2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)

An Ontological Model for Map Data in Automotive Systems

Yogita Suryawanshi*†, Haonan Qiu*†, Adel Ayara† and Birte Glimm*
*Institute of Artificial Intelligence, University of Ulm, Germany, Email: firstname.lastname@uni-ulm.de
†BMW Car IT GmbH, Ulm, Germany, Email: firstname.lastname@bmw.de

***Abstract*—Digital map data is an important source of information for the perception of the environment around cars for advanced driver assistance functions. These functions use map data to acquire information about the road infrastructure beyond the visual horizon of the driver. Embedded software components in today's cars typically use code-based processing of the map data to offer this support to advanced driver assistance functions, but the complexity of automotive systems continues to grow towards the realization of autonomous driving. To facilitate the representation and extraction of knowledge, we explore the feasibility of using ontologies for modelling and processing the map data in cars. We describe the challenges of adequately modelling the knowledge and present a proof of concept implementation that is used in a PC-based simulation to evaluate the knowledge extraction capabilities of this approach considering the requirements of representative advanced driver assistance functions.**

***Index Terms*—Semantic Web, Ontologies, Driver-assistance, Knowledge Modeling**

I. Introduction

Advanced driver assistance (ADAS) functions in today's cars need knowledge about the environment not only in the vicinity of the car but also beyond the visual horizon of the driver. Digital map data provided by the in-car navigation system contains information about the roads ahead of the car. The extracted view representing the streets that the car will most likely drive through is called the *electronic horizon*. The map data, which forms the electronic horizon, is communicated as a stream of messages on the automotive bus to the software components offering advanced driver assistance functions (e.g. speed limit information). The communication is typically based on some map data protocol like the Advanced Driver Assistance System Interface Specification (ADASIS) [1]. The messages sent by the navigation system are assembled by a map data processing component, which constructs the electronic horizon and which provides an advanced perception of the road ahead e.g. by computing the most probable path that the car is likely taking (assuming no destination is set in the navigation system). The map data processing component provides a fixed API (set of functions) that the ADAS functions use to provide their functionalities.

Ontologies and semantic technologies have been used successfully for modelling knowledge in diverse domains [2]–[4]. As the automotive industry heads towards the development of fully autonomous vehicles, the complexity of the environmental knowledge needed for suitably representing the electronic horizon grows significantly. In light of this growing complexity, we investigate the feasibility of using an ontological approach for knowledge representation in automotive systems in order to provide a more flexible and extendable approach compared to the traditional procedural approach. Using ontologies further allows for employing highly optimised reasoners to derive implicit knowledge, which, in the procedural approach, needs to be extracted by writing (often complex) dedicated code. Flexibility is gained by replacing the fixed API with a query interface, e.g. using SPARQL queries. In this paper we highlight the challenges of modeling the highly dynamic map data in such a way that the core knowledge model is independent of changes with respect to the underlying communication protocol.

Basing the system on semantic technologies has many advantages: OWL ontologies provide a rich, flexible, and fully declarative language for modelling map data and a car's environment; additionally, the Semantic Web Rule Language (SWRL) provides an expressive rule language for describing inferences for the data; infrastructures such as ontology editing and reasoning tools are readily available and can be used to support the ontology development; finally, the ADAS functions can easily be adapted and extended without coding new API functions.

In the rest of the paper we assume basic familiarity with RDF, OWL, SWRL, and SPARQL; readers are referred to [5]–[8] for suitable primers.

II. Related Work

To the best of our knowledge, ontologies have not yet directly been used for representing map data in cars. There are, however, ontological approaches for modeling map data and for using ontologies in automotive applications. The OpenStreetMap (OSM) project provides map data in the form of OSM files consisting of elements representing geographic entities and tags representing the characteristics of the elements. An ontology called OSMonto [9] is proposed to model the hierarchy of OSM tags to enable the integration of OpenStreetMap into the Semantic Web. As OSMonto does not model OSM elements, it cannot be used for representing the map data completely. Stadler et al. propose an ontology called LinkedGeoData [10] to model both elements and tags in OSM data. The LinkedGeoData ontology is heavy as it attempts to model all geographic entities and amenities, including those that do not affect driving, e.g. photocopy shops or park benches. On the other hand, the ontology does not model the

