The Role of Ontologies in Emergent Middleware: Supporting Interoperability in Complex Distributed Systems

Gordon S. Blair[1], Amel Bennaceur[2], Nikolaos Georgantas[2], Paul Grace[1], Valerie Issarny[2], Vatsala Nundloll[1], Massimo Paolucci[3]

[1]School of Computing and Communications, Lancaster University, UK
[2]INRIA, CRI Paris-Rocquencourt, France
[3]DOCOMO Euro-Labs, Munich, Germany
gordon@comp.lancs.ac.uk

Abstract. Interoperability is a fundamental problem in distributed systems, and an increasingly difficult problem given the level of heterogeneity and dynamism exhibited by contemporary systems. While progress has been made, we argue that complexity is now at a level such that existing approaches are inadequate and that a major re-think is required to identify principles and associated techniques to achieve this central property of distributed systems. In this paper, we postulate that emergent middleware is the right way forward; emergent middleware is a dynamically generated distributed system infrastructure for the current operating environment and context. In particular, we focus on the key role of ontologies in supporting this process and in providing underlying meaning and associated reasoning capabilities to allow the right run-time choices to be made. The paper presents the CONNECT middleware architecture as an example of emergent middleware and highlights the role of ontologies as a cross-cutting concern throughout this architecture. Two experiments are described as initial evidence of the potential role of ontologies in middleware. Important remaining challenges are also documented.

Keywords: interoperability, ontologies, emergent middleware, system-of-systems

1 Introduction

Interoperability is a fundamental property in distributed systems, referring to the ability for two or more systems, potentially developed by different manufacturers, to work together, including the ability to exchange and interpret action requests and associated data sets. Indeed, interoperability is absolutely foundational—without a solution to interoperability, distributed systems become impossible to develop and evolve. In the first generation of distributed systems, interoperability was relatively straightforward to achieve. Such systems were small-scale, fairly homogenous in terms of languages, operating system platforms and hardware architectures, and also under the control of a single organisation and associated administration team. This

was of course unsustainable and very quickly distributed systems expanded in terms of scale, level of heterogeneity and complexity of administrative control, leading to the Internet-scale distributed systems that we are familiar with today. A number of interoperability solutions emerged both in terms of proposed standards for interoperability and solutions to bridging between standards. Distributed systems have, however, continued to evolve, and we particularly note two important trends:

1. The level of heterogeneity has increased dramatically in recent years with developments such as ubiquitous computing potentially coupled with enhanced modes of interaction (for example using ad hoc networking), mobile computing where an increasing range of mobile devices provide a window on to greater distributed system services, and cloud computing where complex distributed system services are offered in the greater Internet. We refer to this as extreme heterogeneity, whereby the levels of heterogeneity significantly exceed the previous generation of distributed systems in terms of the size and capabilities of end system devices, the operating systems used by different devices, the style of communication protocols employed to provide network-level interoperability, the languages and indeed programming paradigms utilized, and so on. Some observers refer to such systems as Systems of Systems [1], and this certainly captures rather elegantly the complexity of the resultant system structures.
2. The level of dynamism in such systems has also increased significantly, partly as a result of the trends noted above, for example the increasing mobility involved in distributed systems has led to the need to support spontaneous interoperation whereby devices interoperate with services that are discovered in a given location, coupled with solutions that need to be intrinsically context-aware (including of course location-aware access to services). The level of dynamism is also affected by the need for more adaptive and/ or autonomic approaches, again stemming from the complexity of modern distributed systems.

The end result is that it is very difficult to achieve interoperability in such complex distributed systems. Indeed, we can say that distributed systems are in crisis with no principled solutions to interoperability for such complex and dynamic distributed systems structures. Note that we can go further in this analysis and not just consider the ability to interoperate but also the quality of service of interoperability solutions in terms of a range of non-functional properties, for example related to security or dependability. This is a very valid dimension to consider but is beyond the scope of this paper (we return to this in the final section, and in particular our statements on future work).

