicle position and map knowledge

Figure 5.14 shows the graph representation of the scenarios mentioned above. In both cases, there is no relationship hasNextLane between the current lane and the target lane. A combination of relationship of hasNeighbouringLane ∘ hasNextLane, however, between those lanes exists. For example, in scenario A, there is no hasNextLane relationship between Lane2 and Lane3. But, a combination of relationships of hasNeighbouringLane ∘ hasNextLane between Lane2 and Lane3 exists. This pattern applies to scenario B as well. In fact, this pattern of spatial relationships can generally detect longitudinally unreachable lanes due to the transitivity of hasNeighbouringLane and hasNextLane .



(A) Exit lane Lane3 is part of the route.



(B) Lane5 is part of the route.

FIGURE 5.14: Lane network graph representation for the lane change scenarios illustrated in Figure 5.12.

Algorithm 1 summarises the steps used for identifying the necessity of a lane change situation. First, it identifies the part of the route l_i that is not longitudinally reachable from the current lane via the hasNextLane relation. Second, it checks if l_i is reachable via the property combination hasNextLane ∘ hasNeighbouringLane with intermediate lane l, and marks l as the lateral destination and l_i as the final target lane. Third, it calculates the distance from the vehicle's current position to the start point of the target lane as *dis*. Lastly, it compares *dis* with the foresight parameter fp. If *dis* is smaller than fp, then a change lane activity is generated and the current lane c is marked as from-lane of *cla*, and l as to-lane of *cla*. Listing 5.8 shows the corresponding rules.

```
#line 1-2
hasNotLongReachable[?c, ?rs]:-
Route[?route], CurrentLocation[?c], isOnLane[?c,?cl]
NOT EXISTS ?rs IN (hasRouteSegment[?route, ?rs], hasNextLane[?cl, ?rs]).
```

input : c: current location, cl: current lane, cp: current position,
fp: forsight parameter,
$R = l_1 \ldots l_n$: a route with a list of lanes, $n > 0$
output: cla: change lane activity

1 **if** $\exists\, l_i \in R$, *s.t.* $\nexists$ *hasNextLane(cl,*l_i*)* **then**
2 hasNotLongReachable(c,l_i);
3 **if** $\exists\, l$ *s.t. hasNeighbouringLane(cl,l)* $\wedge$ *hasNextLane(l,*l_i*)* $\wedge$ *hasNotLongReachable(cl,*l_i*)* **then**
4 hasTargetLane(c,l_i);
5 hasLateralDestLane(c,l);
6 dis = distance (cp, sp_{l_i}), where sp_{l_i} is the start point of l_i.
7 **if** $dis < fp$ **then**
8 cla=ChangeLaneActivity(cp);
9 fromLane(cla,c);
10 toLane(cla,l) ;

Algorithm 1: Lane change detection.

```
#line 3-5
hasTargetLane[?c, ?rs], hasLateralDestLane[?c,?nbl]:-
hasNotLongReachable[?c, ?rs], isOnLane[?c,?cl],
hasNeighbourLane[?cl,?nbl], hasNextLane[?nbl, ?rs].

#line 6
distToTargetLane[?c,?d]:-
CurrentLocation[?c],isOnLane[?c,?cl],
hasPosition[?c,?cp], hasTargetLane[?c, ?rs],
x[?cp,?cpx],y[?cp,?cpy], length[?cl,?l],
hasNeighbourLane[?cl,?nbl], hasNextLane[?nbl, ?rs],
AGGREGATE(hasNextLane[?nbl,?sl1], hasNextLane[?sl1,?rs],
length[?sl1, ?l]  ON ?nbl BIND SUM(?l) AS ?d2),
remainingDistance[?c,?d1], BIND((?d1+?d2) AS ?d) .

#line 7-10
ChangeLaneActivity[?cla],fromLane[?cla,?cl], toLane[?cla,?nbl]:-
CurrentLocation[?c],isOnLane[?c,?cl],hasPosition[?c,?cp],
hasTargetLane[?c, ?rs],hasLateralDestLane[?c,?nbl],
hasForesightParameter[?c,fp], foresightParameterValue[fp,?v],
distToTargetLane[?c,?d], FILTER(?d<?v), SKOLEM("cla",?cp, AS ?cla).
```

LISTING 5.8: Lane change detection rules

Lane Change Guidance

Once a lane change situation is detected, a vehicle needs to decide a sequence of lanes to follow along their route. In particular, the vehicle needs to know whether it should perform a left-lane-change or right-lane-change manoeuvre. Hence, we use the hasLeftLane and hasRightLane properties, which are fine-grained lateral relationships of, hasNeighbouringLane to derive the lateral lane-change manoeuvres. hasLeftLane is a sup-property of hasDirectLeftLane with transitivity. The same design applies to hasRightLane and hasDirectRightLane.

input : c: current location
output: d:direction, n:change lane number.
1 **if** $\exists$ *cla s.t. ChangeLaneActivity(cla)* $\wedge$ *hasLocation (cla,c)* **then**
2 **if** $\exists$ f,t *s.t. fromLane(cla,f)* $\wedge$ *toLane(cla,t)* **then**
3 **if** *hasDirectRightLane*(f, t) **then**
4 d= right; n = 1;
5 **if** *hasRightLane*(f, L_y) $\wedge$ *hasRightLane*(L_y, t) **then**
6 d= right; n = $|L_y|$ + 1;
7 **if** *hasDirectLeftLane*(f, t) **then**
8 d= left; n = 1;
9 **if** *hasLeftLane*(f, L_y) $\wedge$ *hasLeftLane*(L_y, t) **then**
10 d= left; n = $|L_y|$ + 1;

Algorithm 2: Lane change guidance.

Not only the vehicle needs to know whether it should change to the left or the right, but also the number of lanes it should change. This can be achieved by counting the hasDirectLeftLane or hasDirectRightLane relations between the current lane and the target lateral lane. For example, the expected lane change guidance for the scenario shown in Figure 5.15 is i) from lane2, change one lane right to Lane1; ii) continue driving to Lane3.

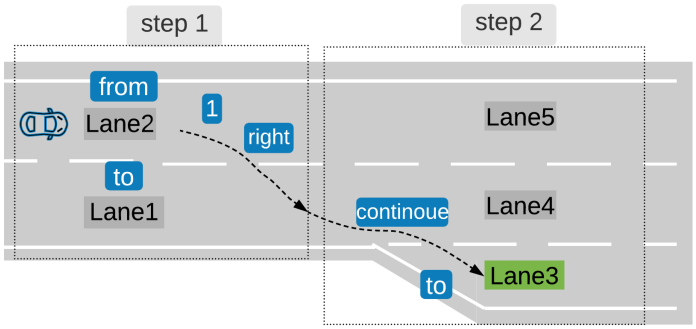


FIGURE 5.15: Right lane change guidance.

Algorithm 2 describes the procedures to generate the manoeuvre guidance. First, if there is a lane change activity *cla* in the current location *c*, then find the starting (from) lane f and the destination (to) lane t (line 1-2). Based on the properties of hasDirectRightLane and hasRightLane, calculate the number of lanes in between f and t (line 3-6). The calculation procedure for the number of lanes to the left side is achieved similarly (line 7-10). Listing 5.9 shows the corresponding rules for lines 1-6 in Algorithm 2.

```
#line 3-4
changeToRightNumOfLane[?cla, 1]:-
ChangeActivity[?cla],hasLocation[?cla,?c], CurrentLocation[?c],
fromLane[?cla,?f], toLane[?cla,?t], hasDirectRightLane[?f,?t].
```

```
#line 5-6
changeToRightNumOfLane[?cla, ?n]:-
ChangeActivity[?cla],hasLocation[?cla,?c], CurrentLocation[?c],
fromLane[?cla,?f], toLane[?cla.?t],
AGGREGATE(hasRightLane[?f, ?u],
hasRightLane[?u, ?t] ON ?f BIND COUNT(DISTINCT ?u) AS ?count),
BIND(?count + 1 AS ?n).
```

LISTING 5.9: Right lane change guidance rules

5.2.4 Continuous Map Processing

For an autonomous vehicle to navigate safely and ensure its passengers' comfort and safety, the navigation system needs to constantly request map data while the vehicle is progressing along a route. With the volume and velocity of map data streams, it becomes infeasible to store and process all incoming information. Hence, only some portion of knowledge is stored in memory for a limited area.

The choice of the stored knowledge is primarily dependent on continuous spatial queries. To achieve this goal, we exploit *a spatial sliding window* over the incrementally updated road environmental knowledge to check if the system needs to *prefetch* map data or delete expired road entities. In addition, prefetching map data for later usage is an important aspect to improve performance for map-based AD functions. The basic idea behind prefetching is to predict which map tiles the vehicle may wish to use "next", or "soon" and request only necessary ones from the map database ahead of time.

Figure 5.16 illustrates an example of the mechanism described above with three snapshots. In each snapshot, the vehicle's external and internal worlds are presented. In snapshot 1, the vehicle initializes with some low-level map data, which results in a high-level road view. As soon as the car moves, it triggers a prefetching query with a spatial window whose forward parameter is set to 5 km. In snapshot 2, the system pre-loads a new map tile. Consequently, the high-level road view is extended with the new spatial knowledge. In snapshot 3, while the system incrementally updates its road environmental knowledge, it also continuously checks if any road parts are "out of window" based on the backward parameter (e.g., 3 km) of the spatial window. The road parts which are not "inside" the spatial window is deleted. We describe the spatial sliding window and the prefetching strategy in the following part of this section in detail.

Spatial Sliding Window

In our approach, we employ a spatial window policy, where the spatial window defines an area according to some spatial properties (e.g., distance) with respect to a reference object. We use spatial rules, namely topological and distance rules, to performance spatial reasoning. The spatial window is relevant to the vehicle's current location with the forward and backward parameters (see Figure 5.17). Figure 5.18 gives two snapshots of the vehicle environment at location `L1` and `L2`. `rp1` — `rp5` represent road parts. At location `L1`, continuous spatial queries are evaluated over `rp1` — `rp3` as they are `inside` the spatial window. At location `L2`, however, `r1` — `r2` is out of the spatial window, which means they are expired. `rp3` is a still valid object, and `rp4` is the new object `inside` the spatial window. Thus, continuous spatial queries would be evaluated over `rp3` and `rp4`.

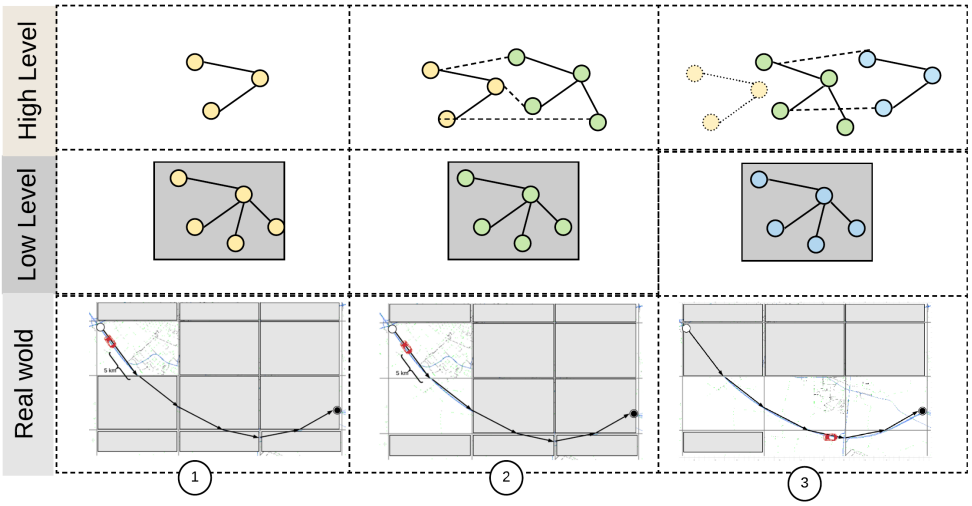


FIGURE 5.16: Snapshots of a vehicle processing along a route and prefetching map tiles.

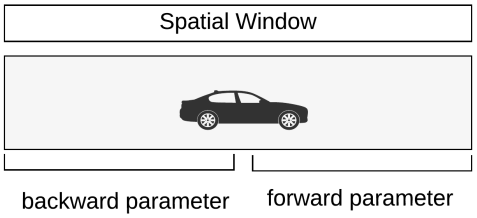


FIGURE 5.17: An example of a route path on map tiles.

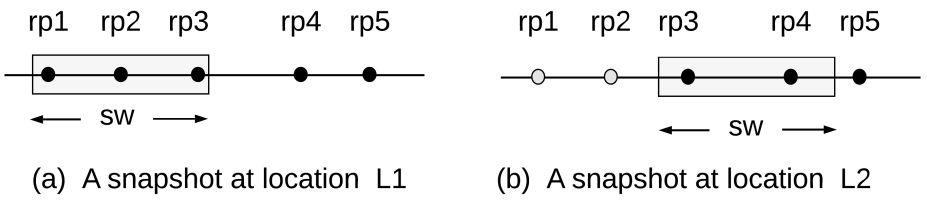


FIGURE 5.18: Spatial sliding window snapshots.

Ontology Module

Figure 5.19 shows the ontology module of the spatial sliding window from HLM. It is modelled as a class `high:SpatialWindow` linked to the class `high:SpatialWindow-Parameter` via object property `high:hasSWParameter`. The class `high:SpatialWindow-Parameter` is described by two data properties `high:forwardParameterValue` and `high:backwardParameterValue`. The design of modelling the parameter as a class allows us to specify different types of sliding window parameters. For example, the highway sliding window could have a bigger region than the city sliding window because the density of map objects in a highway is less than the density of map objects in a city.

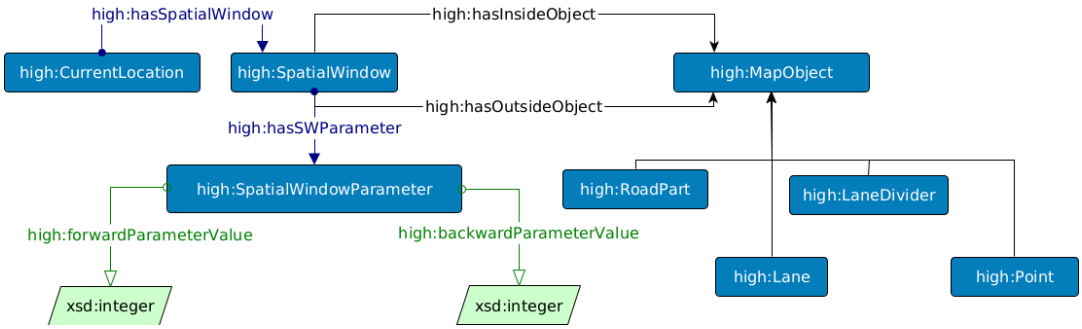


FIGURE 5.19: Ontology module of the spatial window.

We define the `high:hasOutsideObject` object property to identify map objects that are invalid for the spatial window by comparing the distance between the vehicle's position to the previous map objects. If the distance is bigger than the backward parameter, then the map object is considered as "outside" map object. Otherwise, it is considered as an "inside" map object described by `high:hasInsideObject` object property. The `high:forwardParameterValue` is used for prefetching map tiles to ensure sufficient range of map knowledge for the vehicle planning activities, which is explained in the following part of this section. Listing 5.10 shows an example inferring `high:hasInsideObject` and `high:hasOutsideObject` relations between `high:RoadPart` and `high:SpatialWindow`, respectively.

```
hasOutsideObject[?s,?r] :-
CurrentRoadPart[?cr,],hasPrevious[?cr,?r],
Roadpart[?r], distanceToVehicle[?r,?d],
currentLocation[?c], hasSpatialWindow[?c,?s],
SpatialWindow[?s],hasSWparameter[?s,?p],
backwardParameterValue[?p, ?b], FILTER(?d>?b).

hasInsideObject[?s,?r] :-
SpatialWindow[?s],  CurrentRoadPart[?cr,], hasPrevious[?cr,?r],
NOT EXISTS ?r IN (Roadpart[?r], hasOutsideObject[?s,?r]) .
```

LISTING 5.10: Rules defining map objects that are inside the spatial window and outside the spatial window

Prefetching Map Tiles

Prefetching map data includes receiving information on a route, including an origin, a destination, and a set of paths connecting the origin and the designation [106]. It is done by selecting minimum map tiles along a route that satisfy the forward parameter of the spatial sliding window and a threshold value.

Figure 5.20 shows a concrete example of a designed route based on prefetching. The route is constituted by road parts rp1 — rp_n. Each road part is described by its length and tile ID. For example, rp1 has length 1963 m, and it is located in a tile 5062. At this state, only map tile 5062 is loaded for the vehicle, and the remaining distance of the vehicle in $rp1$ is 1937 m. To provide adequate road knowledge ahead, the vehicle needs to prefetch a minimum of future tiles based on the route. In this scenario, prefetching tiles 5059 and 5057 associated with rp2 and rp3 respectively can satisfy the condition as the difference between the remaining distance and forward parameter (e.g.,5 km) is within a threshold (100 m). Other tiles associated with the remaining road parts of the route are not needed at the current location of the vehicle, i.e., tile 5033 for rp_{n-1} and tile 5032 for rp_n.

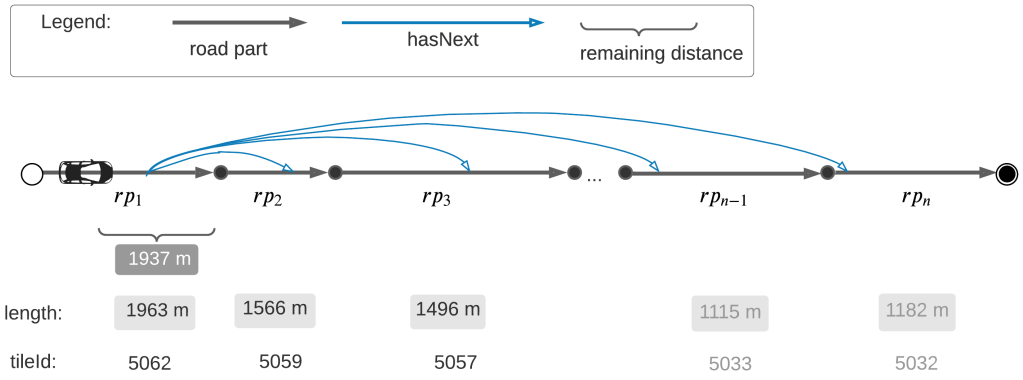


FIGURE 5.20: An example of prefetching map tiles based on a route.

The summation of the remaining distance of the vehicle in rp1 and the length of rp2 and rp3 is 4999 m, and the difference between the summation and the forward parameter is 1 m, which is within the threshold 100 m. Hence, prefetching the tiles associated with rp2 and rp3 is adequate. For rp_{n-1}, the summation is 6114 m, and the difference between the summation and the forward parameter is 114 m, which is bigger than the threshold of 100 m. Hence, rp_{n-1} is not prefetched by the vehicle.

Algorithm 3 illustrates the process of retrieving the needed tile IDs. It starts by, for each i, where $c < i \leq n$, calculating the summation dis_s of the lengths of the current lane rp_c's successors rp_k, where $(c+1) \leq k \leq i$ (Line 1-2). Then it computes the total distance dis by adding the remaining distance d_{re} of the vehicle in rp_c and to dis_s (Line 3). At last, by comparing dis and the threshold t, it returns tile IDs associated with road parts rp_k satisfying the condition (Line 4-5). Listing 5.11 shows the corresponding SPARQL query. Line 2 is modelled using a SPARQL nested query, performing aggregation using `GROUP BY` and `SUM`. Line 4 is modelled using `FILTER` with `ABS` function.

```
SELECT ?tileId
WHERE {
  ?rp_  a :CurrentRoadPart.
  ?rp_ :hasNext ?rp1_2.
  ?rp1_2 :hasNext ?rp2_2.
  #line 2
  { SELECT ?rp2_2 (SUM(?length) AS ?inBwtbLength)
    WHERE {
      ?rp a :CurrentRoadPart.
      ?rp :hasNext ?rp1.
```

input : R={rp_n:$n > 1$}: a route with road parts,
$rp_c \in R$: current road part, d_{re}: remaining distance in rp_c,
f: forward parameter, t: threshold
output: T: a set of tile IDs
1 **for** $c < i \leq n$ **do**
2 $dis_s = \sum_{k=c+1}^{i} length_{rp_k}$;
3 $dis = d_{re} + dis_s$;
4 **if** $|dis - f| \leq t$ **then**
5 T = tileID(rp_k) ;

Algorithm 3: Prefetching the tile IDs.

```
      ?rp1 :hasNext ?rp2_2.
      ?rp1 :length ?length.
    }
  GROUP BY ?rp2_2
  }
  ?curreLoct :remDisOfRoadPart ?remainingDis.
   #line 3
  BIND((?remainingDis + ?inBwtbLength) AS ?distance)
  :spatialWindow :forwardParameterValue ?val.
  #line 4
  FILTER(ABS(?distance-?val) < 100)
  #line 5
  ?rp1_2 :isLocatedIn ?tileId.
}
```

LISTING 5.11: A SPARQL query for prefetching map data tiles.

5.3 Implementation

With the objective of providing efficient knowledge processing and spatial reasoning, we realised the conceptual architecture described in Section 5.2 with decoupled two-level datastore structure using parallel processes shown in Figure 5.21.

High Level The high-level datastore is dedicated to continuously providing location-based decision-making functions based on the vehicle's positions via reasoning using Datalog rules and SPARQL queries. The workflow operated on this level is shown in Figure 5.22. It starts by receiving a route and a vehicle position. Immediately after, the current lane is identified, as well as the needed change for a lane change situation. Based on the defined spatial sliding window, process 1 determines which tiles are required to be prefetched and executes parallel knowledge abstraction processes for each tile in background threads that operate simultaneously. Process 2 determines which high-level road knowledge needs to be deleted. The future tile IDs are retrieved via SPARQL `SELECT` and data deletion is achieved via SPARQL `DELETE`.