978-1-7281-1488-0/19/$31.00 ©2019 IEEE
DOI 10.1109/AIKE.2019.00034

road elements and road structure in enough detail for ADAS functions, e.g. different lanes of and lines on the roads are not modeled. A small topological ontology is defined in the GeoSPARQL standard [11] to represent geospatial data using classes to model geometries of spatial objects. Queries related to the geometries of spatial objects are supported using a geographic query language provided suitable GPS coordinate data is available. Hülsen et al. propose an ontology which models infrastructure elements such as roads, lanes, traffic lights etc. for determining driving behavior in complex traffic situations based on rules [12]. It is assumed that the position of other vehicles is available, whereas this is typically not the case for map data available through the in-car navigation system. Fernandez et al. have proposed an ontology-based intelligent transportation system to offer driver assistance in different traffic situations [13], [14]. Ontologies in this system are used to model road infrastructure elements, vehicle data, weather conditions etc. and to decide the driving behavior of vehicles in different traffic situations. It is assumed that vehicles can exchange information with other vehicles and with infrastructure elements such as traffic signals, signs etc. Another ontology-based approach identifies abnormal situations during driving, where traffic rules can safely be relaxed [15]. Toulni et al. propose an ontology-based approach to manage traffic in urban areas with the help of a vehicular area network (VANET) data transmission system [16] and an ontology-based partial model of map data. Both aforementioned approaches do not model road infrastructure elements like lanes, lines on the road, speed limits, etc. Zhao et al. have proposed three ontologies: a map ontology, a control ontology, and a car ontology to create a knowledge base for determining the driving strategy for an autonomous car [17]. The map ontology uses a static OSM file as input and it does not model road characteristics, e.g. road gradient or (conditional) speed limits.

None of the above approaches predict the path that the car is likely to take in near future, which is an important source of information for ADAS functions. In order to provide such an electronic horizon, one needs to process dynamic message streams about the environment of a car, whereas the above described approaches are all based on static sources for map data like OSM files.

III. A Knowledge Layer for Map Data

As illustrated in Figure 1, we propose an ontology-based knowledge layer between the ADAS functions and the map data provider (in-car navigation system). Since map data communication protocols are prone to changes, we propose a further division into a protocol-dependent ontology and a protocol-independent ontology that provides access to the map data on a more abstract level. The protocol-dependent ontology, also called the *implementation ontology*, models the data based on the map data protocol and is continuously updated as the car moves. The protocol-independent ontology, also called the *core ontology*, is then populated by the implementation ontology using rules. The generic and flexible query interface

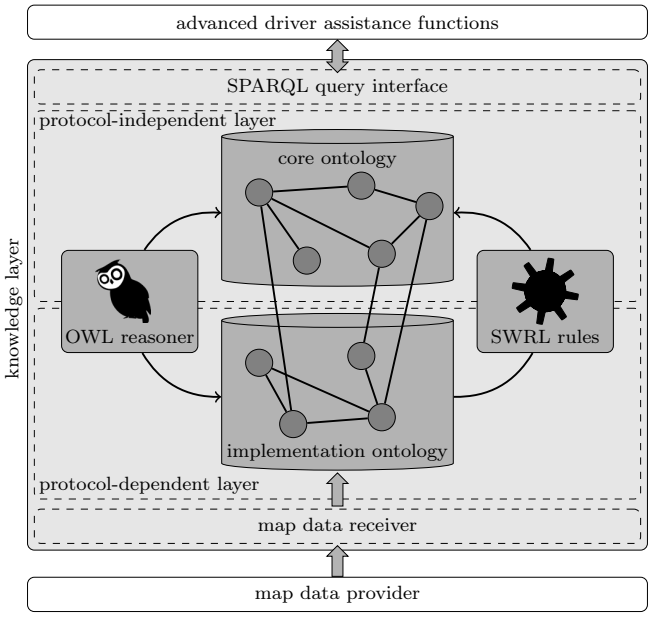

Fig. 1. Ontology-based system architecture

replaces the fixed API offered by the traditional code-based implementation.

A. The Implementation Ontology

The implementation ontology models the map data as communicated via messages by the map data provider and is continuously updated by the map data receiver as messages arrive. Here we assume that the ADASIS protocol represents roads as collections of parts called *road segments*. The messages defined in the protocol fall into three categories:

1) **Segment-info messages** contain map data with information about road segments, e.g., segment identifiers, gradients, ...
2) **Vehicle-info messages** contain information about the vehicle, e.g., current location, current lane, ...
3) **Control messages** for the protocol's state machine logic.