It is interesting to note the definition of interoperability from Tanenbaum [2]:

"The extent by which two implementations of systems or components from different manufacturers can co-exist and work together by merely relying on each other's services as specified by a common standard"

This definition emphasizes the role of a global, or at least common, standard and, while this offers one solution to interoperability, it is not a realistic option for the complex distributed systems of today. For example, competitive pressures have inevitably led to competing standards emerging in the marketplace. Where standards have reached a level of acceptance, for example with web services, it is recognized by the community that they may be problematic for certain operating environments, for example, ubiquitous systems. In addition, any given standard can very quickly

become a legacy system as time elapses and requirements evolve.

We argue that with the above pressures we need a fundamental re-think of distributed systems. In particular, we advocate a solution whereby the necessary middleware to achieve interoperability is not a static entity but rather is generated dynamically as required by the current context. We refer to this as *emergent middleware*. Furthermore, we investigate the key role of ontologies in supporting this process and, in particular, in providing the ability to interpret meaning and associated reasoning capabilities in generating emergent middleware. Ontologies have already been studied in the context of distributed systems, most prominently in the semantic web community, offering a means of interpreting the meaning of data or associated services as they are dynamically encountered in the World-Wide Web. This however limits the scope of ontologies to support the top-level access to data and services. We are interested in a more comprehensive role for ontologies in supporting meaning and reasoning in the distributed systems substrate which supports and enables access to such services, i.e., in the middleware itself, offering a cross-cutting approach where ontologies provide support to fundamental distributed systems engineering elements.

This paper focuses exclusively on the role of ontologies in supporting the concept of emergent middleware (further discussion of the broader area of emergent middleware can be found in [3]). More specifically, the aims of the paper are:

1. To investigate previous work on interoperability in the middleware community and in the semantic web community with a view to seeking a unification between these (to date) largely distinct areas of research;
2. To understand both the role and scope of ontologies in supporting key middleware functions, particularly related to emergent middleware solutions;
3. To investigate more generally the role of ontologies within a general architecture for emergent middleware.

The paper is structured as follows. Section 2 examines the interoperability-related challenges associated with complex distributed systems and the associated responses both from the middleware and the semantic web community. Section 3 moves into the solution space, presenting the key components of an emergent middleware approach, before charting the role of ontologies within this approach. Section 4 presents two experiments, which together provide evidence of the key role ontologies can play in different levels of a middleware architecture. Finally, Section 5 contains an overall analysis and reflections over the experience of working with ontologies in emergent middleware, including the identification of key areas of future work related to this area.

2 The Interoperability Problem Space: Challenges and Responses

The problem space for interoperability must consider the differences of: i) *applications*, and ii) *middleware protocols*. In both cases, there will typically be differences in *data* and *behaviour*:

- Application *data* differs in terms of format and meaning, e.g., the data value of a `price` parameter can be defined in an object or XML document. It can also

mean different things, e.g., the price is in Pounds versus Euros.

- Depending upon application interfaces, the *behaviour* may be significantly different, e.g., multiple operations of one interface performing the same functionality of a single operation of another.
- Middleware protocols providing the same communication abstraction may differ in the *data* format and type model, e.g., different RPC protocols capture data and types using different methods and formats.
- There now exists a broad range of communication abstractions (e.g., publish-subscribe, tuple spaces, message-orientation, group communication) offered by middleware protocols; these exhibit significant *behavioural* differences.

We now examine the responses to these challenges from two distinct communities (the middleware and the semantic web communities) and investigate the extent to which comprehensive application and middleware interoperability has been achieved.

2.1 Response from the Middleware Community

The first responses by the middleware community to address interoperability problems proposed *standards*-based approaches, i.e., common protocols and interface description languages. CORBA, DCOM, and web services are effective examples of this approach. However, as previously described, such solutions are not suited to today's highly complex distributed systems that exhibit extreme heterogeneity and dynamic behaviour. The second set of responses then looked at the challenges of heterogeneous middleware protocols interoperating with one another. One example of this, *software bridge*, acts as a one-to-one mapping between domains; taking messages from a client in one format and then marshalling this to the format of the server middleware. As examples, the OMG created the DCOM/CORBA Inter-working Specification [6]. OrbixCOMet is an implementation of the DCOM-CORBA bridge, while SOAP2CORBA[1] 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.