Low Level Based on the number of returned future tile IDs, the application triggers parallel low-to-high map knowledge process tasks at the low level. Each task processes one map tile, and it is executed on one datastore. The tasks are running in the background to prevent interfering with the functions executed on the high-level datastore. The workflow for each task operated in this level is shown in Figure 5.23. Once the application receives a tile ID, it imports the corresponding low-level map

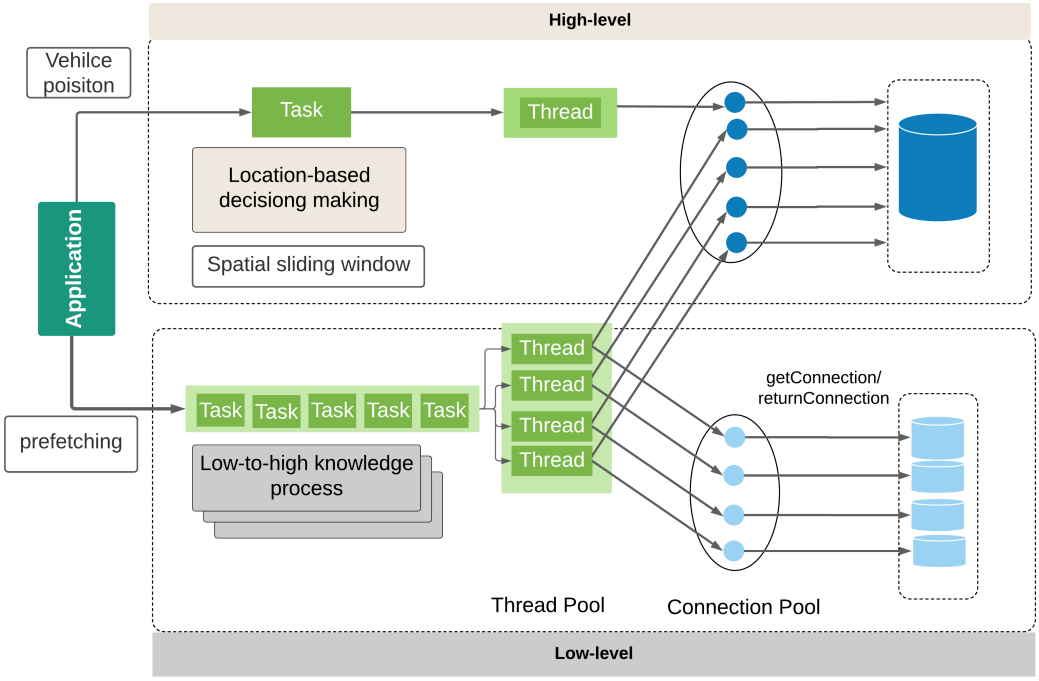


FIGURE 5.21: Implementation of the knowledge-spatial architecture.

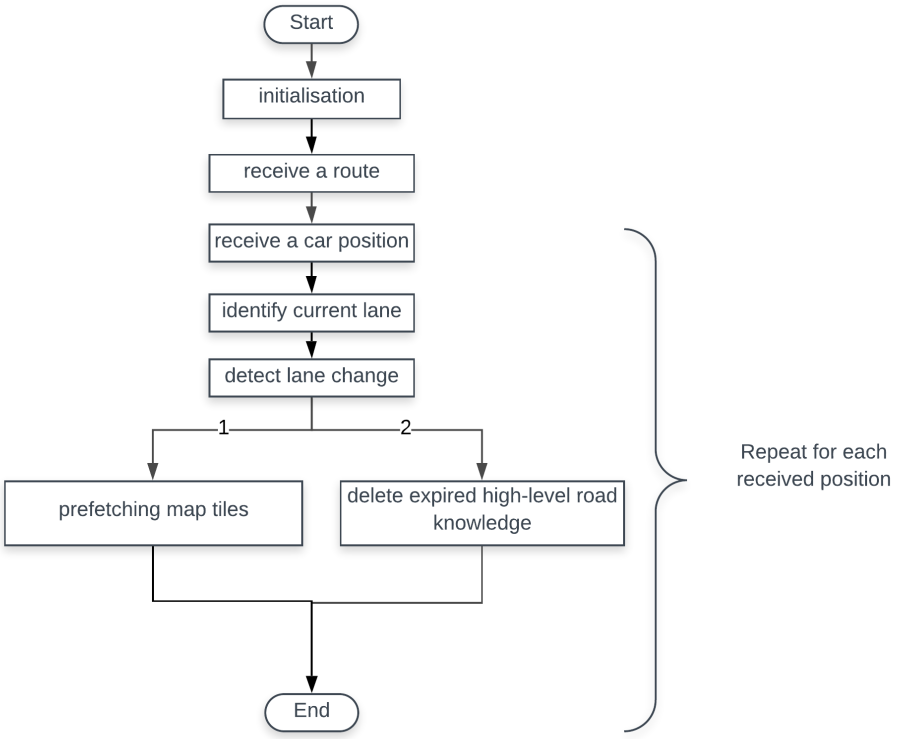


FIGURE 5.22: The method of location-based decision making with update road environment along a route with two parallel processes.

ontology to the datastore. The low-to-high knowledge abstraction starts immediately with defined processing rules. Afterwards, the produced ABox is inserted into the high-level datastore, incrementally updating the high-level view. At last, the datastore is cleared for future use.

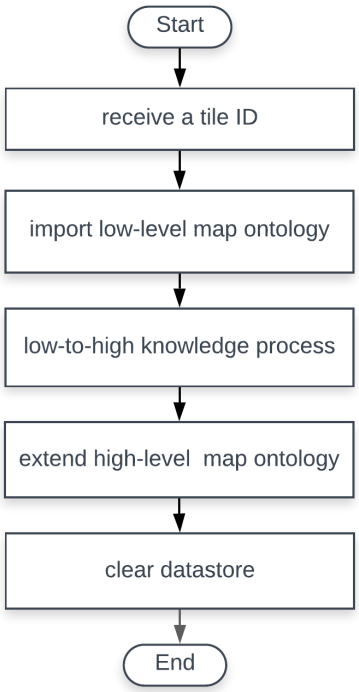


FIGURE 5.23: The workflow of each low-to-high knowledge process.

The low-to-high knowledge process rules are encoded in SPARQL to ensure sufficient computational resources for Datalog reasoning in the high-level datastore. This is because the low-to-high knowledge process is computation-intensive due to the amount of data and rules involved. If the rules are encoded in Datalog, the low-level datastores will compete with the high-level store for computation resources and slow down its reasoning. The correspondence between the Datalog rules and SPARQL queries are presented in Appendix A.3. Parallel processing is achieved with multi-threads and datastore connection pooling. To ensure concurrency, one thread is executed upon exactly one distinct connection from the configured connection pool.

5.4 Evaluation

With the objective of evaluating the efficient reasoning ability and explainability of the proposed approach, we conduct the evaluation based on a vehicle's trace along the A92 highway in Germany using the developed *SmartMapApp* Java application. Figure 5.24 shows the simulation trace of a vehicle along a given route across ten tiles. The data we used are NDS maps converted into RDF triples based on the developed LNDS TBox. Each tile is also referred to as one LNDS ABox. All evaluations were performed on a 64-bit Ubuntu virtual machine with 4 Intel(R) Core(TM) i7-6820HQ CPUs @ 2.70GHz with 15 GB memory. We recorded the computation time after doing a warm-up run by executing the tasks three times sequentially.

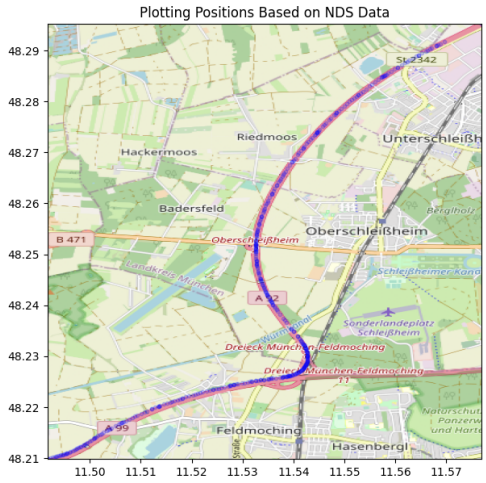


FIGURE 5.24: The vehicle trace ©OpenStreet Map.

5.4.1 Low-to-high Knowledge Process

We report the processing time for each tile using one datastore. The result is shown in Figure 5.25. The figure shows, for each tile, the number of low-level triples (green bar), the resulting number of high-level triples (blue bar), the time for transferring triples from the low-level to the high-level ontology (low2high process time, diamonds), and the time for extending the high-level road environmental view (dot). The computation time for both phases increases regarding the data size. The average processing time for each tile is 1.4s. In addition, as the number of tiles increases, the average processing time increases as well, which delays the update of the HLM ABox and interferes with high-level decision making.

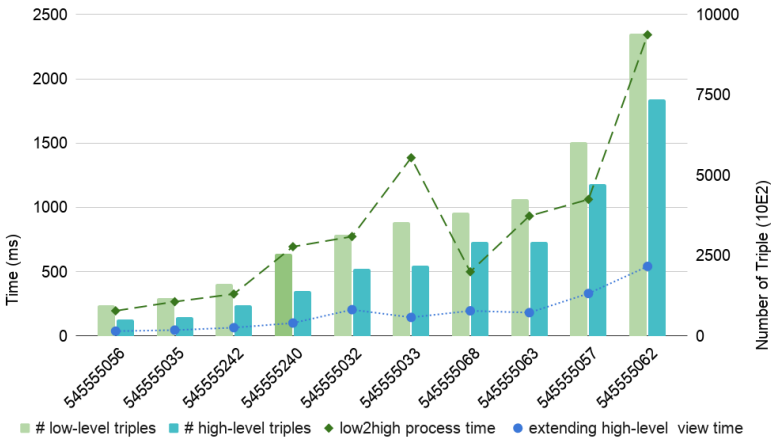


FIGURE 5.25: The low-to-high processing time

With the objective to investigate the performance relationship between the speedup and the number of datastores used in parallel, we did a preliminary test using eight tiles for several datastores ranging from 1 to 8. The result is shown in Figure 5.26. The maximum speedup is achieved when using four datastores. From the number

of 5 to 8 datastores, the speedup is decreasing. This indicates the overhead of parallel datastore execution. A more thorough evaluation is required to find out the correlation between the number of datastores and tiles.

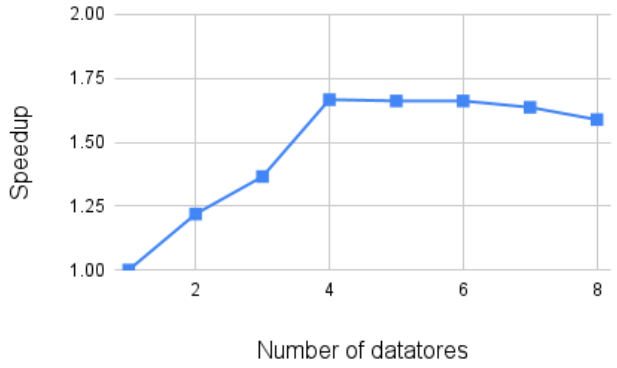


FIGURE 5.26: The speedup measurement of using 1 to 8 datastores simultaneously.

Figure 5.27 shows the low-to-high knowledge processing of 4 tiles using one datastore and four datastores. It took 6 s to finish the task for one datastore, and only then the HLM ABox was updated. However, with four datastores running parallel for each tile in the background, the HLM ABox was updated immediately after each low-to-high process was finished. The overall processing time is approximately equal to the time needed to process the biggest tile (259,842 triples) among the four tiles, about 3 s. This shows the ability of our approach to reducing the knowledge processing time for several tiles, which benefits the map initialisation phase and the prefetching phase.

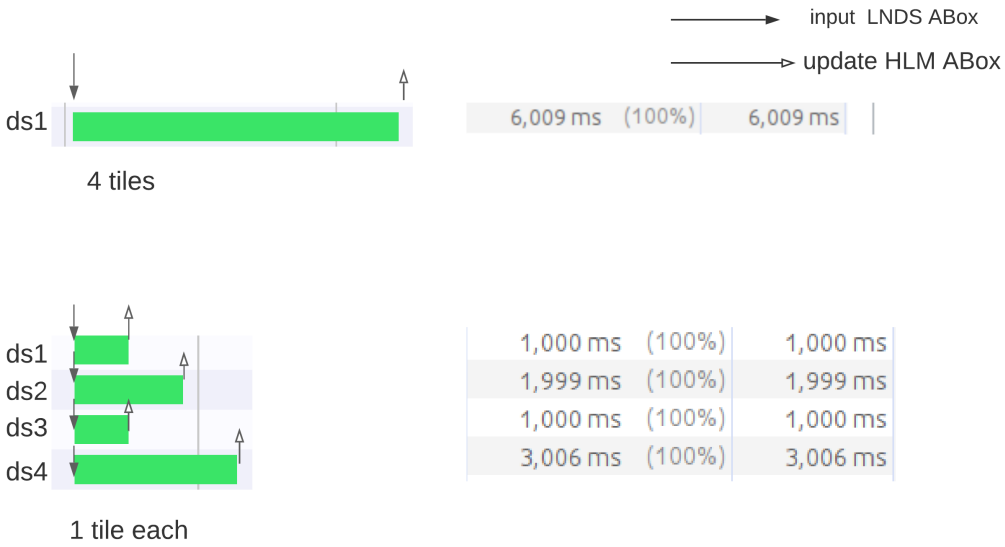


FIGURE 5.27: The comparison between Low-to-high knowledge processing for four tiles running in 1 datastore and four datastores in parallel.

5.4.2 Vehicle Location Awareness

We first evaluate the ability of the ontology module and rules described in Section 5.2.2 to provide location awareness based on the vehicle position. Then we present the performance of the current lane identification. Listing 5.12 shows the SPARQL query used for retrieving the context information of the vehicle position. Based on the current position, the SPARQL queries the types of the lane and road part and length of the lane. Figure 5.28 shows an example of the obtained query result based on the position of the vehicle. The result shows the localised lane, the current lane, which is classified as the left-most lane and regular driving lane. Based on the relationship between lane and road part, the road part as current road part and its type as motorway were inferred.

```
SELECT ?laneType ?length  ?roadType
WHERE {
    ?p a high:CurrentPosition.
    ?p high:isOnLane ?lane.
    ?lane a ?laneType; high:length ?length.

    ?rp a high:RoadPart; high:hasLane ?lane; a ?roadType.
}
```

LISTING 5.12: SPARQL query for location awareness

FIGURE 5.28: Vehicle location awareness at position 48.28721361, 11.56128117 ©Google Map.

We evaluated the performance along a trace and highlighted the first ten positions. The experiment is done for four initialisation cases with the number of tiles ranging from 1 to 4 to investigate the impact of data size on the initialisation and the current lane identification of the first vehicle position. Table 5.1 provides the computation time required for different numbers of tiles. The more tiles are used for initialisation, the more triples are transferred to the HLM ABox. The computation time for identifying the current lane for the first vehicle position also increases. Because the distances between every geometric point and the vehicle position have to be computed to find the minimum one, such that the associated lane is identified. However, the number of tiles used for initialisation was not affected by the average computation time needed for the other positions. This is the result of the optimisation of limiting the search scope for the remaining positions to the neighbouring

lanes of the previously identified lane. In addition, the number of tiles had no effect on the initialisation time, which profited significantly from parallel processing.

TABLE 5.1: Performance for initialisation with 1-4 tiles.

	1 tile	2 tiles	3 tiles	4 tiles
#Triples	137,451	154,706	244,296	274,962
Initialisation time	3074ms	3119ms	3133ms	3398ms
1st Position	621ms	738ms	860ms	970ms
Other positions avg	17ms	11ms	15ms	17ms

5.4.3 Lane Change Evaluation

We evaluate the lane change decision in a real-world scenario. The rules described in Section 5.2.3 are applied to detect the lane change event while the vehicle is progressing along a road. Benefiting from the explainability of rules, the explanation of the lane decision making is provided. Moreover, the manoeuvre steps are generated using SPARQL queries to guide the vehicle in performing a lane change. Figure 5.29 shows a lane change scenario where the vehicle needs to change lanes to exit the highway. The expected manoeuvre is to change one lane to the right, and continue driving to the exit lane. Figure 5.30 shows the RDF graph of the state where the lane change is detected on the road, including the static road knowledge (e.g., lane topological relationships) and the dynamic vehicle knowledge (e.g., current position/lane).

FIGURE 5.29: Visualisation of lane change evaluation ©Google Map. Green lines with the arrow are the lane change manoeuvre sequence.

The inferred result (green arrow) based on the state of road and vehicle knowledge and the lane change decision rules are shown in Figure 5.31. Based on the given route, the target lane and the distance to it are inferred. Because the distance to the target lane is 75.33 m, which is less than the lane change foresight parameter (200 m), this triggered a new instance of `high:ChangeActivity`. For the lane change guidance, the value of `high:changeToRightNumOfLane` is inferred as 1. Because changing one lane to the right enables the vehicle to reach the target lane. Figure 5.32 shows the simulation result for the above described lane change scenario. The manoeuvre steps are retrieved using a SPARQL query (see Listing 5.13) and the explanation is generated with the detailed decision making steps.

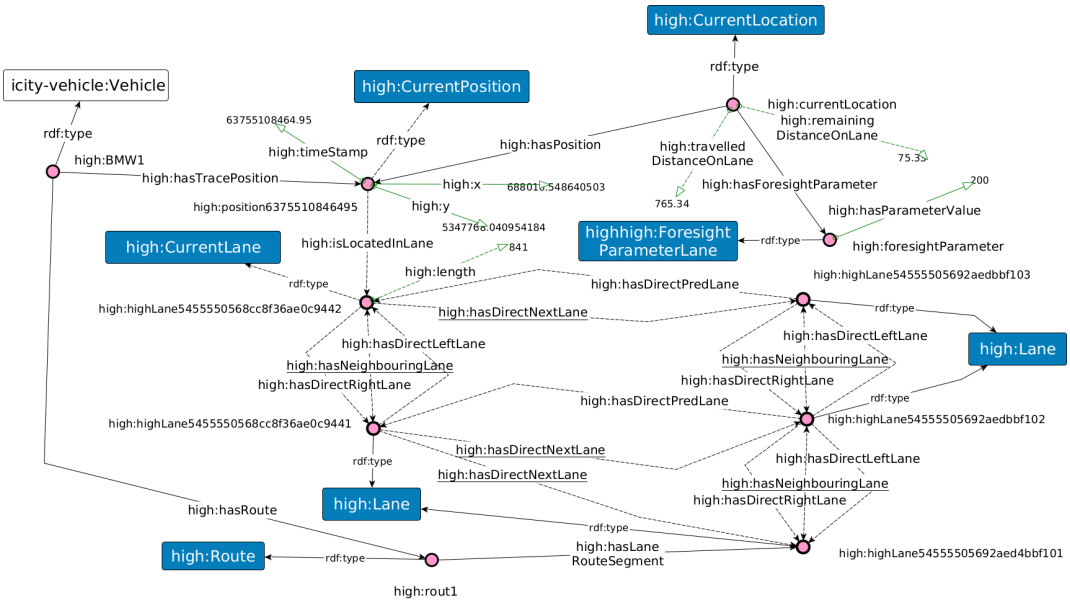


FIGURE 5.30: RDF graph of the snapshot of the vehicle's road knowledge and when the lane change is needed. Solid lines with arrows represent the fact, dashed lines with arrow are inferred knowledge using rules.

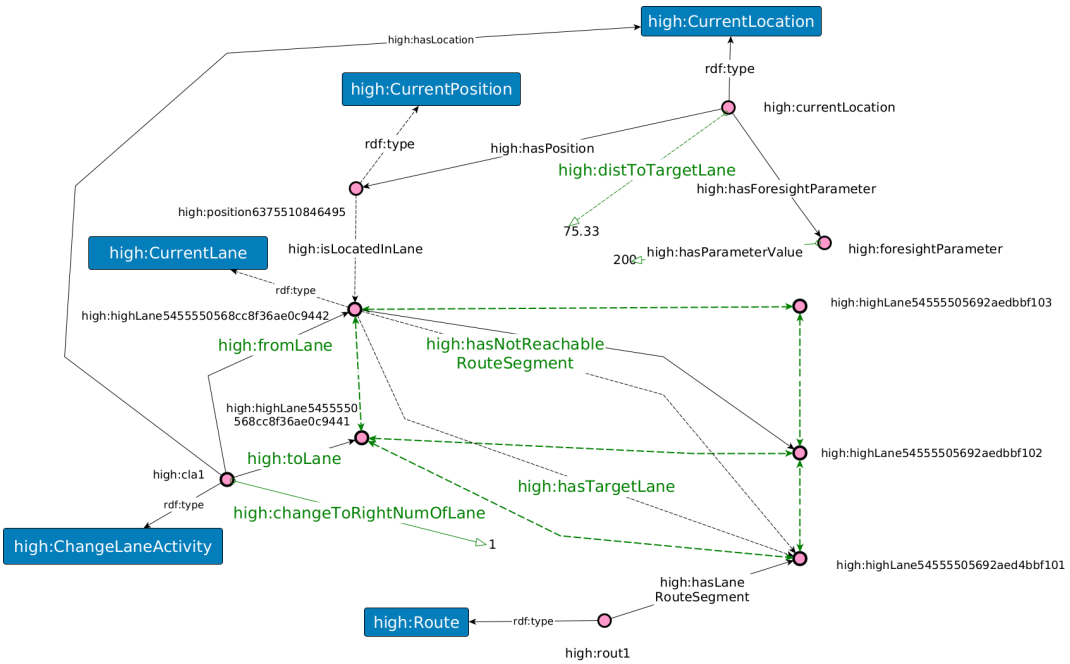


FIGURE 5.31: The inferred result (green property) after applying lane change decision rules in the RDF knowledge shown in Figure 5.30. The vertical green dashed lines represent the `hasNeibouringLane` relation, the horizontal ones represent the `hasDirectNext` relation. We omitted some classes and properties in this figure for simplicity.

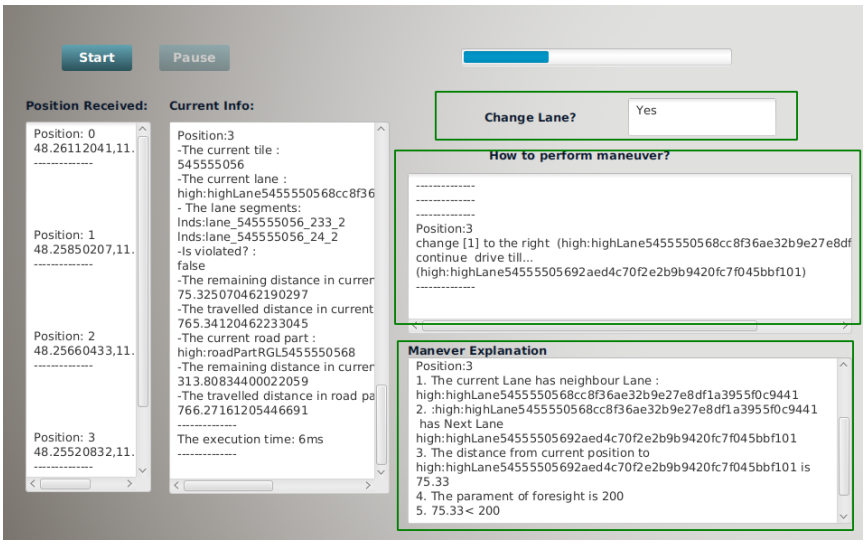


FIGURE 5.32: The developed *SmartMapApp* simulation detects the lane change and provides manoeuvre steps explanation for decision making.

```
SELECT ?tlane ?n ?rl ?tlane
WHERE {
    ?cp a high:CurrentPosition; high:isNeedChangeLane true;
    high:distToTargetLane ?dis.
    ?cl a high:CurrentLane;
        high:changeToRightNumOfLane ?n;
        high:hasRighLane ?rl;
        high:hasTargetLane ?tlane.

    ?rl  high:hasNextLane ?tlane.
}
```

LISTING 5.13: Lane change identification rules

5.4.4 Continuous Map Processing Evaluation

We evaluated continuous map processing using the route with the objective of investigating the reasoning performance in the high-level ontology. Figure 5.33 shows the route with segments located in different tiles. Each route segment has the information about its length, associated tile ID and its direct successor. Table 5.2 shows the triple representation of this route, where the tile ID is described by the data property `high:partitionId`.

