The Role of Ontologies in Enabling Dynamic Interoperability

Vatsala Nundloll, Paul Grace, Gordon S. Blair

School of Computing and Communications, Lancaster University, UK
{nundloll, gracep, gordon}@comp.lancs.ac.uk

Abstract. Advances in the middleware paradigm has enabled applications to be integrated together thus enabling more reliable distributed systems. Although every middleware tries to solve interoperability issues among a given set of applications, nonetheless there still remains interoperability challenges across various middlewares. Interoperability enables diverse systems to work in accordance and extend the scope of services that are provided by individual systems. During an interoperability process, it is imperative to interpret the information exchanged in a correct and accurate manner in order to maintain coherence of data. Hence, the aim of this paper is to tackle this issue of semantic interoperability through an experimental approach using the domain of vehicular ad-hoc networked systems.

Keywords: Interoperability, Ontology, Vehicular Ad-Hoc Networks.

1. Introduction

Middleware technologies have proven a successful solution in ensuring interoperability within distributed systems. However, due to the heterogeneity of applications and environments, a number of different middleware systems have emerged. Due to conflicting standards, communication styles and heterogeneous protocols are unable to interoperate with one another. Solutions to this problem of *middleware heterogeneity* include bridging [14, 15, 16] and interoperability frameworks [17, 19, 20]. However, these are insufficient where interoperability is required 'on-the-fly', i.e., where two heterogeneous systems spontaneously encounter one another at runtime, there is a need to automatically learn the middleware solutions employed and then generate a dynamic bridge between the two systems.

CONNECT[1] is a software framework that dynamically generates these middleware mediators and hence provides *emergent middleware* solutions, which can encounter different networked systems dynamically, enable them to connect together, understand one another and be able to exchange data. Here, a fundamental requirement is the ability to discover and understand the meaning and behaviour of middleware technologies and standards in order to learn and generate the required software. This entails *matching* or comparison between different middleware in order

[1] http://www.connect-forever.eu

to identify how they are different, and based upon this synthesize an appropriate adaptive mapping mechanism to underpin the interoperability solution. To achieve such understanding, we advocate the novel use of ontologies that crosscut middleware solutions; this allows us to obtain semantic knowledge of the middleware in order to allow two different systems to effectively be able to interoperate with each other.

In this paper, we present a dynamic interoperability framework that leverages ontologies in order to provide the following two important capabilities:

- *Classifying Protocols*. This involves discovering and defining the type of messaging protocols from each system.
- *Matching Protocols*. We can observe where the fields of two messages are the same and different to provide the information required to build a bridge.

We use a case-study based evaluation that shows we can understand and match communication protocols from Vehicular Ad-hoc Networks (VANETS) domain.

The paper is structured as follows: Section 2 briefly presents a background on ontologies. Section 3 explains the dynamic framework, and Section 4 details the case study. Section 5 then presents an experiment that validates and evaluates the case study results. Finally, Section 6 discusses related work and then Section 7 concludes the paper and briefly outlines our future work.

2. Background on Ontologies

An ontology is a formal descriptive notation given to concepts that constitute a particular domain. It is a simple but powerful notion used to classify concepts of a domain in the form of a superclass-subclass model and also to define the relationships that exist among these different objects. In addition to making domain assumptions explicit, ontologies also permit reuse and analysis of this domain knowledge. Enabling the classified information regarding a domain to be shared as a common vocabulary across people and applications, ontologies consequently pave the way towards building a supportive infrastructure for information exchange and discovery.

Any kind of domain can be modelled using ontologies, ranging from concrete such as defining a type of bread, to abstract such as defining an organization. Fig. 1 elicits the components that make up an ontology. The *Primitive Concepts*, for instance, denote the different constituents of a domain. On the other hand, the *Defined Concepts* set the criteria to determine whether an object is a member of a certain class. The *Axioms* define the restrictions that are laid on the domain, for example, a particular object O cannot have more than x number of subordinates. These assets can define a domain by structuring its different constituents and defining the relationships that exist among them. Furthermore, the ontologies can also classify this information in order to infer additional meaning to the domain. This is possible through the use of reasoners such as Hermit [1], RacerPro [2] and Fact++ [3].

As an example, referring to the food ontology presented in [23], let us suppose we need to classify the foods into categories such as Healthy Foods and Non-Healthy Foods. Through the means of defined concepts these two classes can thus be created and stored in the ontology. Moreover, through the usage of relations, as explained in Fig. 1, the different foods can be defined in terms of their amount of fat, food energy,

cholesterol, weight and saturated fat. Then, through the use of a reasoner, these various classes of food can be classified as either Healthy Foods or Non-Healthy Foods. The power of the reasoner is that it infers all meanings and facts based upon the semantic meaning provided by the ontology about the concepts of a domain.