Each message consists of a number of signals that represent different attributes. A signal itself consists of bits. Four different segment-info messages together represent the different characteristics of a road segment. These four message types are represented by four classes in the ontology. Similar to messages, the signals within a message fall into two categories:

1) **Simple signals** use all bits to represent one attribute of the road segment (e.g. identifier or gradient). In the ontology, the attribute is connected to the corresponding message class using a data property.
2) **Complex signals** use either one bit or a set of bits to represent different values for a characteristic, e.g., a signal which indicates presence or absence of different types of points of interest (POIs). Thus, a complex signal represents a collection of simple signals/attributes and it is modeled

using an auxiliary class with data properties for representing each attribute. The auxiliary class is associated with the corresponding message class using an object property.

Raw signal values are used in the messages instead of the real physical values to save bandwidth on the automotive buses. The protocol defines conversion formulae to convert the raw values to real physical values. To avoid the necessity of repeated conversions during knowledge extraction, we assign the real values of the signals to the corresponding data properties. Some signal values are also mapped to readable string descriptions. The protocol also defines special raw signal values indicating invalid values and unavailable values. In such cases the signal values are not modeled in the ontology as they do not provide useful information.

Every time a segment-info message is received, the map data receiver creates an instance of the corresponding message class and associates it with the corresponding data properties. Once all four segment-info messages are received, the corresponding individuals are connected with object properties to an instance of the class *Node*, which represents the overall road segment. The data is then abstracted and transferred to the core ontology using SWRL rules and deleted from the implementation ontology to avoid data duplication. When a vehicle-info message is received, a corresponding individual is created in the implementation ontology and associated with the data properties to the received attribute values and then also transferred to the core ontology. Control messages, trigger tasks such as deleting information about road parts that have already been passed by the car.

The ADASIS protocol represents the tree-like road structure using road segment IDs and parent segment IDs. A road segment is represented by an instance of the class *Node* in the implementation ontology and associated via data properties to its ID and parent segment ID. SWRL rules then use the IDs and parent IDs to connect road segments (i.e. instances of the class node) to model the bare minimum partial road structure. The core ontology then models the complete road structure in a generic way based on this minimal road model.

B. The Core Ontology

Unlike the implementation ontology, which uses concepts and relationships to represent messages and signals as used in the map data communication protocol, the core ontology represents real world entities such as road parts, lanes, traffic signs etc. and their relationships in a generic way as illustrated in Figure 2. The mapping from protocol-specific individuals in the implementation ontology to the generic individuals in the core ontology is implemented using SWRL rules. The instances of the abstract class *RoadPart* correspond to instances of the class *Node* in the implementation ontology, which allows for constructing a complete road structure, including directly and indirectly connected road segments, based on the segment and parent segment IDs. A functional object property is used to model the "immediate previous" relationship with its inverse "immediate next". These two object properties are sub-properties of the two transitive properties *hasNext* and *hasPrevious*.

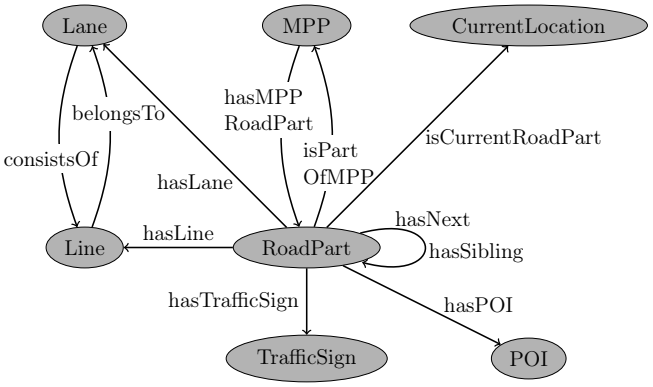

Fig. 2. Modeling of the map data in the core ontology

Apart from the map data receiver, all parts of the knowledge layer use purely declarative knowledge representation techniques. While the map data receiver is still a procedural component, it has a relatively simple structure, which is easy to maintain.

IV. Complex Knowledge Extraction Tasks

SWRL rules are not only used for transferring data between the implementation and the core ontology. Due to their expressivity, they are also used to infer sibling road parts (i.e., different road parts that have the same parent road part), neighboring lanes, and for checking whether the vehicle is approaching or leaving a junction. The obtained ontological knowledge of the core ontology is exposed to the advanced driver assistance functions through a query interface. We categorize the queries into the following three types:

1) Road-part specific queries: Given a road part identifier, provide characteristics of the road part.
2) Look-ahead queries: These queries consider the road parts ahead of the car and involve the identification of change points of characteristics like slope or speed limit in the look-ahead direction or along the most probable path the car is likely taking. The calculation of the distance to a specified POI is also an important query.
3) Look-back queries: These queries consider the road parts already passed by the car. Examples include the calculation of the distance from the last POI or last junction.