We report the result with an excerpt of a simulation result shown in Figure 5.34. The left vertical axis shows execution time. The execution time (line) is under 50 ms on average. The deletion time (area) is almost as low as the execution time. The right vertical axis shows the time for deletion (area) and the background pre-loading process (orange bar). At positions 11, 18 and 29, the prefetching process is triggered. The number of triples loaded at position 18 was the biggest (176,191) and caused the longest processing time (1653 ms). This resulted in a peak of the execution time (104 ms). The number of triplets loaded at position 11 is 72,109, which is less than half of the data loaded at position 18. This is the reason the pre-loading time (479 ms) is also less than half of what is required at position 18, which also had less impact on the execution time. The same reason applies to position 29. Figure 5.35 shows the

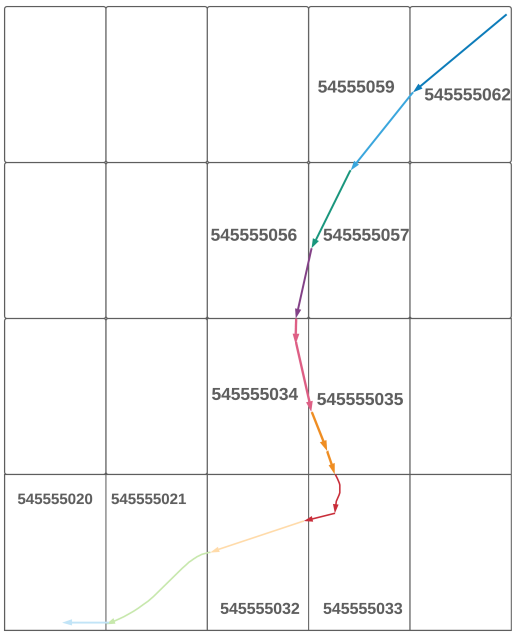


FIGURE 5.33: The vehicle route is constituted by a sequence of route segments marked in arrow.

TABLE 5.2: Triple of the route

Subject	Property	Object
high:route	rdf:type	high:Route
high:route	high:hasRoadPartRouteSegment	high:rp5455506216
high:route	high:hasRoadPartRouteSegment	high:rp5455505904
high:rp5455506216	high:hasDirectRouteSegment	high:rp5455505904
...	...	...
high:route	high:hasRoadPartRouteSegment	high:rp5455502109
high:rp5455506216	rdf:type	high:RoadPart
high:rp5455506216	high:length	1963.62
high:rp5455506216	high:partitionId	545555062
high:rp5455505904	rdf:type	high:RoadPart
high:rp5455505904	high:length	1566.32
high:rp5455505904	high:partitionId	545555059
...	...	...
high:rp5455502109	rdf:type	high:RoadPart
high:rp5455502109	high:length	2010.98
high:rp5455502109	high:partitionId	545555021

size change of the HLM ABox along with the trace corresponding to data loading and deletion events.

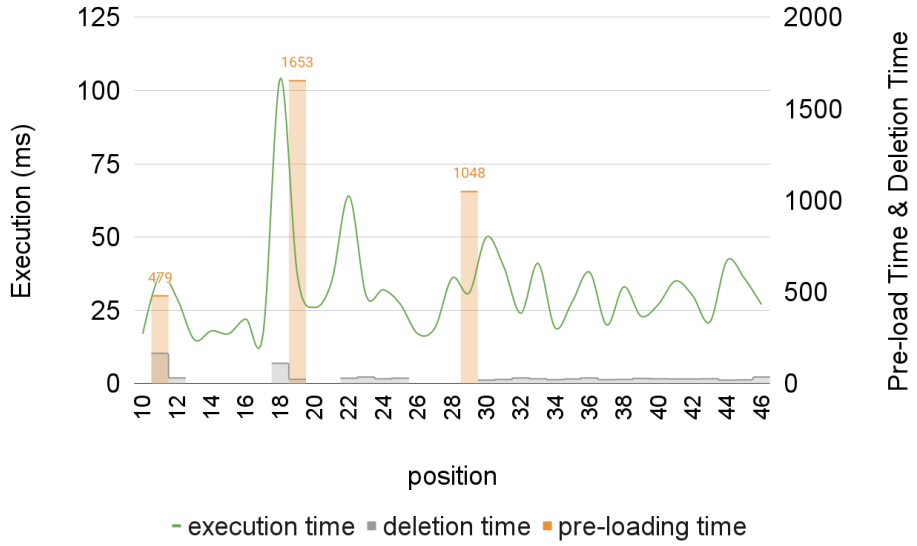


FIGURE 5.34: Spatial reasoning process time.

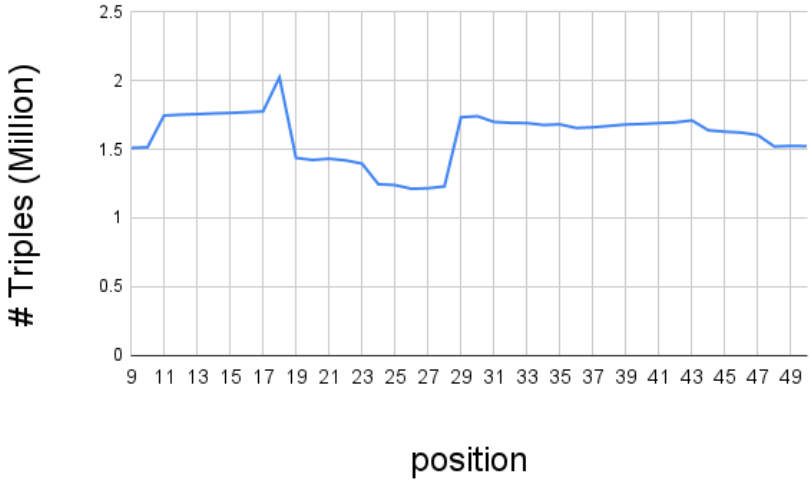


FIGURE 5.35: The size change of the HLM ABox along the trace.

5.5 Solution Deployment Discussion

As for the deployment of the proposed architecture, we discuss two alternatives. The first option is to deploy the entire two-level datastores on the client-side (vehicle) (see Figure 5.36). We refer to this option as *client deployment*. The second option is to partition (federated) the two-level datastores, and deploy the high-level datastore on the client-side, and deploy the group of low-level datastores on the server (cloud) side (see Figure 5.37). We refer to this option as *client-server deployment*. The main difference between these two approaches arises from the deployment of the *Low-to-high processor* component and the related technical realizations.

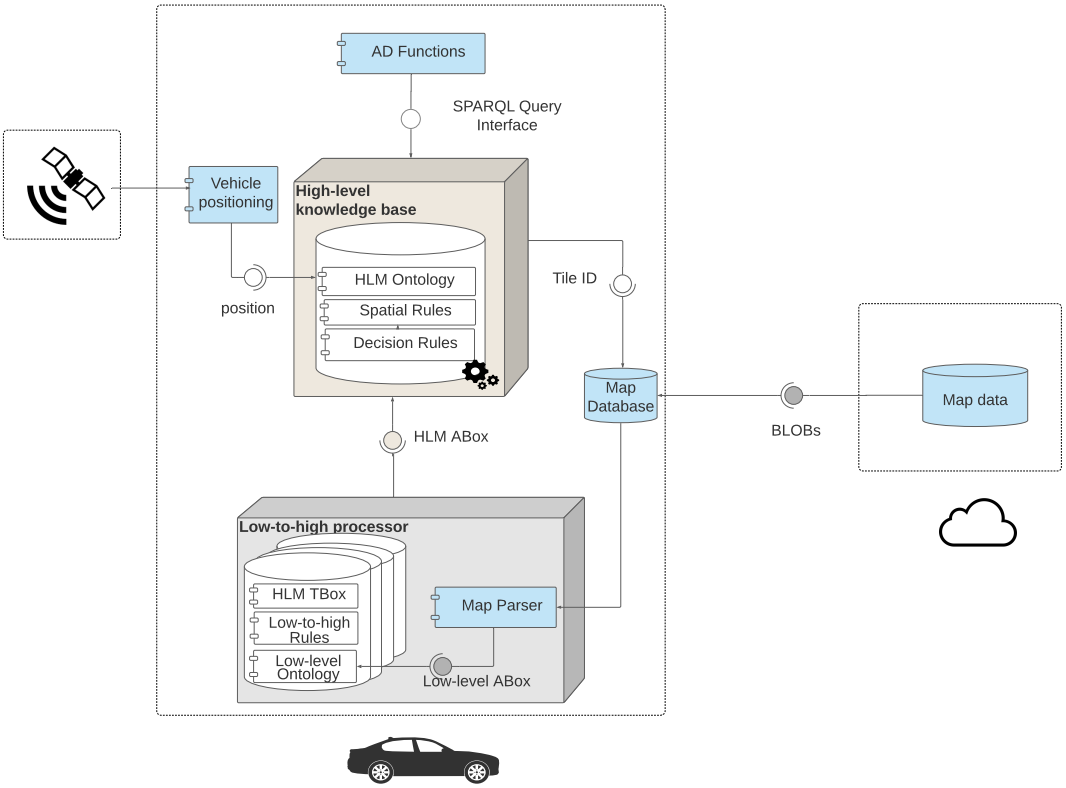


FIGURE 5.36: The client deployment.

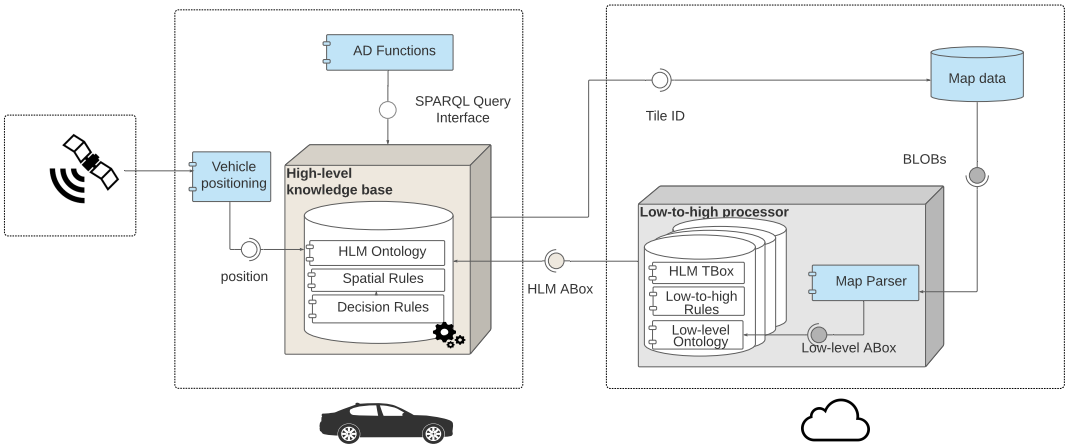


FIGURE 5.37: The client-server deployment.

The Low-to-high processor component is responsible for producing the *HLM ABox* which is the input of the *High-level knowledge base* component. If the Low-to-high processor component is deployed on the client-side, then the data transmitted via cellular network between the client and server is tile BLOBs. If the Low-to-high processor component is deployed on the server-side, the transmitted data between the client and server is the *HLM ABox* in RDF.

5.5.1 Client Deployment

We provide the memory consumption for client-side deployment in Table 5.3. The overall estimated memory consumption of the required components written in C++ is about 250 MB. We use one datastore for the High-level knowledge base component and four datastores for the Low-to-high processor component. The time for generating LNDS Abox using Python parser is 1.75 s on average. The time for producing HLM Abox is 1468 ms on average. The overall time used to produce one tile of the HLM ABox in one datastore is 3,218 ms on average (see Table 5.4). Since the Low-to-high processor is running in the background, the reasoning process in the High-level knowledge base component is not affected.

TABLE 5.3: Memory for client-side deployment.

Parameter	Value	Description
memory	250 MB	C++ application
level of datastores	2	low and high
#datastore	5	1 high-level datastore, 4 low-level datastores

TABLE 5.4: Time for HLM ABox preparation of one tile on average.

Parameter	Value	Description
one tile file size	145 KB	BLOB
preparation of HLM ABox	3,218 ms	LNDS ABox (1750 ms) + HLM ABox (1468 ms)

5.5.2 Client-Server Deployment

In contrast to client deployment, client-server deployment deploys the Low-to-high processor in the server side. This results in much lower memory consumption in the client side (see Table 5.5). This is not a surprise, as it is the result of federated deployment. The memory consumption only comes from the High-level knowledge base using one datastore. According to the memory estimation of the client approach, we estimate that one datastore takes about 50 MB by dividing 250 MB to 5 (#datastore). Since the Low-to-high processor is deployed on the server, the output of this component, an HLM ABox, will be sent to the High-level knowledge base component in the client side. We provide the time related for preparing the HLM ABox in the client side in Table 5.6. The size of one HLM ABox after compression is 166 KB on average. Assuming a data transmission over 4G LTE network is 100 megabits per second, the transmission time is 16 ms. Together with decompression time, the overall estimated time for one tile HLM ABox preparation is about 334 ms on average.

TABLE 5.5: Memory estimation of the client side for the client-server deployment.

Parameter	Value	Description
memory	∼50 MB	application
level of datastores	1	high
#datastore	1	1 high-level datastore

TABLE 5.6: Average time for receiving an HLM ABox of one tile in the client side for the client-server deployment.

Parameter	Value	Description
sending one tile file size	166 KB	compressed
preparation of HLM Abox	334 ms	Receiving 16 ms +decompressing (318 ms)

5.6 Conclusion

In this chapter, we investigated the ontology-based approach for dynamic map processing. We addressed the problem of efficient knowledge processing and spatial reasoning by applying the proposed knowledge-spatial architecture. The knowledge dimension illustrates a knowledge abstraction process from the format-specific and detailed low-level ontologies to the generic high-level ontology. The spatial dimension describes the continuous spatial reasoning process regarding the updated vehicle position and dynamic road environmental knowledge. We used the spatial sliding window and incremental reasoning to perform efficient knowledge update and complex inference.

We reviewed the three aspects of vehicle's situation awareness based on road environmental knowledge, namely location awareness, lane change and continuous map processing. For every aspect, an ontological module and a set of rules are defined to provide context information and facilitate the decision-making process. To realise the knowledge-spatial architecture, we developed a prototype with decoupled two-level datastores in order to provide fast rule evaluation and query answering in the high-level datastore while several low-level datastores running in the background to provide adequate high-level knowledge.

We empirically evaluated the developed prototype using an NDS map to provide a high-level map view. The result shows that the ontology-based approach with rules can efficiently process knowledge and perform spatial reasoning. The explainability and flexibility of using rules are some of these advantages we observe in the proposed approach.

We proposed two deployment choices for the designed solution, namely, a client and a client-server deployment. For the client deployment, the low-to-high processor is deployed on the client. For the client-server approach, however, the low-to-high processor is deployed on the server. The estimated memory consumption and the time for data transmission are presented. Both approaches are feasible for in-car usage.

5.6.1 Remark

Both streaming reasoning and incremental reasoning are designed to derive knowledge from frequently changing data. However, stream reasoning operates over time-based data, while maps contain spatial data. That is to say, and streaming reasoning

techniques cannot be directly applied to processing maps without major modification. Additionally, streaming reasoning is a very young topic of research, and currently supports only primitive reasoning with limited stability [133]. Therefore, incremental reasoning and spatial slide window were applied in this thesis and proved to be applicable for processing dynamic maps.

However, we see potential usage of combining streaming reasoning for time-based sensor data with incremental reasoning over maps containing spatial data. For example, the knowledge of the speed can be derived from the vehicle speed sensor data. With the road environmental knowledge derived from maps, more tactical vehicle driving makeovers can be provided.

Additionally, we observe that ontologies and rules help to set the focus more on the analyses and design of autonomous functions on the knowledge level rather than on map data understanding and implementing various map processing components for similar maps. An example that illustrates the advantage of using ontologies and rules are provided in Appendix A.2.

Chapter 6

Map Quality Assurance

High definition (HD) maps are an essential component in autonomous vehicles. Making HD maps is not only a challenge because of the sheer amount of data that needs to be captured and stored, but also because of the intrinsic accuracy requirements. According to a study carried out by the European Commission (EC) on the integrity and reliability of HD maps, the map data are obtained from multiple sources of varying trustworthiness [57]. As the HD market continues to expand, inconsistent approaches using potentially poor data may increase the risk of hazardous events occurring on the road network. In our work, we identified a data error in a commercially available HD map which caused degradation of the autonomous driving (AD) mode and a driver take-over request. The error is shown in Figure 6.1. The figure on the left-hand side shows the normal state of the road and the AD mode is on, whereas, the figure on the right-hand side shows the road gap caused by the map data error and the AD mode is off.

Usually, a take-over request is conducted for safety reasons when the AD system is approaching its limits due to, for example, weather conditions. In general, a take-over request is a complex and risky process and should be avoided as much as possible. In case of map errors, the request is not even related to system limits. Therefore, the goal of our work is to use ontologies and reasoning to find map errors and provide spatial context for fixing map errors to extend the AD function's availability. Ensuring (general) data quality with an ontology-based approach has been well-studied recently [54, 91, 53]. Yilmaz et al. [186, 187] have even demonstrated the feasibility of using ontological methods for spatial data quality evaluation. The latter work does not, however, consider the HD map-specific concepts such as lane-level geometry and the resulting challenges, and neither are the challenges and possibilities of *handling* violations considered.

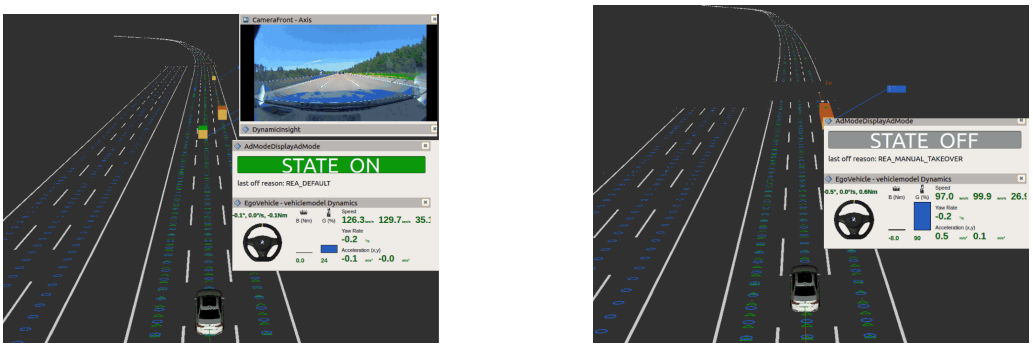


FIGURE 6.1: Snapshots of a normal driving scenario without map errors and active AD mode (left-hand side) and an error scenario (right-hand side) with deactivated AD mode and a driver take-over request due to a gap in the road model

Problem statement. In this chapter, we investigate the problem of HD map quality assurance. This chapter addresses the third level in the general proposed contributions presented in this thesis, i.e., ensuring map data quality with ontologies and rules. Particularly, the research question addressed in this chapter is as follows:

> RQ3: Can an ontology-based approach be utilized to ensure map data quality for road knowledge consistency?

Proposed solution. In this chapter, we tackle the problem of map quality assurance by applying an ontology-based approach. The *Map Quality Violation* (MQV) ontology and a set of constraint rules are used for violation detection. The MQV ontology is used for describing the violation. Constraint rules are used to model the constraints and generate the instances of the violation. Violation handling strategies consider the aspects of tolerance and resolution. The violation tolerance is achieved via RDF graph aggregation and violation resolution is achieved via RDF graph decomposition.

The contributions of this chapter are outlined as follows:

- We present a workflow for ensuring map data quality based on OWL 2 RL ontologies [93] and Datalog rules [10].
- We develop the Map Quality Violation (MQV) ontology and a set of constraint rules for violation detection.
- We demonstrate violation handling strategies via violation tolerance and resolution.
- We evaluate the performance of violation detection and the correctness of violation resolution using RDFox [142] and realistic map data.

The remainder of this chapter is structured as follows. Section 6.1 describes the design of map quality assurance focusing on the MQV ontology and constraint rules, the violation handling strategies is presented in Section 6.2. In Section 6.3, the evaluation result is described which is followed by a discussion in Section 6.4. Finally, concluding remarks for this chapter are presented in Section 6.5.

6.1 Design

In this section, we first introduce the workflow of ontology-based map quality assurance. Then we present the development of the MQV ontology and the constraint rules, which are used along the workflow.

6.1.1 The Workflow

In this section, we present the workflow of ensuring map data quality consisting of: (i) semantic enrichment, (ii) violation detection and (iii) violation handling (see Figure 6.2).

The semantic enrichment process in map data quality assurance focuses on deriving concepts and relationships needed for violation detection using the input data. For example, the bounding concepts on the lane group level as well as their successor relationships are needed to model the *full coverage checking* described in Section 6.1.3. The same design principle is applied to lane boundary full coverage

checking as well. Subsequently, the violation detection operates on the enrichment data via the MQV ontology and the constraint rules. The detected violations are treated based on their severity levels. Currently, critical errors are defined as those conditions that cause the crash of a component or a malfunction. For example, a tile not containing a highway is considered as a critical error. Other severity levels are related to map data completeness and consistency, which are less severe than the critical ones. If a critical error is detected, then the AD mode will be turned off, otherwise errors may be considered to be fixed with violation handling.

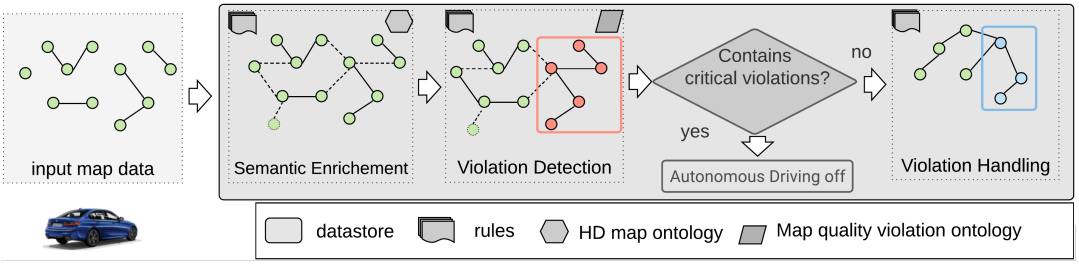


FIGURE 6.2: Workflow diagram of ensuring map data quality

6.1.2 MQV Ontology

In this section, we describe the modelling process of the MQV ontology following the bottom-up approach (see Section 4.1.2). The specific design decisions to cover the map quality are reported, such as an overview of the ontology structure with major classes and its properties. Next, ontologies and ODPs that are reused in MQV are discussed. To the best of our best knowledge, no previous research has been done in the area of HD maps due to the fact that it is a relatively new technology.