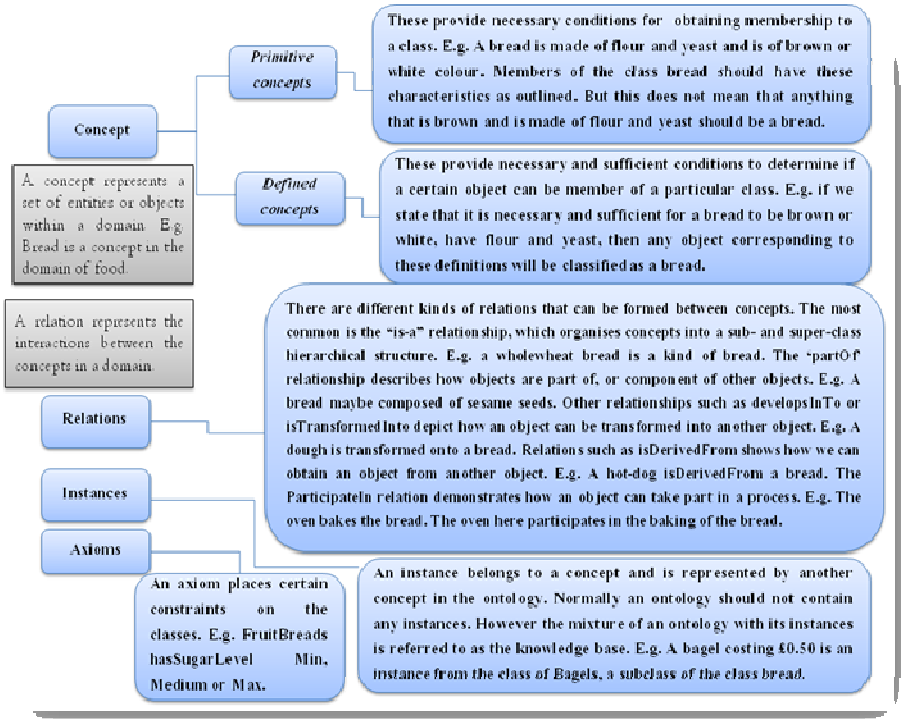

Fig. 1. Ontology Concepts

The OWL language (Ontology Web Language) is the language designed by the W3C in order to build ontologies and is mainly devised for Semantic Web applications. The language formulates the domain information in terms of instances of OWL classes and enables the use of axioms to interpret and manipulate this information.

The software mostly employed to develop ontologies is Protégé, a free, open-source ontology editor [4]. Active development is being carried out to improve the software and its two latest versions are Protégé Version 3.x and Version 4.x. P3.x is shown to be in more stable state to use inference rules within the ontology and hence supports the SWRL rule language and SQWRL query language. SWRL stands for Semantic Web Rule Language and can add more expressivity to the OWL language through the creation of rules. These rules are expressed in terms of OWL concepts (classes, properties, instances). A rule-engine bridge mechanism is provided to embed a rule engine into the Protégé-OWL in order to execute the SWRL rules. One such bridge is the Jess rule engine which can be embedded in P3.x to execute rules and add more expressivity to the OWL language. In addition, the SWRL language has been extended to a query language called SQWRL (Semantic Query Web Rule Language) in order to enable extraction of information from the ontology. The SQWRL library is

packaged with SQL-like built-ins that are used within the SWRL rules in order to execute SQL-like queries to pull out required information. The library further provides new sets of operators classified as Core and Collection operators, which enable basic as well as advanced operations such as select, counting, difference and data aggregation to execute in the rules. An example eliciting the use of a SQWRL query is given below, where a query retrieves all Breads having a price less than £2 from a given ontology:

```
Bread(?b) ^ hasPrice(?b, ?p) ^ swrlb:lessThan(?p, 2) → sqwrl:select(?b, ?p)
```

On the other hand, P4.x supports the latest version of OWL language (OWL 2.0) and is tailored to handle large and complex ontologies. It can produce very expressive ontologies, but it yet cannot provide full support for the creation and execution of inference rules through SWRL and SQWRL. Since the features of P3.x suit more the requirements of our experiment in view of performing queries against our vehicular ontology and enabling matching of different concepts, we have resorted to the usage of Protégé Version 3.4.4 for the purpose of our experiment.

3. Framework for Dynamic Interoperability

Our framework for dynamic interoperability provides mechanisms to achieve emergent middleware (Fig. 2); in this the crosscutting role of ontologies is depicted as central to achieving the objectives. The framework offers three phases of behaviour:

- In the first phase, which regards the *discovery and learning phase*, the ontology is used to give a semantic meaning to the different concepts that are involved in a system. Based on Fig. 1, a system can be defined using the primary and defined concepts together with the axioms available from the ontology. Learning this system involves classifying these defined concepts within the ontology through the use of a reasoner, hence identifying related concepts.
- The second phase, which involves *enabling matching between any two systems*, is achieved through use of semantic rules defined within an ontology. These rules compare the definition of any two concepts classified by the ontology and generate the difference that emanates from the given definitions. This step is crucial as it also shows the degree of similarity/difference that exists between two systems, thus determining the possibility of mapping from one system to another.
- The third step involves the *dynamic synthesis of a mapping mechanism* in order to enable a system A to operate as another system B. The ontology is helpful here to list the missing requirements in A for it to perform as B and vice versa. Once this information is available, the mapping determines how to provide A with the adequate and absent requirements so that it can adapt itself to perform as B.