For road-part specific queries, not all characteristics are always available. This is handled using SPARQL's OPTIONAL keyword for extracting such property values. Some look-ahead queries involve the computation of intermediate results. For example, to identify the distance to characteristic change points, one uses a nested sub-query to find the change points and then computes the distances to these change points in the outer query. The overall distance is the sum of three distances: (i) the remaining distance to be travelled in the current road part, (ii) the sum of the lengths of the road parts between the current road part and the road part of

interest, and (iii) the length of the road part of interest. Analogously, distance calculations are also performed in the look-back direction. These calculations are implemented using SPARQL 1.1 features like the property path expression '+' (for a variable-length path of length at least one) and aggregation (for building the sum).

A. Extracting the Most Probable Path

Some advanced driver assistance functions require knowledge about the most probable path (MPP) the car is taking (assuming no destination is set in the in-car navigation system). The parameters that determine the MPP are defined in the protocol and may include factors such as whether a road part has sibling roads or the curvature of sibling roads. We explore the feasibility of modelling such computations in an ontology-based system using three different approaches:

The query-based approach uses SPARQL queries for computing the MPP. The query uses UNION blocks such that it is guaranteed that only one of the blocks returns a result, which is the next most probable road part for the currently considered road part. Each UNION block matches a condition as defined in the protocol for when a road part is part of the MPP. For example, one UNION block checks whether the current road part has only one succeeding road part. If this is the case, the only succeeding road part is the next most probable road part. Other UNION blocks handle cases where the current road part has more than one succeeding road part and consider, for example, road characteristics to determine the next most probable road part. An iterative execution of the query using procedural code is required to find all road parts of the complete MPP.

The rule-based approach avoids the iterative query execution, which needs to be handled explicitly using code since rule execution is handled implicitly by the reasoner. As described above, the MPP computation involves finding road parts which do *not* have any siblings. This requires the expression of knowledge that is not available, i.e. it requires *negation as failure*, which is not supported by SWRL rules. Hence, it is not possible to directly identify individuals which "do not have a property". As a workaround, we use a temporary data property that is explicitly added to the ontology (using external code) to indicate road parts without siblings. Although SWRL offers a number of built-ins for performing arithmetic operations, it does not directly support the identification of maximum and minimum values from a group of values. This is, however, required for some cases in the decision of whether a road part belongs to the MPP or not. To address this, we use a two-step process with additional temporary data properties: for finding the maximum, the first step identifies the road part which has a property value greater than at least one of the siblings using the *swrlb:greaterThanOrEqual* built-in. These road parts are marked using the temporary data property and their property values are again compared using the *swrlb:lessThan* built-in. After this step, the road part with the maximum property value has only one temporary data property. Other sibling road parts have either two or no temporary data properties. A similar process is used for identifying road parts with minimum property values.

Once a road part satisfies the conditions for being part of the MPP, no further conditions need to be tested. To support this in the rule-based setting, we use a temporary data property to indicate that a condition is not satisfied by any of the checked road parts. This property is used in the antecedent of the rules for the next condition that is to be checked.

The MPP is re-computed only when the current location of the vehicle changes (triggered by the arrival of a corresponding vehicle-info message). To avoid unnecessary rule applications, we assign a temporary data property to all road part individuals only when the current location message is received. This property is used in the antecedent of all rules used to compute the MPP and deleted once the MPP is determined.

The hybrid approach is based on a combination of SWRL rules and a SPARQL query to compute the MPP. This approach only checks whether newly added road parts extend or change the MPP. Hence, less temporary data properties are required as compared to the pure rule-based approach and the number of comparisons required to compare sibling roads is reduced because only the property values of a new road part are compared to an existing most probable road part at the junction. However, this approach needs a complete re-computation of the MPP using the SPARQL query described in the query-based approach when the car leaves the pre-computed MPP. This is due to the fact that the SWRL rules used in this approach cannot be used for MPP computation given a number of existing MPP road parts.

V. Simulation and Evaluation

We developed a simulation to validate and demonstrate the ontology-based map data processing system as shown in Figure 3. An existing tool called "XML runner" is used to generate the map data protocol messages for OSM test maps. The output of this tool is a file containing a stream of messages. This file is converted into a JSON file [18] using the "json-simple" tool-kit[1] to generate a JSON representation of the message stream, which improves readability and facilitates debugging. This JSON file is then used as the input for the implementation ontology. New individuals are created and initialized in the implementation ontology using Java code and the OWL API [19] according to the messages in the input stream. This also involves the conversion of raw signal values to real physical values that are then assigned with the corresponding data properties. Once all messages representing the same road segment are received, the knowledge is transferred to the core ontology using SWRL rules. After this transfer, temporary individuals are deleted and only protocol specific information (about the road parts) is preserved as described earlier. When a new vehicle-info message is received, old individuals used to represent a previous vehicle-info message are deleted and a new individual is created using the OWL API.