Purpose and Scope

The purpose of the MQV ontology is to enable a reasoner to detect HD map data violations and further provide context information of a violation to guide the repair process. Table 6.1 provides the detailed requirement's specification of the MQV ontology.

Capture

The capture of the domain of interest, i.e., Map Quality Violation, is achieved in two steps. First, we analyse the *Map Quality Check List* provided by BMW in the domain of spatial data quality [56]. The *Map Quality Check List* contains 20 items with detailed description of each checking item, such as the goal, priority, test steps and expected behaviours. *Topology consistency*, *geometry precision* and *attribute accuracy* are the three main quality parameters considered for modelling [56]. The Spatial Data Quality Ontology developed by Yilmaz et al. [187] provides the structure of ontological domain knowledge for modelling map quality violations. Second, domain experts with deep knowledge in HD map data support the process of capturing the knowledge of the domain by providing necessary documents and explaining the requirements.

TABLE 6.1: MQV Ontology Requirements Specification Document

MQV Ontology Requirements Specification Document
1. Purpose
The purpose of the MQV ontology is to enable a reasoner to detect HD map data errors, and further provides context information for a violation to guide the repair process.
2. Scope
The ontology has to cover a set of core concepts and relations describing HD map quality based on the provided *Quality Check List* : -Violation: represents the occurrence of map data violations. -Quality parameter: represent the map data quality-related parameters, such as topology, geometry and attributes. -Threshold: represents accepted criteria threshold, such as the distance between two geometry points. -Severity: represents the level of impact of a map data error on the vehicle. There are four different levels: low, medium, high and critical. -Map Object : represents the map object affected by the errors and those which are resolved after repairing.
3. Implementation Language
OWL RL, Datalog
4. Intended End-Users
User 1. Map providers who want to perform map data qaulity assurance. User 2. Navigation systems that need to detect map data errors.
5. Intended Uses
Use 1. provide semantic representation of map data violations. Use 2. provide context information for rule-based map data repair. Use 3. integrate map quality assurance into the map industry ecosystem using semantic web toolchains.
6. Ontology Requirements
a. Non-Functional Requirements
NFR 1. The ontology must follow the naming conventions; NFR 2. The ontology must re-use existing ontologies whenever possible.
b. Functional Requirements: Competency Questions
CQ1. what is the quality parameter types of the violation? CQ2. what error types are related to the violation? CQ3. Is the violation a link/lane/lane group/lane boundary violation? CQ4. what is the severity level of the violation? CQ5. what is the reason of the violation? CQ6. what are the affected/resolved map objects of the violation? CQ7. what is the accepted threshold for the violation?
7. Pre-Glossary of Terms
Violation, Quality Parameter, Violation Type, Severity Level, Affected, Resolved, Reason, Threshold, Map object

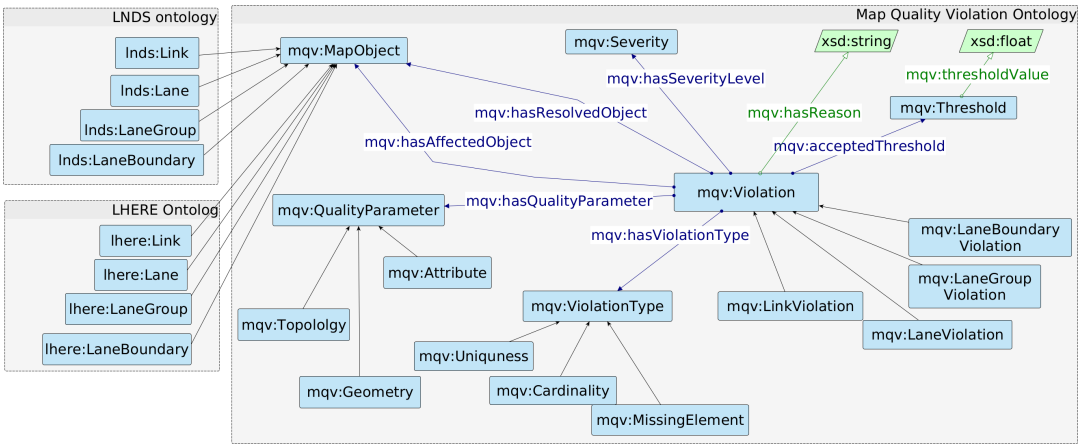


FIGURE 6.3: Classes and properties in the MQV ontology.

Development

An ontology, MQV, describing the map quality violations is detailed in this work. The namespace we used is *mqv*. We first provide an *overview* of the MQV ontology, then we present the *ODPs* used during the development process.

Ontology Overview

In this section, we describe the main classes of the MQV ontology (see Figure 6.3).

mqv:Violation represents the occurrence of map quality violations and is the core element of the MQV ontology. It is further divided into four subclasses depending on the map object type, such as `mqv:LinkViolation`. The reason behind the occurrence of a violation is described via the data property `mqv:hasReason` connecting to `mqv:Violation`.

mqv:QualityParameter expresses the quality parameter class in MQV. It describes the specific quality parameter associated with a violation via the object property `mqv:hasQualityParameter`. With the objective to describe a specific quality parameter of a violation, we model `mqv:Topology`, `mqv:Geometry` and `mqv:Attribute` as the subclasses of `mqv:QualityParameter`. Each type of the parameter can be further specified, describing more fine-grained classes. The advantage of modelling the parameters as classes is that they can be extended to more fine-grained classes. For instance, the class `mqv:Topology` can have a subclass `mqv:Coverage` to describe the coverage, and a subclass `mqv:Selfloop` to describe road network self-loops.

mqv:ViolationType represents the violation types in MQV. The objective is to describe the type of violations via the property `mqv:hasViolationType`. The uniqueness, cardinality and missing elements are the specific violation types under our concern. They are modelled as the sub-classes of `mqv:ViolationType`.

mqv:Threshold depicts the concept of the accepted threshold. It is connected to the class `mqv:Violation` via the object property `mqv:acceptedThreshold`. The specific threshold value is modelled by the data property `mqv:thresholdValue` connecting to `mqv:Violation`.

mqv:Severity expresses the severity class in MQV ontology. It describes the impact level that a map data error has on the AD related functions. The specific levels are represented as the individuals of `mqv:Severity` which are `mqv:Low`, `mqv:Medium`, `mqv:High` and `mqv:Critical`. The `mqv:Severity` class is not an enumeration of individuals (closed class) [92], as the class is intended to be extended with more individuals. If it is modelled as a closed class, it stipulates that `mqv:Low`, `mqv:Medium`,

`mqv:High` and `mqv:Critical` are the only individuals of the class `mqv:Severity` and any additionally added individual of `mqv:Severity` is inferred as equal to one of the existing four individuals.

mqv:MapObject describes the map object affected by the violations. It is defined by connecting to `mqv:Violation` via the object property `mqv:hasAffectedObject`. For the map data repair process, it is required to annotate the repaired map objects. This is done by the object property `mqv:hasResolvedObject` connecting to `mqv:MapObject`. The integration of specific map ontologies such as *LNDS* and *LHERE* can be achieved by defining the subclass relationship via `rdfs:subClassOf` between the classes in the low-level map ontologies and `mqv:MapObject`.

Listing 6.1 shows an example of a violation encoded in Turtle syntax. The triples state that `mqv:v1` is an instance of the class `mqv:LaneGroupViolation` and it has the violation type `mqv:uniqueness`. The quality parameter associated to it is `mqv:attribute`. The affected objects are `lnds:LaneGroup1` and `lnds:LaneGroup2`. The severity level of `mqv:v1` is `mqv:high`, and the violation reason is "same lane group ID".

```
mqv:v1 a  mqv:LaneGroupViolation;
    mqv:hasViolationType mqv:uniqueness;
    mqv:hasQualityParameter mqv:attribute;
    mqv:hasAffectedObject  lnds:LaneGroup1, lnds:LaneGroup2;
    mqv:hasSevertiyLevel mqv:high;
    mqv:hasViolationReason " Same lane group ID".
```

LISTING 6.1: An example of a violation encoded in Turtle

Ontology Design Patterns

We analyse existing ODPs that can be applied to the modelling of the MQV ontology. The *Parameter ODP* is used to model the accepted thread value for the violation. The advantage of modelling the threshold as a class is that it provides the extensibility to further define a more specific threshold class, such as `mqv:MinimumThreshold` with specific value range restriction. The *Class as a Property Value ODP* as featured in [143] is adopted to model the values of the violation type and the quality parameter. This pattern suits the scenario when an existing subsumption class hierarchy is to be reused as a terminology or as a controlled vocabulary to annotate certain elements of other concepts in an ontology. For example, the `mqv:Violation` class can be further specified using the subclasses of `QualityParameter` and `ViolationType`.

Summary of Ontology Characteristics

Table 6.2 reports a summary of the characteristics of MQV ontology. The reuse of ODPs is included in the design process of the ontology.

TABLE 6.2: Summary of the MQV Ontology Characteristics

Name	MQV ontology
size	23 classes, 6 object properties, 2 data properties, 10 individuals, 86 axioms
DL Expressivity	ALEO (D)
Reuse ODP	Parameter, Classe As Property Value
Naming Conventions	CamelCase notation
Methodology	Uschold and Gruninger [177]

6.1.3 Constraint Rules

In this section, we outline the design of *Constraint rules* used for the HD map data validation. The design of violation rules is inspired from the work of Kharlamov et al. [111] where they used constraint rules for industrial use cases.

In our work, the constraint rules are aimed at identifying map data errors. Based on the map data error types, the rules are classified into (1) *topology*, (2) *geometry*, and (3) *attribute* checking rules. Before describing the details of each rule type, we first introduce *Violation Recording Rule Templates* (VRRTs), which provide the patterns for modelling constraint violation detection rules. Table 6.3 shows OWL axioms used to capture the map data quality requirements together with their corresponding *Datalog Constraint Atoms* (DCA). Note that for *Literal Value Restriction*, the Datalog Constrain Atoms can be written as $\mathsf{C}(x)$, $\mathsf{dp}(x, y)$, $\texttt{FILTER}(y = \{\top, \bot\})$ when the data type for the literal is boolean.

TABLE 6.3: Constraint axioms as Datalog Constraint Atoms (DCA). We use C for classes, op for object, dp for data, and p for object or data properties.

OWL Axiom	**Datalog Constraint Atom (DCA)**
Existential Quantification	$\mathsf{C}(x)$, NOT EXISTS y IN $(\mathsf{C}(y), \mathsf{p}(x, y))$
Individual Value Restriction	$\mathsf{C}(x)$, NOT $\mathsf{op}(x, individual)$
Literal Value Restriction	$\mathsf{C}(x)$, NOT $\mathsf{dp}(x, literal)$
HasKey	$\mathsf{C}(x)$, $\mathsf{dp}(x,z)$, $\mathsf{C}(y)$, $\mathsf{dp}(y,z)$, $\texttt{FILTER}(x \neq y)$
Min ($<$)/Max ($>$) Cardinality Restriction	$\mathsf{C}(x)$, $\texttt{AGGREGATE}(\mathsf{p}(x,v)$ ON x BIND $\texttt{count}(v)$ AS $n)$, $\texttt{FILTER}(n \bowtie max), \bowtie \in \{>, <\}$

The occurrence of the constraint violation is recorded using freshly generated instances of `Violation` as shown in the following VRRTs template:

$$\mathsf{Violation}(v),\ \mathsf{hasAffectedObject}(v, x),\ \mathsf{hasReason}(v, \texttt{"}r\texttt{"}) \leftarrow$$
$$\texttt{<DCA>},\ \texttt{BIND}(\texttt{SKOLEM}(\texttt{"}d\texttt{"},\ x)\ \texttt{AS}\ v).$$

The body of the rule template is formed via a DCA with an atom using `SKOLEM` and `BIND` functions. When the DCA is satisfied, the `SKOLEM` function generates an instance based on the identified error object x which is assigned to a variable v via the `BIND` function. The head of the rule template uses the classes and properties defined in MQV ontology. It asserts the newly generated `v` as an instance of the class `Violation`, and links it to the affected object x via the object property `hasAffectObject`. The reason of the violation denoted as string `r` is assigned to v via the data property `hasReason`.

Let's consider a concrete example of the minimum cardinality constraint of lane shape points using the above template. The rule below identifies all the lanes whose shape points are less than 2. The DCA corresponds to Min Cardinality Restriction stated in Table 6.3.

$$\mathsf{Violation}(v),\ \mathsf{hasAffectedObject}(v, l),\ \mathsf{hasReason}(v, \texttt{"MinCardinalityError"}) \leftarrow$$
$$\mathsf{Lane}(l),\ \texttt{AGGREGATE}(\mathsf{hasPoint}(l, p)\ \texttt{ON}\ l\ \texttt{BIND}\ \texttt{count}(p)\ \texttt{AS}\ n),$$
$$\texttt{FILTER}(n < 2),\ \texttt{BIND}(\texttt{SKOLEM}(\texttt{"}d\texttt{"},\ l)\ \texttt{AS}\ v).$$

Topology Checking Rules

Topology checking rules are designed to check the spatial relationships of map objects. A comprehensive formal categorization of binary topological relations between regions, lines, and points has been developed by Egenhofer and Herring [60]. In this work, we model *full coverage* constraints, checking if a set of other map objects fully covers a given map object. For example, Figure 6.4 shows the full coverage constraint between a link and several lane groups. To describe such constraints, we first introduce some basic notations.

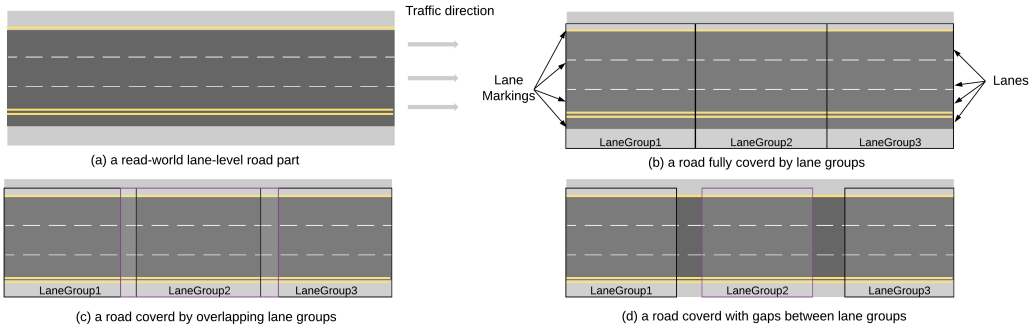


FIGURE 6.4: An example illustrating full coverage constraints between a link (road) and several lane groups. (a) Example of a real-world road; (b) the road is fully covered by lane groups; (c) the road is covered by lane groups that are overlapping with each other; (d) the road is not completely covered by lane groups as there are gaps between them.

Definition 6.1.1. A *line (segment)* $\overline{pq}$ is defined by its *start point* p and its *end point* q, where $p \neq q$. A (base) line $\overline{pq}$ is *fully covered* by a sequence of lines $\overline{u_1v_1} \dots \overline{u_nv_n}$ if $p = u_1$, $q = v_n$, and $v_i = u_{i+1}$ for each $1 \leq i < n$, where $u_i \neq v_i$. We say that there is a *gap at the start* if $p < u_1$, a *gap at the end* if $v_n < q$, a *gap in the middle* if $v_i < u_{i+1}$ for some $1 \leq i < n$, and there is an *overlapping* if $v_i > u_{i+1}$ for some $1 \leq i < n$.

Algorithm 4 presents pseudo-code for checking the full coverage. Given a baseline $\overline{pq}$ and a sequence of line segments L without self-loops, the algorithm first identifies the start and end segment in L (lines 2–3) using the bounding rules. The algorithm checks if there is a gap at the start or end w.r.t. the baseline (lines 4–7). At last, it iterates through the given line segments, and for each segment, it gets the direct next line segment (line 9) through the topology rules, checking for any gaps in the middle (lines 8–10) or any overlapping (lines 11–12).

According to the NDS specification, a link (road) needs to be fully covered by a set of lane groups. We use the following rules to illustrate how Algorithm 4 is implemented. The rules, where we abbreviate hasAffectedObject as hao to check the full coverage of a link are as follows:

$$\begin{aligned}&\mathsf{Violation}(v),\ \mathsf{hao}(v,f),\ \mathsf{hao}(v,lg),\ \mathsf{hasReason}(v,\texttt{"GapAtStart"}) \leftarrow\\ &\quad \mathsf{Feature}(f),\ \mathsf{FeatureRef}(fr),\ \mathsf{refersTo}(fr,f),\\ &\quad \mathsf{hasFeatureRef}(lg,fr),\ \mathsf{StartLaneGroup}(lg),\ \mathsf{hasRange}(lg,r),\\ &\quad \mathsf{startPosition}(r,sp),\ \texttt{FILTER}(sp>0),\\ &\quad \texttt{BIND}(\texttt{SKOLEM}(\texttt{"t}d\texttt{"},\ f,\ lg)\ \texttt{AS}\ v).\end{aligned}$$

$$\begin{aligned}&\mathsf{Violation}(v),\ \mathsf{hao}(v,f),\ \mathsf{hao}(v,lg),\ \mathsf{hasReason}(v,\texttt{"GapAtEnd"}) \leftarrow\\ &\quad \mathsf{Feature}(f),\ \mathsf{numShapeAttrPoint}(f,t),\ \mathsf{FeatureRef}(fr),\ \mathsf{refersTo}(fr,f),\end{aligned}$$

input : $\overline{pq}$: a base line, $L = \{\overline{u_1v_1}, \ldots, \overline{u_nv_n}\}$: a set of line segments
output: L_g, L_o: sets of line segments causing gaps and overlappings, resp.

$L_g = L_o = \emptyset$;
$\overline{u_sv_s}$ = GETSTARTSEGMENT(L) ; // apply bounding rules
$\overline{u_ev_e}$ = GETENDSEGMENT(L) ; // apply bounding rules
if $u_s > p$ **then**
  $L_g = L_g \cup \{\overline{u_sv_e}\}$; // gap at the start
if $v_e < q$ **then**
  $L_g = L_g \cup \{\overline{u_ev_e}\}$; // gap at the end
for $i = 1 \ldots n-1$ **do**
  $\overline{uv}$ = GETDIRECTNEXT($\overline{u_iv_i}$) ; // apply topology rules
  if $v_i < u$ **then**
    $L_g = L_g \cup \{\overline{u_iv_i}, \overline{uv}\}$; // gap in the middle
  else if $v_i > u$ **then**
    $L_o = L_o \cup \{\overline{u_iv_i}, \overline{uv}\}$; // overlapping

Algorithm 4: Check full coverage

hasFeatureRef(lg, fr), EndLaneGroup(lg), hasRange(lg, r),
endPosition(r, ep), FILTER($ep < t$),
BIND(SKOLEM("d", f, lg) AS v).

Violation(v), hao(v, f), hao(v, lg_1), hao(v, lg_2), hasReason(v, "GapInTheMiddle") ←
LaneGroup(lg_1), LaneGroup(lg_2), hasDirectNextLG(lg_1, lg_2),
hasRange(lg_1, r_1), hasRange(lg_2, r_2), endPosition(r_1, ep),
startPosition(r_2, sp), FILTER($ep < sp$),
BIND(SKOLEM("d", f, lg_1, lg_2) AS v).

Violation(v), hao(v, f), hao(v, lg_1), hao(v, lg_2), hasReason(v, "Overlapping") ←
LaneGroup(lg_1), LaneGroup(lg_2), hasDirectNextLG(lg_1, lg_2),
hasRange(lg_1, r_1), hasRange(lg_2, r_2), endPosition(r_1, ep),
startPosition(r_2, sp), FILTER($ep > sp$),
BIND(SKOLEM("d", f, lg_1, lg_2) AS v).

Geometry Checking Rules

Geometry checking rules are designed to check the geometric representation of the links (lanes). The link (lane) model uses an ordered sequence of shape points describing the geometry of a polyline that represents a link (lane). We further subdivide geometry checking rules into *cardinality* and *geometric accuracy* checking rules. Cardinality checking rules use the minimum or maximum cardinality restrictions in VRRTs. Geometric accuracy is checked via the coordinate proximity using distance thresholds [56] to account for different levels of accuracy and precision [191] in the collected map data. Figure 6.5 illustrates a case in which the radius distance of geometric points in two connected lanes should be less than a threshold value, for e.g. 0.5 m. If the radius distance is bigger than an accepted threshold, a violation is generated.

The following rule illustrates the case where a radius distance threshold of the geometric points in two connected lanes is checked.