The role of ontologies spans all the phases required to enable interoperability. We hence advocate and emphasize the importance of using ontologies to define the role or behaviour of a system; these define the types of protocols being deployed by the system and can help bridge the gap between any two different systems trying to interoperate with each other.

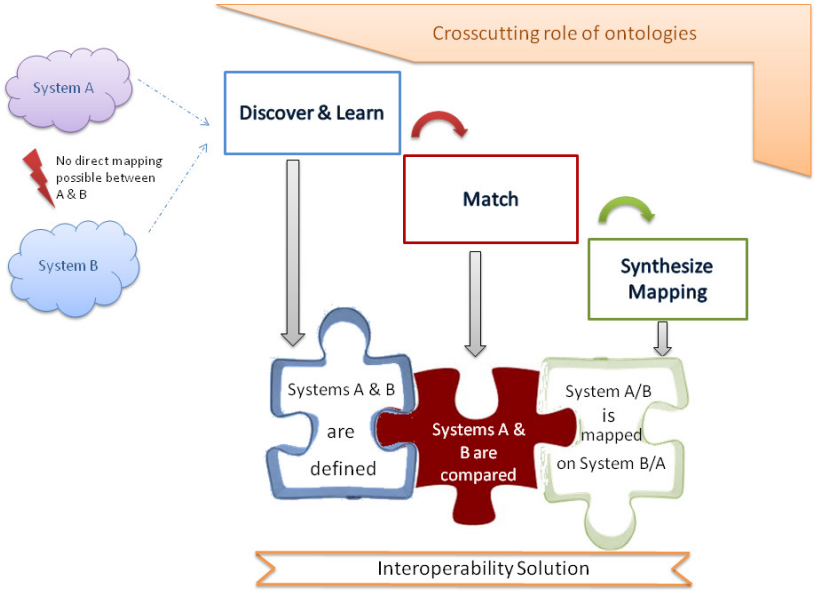

Fig 2. Dynamic Interoperability Framework

4. Case Study on Vehicular Ontology

To enable interoperability between any two systems means dealing with the low-level message exchange between them. Since different systems deal with different message formats, it is imperative to interpret these message formats in a way so that a solution can be devised regarding message exchange between them. The case study that we present in this section is based on the framework explained in section 3 and aims at tackling the interoperability problem at the level of message formats. The main hindrance in exchanging data packets stems from the difference in the packet formats themselves. In this respect, our case study is based on the role played by ontologies in facilitating some level of dynamic semantic interoperability among different packet formats. It shows how we can use ontologies to interpret and enable some level of comparison between message formats from different protocols.

Motivation of the application of Ontologies to VANETs. We chose VANETs as a case-study for our framework, because it is a domain of protocols with heterogeneity of message formats and routing strategies as shown in Fig. 3. Each particular VANET can only interpret the packet formats it has defined for itself. Hence, if we intend to make two different VANETs interoperate with each other, we need a way to be able to interpret the format of incoming packets to a VANET system. To enable this, we define a vehicular ontology to create a vocabulary of the various routing strategies defining their set of requirements. The main idea is to use this ontology to classify unknown incoming packets under the appropriate routing scheme and deduce how to enable the packet to interoperate with the current VANET.

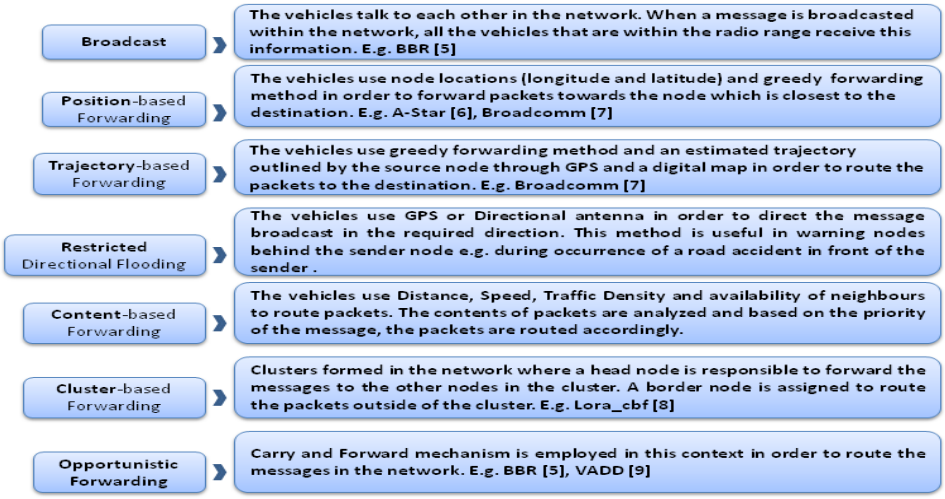

Fig 3. Routing Strategies in VANETs

In Fig. 4 we show the application of our interopeability framework to VANETs. An important element is *the Domain-Component based Model for VANETs*; this is a dynamic middleware for sending, receiving and routing VANET packets—each distinct component serves one specific function within a VANET protocol. This is leveraged to create the emergent middleware between two VANET protocols. We now in turn describe the phases of the interoperability framework.