Alternatively, *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. Enterprise Service Buses (ESB), INDISS [8], uMiddle [9] and SeDIM [10] are examples that follow this philosophy, and these allow differences of both behaviour and data to be overcome. 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.

A radically different response involved *substitution solutions* (e.g., ReMMoC [11] and WSIF [12]); rather than bridging, these embrace the philosophy of speaking the

[1] http://soap2corba.sourceforge.net/

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. This approach allows interoperation among different abstractions and protocols. However, as with 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 servers.

The limitation of all the above responses is that they ignore the heterogeneity of the application, assuming that there are no differences, due to the adoption of a common interface. In complex systems, this is clearly not the case.

2.2 Response from the Semantic Web Community

The semantic web community's responses to the interoperability problem are based upon the principles of reasoning about and understanding how different systems can work together. Their key contribution is *ontologies*. An ontology is defined as a logic theory, and more precisely as a tuple <A, L, P>, where A is a set of axioms, L is a language in which to express these axioms, and P is a proof theory, that supports the automatic derivation of consequences from the axioms. In turn, the proof theory P allows us to derive consequences, which extract relations that have never been stated explicitly, but that are implicit in the description of the systems. Ultimately, the proof theory allows recognition of the deeper "semantic" similarity between structures that are syntactically very different.

The work in *semantic web services* demonstrates how ontologies can be used to address interoperability problems at the application level. Specifically, ontologies have been used during discovery to express the capabilities of services, as well as the requests for capabilities; in this case, the proof theory recognizes whether a given capability fits a given request. A number of semantic middleware technologies provide this ability, e.g., the Task Computing project [13], and the Integrated Global Pervasive Computing Framework [14]. One important solution, EASY [15], implements efficient, semantic discovery and matching to foster interoperability in pervasive networking environments. Further, ontologies have been used during composition to address the problem of application data interoperability, as well as the problem of recognizing whether the conditions for executing the service indeed hold. The limitation of these responses lies in the assumption of a specific middleware, namely web services. There is a need to represent heterogeneous middleware and networking environments, which is almost completely absent in the semantic web services work.

Ontologies introduce a new meta-level, which can produce its own interoperability problems. Heterogeneous ontologies push the interoperation problem one level up. The computational complexity of the proof theories, which is often beyond exponential, makes ontologies resource expensive. Finally, there is a problem of generating the ontologies. The problems listed here are fundamental problems with which the semantic web at large is grappling, and fortunately a number of partial solutions exist that mitigate these problems. For example, ontology matching can be used to address the problem of different ontologies, and smart and efficient inference

engines are now available. As a result, ontologies may be used effectively to automatically address many interoperability problems.

2.3 Summary

It is clear that semantic technologies and interoperability middleware have mostly been developed in isolation by distinct communities. The middleware community made assumptions of common application interfaces and focused on middleware behaviour and data heterogeneity. The semantic web community made the opposite assumption, that there was a common middleware, and the solutions focused on differences in application behaviour and data.

In our view, semantic technologies and interoperability middleware must be comprehensively combined to enable emergent middleware, that is, on-the-fly generation of the middleware that allows networked systems to coordinate to achieve a given goal. Semantic technologies bring the necessary means to rigorously and systematically formalize, analyze and reason about the behaviour of digital systems. Semantic web service technologies have further highlighted the key role of process mediation in an Internet-scale open network environment where business processes get composed out of services developed by a multitude of independent stakeholders. Then, in a complementary way, interoperability middleware solutions hint towards an architecture of emergent middleware that mediates interaction among networked systems that semantically match while possibly behaviourally mismatching, from the application down to the network layer.