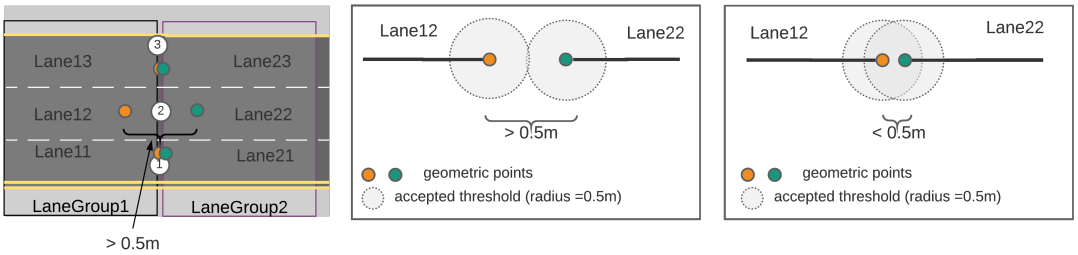


FIGURE 6.5: An example of lane geometric accuracy with a threshold

$$\begin{aligned}&\mathsf{Violation}(v),\ \mathsf{hao}(v,u_p),\ \mathsf{hao}(v,v_q),\ \mathsf{hasReason}(v,\text{"GeometryError"}) \leftarrow \\ &\quad \mathsf{Lane}(p),\ \mathsf{Lane}(q),\ \mathsf{hasDirectNext}(p,q),\ \mathsf{endPoint}(p,u_p),\ \mathsf{startPoint}(q,v_q),\\ &\quad \mathsf{CoordinateDistance}(c),\ \mathsf{hasSource}(c,q),\mathsf{hasTarget}(c,p),\ \mathsf{distance}(c,d),\\ &\quad \mathsf{Threshold}(t),\ \mathsf{thresholdValue}(t,v_t),\ \texttt{FILTER}(d > v_t),\\ &\quad \texttt{BIND}(\texttt{SKOLEM}(\text{"}d\text{"},\ u_p,\ v_q)\ \texttt{AS}\ v).\end{aligned}$$

Attribute Accuracy Checking Rules

In order to check if the recorded attributes of the map data representing real-world entities are correct and consistent, we used a set of accuracy checking rules. The attributes could be feature classifications, text information for feature names, or descriptions, which ought to be consistent with each other. For example, if a road is classified as a motorway, it should also have a controlled-access designed for high-speed vehicular traffic. Controlled-access is modelled as a data property with a Boolean value. Hence, the corresponding violation detection rule can be modelled using a literal value restriction in the VRRT. The concrete rule is modelled as follows:

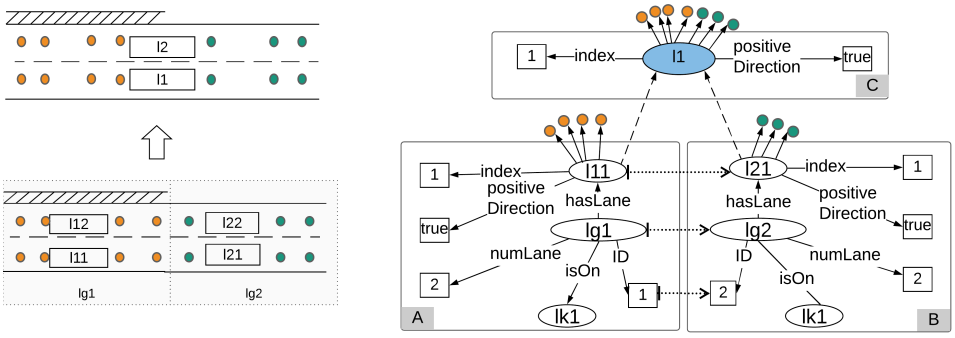
$$\begin{aligned}&\mathsf{Violation}(v),\ \mathsf{hao}(v,f),\mathsf{hasReason}(v,\text{"AttributeError"}) \leftarrow \\ &\quad \mathsf{Tile}(t),\ \mathsf{hasFeature}(t,f),\ \mathsf{Motorway}(f),\\ &\quad \mathsf{isControlledAccess}(f,c),\ \texttt{FILTER}(c = \bot),\\ &\quad \texttt{BIND}(\texttt{SKOLEM}(\text{"}d\text{"},\ p,\ n,\ q)\ \texttt{AS}\ v).\end{aligned}$$

6.2 Violation Handling

Violations are handled based on the severity level. If a critical violation is detected during the map pre-loading phase, the autonomous driving mode is switched off and the control is handed over to a driver in the corresponding region. For non-critical violations, we rely on *violation tolerance* and *violation resolution* strategies considering the spatial relations. Violation tolerance is feasible because errors in the low-level (raw) data do not necessarily affect the decision taken at the knowledge (human-perceivable) level in intelligent systems [185]. In cases where the violations cannot be tolerated, spatial knowledge, e.g., topological and geometric relations, can be used to resolve violations [129, 11]. These strategies allow us to support the autonomous driving applications, even in the presence of low-level data errors.

To achieve knowledge level consistency, we apply graph aggregation [128] for violation tolerance and the graph decomposition [20] for violation resolution. Essentially, these strategies take advantage of graph structure similarity, which is captured by the notion of *isomorphisms* :

Definition 6.2.1 (RDF Graph Isomorphism [50])**.** Let G_1 and G_2 be RDF graphs with V_1 and V_2 as the induced vertices of G_1 and G_2, respectively. We say that G_1 and G_2 are *isomorphic*, if there is a bijection $\mu\colon V_1 \rightarrow V_2$ such that $\mu(b) \in B$ for each



(a) lanes aggreation visual representation.

(b) lanes aggreation graph representation

FIGURE 6.6: A violation-free example of RDF graph aggregation over lanes

$b \in V_1 \cap B$, $\mu(\ell) \in L$ for each $\ell \in V_1 \cap L$, $\mu(v) \in I$ for each $v \in V_1 \cap I$, and, for each triple $(s, p, o) \in G_1$, $(\mu(s), p, \mu(o)) \in G_2$. We call such μ an *isomorphism* between G_1 and G_2.

Based on isomorphism, we introduce graph aggregation and its use for *violation tolerance*. Apart from its use in violation tolerance, graph aggregation is helpful in itself to obtain a higher-level view of the map data, with a focus on the details that are important for autonomous driving.

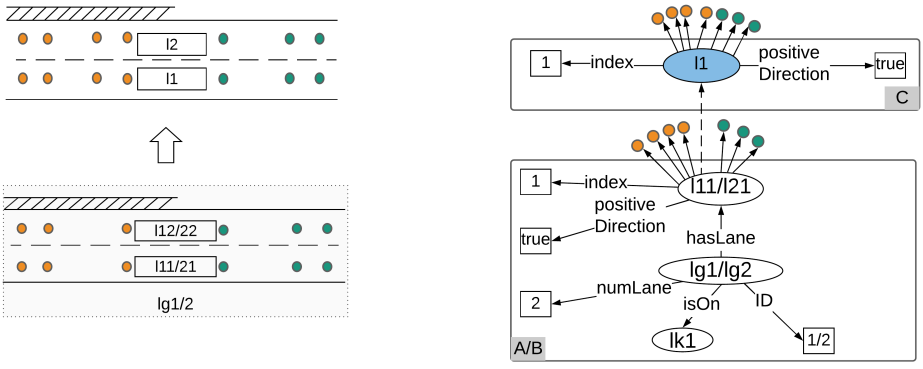
Definition 6.2.2 (RDF Graph Aggregation)**.** Let G_1, G_2, and G be RDF graphs with vertices V_1, V_2, and V, respectively, such that G_1 is isomorphic to G_2 witness by the isomorphism μ. A (partial) function $\alpha\colon V_1 \cup V_2 \to V$ is an *abstraction function w.r.t. G_1, G_2, and G* if, for each $v \in V$, there are nodes $v_1 \in V_1$ and $v_2 \in V_2$ such that $\mu(v_1) = v_2$ and $\alpha(v_1) = \alpha(v_2) = v$. If an abstraction function w.r.t. G_1, G_2, and G exists, we call G an *aggregation graph* of G_1 and G_2.

We generalise the notion of an abstraction graph to a set of pairwise isomorphic graphs $G_1, \ldots, G_n$ in a natural way.

Figure 6.6 shows an example where we apply graph aggregation over the lanes of two lane groups with ID 1 and 2. We abbreviate lane as l, laneGroup as lg, and link as lk, e.g., l11 stands for lane11. Subfigure (a) shows a map visualisation, while (b) shows the corresponding graph representation, and the aggregation is shown in the upper part. Note that the mapping with a dotted line shows the isomorphism between graph A and graph B (we omit the mapping for identical values such as a mapping from true in graph A to true in graph B). The dashed lines show the abstraction function, where we again omit the identical value mappings. The abstraction function only maps the lanes (l11 in graph A and l21 in graph B) as well as the lane index and the direction attribute. The lane aggregation aligns with the human perception of `l11` and `l21` as one continuous lane.

6.2.1 Violation Tolerance.

Figure 6.7 shows an example with a violation, which consists of a duplicate lane group ID. More precisely, lg1 and lg2 both have ID 1 in the map data. As a result, the map data is parsed as containing just one lane group (with ID 1), which also causes l11 and l21 to be considered equal as they both have ID 1 and belong to the lane group with ID 1. Hence, we get identical RDF graphs for l11 (graph A) and



(a) lane aggregation visual representation with lane group ID dupicated

(b) lane aggregation graph representation with lane group ID dupicated

FIGURE 6.7: An example of lane aggregation with lane group ID uniqueness violation

l21 (graph B), which is a special case of RDF graph isomorphism. Applying the abstraction function (as in Figure 6, dashed line) results, however, in the same (correct) aggregation graph (graph C) as for the violation-free scenario shown in Figure 6.6. Hence, the RDF graph aggregation can tolerate some data errors.

6.2.2 Violation Resolution

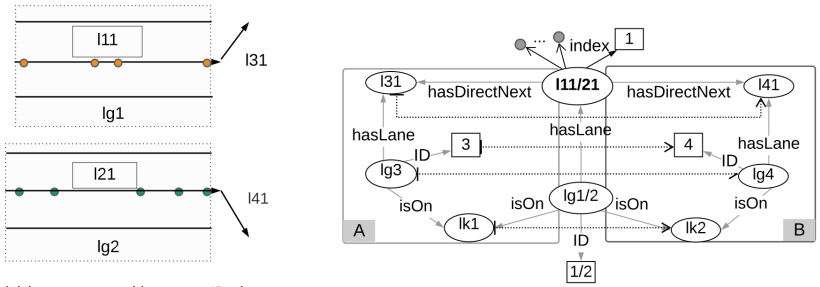
We illustrate how violations can be resolved (in particular, lane ambiguity) using graph decomposition.

Definition 6.2.3 (RDF Graph Decomposition)**.** The *RDF decomposition* of an RDF graph G is a collection of edge-disjoint, isomorphic subgraphs $G_1, \ldots, G_n$ of G such that every edge of G belongs to exactly one G_i, $1 \leq i \leq n$. We denote such a decomposition of G as $\hat{G} = \{G_1, G_2, \ldots, G_n\}$.

Figure 6.8 shows an example of lane ambiguity also caused by a lane group ID duplication. Subfigure (a) shows a normal map visualisation of lg1 and lg2 located in separate roads. Subfigure (b) shows the graph representation resulting from the duplicate ID of lg1 and lg2 which causes l11 and l21 to merge into one lane instance having both lanes' spatial relationships, such as associated points, links and successor lanes. Based on the graph structure of the ambiguous graph, there exists a mapping between subgraphs A and B, which indicates the application of RDF decomposition. Hence, we apply RDF graph decomposition to fix the topology and distance measurements to restore it's geometry. Figure 6.9 shows the concrete steps: (1) violation detection, (2) topology correction, and (3) assignment of geometric points.

In Step 1, a topology violation is detected if a lane group is associated with two disconnected links. This is modelled by checking the existence of a connection between links associated to a lane group using an existential qualification in a VRRT, and an instance of LaneViolation is generated having topology (an instance of class Topology) as its QualityParameter. The concrete rule is as follows:

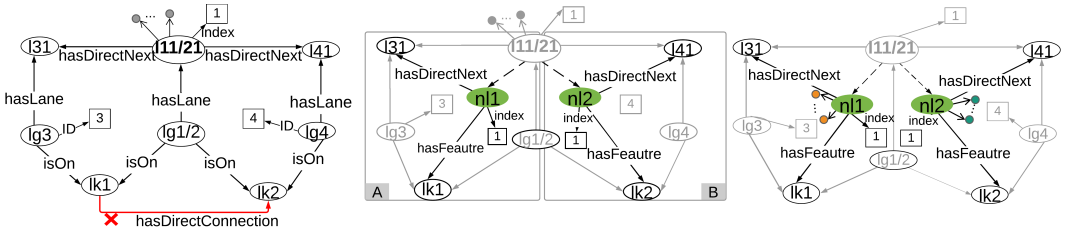
$$\begin{aligned}&\mathsf{LaneViolation}(v),\ \mathsf{hao}(v,l),\ \mathsf{hao}(v,f_1),\ \mathsf{hao}(v,f_2),\ \mathsf{hasQualityParameter}(v,\mathsf{topology}) \leftarrow\\ &\quad \mathsf{LaneGroup}(lg),\ \mathsf{hasLane}(lg,l),\ \mathsf{isOn}(lg,f_1),\ \mathsf{isOn}(lg,f_2),\\ &\quad \texttt{FILTER}(f_1 \neq f_2),\ \texttt{NOT}\ \mathsf{hasDirectConnection}(f_1,f_2),\\ &\quad \texttt{BIND}(\texttt{SKOLEM}(\texttt{"}d\texttt{"},\ l)\ \texttt{AS}\ v).\end{aligned}$$



(a) lane groups with correct IDs in differnt roads visulisation

(b) lane group graph representation with ID dupicated

FIGURE 6.8: An example of lane ambiguity caused by lane group ID duplication.



(1) Violation detection

(2) Topology correction

(3) Geometric shape points assgiment

FIGURE 6.9: Lane ambiguity violation resolution steps

In Step 2, the topology correction is achieved via the graph decomposition and relationship establishment. The original graph of l11/21 can be decomposed into isomorphic subgraphs A and B. Two new lane instances (nl1 and nl2) are generated with the correct topological relationships.

$$\begin{aligned}\mathsf{NewLane}(l),\ \mathsf{hasFeature}(n,f),\ \mathsf{hasDirectNext}(l,n),\ \mathsf{hasOriginalLane}(l,m) \leftarrow\\ \mathsf{LaneViolation}(v),\ \mathsf{hasQualityParameter}(v,\mathsf{topology}),\ \mathsf{hao}(v,m),\\ \mathsf{hasLane}(lg_1,m),\ \mathsf{isOn}(lg_1,f),\ \mathsf{LaneGrp}(lg_2),\ \mathsf{hasLane}(lg_2,n),\\ \mathsf{isOn}(lg_2,f),\ \mathsf{hasDirectNext}(m,n),\ \mathsf{index}(m,i),\\ \texttt{BIND}(\texttt{SKOLEM}(\texttt{"}d\texttt{"},\ f,\ i)\ \texttt{AS}\ l)\end{aligned}$$

In Step 3, the geometric shape point assignment is achieved via a point grouping strategy, which compares the distance from each shape point of the lane to the first and last shape point of the lane associated links. The shape points are then grouped if the difference between the calculated distance and the links' length is within a threshold, e.g., $10\,m$.

$$\begin{aligned}\mathsf{hasPossibleLanePoint}(f,p) \leftarrow\\ \mathsf{LaneViolation}(v),\ \mathsf{hasQualityParameter}(v,topology),\ \mathsf{hao}(v,l),\ \mathsf{hasShapePoint}(l,p),\\ \mathsf{hasFeature}(l,f),\ \mathsf{length}(f,n),\ \mathsf{hasFirstShapePoint}(f,u),\ \mathsf{hasLastShapePoint}(f,o),\\ \mathsf{CoordinateDistance}(d_1),\ \mathsf{hasSource}(d_1,p),\ \mathsf{hasTarget}(d_1,u),\ \mathsf{distance}(d_1,t_1),\\ \mathsf{CoordinateDistance}(d_2),\ \mathsf{hasSource}(d_2,p),\ \mathsf{hasTarget}(d_2,o),\ \mathsf{distance}(d_2,t_2),\\ \texttt{FILTER}(\texttt{ABS}((t_1+t_2)-n)<10).\end{aligned}$$

While assigning the point groups to correct new lanes, geometric point grouping is verified by comparing the number of points in each group to the total number of points of the original lane to prevent wrong point group assignments.

$$\begin{aligned}\mathsf{hasShapePoint}(n,p) \leftarrow\\ \mathsf{NewLane}(n),\ \mathsf{hasOriginalLane}(n,l),\ \mathsf{numPoints}(l,u),\ \mathsf{hasFeature}(n,f),\\ \mathsf{hasPossibleLanePoint}(f,p),\ \mathsf{numPossibleLanePoints}(f,m),\ \texttt{FILTER}(m<u).\end{aligned}$$

6.3 Evaluation

In this section, We first show the performance of semantic enrichment and violation detection and then we evaluate the correctness of violation handling.

6.3.1 Violation Detection

We used 16 adjacent real map tiles along the Federal Motorway 92 (Bundesautobahn 92) in Germany for violation detection evaluation. The evaluation was performed on a 64-bit Ubuntu virtual machine with 8 Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz running at 33MHz with 15 GB memory. We first show the performance of semantic enrichment and violation detection and then we evaluate the correctness of violation handling. The recorded the computation time after doing a warm-up run by executing the tasks 3 times sequentially.

The result of the violation detection is summarised in Figure 6.10. In general, the computation time for total violation checking increases with respect to the data size. The average number of input triples is 117,733 and the average execution time of violation detection is 132 ms. With the objective to investigate what type of violation checking rules cost more time, we divided the time measurements into three parts according to the rule types, namely topology, geometry and attribute. As shown in the Table 6.4, the geometry checking rules take up to 50.47% of the total violation detection time, whereas, topology checking rules take 32.77% and attribute checking rules take 16.77%. This is because of the nature of the HD maps, which use a large number of shape points to describe the shapes of lanes and roads.

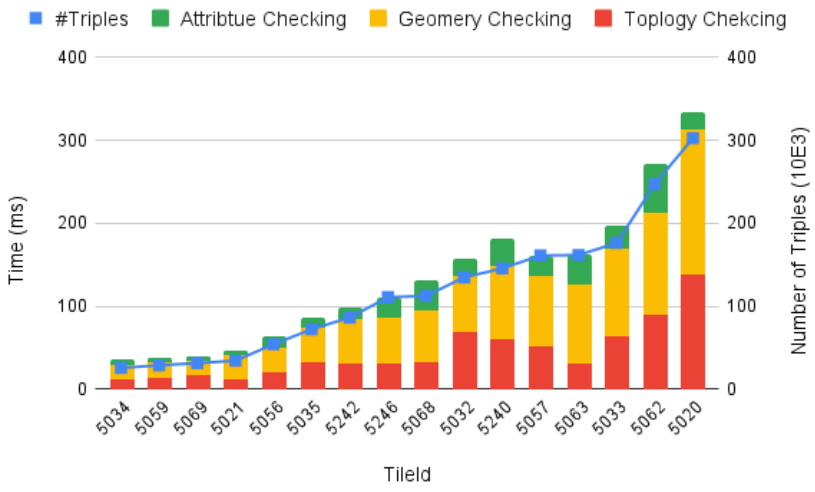


FIGURE 6.10: Performance of violation detection over real map data; the left-hand side scale shows the execution time for the violation detection (column) in milliseconds. The time for topology, geometry and attribute checking rules are shown as segments of each column. The number of input triples is shown as a line using the scale on the right-hand side.

6.3.2 Violation Handling

We consider the use case of lane group ID uniqueness violations to evaluate violation handling strategies. We show the result of the violation tolerance over an error on a

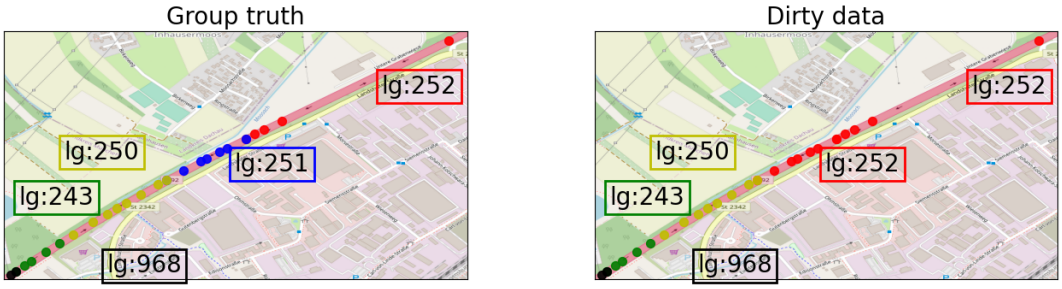
TABLE 6.4: Number of rules used for violation detection

Rule Type	#Rule	AVG Time(%)
Topology Checking	10	32.77
Geometry Checking	14	50.47
Attribute Checking	13	16.77

highway consisting of several lane groups, and violation resolution over an error on two lanes groups on separated roads.

Violation Tolerance

The error in Figure 6.1 occurred in the map data containing a highway in Germany. Part of this highway is represented by five continuous lane groups with the same number of lanes. Two of the lane groups have the same ID, which caused the degradation of the autonomous driving mode. Figure 6.11a shows the ground truth of the data. The dots represent the sequence of shape points for the lane with index 3 in each lane group. For example, the red dots represent the shape points of the lane with index 3 in lane group 252. The effect of duplicated lane group ID 251 in 252 is shown in Figure 6.11b where the existence of lane group 251 is represented by lane group 252.



(A) Ground truth.

(B) Dirty data where lane group 251 is 252.

FIGURE 6.11: Violation tolerance of lane group ID duplication. The shape points of the lanes with index 3 are visualized in dots.

We applied graph aggregation over both the ground truth and dirty data. The lane aggregation results agree on both inputs (see Table 6.5). Figure 6.12 shows the test results from the *SmartMap* application where the lane group ID duplication violation is detected and resolved in the successor lanes of the current lane.

TABLE 6.5: The violation tolerance over a lane using graph aggregation on ground truth and dirty data

	lane	length (m)	#points	successor	link
ground truth	BE9D6	2034	54	563E	02
dirty data	BE9D6	2034	54	563E	02

According to the combination ${}^{n}C_{k} = \frac{n!}{k!(n-k)!}$, where n is the number of lane groups, and k is the number of lane groups involved in ID duplication, there are

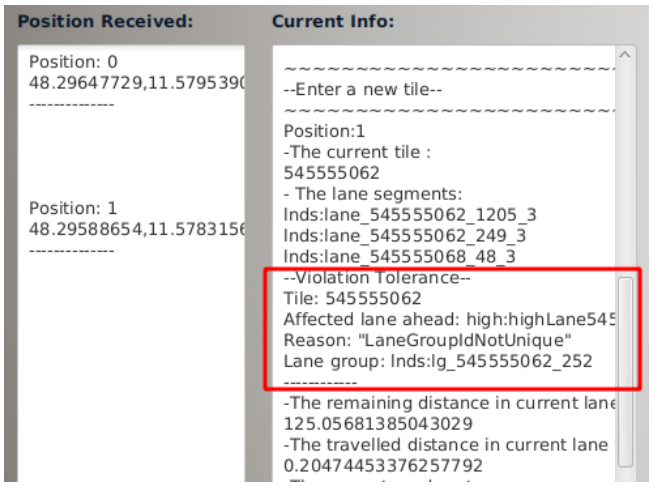
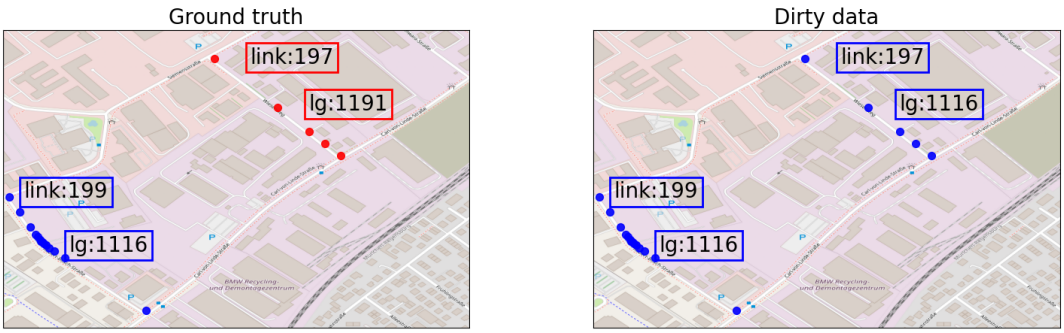


FIGURE 6.12: Violation tolerance tested in *SmartMap* based on the dirty data in Figure 6.11b

10 ID duplication combinations in total when $n = 5$ and $k = 2$. The lane aggregation, over the other 9 dirty data cases, all resulted in the same result. The conducted tests show that the lane group ID issue can be solved.