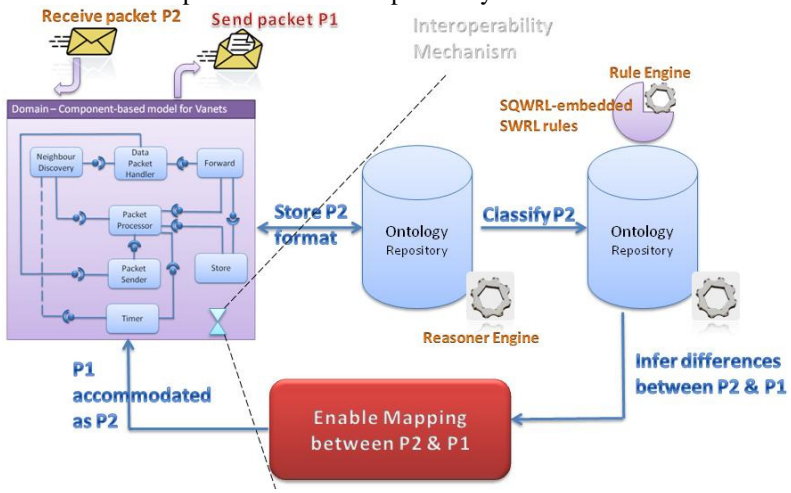

Fig. 4 Applying the dynamic interoperability framework to VANETs

Phase 1, Discover & Learn: *Defining the VANET domain in the ontology*. The first step is to define the VANET domain within the ontology, which is part of the first phase regarding discovery and learning. This ontology contains all of the meanings of the different routing strategies applicable to VANETs, together with the definition of known packet formats. As can be seen in Fig. 5, which shows part of the vehicular ontology, existing packet formats are defined and stored within the ontology. In this

case, they are stored as subclasses of a class called NamedPackets. For instance, one of these is *BBRPacket* which is a protocol performing Broadcast and is derived from the protocol *BBR* [5]. Referring to Fig. 4, let us assume that our VANET system sends packets P_1, suppose a broadcast-based packet, the format of which is already defined by the ontology. Upon receiving packet P_2 with a new unknown format, suppose a trajectory-based packet, the system enables this new format to be defined and stored within the ontology repository.

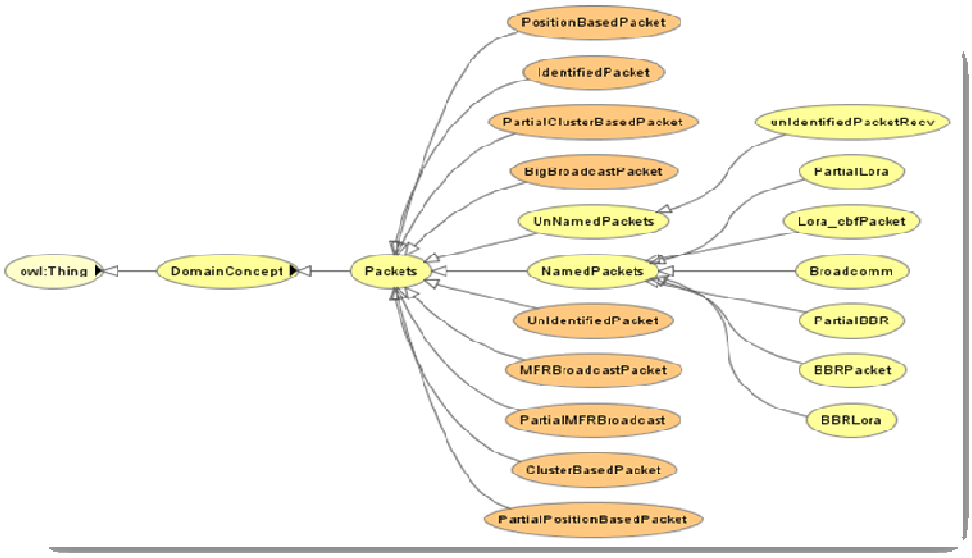

Fig. 5 The VANET Ontology

Phase 1, Discover & Learn: *Identifying a VANET protocol*. The presence of a reasoner engine embedded within the ontology tool enables to infer the meaning of a packet. As a result, the packet is classified under the most appropriate routing strategy. This classification is an important step as it helps to establish a ground for comparison between packets belonging to different routing categories. Part of the inferred ontology is shown in Fig. 6, where the class *BBRPacket* has been properly classified as an *IdentifiedPacket* and also as an *MFRBroadcastPacket*. The requirements for *MFRBroadcastPacket* are the fields *CommonNeighbourNo* and *NeighbourList*. These fields form part of the format of a *BBRPacket* and hence the reasoner is able to classify the latter as being of type *MFRBroadcast* packet (Most Forwarding Broadcast). On the other hand, the class *IdentifiedPacket* denotes that the incoming packets contain fields that are known. It is possible that an incoming packet does not correspond to any of the routing strategies defined within the ontology, yet contains fields that have been already defined by the ontology. In this case, the packet is an *IdentifiedPacket* and this classification is enough to show that information can be extracted from the packet using SQWRL mechanisms (as shown later in the section).