[1] https://cliftonlabs.github.io/json-simple/

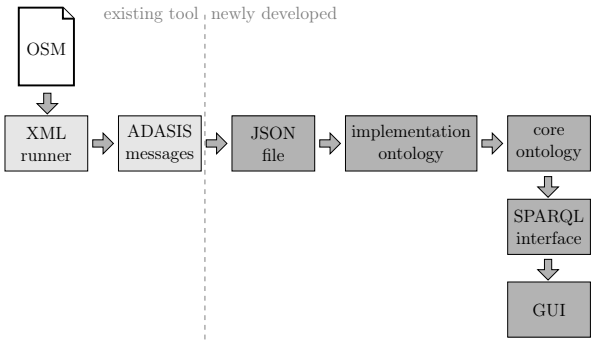

Fig. 3. Data flow in the simulation

TABLE I
AVERAGE QUERY RESPONSE TIME IN MS FOR ROAD-PART-SPECIFIC, LOOK-AHEAD, AND LOOK-BACK QUERIES

road-part-specific queries	road part is present	road part is missing	look-ahead queries	
traffic signs	61.8	61.6	distance to POI	108.2
topology	62.0	59.8	foresight speed	62.2
speed limit	62.2	62.8	foresight slope	61.4
speed	60.4	60.8		
branch probability	65.6	63.6	look-back queries	
street type	62.4	61.0	distance to last POI	67.4
traffic light	60.6	61.2		
average time	62.6	61.5		

If a new control message is received, the protocol state machine control logic is executed using Java code according to the command in the control message. For example, in case of a kill-all command, all individuals in both ontologies are deleted. Two individuals are used in the implementation ontology to represent the current and the previous control message in order to facilitate the state machine control which involves comparing the message counters received in the current and the previous control message.

The input JSON File is parsed message by message to the end of the file. In this way, the map data protocol message stream in the car is simulated and modeled using ontologies.

The described simulation is used to evaluate the proposed system and representative queries from each category (road-part specific, look-ahead, and look-back queries). The evaluation was performed on a 64-bit Windows-7 laptop with an i7-4910MQ CPU running at 2.9GHz with 16GB memory. The OWL API v5.1.0 was used for interacting with the ontologies. The Pellet reasoner [20] was used for reasoning and executing SWRL rules. The SPARQL interface of the Pellet reasoner is used to execute the SPARQL queries. The simulation code is implemented in Java 8. A test map was generated using the Java OpenStreetMap (JOSM) editor to provide the input map data for the system evaluation. The map consists of 60 road segments with a most probable path composed of at most 20 road segments. Features like traffic lights, traffic signs etc. were assigned to the road segments randomly. The system evaluation uses several metrics as described next.

Result Validity: The results of the tested queries and the MPP computation were compared manually with the expected results and all comparison results were positive.

Query Response Time: The execution time for all the SPARQL queries used for knowledge extraction was measured. Initially, all the queries were executed sequentially five times for system warm-up. The query execution time was not measured during the warm-up phase. The system warm-up avoids skewed measurements due to overheads associated with the Java Virtual Machine (JVM) initialization.

Table I shows the average response times for the road-part-specific queries, the look-ahead and the look-back queries. For the road-part-specific queries, we consider two scenarios (i) when the input road part is present and (ii) when the input road part is not present. The query response time is calculated as the average of ten response times for each query.

The results show that the average query response time is 62.6 milliseconds when the road part information is available in the ontology. When the road part information is not presented, the average query response time is 61.5 milliseconds. The first step in these queries is to find the road part matching the input segment identifier. When the road part is not available in the ontology, no further processing is required to extract the values of the characteristics specified in the query like length, speed limit, etc. Hence, the query response time is less in this case as compared to the case when the road part information is present in the ontology. This difference is, however, less than 1%. This shows that a substantial amount of time is required for the general query set-up and for identifying the road part individual matching the segment identifier specified in the query.

The right hand side of Table I shows the average time required for the execution of the look-ahead and look-back queries. The time required for the look-ahead query to determine the distance to a points of interest (POI) is considerably larger than the time required for the corresponding look-back query (distance to last POI). This is due to the fact that the look-back query calculates the distance to only one point of interest in the unique look-back direction. Hence, the query has at most one answer. The distance to POI look-ahead query, however, considers all points of interest in the look-ahead direction. This query returns all road parts with the specified point of interest (e.g. a petrol station) in the look-ahead direction and their distances from the current road part.