Violation Resolution

We evaluated the resolution strategy over two lane groups containing only one lane allocated to different links. Figure 6.13a shows the ground truth of the lanes in lane group 1116 and 1191 with their associated links. The effect of lane group ID 1116



(A) Ground truth

(B) Dirty data where lane group 1191 is 1116.

FIGURE 6.13: The shape points of the lanes with index 0 are visualized in dots.

duplication is visualized in Figure 6.13b. As the figure shows, the duplicated ID of 1116 caused an ambiguity of the lane, namely, the very same lane in lane group 1116 has two different map representations.

After applying the graph decomposition over the dirty data using rules (see Section 6.2.2), the relationship was restored with the newly generated lane instances `lnds:168747` and `lnds:168748` shown in Figure 6.14. Table 6.6 summarises the ground truth, dirty data and resolved violation result of lane group ID 1116 duplication. The identical values of the ground truth (Row 1 and Row 2) and the resolved violation (Row 4 and Row 5) show that the RDF graph decomposition using spatial relations can resolve violations caused by lane group ID duplication.

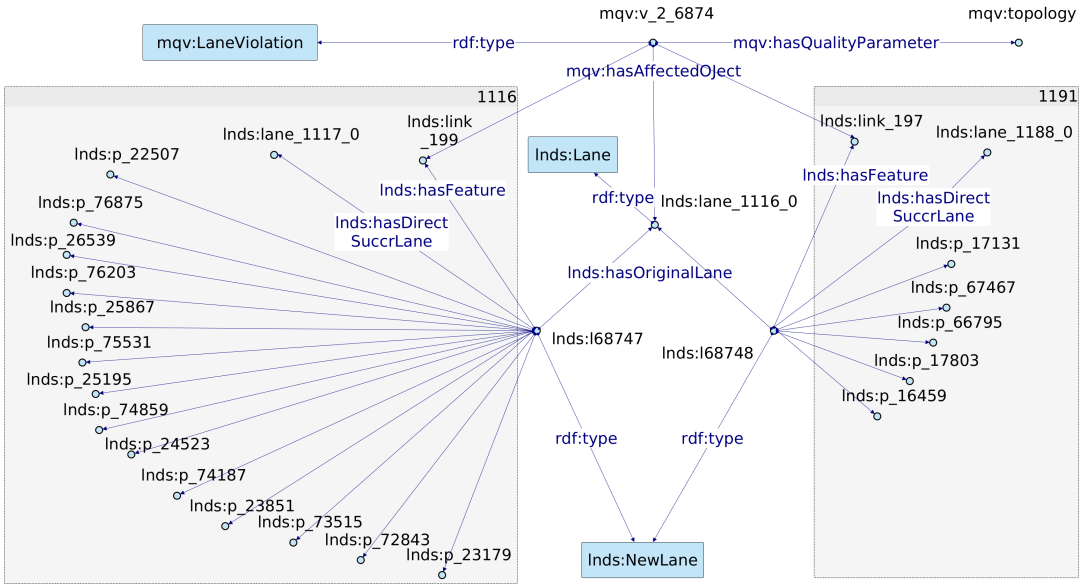


FIGURE 6.14: The RDF graph representation of a resolved violation with two newly generated lanes associated with the correct properties.

TABLE 6.6: The resolution of a lane violation using graph decomposition

	lane	length (m)	#points	successor	link
ground truth	1116_0	358	14	1117_0	199
	1191_0	314	5	1118_0	197
dirty data	1116_0	314, 358	19	1117_0, 1188_0	199, 197
resolved violation	68747	358	14	1117_0	199
	68748	314	5	1188_0	197

6.4 Discussion

Overall, the results demonstrate that our approach can improve the map data quality, resulting in an improved error-tolerance of AD systems. On the one hand, the performance of the violation detection allows the deployment of the proposed solution using client-server approach described in previous chapter (see Section 5.5.2) to check the map data before sending it to the car or client only approach (see Section 5.5.1) in case of loss of connectivity with the back-end. On the other hand, the evaluation of the violation handling strategies has shown that we could avoid the deactivation of the AD mode by detecting the error and correcting the map data in both cases of the lane group error.

As we reported, the results are promising. There are limitations of our violation handling approach, which can be extended in the future. First, our strategy applies only to the lane groups with similar structure (e.g. the same number of lanes). This is because graph aggregation and decomposition are based on pattern matching, namely finding similar graphs. Second, for the lane groups containing more than one lane, the topological relationships (e.g. links, successors) can be repaired using the RDF graph decomposition but not the relationships between shape points and each lane in a lane group. To repair the associations between shape points and corresponding lanes in one lane group, the in-depth distance patterns between shape points needs to be studied based on statistical analysis.

6.5 Conclusion

In this chapter, we investigated the applicability of an ontology-based approach for ensuring map data quality. We proposed a workflow considering semantic enrichment, violation detection and violation handling. Semantic enrichment is described in Chapter 4 which provides the needed spatial knowledge for violation detection and handling. To achieve the violation detection, we reviewed the provided HD Map Quality Check List in the context of spatial data quality and defined a Map Quality Violation Ontology. We identified the key OWL axioms to capture the map data integrity constraints. Since those constraints cannot be captured by OWL 2 RL, we modelled them using Datalog rules, providing the correspondence between the Datalog constraint atoms and OWL axioms. The violation handling strategies focused on the error caused by lane group ID duplication. We presented violation tolerance using graph aggregation and violation resolution using graph decomposition. The set of Datalog rules are utilised to model the violation handling strategies.

We evaluated the performance of violation detection and identified that geometry checking rules take the majority of the time due to the number of geometry checking rules and shape points involved. We tested the correctness of violation handling over two scenarios where lane group ID duplication occurs. In the first scenario, the lane group ID duplication occurred in two lane groups on the same road. The RDF graph aggregation was applied on the ground truth and dirty data, and the aggregated results agree on both inputs. For the second scenario, the lane group ID duplication occurred in two lane groups containing one lane on two separate roads. The result shows that the violation can be resolved by graph decomposition.

The overall results shows that our approach can successfully check the quality of the map data and suggests that violation handling is even feasible on-the-fly using client deployment (see Section 5.5.1), avoiding the autonomous driving mode's deactivation. Moreover, we observe some advantages of using an ontology-based

approach for map data quality assurance. Since the RDF graph representation of the map data naturally aligns with the road (lane) network model which is the underlying model of all HD maps. This provides the advantage of using existing graph techniques for map data analysis and for deriving new knowledge. The extensibility and flexibility of using an ontology and rules to model constraints eases the process of constraint development, and with the fixed vocabularies used in constraints, an automatic violation rule generation can be achieved.

Chapter 7

Conclusion and Outlook

In this thesis, we investigate the problem of the map modelling and processing for autonomous vehicles. In order to solve this problem, we propose an ontology-based approach focusing on map data integration and processing considering data quality assurance. The ontology-based approach enables a shift from map data-oriented functional design to knowledge-centered ontology design. The discussion of the research challenges, the research questions, as well as the contributions, are presented in Chapter 1. Necessary background concepts are examined in Chapter 2. An overview of state-of-the-art approaches related to the main problem addressed in this thesis is presented in Chapter 3. Then, the subsequent three core chapters of the thesis describe and evaluate the proposed ontology-based approach for map data integration, processing and quality assurance. Finally, in this chapter, the thesis is concluded by revisiting the research question.

To this end, the achieved results are examined in Section 7.1 and some learned lessons are highlighted in Section 7.2. Section 7.3 provides possible avenues for future work.

7.1 Revisiting the Research Question

In order to conduct the work of this thesis, the research problem is divided into three research questions. The objective of the first research question is to investigate whether an ontology-based approach is capable of providing a generic map model ontology using different map formats for autonomous driving functions:

> RQ1: Can an ontology-based approach solve the map data integration problem and provide a generic and unified map model?

This research question is addressed in Chapter 4. We interpret the ontologies as a key part for map data integration. Thus, we adopt the Global as View (GAV) principle–an ontology-based data integration approach for integrating different low-level map data formats into a generic high-level map model. We propose a novel methodology that combines a top-down approach and a bottom-up approach for building the unified high-level ontology and the specific low-level ontologies iteratively with respect to fulfilling ontology development requirements. The methodology is applied for the construction of the two-level map ontologies. First, the high-level map ontology HLM covers static road knowledge and dynamic vehicle information. The generic concepts and spatial relationships of roads and lanes are

modelled in this ontology. This permits to provide a common representation of the road knowledge over different types of map data in various formats. Second, the low-level map ontologies, here LNDS and LHERE, covers NDS and HERE Live HD maps, respectively. The LNDS ontology provides the main concepts and relationships for capturing the NDS format to support the knowledge abstraction process and eventually populate the HLM ontology. The LHERE ontology is developed similarly. The simple transformation rules and complex transformations rules are used for transferring the LNDS ontology and the LHERE ontology to the HLM ontology respectively. Finally, we evaluate the methodology via three use cases. The results demonstrate that the two-level ontological maps and rule-based knowledge processing is capable of providing: 1) semantic integration among different maps; 2) reducing data size via layered abstraction, and 3) a generic map view over different map formats for autonomous vehicles. The flexible schema representation and query language are additional advantages that we observed in the proposed approach.

RQ2: Can an ontology-based approach perform efficient knowledge processing and spatial reasoning while the knowledge base is continuously changing?

In Chapter 5 this research question is positively answered by demonstrating that the ontology-based approach is capable of providing adequate road knowledge around the vehicle for answering autonomous driving functions (e.g., lane change guidance) while it's progressing along a route. We proposed a knowledge-spatial architecture to achieve the efficiency requirement. The knowledge dimension illustrates a knowledge abstraction process from the format-specific and detailed low-level ontologies to the generic high-level ontology. The spatial dimension describes the continuous spatial reasoning process with respect to the updated vehicle position and dynamic road environmental knowledge. We used a spatial sliding window and incremental reasoning to perform efficient knowledge updates and complex inferencing. To realize the knowledge-spatial architecture, we developed a prototype with decoupled two-level datastores in order to provide fast rule evaluation and query answering in the high-level datastore while several low-level datastores running in the background provide the needed high-level map knowledge ahead of the vehicle. The prototype is evaluated by using an NDS map to provide a high-level map view. The result shows that the ontology-based approach with rules can efficiently process knowledge and perform spatial reasoning. The explainability and flexibility of using rules are some of the advantages we observe in the proposed approach. Finally, we proposed two deployment choices of the designed solution, namely, a client and a client-server deployment. The related technical feature analysis such as memory consumption and data transmission time is presented. Both approaches are feasible for in-car usage. A concrete decision can then be made based on the specific requirements of the considered use cases.

RQ3: Can an ontology-based approach be utilized to ensure map data quality for road knowledge consistency?

In Chapter 6, this research question is addressed by proposing a workflow for detecting and fixing map data errors using ontologies and rules. The workflow consists of three stages, namely semantic enrichment, violation detection and violation handling. Semantic enrichment provides the needed spatial knowledge for violation detection and handling. The violation detection relies on the developed Map Quality Violation (MQV) ontology and a set of map data constraint rules. We identified the key OWL axioms to capture the map data integrity constraints. Since those constraints cannot be captured by OWL 2 RL, we modelled them using Datalog rules, providing the correspondence between the Datalog constraint atoms and OWL axioms. The violation handling strategies focused on the error caused by lane group ID duplication. We presented violation tolerance using graph aggregation and violation resolution using graph decomposition. The evaluation of the proposed approach focused on the performance of violation detection and the correctness of violation handling. The observed results indicate that the proposed approach can successfully check the quality of the map data and suggests that violation handling is even feasible on-the-fly using client deployment (see Section 5.5.1), avoiding the autonomous driving mode's deactivation. The extensibility and flexibility of using an ontology and rules to model constraints eases the process of constraint development, and with the fixed vocabularies used in constraints, an automatic violation rule generation can be achieved.

7.2 Lesson Learned

The adoption of semantic technologies in autonomous driving applications is still facing the challenge of bridging the gap between vehicles and semantics disciplines. We observe that, heretofore, the impact made by semantic technologies in vehicles is limited for industrial usage. Established productive autonomous driving solutions use traditional data-oriented approaches for the optimization of route planning, task planning, and manipulation problems. Through the work presented in this thesis, we pave the way for a productive application of semantic technologies to enhance the development of autonomous driving functions. For example, the ability to easily integrate various map formats for driving function realizations has always been a requested feature reported by engineers working on vehicle foresight. Human drivers understand the digital maps based on the real-world road. To enable the vehicle to have such cognition of the road, knowledge, and reasoning are required. Overall, we have observed the following practical findings from our study:

1. An ontology-based approach allows engineers to focus more on the analyses and design of autonomous functions on the knowledge level rather than on map data understanding and implementing various map processing components for similar maps.

2. Reasoning based on a rule-engine and an ontology has been applied in various scenarios. In the light of having a well-functioning and scalable autonomous driving scenario, using RDF triples, Datalog rules and SPARQL queries turn out to be a valid choice.

3. Rules need to be created and maintained in a modulated and decoupled fashion. The low-level semantic enrichment, low-to-high knowledge process, and the high-level spatial reasoning rules should be decoupled from each other, but aligned as modular components which can be extended or replaced, if, e.g., other or new map transformations are desired.

4. The RDF graph representation of the map data naturally aligns with the road (lane) network model, which is the underlying model of all HD maps. This provides the advantage of using existing graph techniques for map data analysis, using derived new spatial knowledge.

7.3 Outlook

In the development of this thesis, we focus on semantically representing the map standard NDS and the specific commercial HERE Live HD map and integrating them into the generic high-level map ontology. However, we are aware there exists a multitude of map formats that are available for autonomous driving systems. Nevertheless, the methodology presented in Chapter 4 for modelling the low-level ontologies and related transformation can be easily applied to other map formats in the domain.

Typically, there exists a need for combining sensor data and digital maps for vehicle perception, planning and control functions. To jointly use the different sources of data, the semantic meaning needs to be extracted and combined considering the application scenario. Thus, we envision to combine streaming reasoning for time-based sensor data with incremental reasoning over map data. For example, the lane change guidance can be refined based on the vehicle speed sensor data and the map data such that, more tactical vehicle driving manoeuvre with speed adjustments can be provided.

Furthermore, before the prototype goes to the production process, there are several important steps that need to be conducted. First, safety critical applications in cars need to have deterministic timing behaviour. The developed *SmartMap* prototype can be evaluated further for determinism and real-time performance. Second, the prototype has to become much more mature and improve from the scientific driven prototype to a stable solution. Finally, an infrastructure for rule management, in particular, for rule maintenance and reuse, needs to be provided as well.

Appendix A

Appendix

A.1 RDF Graph Representation of Lanes

In the following the RDF graph representations for the use case of the lane unified view (see Section 4.6.3) are presented. Figure A.1 shows the RDF graph representation of the aggregated high-level lane laneA1 over LNDS instances. The instance `high:LaneA1` has type `high:Lane`, data property `high:index` with value 1 and data property `high:length` with value 2008. Figure A.2 shows the corresponding RDF graph representation of the aggregated high-level lane laneB1 over LHERE instances. Similar to `high:LaneA1`, `high:LaneB1` is also an instance of the class `high:Lane` and has data property `high:index` with value 1 and data property `high:length` with value 2388. This shows that the high-level HLM ontology can unify and provide a generic lane representation using different low-level map ontologies.

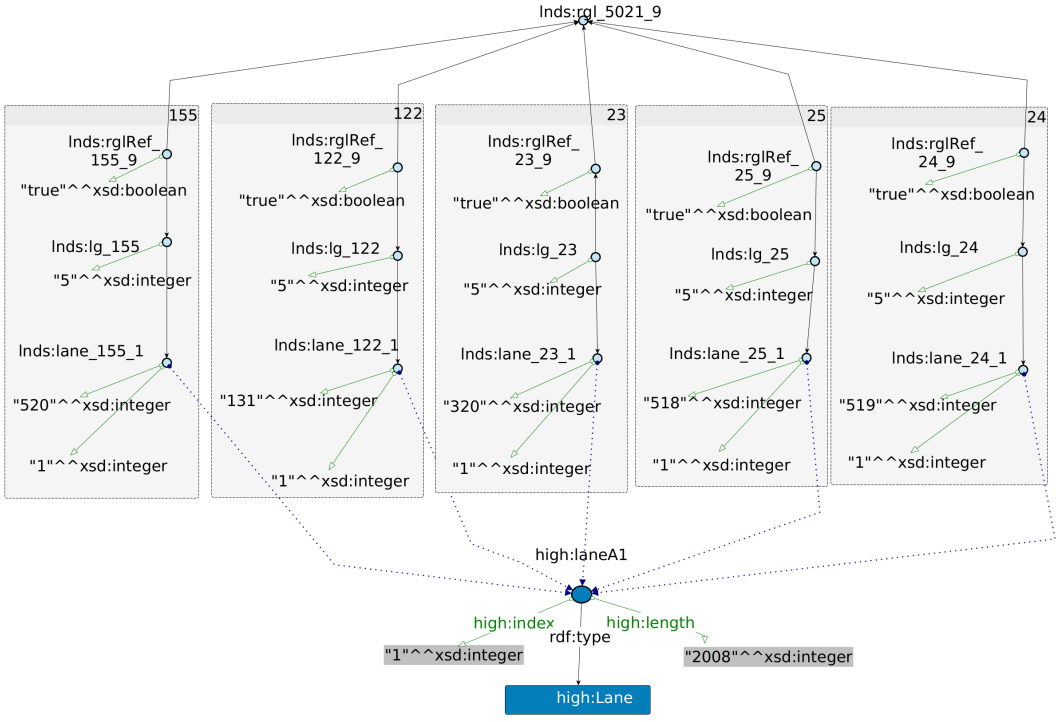


FIGURE A.1: The RDF graph representation of LNDS lane aggregation. The box with lane group ID contains the graph related to the aggregatable lane. We omitted the properties in the graph.

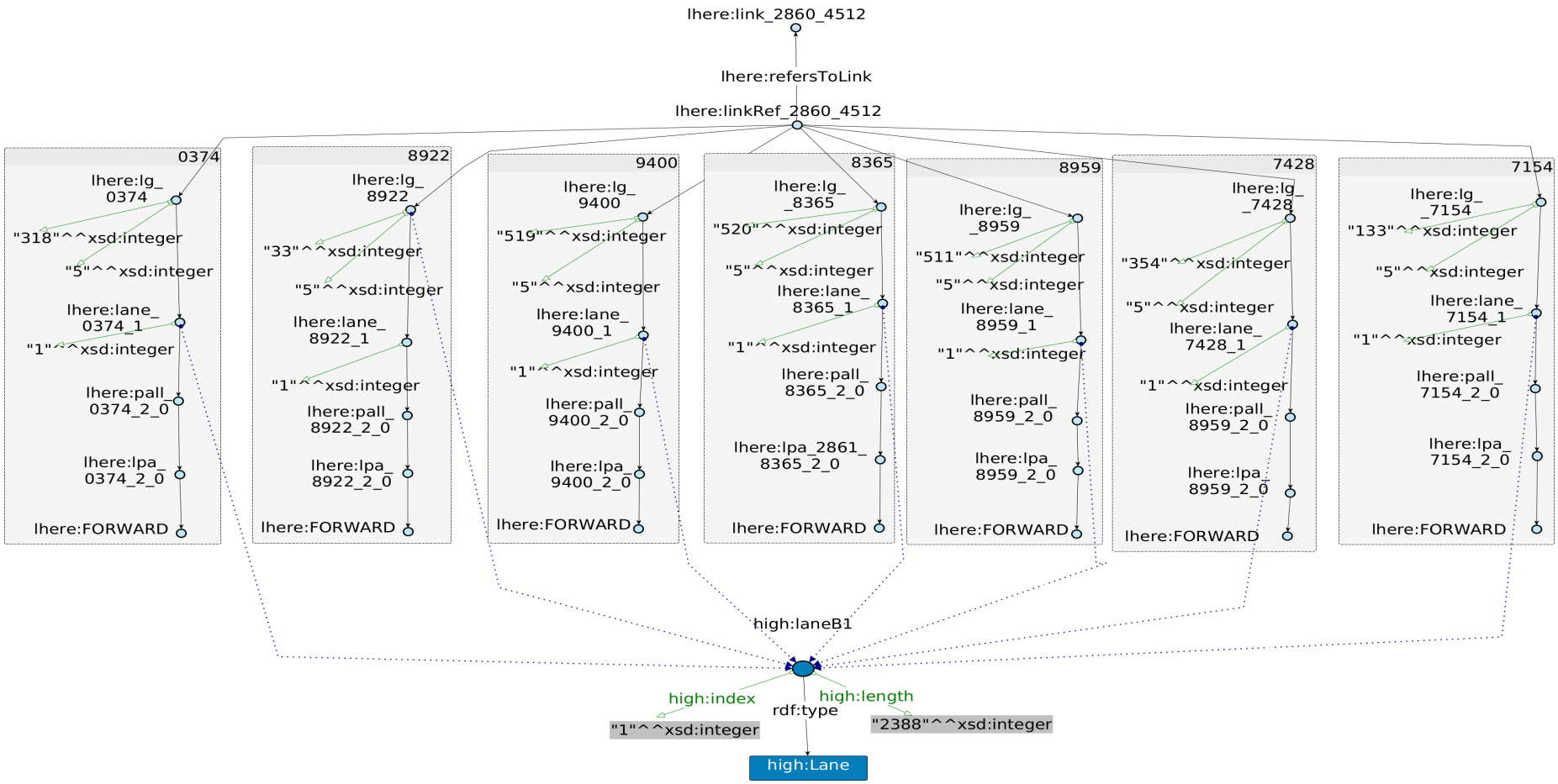


FIGURE A.2: The RDF graph representation of LHERE lane aggregation. The box with last four digits lane group ID contains the graph related to the aggregatable lane. We omitted the properties in the graph.

A.2 Example of Imperative and Declarative Programs

Imperative programming requires developers to define step by step how code should be executed. For example, algorithm 5 shows the typical steps used to output connected lanes using sort-merge algorithm. It is executed at the map data structure level. In other words, the algorithm need to be re-implemented for every map formats whose schema is different from each other.

input : R: a list containing lane destination connection ID,
S: a list containing lane source connection ID
output: C: list of the pair of two connected lanes

```
1  if R and S are not sorted then
2      sort R and S;
3  i ← 1; j ← 1;
4  while i ≤ |R| ∧ j ≤ |S| do
5      if R[i] = S[j] then
6          C ← add(R[i], S[j])
7      else if R[i] > S[j] then
8          j ← j + 1
9      else if R[i] < S[j] then
10         i ← i + 1
```

Algorithm 5: Sort-merge algorithm used for lane connectivity.