Phase 2, Match*: Dynamic Bridging between P_1 and P_2*. Once the classification process is done, the packet P_2 (Fig. 4) can be compared to the existing packet P_1 through an intuitive mechanism which makes use of SWRL rules and SQWRL query

rules within the ontology itself. These rules enable to deduce the difference that lies between the packet formats P_1 and P_2. For instance, let us assume P_1 to be a *BBR* packet [5] (designed for performing Broadcast-based routing) and P_2 to be a Broadcomm [7] packet (designed for performing cluster-based routing). At this stage, both packets P_1 and P_2 have already been classified under the appropriate routing scheme by the ontology. As can be seen in Fig. 7, which details out the packet format of both BBR (P_1) and Broadcomm (P_2), there is no direct mapping possible.

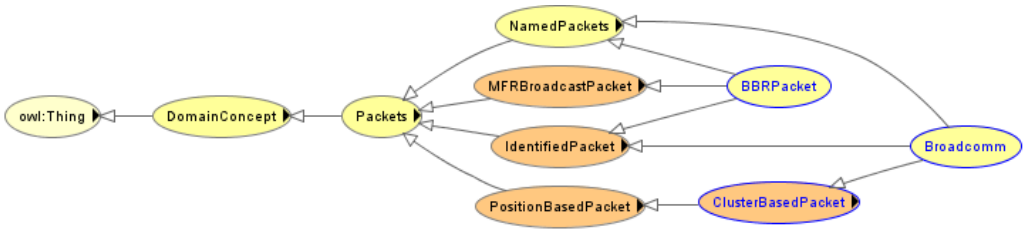

Fig 6. Inferred Vehicular Ontology

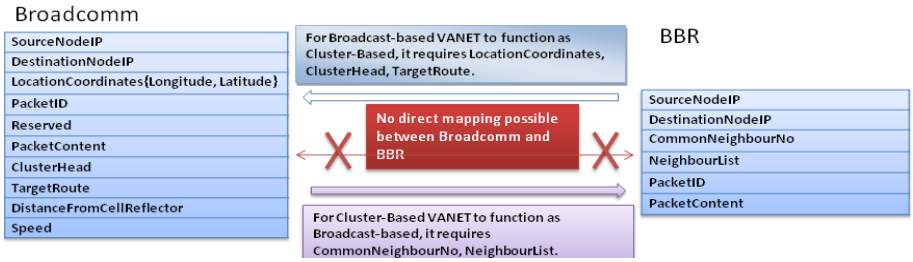

Fig. 7. Mapping BBR and Broadcomm packet formats

Phase 2, Match*: Role of SQWRL.* In order to enable some kind of comparison between them, a rule-based mechanism needs to be deployed within the ontology in order to provide the adequate reasoning to enable matching. We need to make use of SQWRL like queries to retrieve the required information from the ontology. The following SQWRL rule formulates a comparison between *BBR* and *Broadcomm* in order to find out which fields are different between them:

$$BBRPacket(?b) \wedge hasFields(?b, ?f) \wedge Broadcomm(?p) \wedge hasFields(?p, ?pf) * sqwrl:makeBag(?bag, ?f) \wedge sqwrl:makeBag(?bagt, ?pf) * sqwrl:difference(?diff, ?bagt, ?bag) \wedge sqwrl:element(?e, ?diff) \rightarrow sqwrl:selectDistinct(?p, ?e)$$

The SQWRL query states that if *b* is a *BBR* packet and has fields represented by *f*, create a set of all these fields called *bag*. On the other hand, if *p* is a *Broadcomm* packet and has fields denoted by *pf*, create a set of such fields called *bagt*. Then find the difference between these two bags and in such a case, select those fields that have been found to be in *Broadcomm* packet *p* but not in *BBR* packet *b*. The result of this query is the set of fields missing from *BBR* for it to function as a Broadcomm packet. As shown in Fig. 7, the fields lacking in BBR are: *LocationCoordinates*, *TargetRoute* and *ClusterHead* and this information is retrieved via the SQWRL query.

The OWL language enhanced with the use of SWRL and SQWRL results in creating an expressive vehicular ontology, which determines the nature of a packet given the field descriptions. Furthermore, it enables a comparison of any two particular packets and thus provides the difference between them in terms of the fields that are missing. Once the matching of the packets has been achieved, this leads to the next step, which is to perform the mapping between these two packets.

Phase 3, Synthesize Mapping*: Mapping P_2 onto P_1.* Once the difference in the two packet formats P_1 and P_2 have been provided via the ontology, this final step entails engineering an adaptive mapping mechanism to enable P_1 to function as P_2. For example, if we need to enable *BBR* to function as *Broadcomm*, the step will determine how to provide the missing fields in the *BBR* packet, which are *Location Coordinates*, *Cluster Head* and *Target Route*. These are among the set of fields required for Broadcomm to perform cluster-based routing and are lacking from *BBR*. To detail how to enable this mapping mechanism is not within the scope of this paper as it is included in part of our future work regarding the interoperability process.