Ontology Parameters: The first part of Table II shows the number of data properties, object properties and axioms used by the three MPP computation approaches. The rule-based approach uses the most data properties due to used temporary data properties for the MPP computation. This approach also uses the maximum number of axioms. The query-based approach requires the least number of data properties and axioms, because it does not use any temporary data properties and rules for the MPP computation.

System Complexity: The second part of Table II shows the

TABLE II
(1) STATISTICS ABOUT THE ONTOLOGY, (2) THE NUMBER OF REQUIRED RULES AND QUERIES FOR MPP COMPUTATION, (3) THE AVERAGE MPP COMPUTATION TIME IN MS, AND (4) THE MAP LOADING TIME IN S

		SPARQL	SWRL	Hybrid
(1)	data properties	155	165	158
	object properties	34	34	34
	axioms	859	879	873
(2)	MPP SPARQL queries	1	0	1
	auxiliary SPARQL queries	1	5	5
	SWRL rules	0	21	6
	temporary properties	0	10	3
(3)	initialization (ms)	2,788.71	2,908.40	3,718.92
	MPP computation (ms)	692.91	2,510.28	14.86
(4)	map loading (s)	199	346	190

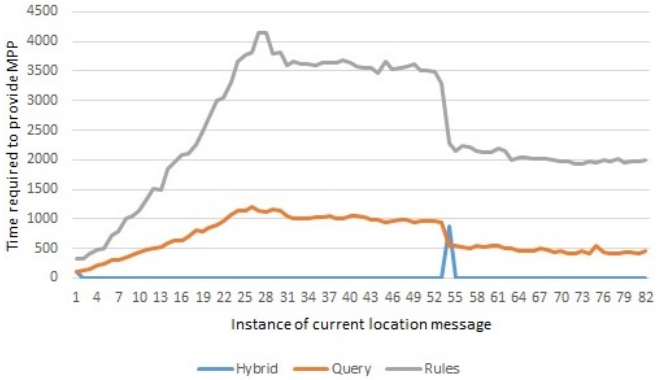

Fig. 4. MPP computation time for each approach

complexity of the three MPP computation approaches in terms of the number of rules, queries and temporary data properties used by these approaches. The SPARQL query-based approach models the MPP with only one SPARQL query and requires no rules and temporary data properties, but external code to iteratively run the query until all road parts of the MPP are found. The SWRL rule-based approach uses 21 SWRL rules for modelling the MPP computation with 5 helper SPARQL queries and 10 temporary data properties. The hybrid approach uses 1 SPARQL query and 6 SWRL rules for modelling the MPP computation with only 3 temporary properties and 5 helper SPARQL queries.

MPP Computation Time: As described in Section IV, the MPP computation is split into different stages in all the three approaches used for modeling the MPP computation. In the query-based approach and the rule-based approach, all the computations required by the MPP algorithm are executed only after reception of the message containing the information about the car's current location. In the hybrid approach, the major part of the MPP computation algorithm is executed whenever new road parts are added to the ontology. After receiving the information about the current location of the car, only the remaining small part of the MPP computation is executed, i.e. the approach performs an incremental MPP maintenance. Hence, we measure the time required for adding a new road part along with the time required to compute the MPP after receiving the information about the car's current location. These timing measurements are taken during the message streaming in the simulation system, when a new road part is added to the ontology and when the current location information is received.

Part (3) of Table II shows that the time required for initializing a newly created instance of the class *RoadPart* is different for the three MPP computation approaches. The hybrid approach needs the maximal road part initialization time. This is because an additional set of SWRL rules is executed during the initialization of the newly created *RoadPart* instance to check whether it belongs to the MPP as described in Section IV.

This part also shows that the road part initialization time is higher in the rule-based approach compared to the query-based approach. As described in Section IV, the rule-based approach and the query-based approach compute the MPP only when the current location information is received. In the rule-based approach, however, some SWRL rules are executed during the initialization of a newly created *RoadPart* instance for inferences, which are required later for the MPP computation e.g. checking if the new *RoadPart* instance has siblings.

One can also observe that the average time required to provide the MPP after the reception of the current location message is lowest in the hybrid approach. This is due to the fact that the MPP is already calculated in the hybrid approach during the road part initialization. Hence, the MPP is not re-computed after the reception of every current location message in this approach unlike in the other two approaches. As described in Section IV, unless the car leaves the calculated MPP, the hybrid approach merely provides the pre-computed MPP when the current location message is received.