3 The Solution Space

The realisation of emergent middleware faces significant challenges, which we are in particular investigating as part of the CONNECT project [3]: i) discovering what is there in terms of application and middleware behaviour and data, ii) enhancing this information using learning techniques, and iii) reasoning upon the required mediation and synthesizing the resulting software to enable interoperability between heterogeneous networked systems. In this section, we first introduce the architecture of the generated emergent middleware, and then we present the ontology-based models of the networked systems used by *Enablers*, i.e., active software entities that collaborate to realise the emergent middleware. Finally, we describe the architecture of Enablers that need to be deployed in the network toward allowing networked systems to interact seamlessly.

3.1 Architecture of Emergent Middleware

Building upon previous interoperability middleware solutions [8, 10, 16], the architecture of the emergent middleware (as shown in Fig. 1) decomposes into: (i) *message interoperability* that is dedicated to the interpretation of messages

from/toward networked systems (listeners parse messages and actuators compose messages) and (ii) *behavioural interoperability* that mediates the interaction protocols run by the communicating networked systems by translating messages from one protocol to the other, from the application down to the middleware and further to the network layer.

However, interoperability can only be achieved based on the unambiguous specification of networked systems' behaviour, while not assuming any a priori design-time knowledge about the given systems. This is where the key role of semantic technologies, i.e., ontologies, comes into play. As discussed in the next section, ontological concepts are employed to characterise the semantics of exchanged messages, from the application down to the network layer, and thus allow the analysis of and reasoning about the external actions performed by systems. This is a major step in the realization of interoperability, since it allows the mediation of interaction protocols at all layers, provided their respective functionalities semantically match.

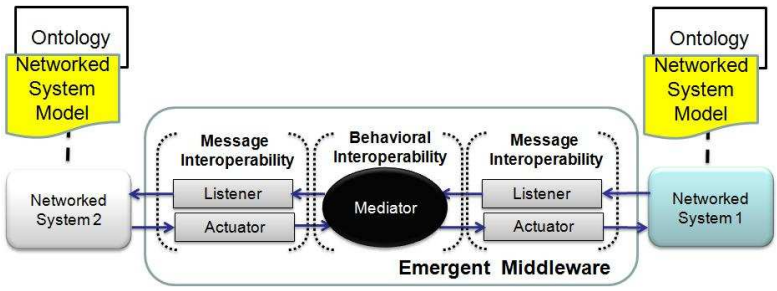

Fig. 1. The emergent middleware architecture

3.2 Ontology-based Networked System Model

The *networked system model* builds upon semantic technologies and especially semantic web services ontologies [17]. Fig. 2 depicts key elements of the system model with ontologies cross-cutting these elements. The model decomposes into:

- The *Affordances* (*aka capabilities* in OWL-S[2]) provide a macroscopic view of networked system features. An affordance is specified using ontology concepts defining the semantics of its *functionality* and of the associated *inputs* and *outputs*. Essentially, the affordance describes the high-level roles a networked system plays, e.g., 'prints a document'. This allows semantically equivalent action-relationships/interactions with another networked system to be matched; in short, they are doing the same thing. Then, provided the matching of affordances that are respectively required and provided by two networked systems, it should be possible to synthesize an emergent middleware that allows the networked systems to coordinate toward the realization of the affordance despite possible mismatches in the messages they exchange and even their behaviour. In practice, networked systems do not advertise affordances but rather interfaces, as discussed below. Nevertheless, recent advances on learning techniques,

[2] http://www.w3.org/Submission/OWL-S/

combining solutions to the cohesion of system interfaces [18] and semantic knowledge inference [19], provide base ground that can be exploited to support the automated inference of affordances from interfaces, although this remains area for future work.