5. Dynamic Interoperability Experiments

5.1 Methodology

The case study above explains how interoperability can be tackled between two specific VANETs, which are cluster-based (*Broadcomm*) and broadcast-based (*BBR*). In order to validate this case study, we have conducted an experiment to enable the same interoperability procedures (i.e. discovery/learning and matching) to execute at run-time. The experiment consists of tackling interoperability between other systems and our VANET system at run time through the use of our vehicular ontology. In order to enable the experiment at run time, we have made use of java-based programs and Protege-owl API [10] in order to manipulate the ontology at run time. The version of the Protege ontology tool that we have used is Protege3.4.4 [4] which provides full support to apply SWRL rules and SQWRL-based queries.

In order to interpret incoming packets, we read those incoming packets and extract their field labels from their format at run time. These field labels are then stored in a text file. Another java program loads and manipulates the vehicular ontology at run time. The field names, stored in the text file, are then fed as input to the ontology which creates a new packet based on these values. We have used the reasoner Pellet [11] in order to classify the packets defined within the ontology. If the new packet contains fields which are identified by the ontology, then it is classified under a class called *IdentifiedPacket*, as shown in Fig. 8. Otherwise, the packet is ranked under *UnIdentifiedPacket*. Moreover, if the packet corresponds to the requirements of a given routing scheme, the reasoner classifies the packet under the appropriate routing class. However, if the packet partly corresponds to these requirements, it is classified as partially fulfilling the role of that routing scheme by the reasoner. Part of the resulting inferred version of the ontology is displayed in Fig. 6.

We carried out test runs with different packet formats and these are displayed in Table 1. For each new incoming packet, the java program creates a new packet class

in the ontology at runtime, displayed in the *ObjectName* column in Table 1. All the new packet objects are initially created as subclasses of the class *UnNamedPackets*, pictured in Fig. 8. The reasoner then classifies them as identified or non-identified packets and also ranks them under the appropriate routing scheme.

Test Cases	Packet Formats	Object Name
0	{CommonNeighbourNo, DestinationIP}	UnIdentifiedPacketRecv0
1	{CommonNeighbourNo, DestinationIP, Longitude, Latitude, ClusterHead, TargetRoute}	UnIdentifiedPacketRecv1
2	{BroadcastMeter, Distancee, DestinationIp}	UnIdentifiedPacketRecv2
3	{CommonNeighbourNo, UnknownField1, DestinationIP, UnknownField2}	UnIdentifiedPacketRecv3
4	{UnknownField1, UnknownField2, UnknownField3, UnknownField4}	UnIdentifiedPacketRecv4

Table 1. Test Cases

Fig. 8. Inferred Vehicular Ontology with the generated Test cases

5.2 Experiment Results

Fig. 8 portrays the resulting ontology after the generation and classification of these test cases by the reasoner at run time. The first test run, *UnIdentifiedPacketRecv0*, consists of only the fields *CommonNeighbourNo* and *DestinationIP*, whereby *CommonNeighbourNo* is among the set of fields required for performing the MFRBroadcast routing. Therefore, the reasoner classifies the packet *UnIdentifiedPacketRecv0* under the class *PartialMFRBroadcast*, which implies that

this packet can partially provide requirements for performing *MFRBroadcast* routing. It is also classified as an *IdentifiedPacket* since it contains known fields. In the second test case, *UnIdentifiedPacketRecv1*, the fields *Longitude*, *Latitude* and *TargetRoute* are required for performing a Position-based routing. On the other hand, *TargetRoute* is also required among other set of fields to perform a Cluster-based routing. Therefore, the packet *UnIdentifiedPacketRecv1* is classified under *ClusterBasedPacket* which, in turn, is a subclass of *PositionBasedPacket* and is also ranked as an IdentifiedPacket. In test case 3, although the packet, *UnIdentifiedPacketRecv2*, contains known fields such as *BroadcastMeter*, *Distance* and *DestinationIP*, the packet is not classified under any routing strategy since these fields are not defined as critical in the running of any routing strategy. However, because the packet has been identified as containing existing fields, it has been categorized under the *IdentifiedPacket* class. In addition to containing the same fields as in test case 1, the fourth test case, *UnIdentifiedPacketRecv3*, also contains 2 unknown fields. Consequently, it is classified both under *PartialMFRBroadcast* and *UnIdentifiedPacket*. Finally, the fifth test case, *UnIdentifiedPacketRecv4*, contains all unknown fields and hence, is classified as *UnIdentifiedPacket*.