With knowledge-based programming, the rules and ontologies are used to infer the desired knowledge instead of focusing on processing map data step-wise depending on specific map formats. For instance, one single rule can achieve the same goal as the Algorithm 5 does (see Listing A.1), without understand how map data is structured. Notably, the rule directly reflects the human reasoning process of lane connectivity as *if one lane's destination ID is equal to another lane's source ID, then there are in the successor relationship*. Rule-based programming may also axiomatise recursive qualities like lane reachability with only two rules (see Listing A.2), but imperative programming is more complicated to achieve. Hence, knowledge-enabled programming can help to separate knowledge from the data processing, and enable programmers to write high-level functions.

```
hasDirectNextLane[?r,?s]:-
Lane[?r], hasDestination[?r,?id],
Lane[?s], hasSource[?s,?id].
```

LISTING A.1: Lane connectivity rule

```
hasNextLane[?r,?s]:- hasDirectNextLane[?r,?s].

hasNextLane[?r,?z]:- hasDirectNextLane[?r,?s], hasNextLane[?s,?z].
```

LISTING A.2: Lane reachability rules

A.3 Correspondence of Datalog Rules and SPARQL Queries

Datalog rules are translated into SPARQL queries to ensure the efficiency of parallel processing due to the evaluation mechanism of the underlying data store. That is

rule evaluations in different data stores compete for computation resources while the SPARQL queries are evaluated independently. However, there is no one-to-one correspondence between Datalog rules and SPARQL queries. For example, the recursion can be expressed in Datalog but not in SPARQL; the `OPTIONAL` of SPARQL can not be expressed in Datalog (see Section 2.3.4 for details).

A.3.1 Primitive Attribute Rules

(1) Primitive relationship rules infer relationships between individuals.

Datalog: $\mathsf{op}(x,y) \leftarrow \mathsf{C}_1(x),\ \mathsf{dp}(x,z),\ \mathsf{C}_2(y),\ \mathsf{dp}(y,z).$

SPARQL:
```
INSERT {?x :op ?y}
WHERE { ?x a :C1; :dp ?z.?y a :C2; :dp ?z.}
```

(2) Primitive attribute rules infer an attribute of an individual.

Datalog: $\mathsf{dp}_2(x,v) \leftarrow \mathsf{C}(x),\ \mathsf{dp}_1(x,z),\ \mathsf{dp}_2(y,v),\ \mathsf{dp}_1(y,z).$

SPARQL:
```
INSERT {?x :dp2 ?v}
WHERE {?x a :C; :dp1 ?z. ?y :dp2  ?v; :?dp1 ?z.}
```

A.3.2 Transfer Rules

(1) Simple rules create, for each individual of a certain type in one ontology, a new individual of a certain type in another ontology.

Datalog: $\mathsf{C}_2(n),\ \mathsf{op}_2(n,u),\ \mathsf{dp}_2(n,v) \leftarrow$
$\mathsf{C}_1(x),\ \mathsf{op}_1(x,u), \mathsf{dp}_1(x,v),\ \mathtt{BIND}(\mathtt{SKOLEM}(\text{"}c\text{"},\ x)\ \mathtt{AS}\ n).$

SPARQL:
```
INSERT {?n a :C2; :op2 ?u; :dp2 ?v}
WHERE {?x a :C1; :op1 ?u; :dp1 ?v.
       BIND(SKOLEM("c",?x) AS ?n)
       }
```

(2) Complex rules create a new individual in one ontology based on several individuals in another ontology.

Datalog: $\mathsf{C}_2(w) \leftarrow \mathsf{C}_1(x),\ \mathsf{op}_1(x,z),\ \mathsf{dp}_1(x,v),$
$\mathsf{C}_1(y),\ \mathsf{op}_1(y,z),\ \mathsf{dp}_1(y,v),\ \mathtt{BIND}(\mathtt{SKOLEM}(\text{"}c\text{"},\ x,\ y)\ \mathtt{AS}\ w).$

SPARQL:
```
INSERT {?w a :C2.}
WHERE {?x a :C1; :op1 ?z; :dp1 ?v.
       ?y a :C1; :op1 ?z; :dp1 ?v.
       BIND(SKOLEM("c",?x,?y) AS ?w)
       }
```

A.3.3 Spatial Rules

(1) Bounding rules infer the boundaries of an area or the range of a line, such as a start/end point or the left or right-most lane. Aggregation functions (e.g., `MIN` or `MAX`) can be used to identify an individual with a minimal or maximal bounding value.

Datalog: $\mathsf{C}_2(z) \leftarrow \mathsf{C}_1(x)$, `AGGREGATE`$(\mathsf{dp}_1(x,v)$ `ON` x
`BIND MAX`(v) `AS` $m)$, $\mathsf{C}_1(z)$, $\mathsf{dp}_1(z,m)$.

SPARQL:
```
INSERT {?x a :C2.}
WHERE {?x a :C1; :dp1 ?v.}
ORDER BY DESC(?v) LIMIT 1
```

Such rules might also use (stratified) negation to identify individuals without some properties:

Datalog: $\mathsf{C}_2(x) \leftarrow \mathsf{C}_1(x)$, `NOT EXISTS` y `IN` $(\mathsf{C}_1(y)$, $\mathsf{op}_1(x,\ y))$.

SPARQL:
```
INSERT {?x a :C2.}
WHERE {?x a :C1.
       FILTER NOT EXISTS {?y a :C1. ?x :op1 ?y}
       }
```

(2) Topological rules refer to topological relations, more specifically, lateral (left-/right) and longitudinal (predecessor/successor) relations. Reachability can naturally be expressed using recursive rules.

Datalog: $\mathsf{op}_1(x,y) \leftarrow \mathsf{op}_2(x,y)$
$\mathsf{op}_1(x,z) \leftarrow \mathsf{op}_2(x,y)$, $\mathsf{op}_1(y,z)$

SPARQL: No corresponding SPARQL recursive queries.

(3) Distance rules refer to the spatial arrangement of objects, such as the distance to a point of interest.

Coordinate distance rules indicate the distance between two points using coordinates:

Datalog:
$$\begin{aligned}&\mathsf{A}(d),\ \mathsf{hasSource}(d,s),\ \mathsf{hasTarget}(d,t),\ \mathsf{distance}(d,z) \leftarrow\\ &\quad \mathsf{C}_1(s),\ \mathsf{x}(s,x_s),\ \mathsf{y}(s,y_s),\ \mathsf{C}_1(t),\ \mathsf{x}(t,x_t),\ \mathsf{y}(t,y_t),\\ &\quad \texttt{BIND}(\texttt{sqrt}((x_s-x_t)^2+(y_s-y_t)^2)\ \texttt{AS}\ z),\\ &\quad \texttt{BIND}(\texttt{SKOLEM}(\texttt{"}d\texttt{"},\ s,\ t)\ \texttt{AS}\ d).\end{aligned}$$

SPARQL:
```
INSERT {?d a :A; :hasSource ?s;
        :hasTarget ?t; :distance ?z.
       }
WHERE { ?s a :C1; :x ?xs;:y ?ys.
        ?t a :C1; :x ?xt;:y ?yt.
        BIND(sqrt((?xs-?xt)*(?xs-?xt)
             +(?ys-?yt)*(?ys-?yt)) AS ?z)
        BIND(SKOLEM("d",?s,?t) AS ?d)
      }
```

Length distance rules refer to the spatial arrangement of objects, such as the distance to a point of interest.

Datalog:
$$\begin{aligned}&\mathsf{A}(d),\ \mathsf{hasSource}(d,s),\ \mathsf{hasTarget}(d,t),\ \mathsf{distance}(d,z) \leftarrow\\ &\quad \mathsf{C}_1(s),\ \mathsf{length}(s,v),\\ &\quad \texttt{AGGREGATE}(\mathsf{op}_1(s,p),\ \mathsf{op}_1(p,t)),\ \mathsf{length}(p,l)\ \texttt{ON}\ s\\ &\quad \texttt{BIND SUM}(l)\ \texttt{AS}\ u),\ \texttt{BIND}((v+u)\ \texttt{AS}\ z),\\ &\quad \texttt{BIND}(\texttt{SKOLEM}(\texttt{"}f\texttt{"},\ s,\ t)\ \texttt{AS}\ d).\end{aligned}$$

SPARQL:
```
INSERT {?d a :A; :hasSource ?s;
        :hasTarget ?t; :distance ?z.
       }
WHERE {?s a :C1; :length ?v.
        {
          SELECT ?s (SUM(?l) AS ?u)
          WHERE{?s :op1 ?p. ?p :op1 ?t. ?p :length ?l
          } GROUP BY ?s
        }
        BIND ((?v+?u) AS ?z)
        BIND(SKOLEM("f", ?s,?t) AS ?d)
      }
```

Bibliography

[1] Us9726505b2 - methods and systems for generating a horizon for use in advanced driver assistance system (adas) - google patents. https://patents.google.com/patent/US9726505. (Accessed on 04/11/2021).

[2] Openstreetmap. https://www.openstreetmap.org/#map=18/48.25292/11.53241, . (Accessed on 05/12/2021).

[3] Openstreetmap. https://www.openstreetmap.org/#map=15/48.2897/11.5681, . (Accessed on 06/21/2021).

[4] Swi-prolog for the (semantic) web. https://www.swi-prolog.org/web/index.html. (Accessed on 06/24/2021).

[5] Ieee standard for developing software life cycle processes. *IEEE Std 1074-1995*, pages 1–106, 1996. doi: 10.1109/IEEESTD.1996.79663.

[6] Bim – new rules of measurement ontology for construction cost estimation. *Engineering Science and Technology, an International Journal*, 20(2):443–459, 2017. ISSN 2215-0986. doi: https://doi.org/10.1016/j.jestch.2017.01.007.

[7] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. Sw-store: a vertically partitioned dbms for semantic web data management. *The VLDB Journal*, 18 (2):385–406, 2009.

[8] A. Abdalla, Y. Hu, D. Carral, N. Li, and K. Janowicz. An ontology design pattern for activity reasoning. In *WOP*, pages 78–81, 2014.

[9] ABI Research. The future of maps:technologies, processes, and ecosystem. https://www.here.com/sites/g/files/odxslz166/files/2019-10/CR-HERE-108%20v1_1.pdf, 2018. (Accessed on 06/21/2021).

[10] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*, volume 8. Addison-Wesley, 1995.

[11] T. Ai, M. Yang, X. Zhang, and J. Tian. Detection and correction of inconsistencies between river networks and contour data by spatial constraint knowledge. *Cartography and Geographic Information Science*, 42(1):79–93, 2015.

[12] W. Ali, M. Saleem, B. Yao, A. Hogan, and A.-C. N. Ngomo. A survey of rdf stores & sparql engines for querying knowledge graphs. *arXiv preprint arXiv:2102.13027*, 2021.

[13] R. Angles and C. Gutierrez. The expressive power of sparql. In *International Semantic Web Conference*, pages 114–129. Springer, 2008.

[14] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. Ep-sparql: a unified language for event processing and stream reasoning. In *Proceedings of the 20th international conference on World wide web*, pages 635–644, 2011.

[15] ArcGIS. Arcgis data reviewer | data quality management. https://www.esri.com/en-us/arcgis/products/arcgis-data-reviewer/overview, 2020. (Accessed on 08/13/2020).

[16] M. Arenas, C. Gutierrez, and J. Pérez. Foundations of rdf databases. In *Reasoning Web International Summer School*, pages 158–204. Springer, 2009.

[17] A. Armand, D. Filliat, and J. Ibañez-Guzman. Ontology-based context awareness for driving assistance systems. In *2014 IEEE intelligent vehicles symposium proceedings*, pages 227–233. IEEE, 2014.

[18] A. Armand, J. Ibanez-Guzman, and C. Zinoune. *Digital Maps for Driving Assistance Systems and Autonomous Driving*, pages 201–244. Springer, 2017. ISBN 978-3-319-31895-0.

[19] A. Armand, J. Ibanez-Guzman, and C. Zinoune. Digital maps for driving assistance systems and autonomous driving. In *Automated Driving*, pages 201–244. Springer, 2017.

[20] S. Arumugam, I. Hamid, and V. Abraham. Decomposition of graphs into paths and cycles. *Journal of Discrete Mathematics*, 2013, 2013.

[21] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003. ISBN 0-521-78176-0.

[22] R. Bader. *Proactive Recommender Systems in Automotive Scenarios*. PhD thesis, Technische Universität München, 2013.

[23] M. Baglioni, M. V. Masserotti, C. Renso, and L. Spinsanti. Building geospatial ontologies from geographical databases. In *International Conference on GeoSpatial Sematics*, pages 195–209. Springer, 2007.

[24] G. Bagschik, T. Menzel, and M. Maurer. Ontology based scene creation for the development of automated vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1813–1820. IEEE, 2018.

[25] D. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. Stream reasoning: Where we got so far. In *NeFoRS 2010: 4th International Workshop on New Forms of Reasoning for the Semantic Web: Scalable and Dynamic*, 2010.

[26] D. F. Barbieri, D. Braga, S. Ceri, E. D. VALLE, and M. Grossniklaus. C-sparql: a continuous query language for rdf data streams. *International Journal of Semantic Computing*, 4(01):3–25, 2010.

[27] M. Beetz. Cognition-enabled autonomous robot control for the realization of home chore task intelligence. In P. van Emde Boas, F. C. A. Groen, G. F. Italiano, J. Nawrocki, and H. Sack, editors, *SOFSEM 2013: Theory and Practice of Computer Science*, pages 106–106, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-35843-2.

[28] R. Behrens, T. Kleine-Besten, W. Pöchmüller, and A. Engelsberg. *Digitale Karten im Navigation Data Standard Format*, pages 513–523. Springer Fachmedien Wiesbaden, Wiesbaden, 2015. ISBN 978-3-658-05734-3. doi: 10.1007/978-3-658-05734-3_27. URL https://doi.org/10.1007/978-3-658-05734-3_27.

[29] P. Bellini, I. Bruno, A. Cavaliere, D. Cenni, M. DiClaudio, G. Martelli, S. Menabeni, P. Nesi, G. Pantaleo, and N. Rauch. Km4city: Smart city ontology and tools for city knowledge exploitation. In *European Data Forum*, 2015.

[30] D. Bétaille and R. Toledo-Moreo. Creating enhanced maps for lane-level vehicle navigation. *IEEE Transactions on Intelligent Transportation Systems*, 11(4): 786–798, 2010.

[31] E. Blomqvist, P. Hitzler, K. Janowicz, A. Krisnadhi, T. Narock, and M. Solanki. Considerations regarding ontology design patterns. *Semantic Web*, 7(1):1–7, 2016.

[32] I. Bratko. *Prolog programming for artificial intelligence*. Pearson education, 2001.

[33] L. Bravo and M. A. Rodriguez. Formalization and reasoning about spatial semantic integrity constraints. *Data & Knowledge Engineering*, 72:63–82, 2012.

[34] D. Buckley, C. F. Canada, C. Reid, and Associates. *The GIS Primer: An Introduction to Geographical Information Systems*. Forestry Canada, 1990. URL `https://books.google.de/books?id=zrUItwAACAAJ`.

[35] M. Buechel, G. Hinz, F. Ruehl, H. Schroth, C. Gyoeri, and A. Knoll. Ontology-based traffic scene modeling, traffic regulations dependent situational awareness and decision-making for automated vehicles. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1471–1476. IEEE, 2017.

[36] D. Burgstahler, C. Peusens, D. Böhnstedt, and R. Steinmetz. Horizon. kom: A first step towards an open vehicular horizon provider. In *VEHITS*, pages 79–84, 2016.

[37] J.-P. Calbimonte, O. Corcho, and A. J. Gray. Enabling ontology-based access to streaming data sources. In *International Semantic Web Conference*, pages 96–111. Springer, 2010.

[38] R. F. Calhau and R. d. A. Falbo. An ontology-based approach for semantic integration. In *2010 14th IEEE International Enterprise Distributed Object Computing Conference*, pages 111–120, 2010. doi: 10.1109/EDOC.2010.32.

[39] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The dl-lite family. *Journal of Automated reasoning*, 39(3):385–429, 2007.

[40] O. Carsten, K. Kircher, and S. Jamson. Vehicle-based studies of driving in the real world: The hard truth? *Accident Analysis & Prevention*, 58:162–174, 2013.

[41] B. Cavazza, R. Gandia, F. Antonialli, A. Zambalde, I. Nicolaï, J. Sugano, and A. Neto. Management and business of autonomous vehicles: A systematic integrative bibliographic review. *International Journal of Automotive Technology and Management*, 19:31, 01 2019. doi: 10.1504/IJATM.2019.098509.

[42] M. Cheatham and C. Pesquita. *Semantic Data Integration*, pages 263–305. Springer International Publishing, Cham, 2017.

[43] W. Chen and L. Kloul. An ontology-based approach to generate the advanced driver assistance use cases of highway traffic. In *10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management.*, 2018.

[44] T. Chiang. Mapping the future: High-definition map generation for self-driving cars. 2018.

[45] M. Codescu, G. Horsinka, O. Kutz, T. Mossakowski, and R. Rau. Osmonto-an ontology of openstreetmap tags. *State of the map Europe (SOTM-EU)*, 2011, 2011.

[46] E. Commission. European commission, care project: Road safety evolution in the eu. `https://ec.europa.eu/transport/road_safety/sites/default/files/pdf/observatory/historical_evol.pdf`. (Accessed on 06/21/2021).

[47] O. Corcho, M. Fernández-López, and A. Gómez-Pérez. Methodologies, tools and languages for building ontologies. where is their meeting point? *Data & knowledge engineering*, 46(1):41–64, 2003.

[48] I. F. Cruz and H. Xiao. The role of ontologies in data integration. *JOURNAL OF ENGINEERING INTELLIGENT SYSTEMS*, 13:245–252, 2005.

[49] I. F. Cruz, H. Xiao, et al. The role of ontologies in data integration. *Engineering intelligent systems for electrical engineering and communications*, 13(4):245, 2005.

[50] R. Cyganiak, D. Wood, and M. Lanthaler. RDF 1.1 Concepts and Abstract Syntax. `https://www.w3.org/TR/rdf11-concepts/`, 2014. (Accessed on 09/14/2020).

[51] B. Daniel et al. Remotehorizon. kom: Dynamic cloud-based ehorizon. *Automotive meets Electronics (AmE 2016)*, 2016.

[52] G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati. Using ontologies for semantic data integration. In *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years*, pages 187–202. Springer, 2018.

[53] B. De Meester, P. Heyvaert, D. Arndt, A. Dimou, and R. Verborgh. Rdf graph validation using rule-based reasoning. *Semantic Web Journal*, 2020.

[54] J. Debattista, C. Lange, and S. Auer. daq, an ontology for dataset quality information. In *LDOW*, 2014.

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

[56] R. Devillers and R. Jeansoulin. *Fundamentals of spatial data quality*. ISTE Publishing Company, 2006.

[57] Directorate-General for Defence Industry and Space (European Commission). Study on the integrity and reliability of high definition (hd) maps for connected and automated driving (cad) - publications office of the eu. `https://op.europa.eu/en/publication-detail/-/publication/a16e08f5-d5f8-11ea-adf7-01aa75ed71a1/language-en`. (Accessed on 05/26/2021).

[58] V. Dubinin, V. Vyatkin, C.-W. Yang, and C. Pang. Automatic generation of automation applications based on ontology transformations. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–4. IEEE, 2014.

[59] S. Durekovic, A. BRACHT, B. RAICHLE, M. RAUCH, J. REQUEJO, D. TOROPOV, and A. VARCHMIN. Adasis v2 protocol. *Bosch Engineering GmbH Internal, BEG-VS/EBI1*, 2012.

[60] M. J. Egenhofer and J. Herring. Categorizing binary topological relations between regions, lines, and points in geographic databases. *The*, 9(94-1):76, 1990.

[61] B. Eine, M. Jurisch, and W. Quint. Ontology-based big data management. *Systems*, 5(3):45, 2017.

[62] T. Eiter, J. Z. Pan, P. Schneider, M. Šimkus, and G. Xiao. A rule-based framework for creating instance data from openstreetmap. In B. ten Cate and A. Mileo, editors, *Web Reasoning and Rule Systems*, pages 93–104, Cham, 2015. Springer International Publishing.

[63] T. Eiter, H. Füreder, F. Kasslatter, J. X. Parreira, and P. Schneider. Towards a semantically enriched local dynamic map. *International Journal of Intelligent Transportation Systems Research*, 17(1):32–48, 2019.

[64] F. Ekaputra, M. Sabou, E. Serral Asensio, E. Kiesling, and S. Biffl. Ontology-based data integration in multi-disciplinary engineering environments: A review. *Open Journal of Information Systems*, 4(1):1–26, 2017.

[65] F. J. Ekaputra, M. Sabou, E. Serral, and S. Biffl. Knowledge change management and analysis during the engineering of cyber physical production systems: A use case of hydro power plants. In *Proceedings of the 12th International Conference on Semantic Systems*, SEMANTiCS 2016, page 105–112, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450347525. doi: 10.1145/2993318.2993325.

[66] B. El Asmar, S. Chelly, N. Azzi, L. Nassif, J. El Asmar, and M. Färber. Aware: A situational awareness framework for facilitating adaptive behavior of autonomous vehicles in manufacturing. In *International Semantic Web Conference*, pages 651–666. Springer, 2020.

[67] M. R. Endsley. Measurement of situation awareness in dynamic systems. *Human factors*, 37(1):65–84, 1995.

[68] S. Esswein, S. Goasguen, C. Post, J. Hallstrom, D. White, and G. Eidson. Towards ontology-based data quality inference in large-scale sensor networks. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 898–903. IEEE, 2012.

[69] T. ETSI. Intelligent transport systems (its); vehicular communications; basic set of applications; definitions. *Tech. Rep. ETSI TR 102 6382009*, 2009.

[70] W. FABER, G. PFEIFER, N. LEONE, T. DELL'ARMI, and G. IELPA. Design and implementation of aggregate functions in the dlv system. *Theory and Practice of Logic Programming*, 8(5-6):545–580, 2008. doi: 10.1017/S1471068408003323.

[71] R. Falco, A. Gangemi, S. Peroni, D. Shotton, and F. Vitali. *Modelling OWL Ontologies with Graffoo*, pages 320–325. Springer International Publishing, Cham, 2014. doi: 10.1007/978-3-319-11955-7_42.

[72] M. Fernández-López and A. Gómez-Pérez. Overview and analysis of methodologies for building ontologies. *The knowledge engineering review*, 17(2):129, 2002.