The second row of part (3) of the table also illustrates that the rule-based approach requires the highest average time for providing the MPP after the reception of the current location message. This is due to the fact that this approach uses some intermediate OWL API-based operations for the deletion and modification of some temporary data properties as described in Section IV. Some SPARQL queries are also executed for housekeeping in this approach. Due to these overheads, this approach needs the highest time to provide the MPP. The query-based approach does not need intermediate operations for management of temporary data properties like the rule-based approach. Hence, the query-based approach needs less time than the rule-based approach to provide the MPP.

Figure 4 shows the variation of the time required to provide the MPP during the reception of the protocol messages for the test map. The time required to provide the MPP is recorded at 82 different points during the reception of the input message stream. This means that the current location information was received 82 times during the streaming and, hence, the MPP was provided 82 times. Each point on the x-axis corresponds to the reception of a current location message. The figure shows that the MPP computation time increases as messages

representing new road parts keep coming in until a certain point, for both the rule-based and the query-based approaches. This can be seen near point 26 on the x-axis in Figure 4. At this point, the MPP length is at its maximum. After this point, the time required to provide the MPP decreases gradually for both the rule-based and the query-based approach as the car moves ahead and further old road parts become irrelevant for the MPP computation. There is a drastic decrease in the time required to provide the MPP around point 53 on the x-axis in Figure 4, because the car takes a diversion, which has few road parts ahead in the test map. This is again followed by a gradual decrease in the MPP computation time due to old road parts in the diversion becoming irrelevant for the MPP computation.

As shown in Figure 4, the time required to provide the MPP stays fairly constant throughout the test map for the hybrid approach. This is due to the fact that this approach uses a-priori knowledge about the MPP as described in Section IV. When the car moves away from the pre-computed MPP, the MPP needs to be re-computed from scratch using a SPARQL query in this approach. This leads to an increase in the time to provide the MPP as shown by the spike around point 53 on the x-axis in Figure 4. As the car continues to drive along the new MPP, the time required to provide the MPP is reduced again and stays fairly constant.

Note that when the car leaves the pre-computed MPP, the hybrid approach uses the same SPARQL query used by the query-based approach for MPP computation. However, the MPP computation time in the hybrid approach is higher as shown around point 53 on the x-axis. This is due to the fact that the query-based approach computes the MPP immediately after receiving the current location information. The hybrid approach first needs to check if the car left the pre-computed MPP before executing the query for the MPP computation, which explains the higher time requirement.

Map Loading Time: The map loading time is the time required for loading the test map completely in the ontology. As shown in part (4) of Table II, the map loading time is lowest for the hybrid approach and highest for the rule-based approach. The SWRL rules used in these two approaches increase the reasoning time. However, as described earlier, the hybrid approach does not compute the MPP every time a message containing the current location information is received. Hence, the map loading time for the hybrid approach is not as high as for the rule-based approach. Although the query-based approach does not use SWRL rules for the MPP computation, it computes the MPP using an iterative query execution every time a message containing the current location information is received. Hence, the map loading time of the query-based approach is higher than for the hybrid approach.

Coding Complexity: More lines of Java code are required for the query-based and hybrid approach as compared to the rule-based MPP computation approach. This is due to the fact that the rule-based approach models the majority of the MPP computation logic in the ontology itself. Hence, it requires only a small amount of external housekeeping code.

The evaluation results show that the hybrid approach offers the best compromise among the evaluation parameters. It offers the least MPP computation time and map loading time with moderate system and coding complexity.

VI. Conclusions and Outlook

We have explored the possibility of using ontologies for modeling map data and its processing for automotive systems using a case study of the protocol used to communicate the map data at BMW. A proof-of-concept simulation demonstrates the feasibility of this approach. The ability to support the knowledge extraction requirements of representative advanced driver assistance functions is evaluated using representative SPARQL queries over the map data in the dynamically updated ontology. The challenge of computing the most probable path for a car is solved and evaluated with three different approaches (rule-based, query-based, and hybrid). The proposed system is evaluated in terms of the system complexity, result validity, query response time and algorithm execution time.