5.3 Evaluation

When using BBR, which requires fields such as *CommonNeighbourNo*, *NeighbourList*, and *DestinationIP* address to perform broadcast-based routing, then test case 0 will require *NeighbourList* to operate as *BBR* and the system will then be able to route the packet. The matching mechanism through the use of SQWRL queries as explained earlier indicates that the latter field is required to enable the existing VANET to route the packet. The mapping mechanism will eventually determine how to enable test case 0 to function as *BBR*. On the other hand, if test case 0 is compared against *Broadcomm*, there are more missing fields since *Broadcomm* requires more fields to operate. Furthermore, if we take test case 3, although a few of the fields have been identified to enable this packet to be partially classified under *MFRBroadcast* routing scheme, the lack of information about the unidentified fields acts as a hindrance to properly identify the format. We may need additional mechanisms to interpret the fields that are unknown, which we consider as part of our future work.

In this experiment, we have tried to deal with the problem of interoperability in the domain of VANETs and have been able to show that this problem can be tackled to a certain degree. The results of this experiment demonstrate that the use of ontology combined with SWRL and SQWRL can help perform a matching between any two packets. This forms the basis of comparing any two concepts, which is the starting point for handling interoperability between them. If we try to expand this idea in a much broader context where different networked systems are trying to interoperate with one another, we would need to create an ontology for every such system in order to capture the meaning of the concepts present within the domain. Thus, the deployment of ontologies creates yet another challenge which is the differences arising among the different ontologies of a same general domain, making their manipulation even more complex and difficult.

To deal with such different application ontologies, a new type of ontology is surfacing called the Reference ontology [24], which aims at providing links between heterogeneous ontologies. However, the authors in paper [24] argue that if ontologies expand a particular reference ontology in a coherent way, then matching their different concepts can be made easier. Providing an initial matching between distinct ontologies of a general domain through a reference ontology is indispensable. If we are moving towards the inception of an emergent middleware to tackle dynamic interoperability, then the reference ontology can provide a benchmark to compare related ontologies and hence facilitate matching the different concepts through the application of SWRL and SQWRL-like rules.

6. Related Work

Universal interoperability is a long-standing objective of distributed systems research. The traditional approach to resolve interoperability problems is to agree on a standard, i.e., everyone uses the same protocols and interface description languages; CORBA[12], DCOM[13], and Web Services are good examples of this approach. For situations where systems can agree to a common standard, the approaches are highly effective. However, for both long-lived and universal interoperability these solutions have demonstrably failed, indeed, future attempts at such global standards are destined to fail too. Such one size fits all standards and middleware platforms cannot cope with the extreme heterogeneity of distributed systems, e.g., from sensor applications through to large-scale Internet applications; and a single communication paradigm, e.g. RPC, cannot meet all application requirements. Moreover, new distributed systems and applications emerge fast, while standards development is a slow, incremental process; it is likely that new technologies will appear that will make a pre-existing standard obsolete. Finally, new standards do not typically embrace an existing legacy standard, which leads to immediate interoperability problems.

Software bridges have been proposed to enable communication between different middleware environments. The bridge acts as a one-to-one mapping between domains; it will take messages from a client in one format and then marshal this to the format of the server middleware; the response is then mapped to the original message format. Many bridging solutions have been produced between established commercial platforms. The OMG created the DCOM/CORBA Inter-working specification [14]. OrbixCOMet [15] is an implementation of the DCOM-CORBA bridge, while SOAP2CORBA [16] bridges SOAP and CORBA middleware. Further, Model Driven Architecture advocates the generation of such bridges to underpin deployed interoperable solutions. However, developing bridges is a resource intensive, time-consuming task, which for universal interoperability would be required for every protocol pair; further a future protocol requires a mapping to every existing protocol. Finally, software bridges must normally be deployed and available in the network; for many environments (particularly resource-constrained) this is not possible.

Intermediary-based solutions take the ideas of software bridges further; rather than a one-to-one mapping, the protocol or data is translated to an intermediary

representation at the source and then translated to the legacy format at the destination (and vice versa for a response). Enterprise Service Buses (ESB), INDISS [17], uMiddle [18] and SeDIM [19] are examples that follow this philosophy. However, this approach suffers from the greatest common divisor problem, i.e., between two protocols the intermediary is where their behaviour matches, they cannot interoperate beyond this defined subset. As the number of protocols grows this common divisor then becomes smaller such that only limited interoperability is possible.

Substitution solutions (e.g., ReMMoC [20] and WSIF [21]) embrace the philosophy of speaking the peer's language. That is, they substitute the communication middleware to be the same as the peer or server they wish to use. A local abstraction maps the behaviour onto the substituted middleware. Like for software bridges this is particularly resource consuming; every potential (and future) middleware must be developed such that it can be substituted. Further, it is generally limited to client-side interoperability with heterogeneous server.

Semantic Middleware solutions e.g. S-ARIADNE [22] employs efficient, semantic, service-oriented solutions to achieve interoperability; this is utilized at the discovery and matching stage to ensure that only services that semantically match attempt to interoperate with one another; hence, concerning itself with only the application data and function differences, not the heterogeneity of middleware (indeed a common middleware platform is required).