[73] C. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem artificial intelligence 19 (1): 17-37, 1982.

[74] C. Fürber and M. Hepp. Towards a vocabulary for data quality management in semantic web architectures. In *Proceedings of the 1st International Workshop on Linked Web Data Management*, pages 1–8, 2011.

[75] M. Galvani. History and future of driver assistance. *IEEE Instrumentation Measurement Magazine*, 22(1):11–16, 2019. doi: 10.1109/MIM.2019.8633345.

[76] R. Gayathri and V. Uma. Ontology based knowledge representation technique, domain modeling languages and planners for robotic path planning: A survey. *ICT Express*, 4(2):69 – 74, 2018. ISSN 2405-9595. SI on Artificial Intelligence and Machine Learning.

[77] S. Germano, T.-L. Pham, and A. Mileo. Web stream reasoning in practice: on the expressivity vs. scalability tradeoff. In *International Conference on Web Reasoning and Rule Systems*, pages 105–112. Springer, 2015.

[78] S. Geyer, M. Baltzer, B. Franz, S. Hakuli, M. Kauer, M. Kienle, S. Meier, T. Weißgerber, K. Bengler, R. Bruder, et al. Concept and development of a unified ontology for generating test and use-case catalogues for assisted and automated vehicle guidance. *IET Intelligent Transport Systems*, 8(3):183–189, 2013.

[79] G. Gottlob, G. Orsi, A. Pieris, and M. Simkus. Datalog and its extensions for the semantic web. In T. Eiter and T. Krennwallner, editors, *Reasoning Web 2012*, pages 54–77. 2012. doi: 10.1007/978-3-642-33158-9_2. URL `http://link.springer.com/chapter/10.1007%2F978-3-642-33158-9_2`.

[80] GraphDB. Reasoning — graphdb se 9.3 documentation. `http://graphdb.ontotext.com/documentation/standard/reasoning.html#consistency-checks`, 2020. (Accessed on 08/09/2020).

[81] S. Greco and C. Molinaro. Datalog and logic databases. *Synthesis Lectures on Data Management*, 7(2):31–32, 2015.

[82] S. Greco and C. Molinaro. Datalog and logic databases. *Synthesis Lectures on Data Management*, 7(2):36, 2015.

[83] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proceedings of the 12th international conference on World Wide Web*, pages 48–57, 2003.

[84] D. Gruyer, R. Belaroussi, and M. Revilloud. Accurate lateral positioning from map data and road marking detection. *Expert Systems with Applications*, 43:1–8, 2016.

[85] N. Guarino. *Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 1st edition, 1998.

[86] R. V. Guha and D. Brickley. RDF vocabulary description language 1.0: RDF schema, 2004. URL `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`.

[87] G.-P. Gwon, W.-S. Hur, S.-W. Kim, and S.-W. Seo. Generation of a precise and efficient lane-level road map for intelligent vehicle systems. *IEEE Transactions on Vehicular Technology*, 66(6):4517–4533, 2016.

[88] V. Haarslev, K. Hidde, R. Möller, and M. Wessel. The racerpro knowledge representation and reasoning system. *Semantic Web*, 3(3):267–277, 2012.

[89] K. Hammar. *Content ontology design patterns: Qualities, methods, and tools*, volume 1879. Linköping University Electronic Press, 2017.

[90] HERE. Here_hd_live_map_one_pager.pdf. `https://www.here.com/sites/g/files/odxslz166/files/2018-11/HERE_HD_Live_Map_one_pager.pdf`. (Accessed on 03/21/2021).

[91] P. Heyvaert, A. Dimou, B. De Meester, and R. Verborgh. Rule-driven inconsistency resolution for knowledge graph generation. *Semantic Web Journal*, pages 1071–1086, 2019.

[92] P. Hitzler, M. Krötzsch, B. Parsia, P. Patel-Schneider, and S. Rudolph. Owl 2 web ontology language primer (second edition). `https://www.w3.org/TR/owl2-primer/#Enumeration_of_Individuals`. (Accessed on 05/10/2021).

[93] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph. OWL 2 Web Ontology Language: Primer (2nd Edition). `http://www.w3.org/TR/owl2-primer/`, 27 Oct 2009.

[94] I. Horrocks, P. F. PatelSchneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. Swrl: A semantic web rule language combining owl and ruleml. w3c member submission, 21 may 2004, 2004. URL `https://www.w3.org/Submission/2004/SUBM-SWRL-20040521/`.

[95] I. Horrocks, A. Fokoue, B. C. Grau, B. Motik, and Z. Wu. OWL 2 web ontology language profiles (second edition), 2012. URL `http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/`.

[96] W. Huang. Knowledge-based geospatial data integration and visualization with semantic web technologies. In *2019 Doctoral Consortium at the 18th International Semantic Web Conference, ISWC-DC 2019*, volume 2548, pages 37–45. CEUR-WS, 2019.

[97] Z. Huang, S. Qiao, N. Han, C.-a. Yuan, X. Song, and Y. Xiao. Survey on vehicle map matching techniques. *CAAI Transactions on Intelligence Technology*, 6(1):55–71, 2021. doi: https://doi.org/10.1049/cit2.12030. URL `https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cit2.12030`.

[98] M. Hülsen, J. M. Zöllner, N. Haeberlen, and C. Weiss. Asynchronous real-time framework for knowledge-based intersection assistance. In *14th Int. IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1680–1685, 2011.

[99] M. Hülsen, J. M. Zöllner, and C. Weiss. Traffic intersection situation description ontology for advanced driver assistance. In *IEEE Intelligent Vehicles Symposium*, pages 993–999, 2011.

[100] B. Hummel, W. Thiemann, and I. Lulcheva. Scene understanding of urban road intersections with description logic. In *Dagstuhl Seminar Proc.*, 2008.

[101] V. Ilci and C. Toth. High definition 3d map creation using gnss/imu/lidar sensor integration to support autonomous vehicle navigation. *Sensors*, 20(3): 899, 2020.

[102] M. Imran and R. I. Young. Reference ontologies for interoperability across multiple assembly systems. *International Journal of Production Research*, 54(18): 5381–5403, 2016.

[103] S. international. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles (j3016_201806). *SAE International*, 2018.

[104] ISO/TC 204. Intelligent transport systems-Local dynamic map. `https://www.iso.org/obp/ui/#iso:std:69433:en`, 2018. (Accessed on 05/27/2020).

[105] R. Jasper and M. Uschold. A framework for understanding and classifying ontology applications. In *Proceedings 12th Int. Workshop on Knowledge Acquisition, Modelling, and Management KAW*, volume 99, pages 16–21, 1999.

[106] I. Kalai, M. Siliski, J. Morrison, K. Ito, and A. T. Miller. Pre-fetching map tile data along a route, Aug. 12 2014. US Patent 8,803,920.

[107] T. Karbe. Crystal deliverables-state of the art for automotive ontology.pdf. `https://www.crystal-artemis.eu/fileadmin/user_upload/Deliverables/CRYSTAL_D_308_010_v1.2.pdf`. (Accessed on 06/23/2021).

[108] M. Katsumi. Vehicle ontology. `https://enterpriseintegrationlab.github.io/icity/Vehicle/doc/index-en.html`. (Accessed on 04/19/2021).

[109] M. Katsumi and M. Fox. Ontologies for transportation research: A survey. *Transportation Research Part C: Emerging Technologies*, 89:53 – 82, 2018. ISSN 0968-090X. doi: https://doi.org/10.1016/j.trc.2018.01.023.

[110] B. Ketsman and C. Koch. Datalog with negation and monotonicity. In *23rd International Conference on Database Theory (ICDT 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[111] E. Kharlamov, B. C. Grau, E. Jiménez-Ruiz, S. Lamparter, G. Mehdi, M. Ringsquandl, Y. Nenov, S. Grimm, M. Roshchin, and I. Horrocks. Capturing industrial information models with ontologies and constraints. In *International Semantic Web Conference*, pages 325–343. Springer, 2016.

[112] R. Kohlhaas, T. Bittner, T. Schamm, and J. M. Zöllner. Semantic state space for high-level maneuver planning in structured traffic scenes. *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1060–1065, 2014.

[113] S. Komazec, D. Cerri, and D. Fensel. Sparkwave: continuous schema-enhanced pattern matching over rdf data streams. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, pages 58–68, 2012.

[114] O. Kovalenko and J. Euzenat. Semantic matching of engineering data structures. In *Semantic web technologies for intelligent engineering applications*, pages 137–157. Springer, 2016.

[115] P. Langley, J. E. Laird, and S. Rogers. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, 10(2):141–160, 2009.

[116] 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 *International Semantic Web Conference*, pages 370–388. Springer, 2011.

[117] T. B. LEE. Waymo finally launches an actual public, driverless taxi service | ars technica. `https://arstechnica.com/cars/2020/10/waymo-finally-launches-an-actual-public-driverless-taxi-service/`. (Accessed on 01/15/2021).

[118] J. R. Lewis and J. Sauro. *The Factor Structure of the System Usability Scale*, pages 94–103. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. doi: 10.1007/978-3-642-02806-9_12.

[119] Y. Li, J. Tao, and F. Wotawa. Ontology-based test generation for automated and autonomous driving functions. *Information and software technology*, 117: 106200, 2020.

[120] H.-K. Lin and J. A. Harding. A manufacturing system engineering ontology model on the semantic web for inter-enterprise collaboration. *Computers in Industry*, 58(5):428–437, 2007.

[121] R. Liu, J. Wang, and B. Zhang. High definition map for automated driving: Overview and analysis. *Journal of Navigation*, 73(2):324–341, 2020.

[122] Y. Liu, X. Wang, L. Li, S. Cheng, and Z. Chen. A novel lane change decision-making model of autonomous vehicle based on support vector machine. *IEEE Access*, 7:26543–26550, 2019. doi: 10.1109/ACCESS.2019.2900416.

[123] M. F. Lopez, A. G. Perez, and N. Juristo. METHONTOLOGY: from Ontological Art towards Ontological Engineering. In *Proceedings of the AAAI97 Spring Symposium*, pages 33–40, Stanford, USA, 1997.

[124] M. F. López, A. Gómez-Pérez, J. P. Sierra, and A. P. Sierra. Building a chemical ontology using methontology and the ontology design environment. *IEEE Intelligent Systems and their applications*, 14(1):37–46, 1999.

[125] B. Lorenz, H. J. Ohlbach, and L. Yang. Ontology of transportation networks, 2005.

[126] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang. Map-matching for low-sampling-rate gps trajectories. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 352–361, 2009.

[127] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang. Proceedings of the 17th acm sigspatial international conference on advances in geographic information systems. 2009.

[128] F. Maali, S. Campinas, and S. Decker. Gagg: A graph aggregation operator. In *The Semantic Web. Latest Advances and New Domains*, pages 491–504. Springer, 2015.

[129] I. Majic, E. Naghizade, S. Winter, and M. Tomko. Discovery of topological constraints on spatial object classes using a refined topological model. *Journal of Spatial Information Science*, 2019(18):1–30, 2019.

[130] S. Mäs. Reasoning on spatial semantic integrity constraints. In *International Conference on Spatial Information Theory*, pages 285–302. Springer, 2007.

[131] MathWorks. Here hd live map layers - matlab & simulink - mathworks deutschland. `https://de.mathworks.com/help/driving/ug/here-hd-live-map-layers.html`. (Accessed on 03/18/2021).

[132] M. Maurer, J. C. Gerdes, B. Lenz, and H. Winner. *Autonomous Driving*, volume 10. Springer, 2016.

[133] A. Mileo, M. Dao-Tran, T. Eiter, and M. Fink. Stream reasoning. 2017.

[134] E. Miller. icity: Urban informatics for sustainable metropolitan growth; a proposal funded by the ontario research fund, research excellence. *University of Toronto Transportation Research Institute*, 2014.

[135] M. F. Mokbel, X. Xiong, M. A. Hammad, and W. G. Aref. Continuous query processing of spatio-temporal data streams in place. *GeoInformatica*, 9(4):343–365, 2005.

[136] P. Morignot and F. Nashashibi. An ontology-based approach to relax traffic regulation for autonomous vehicle assistance. *arXiv preprint arXiv:1212.0768*, 2012.

[137] M.-A. Mostafavi, G. Edwards, and R. Jeansoulin. An ontology-based method for quality assessment of spatial data bases. In *Third International Symposium on Spatial Data Quality*, volume 1/28a of *Geoinfo series*, pages 49–66, Apr. 2004.

[138] B. Motik, Y. Nenov, R. Piro, I. Horrocks, and D. Olteanu. Parallel owl 2 rl materialisation in centralised, main-memory rdf systems. In *Informal Proceedings of the 27th International Workshop on Description Logics*, volume 1193. CEUR Workshop Proceedings, 2014.

[139] I. S. Mumick and O. Shmueli. How expressive is stratified aggregation? *Annals of Mathematics and Artificial Intelligence*, 15(3):407–435, 1995.

[140] Navigation Data Standard . Nds white paper - the benefits of a common map data standard for autonomous driving. `https://nds-association.org/wp-content/uploads/2019/06/NDS-White-Paper__Benefits-of-Map-Data-Standard-for-Autonomous-Driving.pdf`. (Accessed on 03/21/2021).

[141] NDS. The standard for map data - NDS Association. `https://nds-association.org/`. (Accessed on 01/27/2021).

[142] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, and J. Banerjee. RDFox: A Highly-Scalable RDF Store. pages 3–20, 10 2015.

[143] N. Noy. Representing classes as property values on the semantic web. `https://www.w3.org/TR/swbp-classes-as-values/`. (Accessed on 05/10/2021).

[144] N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. 2008.

[145] A. Olivares-Alarcos, D. Beßler, A. Khamis, P. Gonçalves, M. Habib, J. Bermejo, M. Barreto, M. Diab, J. Rosell, J. Quintas, J. Olszewska, H. Nakawala, E. Pignaton de Freitas, A. Gyrard, S. Borgo, G. Alenyà, M. Beetz, and H. Li. A review and comparison of ontology-based approaches to robot autonomy. *The Knowledge Engineering Review*, 34, 12 2019. doi: 10.1017/S0269888919000237.

[146] Oracle. Graph database | oracle. `https://www.oracle.com/database/graph/`. (Accessed on 06/25/2021).

[147] P. Patel-Schneider, M. Krötzsch, P. Hitzler, S. Rudolph, and B. Parsia. OWL 2 web ontology language primer (second edition), 2012. URL `http://www.w3.org/TR/2012/REC-owl2-primer-20121211/`.

[148] S. Peroni, D. Shotton, and F. Vitali. The live owl documentation environment: a tool for the automatic generation of ontology documentation. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 398–412. Springer, 2012.

[149] S. Peroni, D. Shotton, and F. Vitali. Tools for the automatic generation of ontology documentation: A task-based evaluation. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 9(1):21–44, 2013.

[150] H. S. Pinto, S. Staab, and C. Tempich. Diligent: Towards a fine-grained methodology for distributed, loosely-controlled and evolving engineering of ontologies. In *In Proceedings of the 16th European Conference on Artificial Intelligence (ECAI*, pages 393–397. IOS Press, 2004.

[151] R. Regele. Using ontology-based traffic models for more efficient decision making of autonomous vehicles. In *4th Int. IEEE Conf. on Autonomic and Autonomous Systems (ICAS'08)*, pages 94–99, 2008.

[152] N. Regnauld. Generalisation and data quality. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 91–94, 2015.

[153] C. Ress, D. Balzer, A. Bracht, S. Durekovic, and J. Löwenau. Adasis protocol for advanced in-vehicle applications. In *15th World Congress on Intelligent Transport Systems*, page 7, 2008.

[154] C. Ress, A. Etemad, D. Kuck, and J. Requejo. Electronic horizon-providing digital map data for adas applications. In *2nd International Workshop on Intelligent Vehicle Control Systems*, pages 40–49, 2008.

[155] M. Rinne, E. Nuutila, and S. Törmä. Instans: High-performance event processing with standard rdf and sparql. In *11th International Semantic Web Conference ISWC*, volume 914, pages 101–104. Citeseer, 2012.

[156] M. Rodriguez-Muro, R. Kontchakov, and M. Zakharyaschev. Ontology-based data access: Ontop of databases. In *International Semantic Web Conference*, pages 558–573. Springer, 2013.

[157] M.-C. Rousset, M. Atencia, J. David, F. Jouanot, O. Palombi, and F. Ulliana. Datalog revisited for reasoning in linked data. In *Reasoning Web International Summer School*, pages 121–166. Springer, 2017.

[158] A. Sawsaa and J. Lu. Building information science ontology (ois) with methontology and protégé. *Journal of Internet Technology and Secured Transaction*, 1: 100–109, 12 2012. doi: 10.20533/jitst.2046.3723.2012.0014.

[159] M. Schreier. Environment representations for automated on-road vehicles. *at-Automatisierungstechnik*, 66(2):107–118, 2018.

[160] A. Seaborne and E. Prud'hommeaux. SPARQL query language for RDF. W3C recommendation, W3C, Jan. 2008. http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[161] H. G. Seif and X. Hu. Autonomous driving in the icity—hd maps as a key challenge of the automotive industry. *Engineering*, 2(2):159 – 162, 2016. ISSN 2095-8099.

[162] D. Seipel. Knowledge engineering for hybrid deductive databases. In *WLP / WFLP*, 2017.

[163] M. Shaw, L. T. Detwiler, N. Noy, J. Brinkley, and D. Suciu. vsparql: A view definition language for the semantic web. *J. of Biomedical Informatics*, 44(1): 102–117, 2011.

[164] N. D. Standard. Nds open lane model. `http://www.openlanemodel.org/`, . (Accessed on 03/15/2021).

[165] N. D. Standard. Navigation Data Standard (NDS) - The worldwide standard for map data in automotive eco-systems. `https://nds-association.org/`, . (Accessed on 03/15/2021).

[166] O. Standards. Geosparql - a geographic query language for rdf data | ogc. `https://www.ogc.org/standards/geosparql`. (Accessed on 06/23/2021).

[167] M. Stocker and M. Smith. Owlgres: A scalable owl reasoner. In *OWLED*, volume 432, 2008.

[168] R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1-2):161–197, 1998. doi: 10.1016/S0169-023X(97)00056-6.

[169] X. Su, E. Gilman, P. Wetz, J. Riekki, Y. Zuo, and T. Leppänen. Stream reasoning for the internet of things: Challenges and gap analysis. In *Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics*, pages 1–10, 2016.

[170] M. C. Suárez-Figueroa. Neon methodology for building ontology networks: Specification, scheduling and reuse. Ontology Engineering Group, June 2010. URL `http://oa.upm.es/3879/`.

[171] M. C. Suárez-Figueroa, A. Gómez-Pérez, and B. Villazón-Terrazas. How to write and use the ontology requirements specification document. In *Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems: Part II*, OTM '09, pages 966–982, Berlin, Heidelberg, 2009. Springer-Verlag.

[172] Y. Suryawanshi, H. Qiu, A. Ayara, and B. Glimm. An ontological model for map data in automotive systems. In *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 140–147. IEEE, 2019.

[173] B. Swartout, R. Patil, K. Knight, and T. Russ. Toward Distributed Use of Large-Scale Ontologies. In *Ontological Engineering, AAAI-97 Spring Symposium Series*, pages 138–148, 1997.

[174] The W3C SPARQL Working Group. SPARQL 1.1 overview, 2013. URL `http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/`.

[175] S. Ulbrich, A. Reschka, J. Rieken, S. Ernst, G. Bagschik, F. Dierkes, M. Nolte, and M. Maurer. Towards a functional system architecture for automated vehicles, 2017.

[176] S. Ulbrich, A. Reschka, J. Rieken, S. Ernst, G. Bagschik, F. Dierkes, M. Nolte, and M. Maurer. Towards a functional system architecture for automated vehicles. *arXiv preprint arXiv:1703.08557*, 2017.

[177] M. Uschold and M. Gruninger. Ontologies: principles, methods and applications. *The Knowledge Engineering Review*, 11(2):93–136, 1996. doi: 10.1017/S0269888900007797.

[178] S. Vallières, J. Brodeur, and D. Pilon. Spatial integrity constraints: a tool for improving the internal quality of spatial data. *Fundamentals of Spatial Data Quality*, pages 161–178, 2006.

[179] F. Van Harmelen, V. Lifschitz, and B. Porter. *Handbook of knowledge representation*. Elsevier, 2008.

[180] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (Second Edition) - W3C Recommendation 11 December 2012, Dec. 2012. URL `http://www.w3.org/TR/owl2-overview/`.

[181] H. Wache, T. Voegele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information-a survey of existing approaches. In *Ois@ ijcai*, 2001.

[182] C. Wang, Q. Sun, Z. Li, and H. Zhang. Human-like lane change decision model for autonomous vehicles that considers the risk perception of drivers in mixed traffic. *Sensors*, 20(8):2259, 2020.

[183] F. Wang, S. Mäs, W. Reinhardt, and A. Kandawasvika. Ontology based quality assuranc e for mobile data acquisition. In *EnviroInfo*, pages 334–341, 2005.

[184] Y. Xing, C. Lv, H. Wang, H. Wang, Y. Ai, D. Cao, E. Velenis, and F.-Y. Wang. Driver lane change intention inference for intelligent vehicles: Framework, survey, and challenges. *IEEE Transactions on Vehicular Technology*, 68(5):4377–4390, 2019. doi: 10.1109/TVT.2019.2903299.

[185] X. Xu and H. H. Huang. Exploring data-level error tolerance in high-performance solid-state drives. *IEEE Transactions on Reliability*, 64(1):15–30, 2014.

[186] C. Yilmaz and Ç. Cömert. Ontology based quality evaluation for spatial data. *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, 40, 2015.

[187] C. Yilmaz, C. Comert, and D. Yildirim. Sdqo and sfo, ontologies for spatial data quality. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W13:1275–1280, 2019.

[188] L. Zhao, R. Ichise, S. Mita, and Y. Sasaki. Core ontologies for safe autonomous driving. In *International Semantic Web Conference (Posters & Demos)*, 2015.

[189] L. Zhao, R. Ichise, Z. Liu, S. Mita, and Y. Sasaki. Ontology-based driving decision making: A feasibility study at uncontrolled intersections. *IEICE Transactions on Information and Systems*, E100.D(7):1425–1439, 2017.

[190] L. Zheng, B. Li, H. Zhang, Y. Shan, and J. Zhou. A high-definition road-network model for self-driving vehicles. *ISPRS International Journal of Geo-Information*, 7(11):417, 2018.

[191] L. Zheng, B. Li, B. Yang, H. Song, and Z. Lu. Lane-level road network generation techniques for lane-level maps of autonomous vehicles: A survey. *Sustainability*, 11(16):4511, 2019.

[192] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller, et al. Making bertha drive—an autonomous journey on a historic route. *IEEE Intelligent transportation systems magazine*, 6(2):8–20, 2014.