Based on the evaluation results, we conclude that the map data and its processing as used in cars can be modeled successfully using ontologies and semantic web technologies such as SPARQL queries and SWRL rules. However, some improvements and extensions to the prototype proposed in this paper are required, before it can be used in real world applications. The major hurdles in using ontology-based processing in cars are the resource constraints on the electronic control units such as lack of file systems. Furthermore, we have used tools that are implemented in Java, whereas the software for resource constrained embedded systems in cars is typically developed using programming languages such as C or C++. A possible extension of this work is to implement the presented ontology-based map data processing system using a reasoner such as RDFox [21], which is implemented in C++. This may also improve the performance, which is an important factor for automotive embedded system software.

In conclusion, by adding an additional knowledge layer to the system, we can successfully decouple the low-level map data processing functions and the high-level driver assistance functions. The provided query interface replaces the fixed application programming interface. This allows for easily adding and changing driver assistance functions as these functions can now flexibly request the required knowledge via queries over a protocol-independent knowledge base. The split of a protocol-dependent implementation and a protocol-independent core ontology further allows for changing the underlying protocol or adding support for other protocols without exposing this to the driver assistance functions, as these function purely query the generic knowledge in the core ontology. Hence, the move towards a knowledge based system has significant potential to improve the maintainability of the overall in-car map processing system.

REFERENCES

[1] 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*, 2008, p. 7.

[2] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing?" *International journal of human-computer studies*, vol. 43, no. 5-6, pp. 907–928, 1995.

[3] M. Uschold and M. Gruninger, "Ontologies: Principles, methods and applications," *The knowledge engineering review*, vol. 11, no. 2, pp. 93–136, 1996.

[4] O. Bodenreider, "Biomedical ontologies in action: role in knowledge management, data integration and decision support," *Yearbook of medical informatics*, vol. 17, no. 01, pp. 67–79, 2008.

[5] G. Schreiber and Y. Raimond, Eds., *RDF 1.1 Primer*, 27 October 2014.

[6] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, Eds., *OWL 2 Web Ontology Language: Primer*, 27 October 2009.

[7] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, 2004.

[8] *SPARQL 1.1 Overview*, 21 March 2013.

[9] M. Codescu, G. Horsinka, O. Kutz, T. Mossakowski, and R. Rau, "OSMonto–an ontology of OpenStreetMap tags," *State of the map Europe (SOTM-EU)*, vol. 2011, 2011.

[10] C. Stadler, J. Lehmann, K. Höffner, and S. Auer, "LinkedGeoData: A core for a web of spatial open data," *Semantic Web*, vol. 3, no. 4, pp. 333–354, 2012.

[11] T. O. G. Consortium, "GeoSPARQL-a geographic query language for RDF data," viewed on 14.12.2018, http://www.opengeospatial.org/standards/geosparql.

[12] M. Hülsen, J. M. Zöllner, and C. Weiss, "Traffic intersection situation description ontology for advanced driver assistance," in *Intelligent Vehicles Symposium (IV)*. IEEE, 2011, pp. 993–999.

[13] S. Fernandez, T. Ito, and R. Hadfi, "Architecture for intelligent transportation system based in a general traffic ontology," in *Computer and Information Science*. Springer, 2016, pp. 43–55.

[14] S. Fernandez, R. Hadfi, T. Ito, I. Marsa-Maestre, and J. R. Velasco, "Ontology-based architecture for intelligent transportation systems using a traffic sensor network," *Sensors*, vol. 16, no. 8, p. 1287, 2016.

[15] P. Morignot and F. Nashashibi, "An ontology-based approach to relax traffic regulation for autonomous vehicle assistance," in *Proceedings of the 12th IASTED International Conference on Artificial Intelligence and Applications*, 2013.

[16] H. Toulni, B. Nsiri, M. Boulmalf, and T. Sadiki, "An ontology based approach to traffic management in urban areas," *International Journal of Systems Applications, Engineering & Development*, vol. 9, 2015.

[17] L. Zhao, R. Ichise, S. Mita, and Y. Sasaki, "Ontologies for advanced driver assistance systems," *Journal of the Japanese Society for Artificial Intelligence (JSAI), Technical Report, Proceedings of the 35th Semantic Web and Ontology Study Group*, 2015.

[18] *Standard ECMA-404: The JSON Data Interchange Syntax*, 2017, 2nd edition.

[19] M. Horridge and S. Bechhofer, "The OWL API: a Java API for OWL ontologies," *Semantic Web Journal*, vol. 2, pp. 11–21, 2011.

[20] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz, "Pellet:a practical OWL-DL reasoner," *Journal of Web Semantics*, vol. 5, no. 2, pp. 51–53, 2007.

[21] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, and J. Banerjee, "RDFox: A highly-scalable RDF store," in *Proceedings of the International Semantic Web Conference (ISWC)*. Springer, 2015, pp. 3–20.