There is a distinct disconnection between the mainstream middleware work and the work on semantic interoperability. Our solution embraces and integrates the ideas of both (as far as we are aware it is the first to employ ontologies to resolve communication protocol interoperability); because of this, we argue that it is better placed to achieve long-lived, universal interoperability. By employing ontologies to classify, match and map communication protocols, we have the ability to automatically generate a bridge between two legacy communication protocols. The nature of the solution means that the problems of standards, exhaustive resource requirements, and minimal matches can be overcome.

7. Conclusions

This paper puts forward the approach of using ontologies to handle the semantic differences that arise between different systems, in order to enable them to interoperate. We have elicited three major steps required for performing interoperability, which are: discovery/learning, matching and synthesis of mapping. We have also elicited the role ontology plays in handling these steps and we advocate that the ontology has a crosscutting role in the steps involved during the interoperability process. All three processes are intertwined and each step provides the necessary input to perform the next step. For instance, the result from the matching process provides a sound notion on the type of mapping that needs to be performed. We have been able to validate the discovery/learning and matching phases through a case study on VANETs, however, the synthesis of mapping between two different systems remains part of our future work.

Furthermore, we also intend to extend in the future our experiment on VANETs through the investigation of user-defined SWRL built-ins in order to compare the data types of the fields in a message to achieve richer interoperability between protocols through the handling of data heterogeneity. We also plan to explore a wider range of middleware protocols including traditional technologies where bridging has been attempted e.g., RPC protocols, message-based platforms and service discovery.

References

1. http://hermit-reasoner.com/
2. http://www.racer-systems.com/products/racerpro/
3. http://owl.man.ac.uk/factplusplus/
4. http://protege.stanford.edu/
5. Zhang, M and Wolf, R. 2007. Border Node Based Routing Protocol for VANETs in Sparse and Rural Areas. IEEE Globecom Autonet Workshop (Washington DC, Nov. 2007), 1-7.
6. Liu, G., Lee, B, Seet, B., et al.. A Routing Strategy for Metropolis Vehicular Communications, in Proc. Int. Conference on Information Networking (ICOIN), Feb 2004.
7. Durresi, M., Durresi, A., and Barolli, L. 2005. Emergency Broadcast Protocol for Inter-Vehicle Communications, in Proc. 11th International ICPADS Conference Workshops.
8. Santos, R., Edwards, A., Edwards, R., and Seed, N. 2005, Performance evaluation of routing protocols in vehicular ad-hoc networks, Int. J. Ad Hoc Ubiquitous Comput. 1, 1/2, 80-91.
9. Zhao J. and Cao, G. 2006. VADD: Vehicle-assisted data delivery in vehicular ad hoc networks, in Proc. 25th IEEE International Conference on Computer Communications.
10. http://protege.stanford.edu/plugins/owl/api/
11. http://clarkparsia.com/pellet/
12.Object Management Group, "The common object request broker: Architecture and specification Version 2.0," Technical Report 1995.
13. D. Booth et al. (2004, Feb) W3C Working Group Note, http://www.w3.org/TR/ws-arch/
14. Object Management Group, "COM/CORBA Interworking Specification Part A & B," 1997.
15. Iona Tech. (1999) OrbixCOMet http://www.iona.com/support/whitepapers/ocomet-wp.pdf.
16. L. Brueckne. (2010, January) SOAP2CORBA. [Online]. http://soap2corba.sourceforge.net
17. Y. Bromberg, V. Issarny, "INDISS: Interoperable Discovery System for Networked Services," in Proc of the IFIP/ACM/Usenix Int. Middleware Conference, France, 2005.
18. J. Nakazawa, H. Tokuda, W. Edwards, and U. Ramachandran, "A Bridging Framework for Universal Interoperability in Pervasive Systems," in Proc of 26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006), Lisbon, Portuga, 2006.
19. C. Flores, G. Blair, and P. Grace, "An Adaptive Middleware to Overcome Service Discovery Heterogeneity in Mobile Ad Hoc Environments," IEEE Dist Sys Online, July 2007.
20. P. Grace, G. Blair, S. Samuel, "A Reflective Framework for Discovery and Interaction in Heterogeneous Mobile Environments," ACM SIGMOBILE Review, Jan. 2005.
21. M. Duftler, N. Mukhi, S. Slominski, and S. Weerawarana, "Web Services Invocation Framework (WSIF)," in Proc. OOPSLA 2001 Workshop on OO Web Services, Florida, 2001.
22. S. Ben Mokhtar, A. Kaul, N. Georgantas, V. Issarny. “Efficient Semantic Service Discovery in Pervasive Computing Environments”, in Proc. of ACM/IFIP/USENIX 7th International Middleware Conference, Melbourne, Australia, November 2006.
23. J. Cantais, D. Dominguez, V. Gigante, L.Laera, V. Tamma, “An example of food ontology for diabetes control”, in Proc. of the ISWC 2005 workshop on Ontology Patterns for the Semantic Web, Galway, Ireland, November 2005
24. C. Wang, K. He, Y. He, “MFI4Onto: Towards Ontology Registration on the Semantic Web”, 6th IEEE Int. Conference on Computer and Information Technology (CIT'06), 2006.