- *The Interface* provides a refined or microscopic view of the system by specifying finer actions or methods that can be performed by/on the networked system, and used to implement its affordances. Each networked system is associated with a unique interface. However, there exist many interface definition languages and actually as many languages as middleware solutions. Nevertheless, existing languages may easily be translated into a common IDL so as to allow the matchmaking of interfaces [20]. Still, a major requirement and challenge are for interfaces to be annotated with ontology concepts so that the semantics of embedded actions can be reasoned upon. While this is already promoted by web services standards (e.g., SA-WSDL[3]), it still remains an exception for middleware solutions at large. Here too, research on advanced learning techniques can lead to automated solutions to the semantic annotation of syntactic interfaces [22].
- The *Behaviour* describes how the actions of the interface are co-ordinated to achieve a system's affordance, and in particular how these are related to the underlying middleware functions. The language used to specify the behaviour of networked systems revolves around process algebra enriched with ontology knowledge, so as to allow reasoning about their behavioural matching based on the semantics of their actions, and subsequently support the generation of the emergent middleware. Such behaviour description has been acknowledged as a fundamental element of system composition in open networks in the context of the Web[4]. However, in the vast majority of cases, networked systems do not advertise their behaviour. On the positive side, different techniques have emerged to learn the interaction behaviour of systems, either reactively or proactively [23, 24, 33]. Still, major research challenges remain in the area, as provided techniques need to be made more efficient as well as be improved, considering, e.g., the handling of data and non-functional properties.

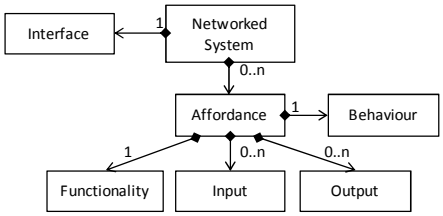

Fig. 2. The networked system model

[3] http://www.w3.org/TR/sawsdl/
[4] www.w3.org/TR/wscl10/

3.3 Enablers for Emergent Middleware

The realization of emergent middleware is supported by cooperating core Enablers as depicted in Fig. 3.

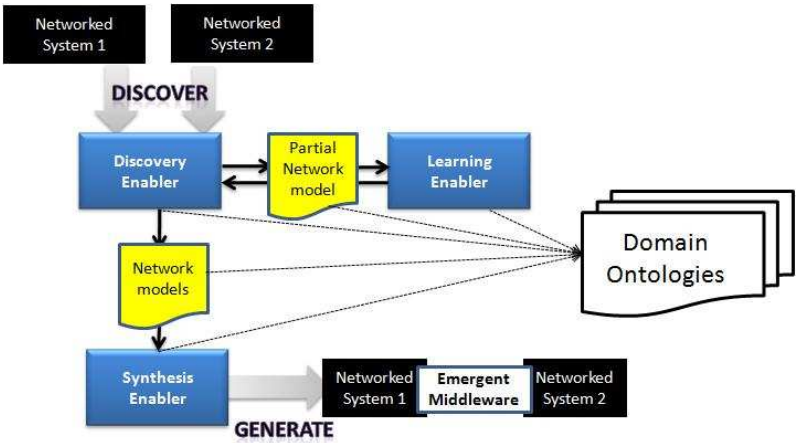

Fig. 3. The architecture of the emergent middleware Enablers

The *Discovery Enabler* receives both the advertisement messages and lookup request messages that are sent within the network environment by the networked systems. The enabler obtains this input by listening on known multicast addresses (used by legacy discovery protocols), as common in interoperable service discovery [25]. These messages are then processed; information from the legacy messages is extracted. At this stage, the networked system model includes at least the interface description, which can be used to infer the ontology concepts associated to the affordance in the case they are not specified. The semantic matching of affordances is then performed to determine whether two networked systems are candidates to have an emergent middleware generated between them. The semantic matching of affordances is based on the *subsumption* relationship possibly holding between concepts of the compared affordances [26]; briefly, the functionality of a required affordance matches a provided one if the former is subsumed by the latter. Other semantic relations such as sequence [29] or part-whole[5] can also be beneficial to concept matching. On a match, the process of emergent middleware generation is started; the current networked system model is sent to the Learning Enabler, which adds more semantic knowledge to it. On completion of the model, the Discovery Enabler sends this to the Synthesis Enabler.

More specifically, the *Learning Enabler* attaches semantic annotations to the interface, and uses active learning algorithms to dynamically determine the interaction behaviour associated to an affordance. Interaction behaviour learning is built upon the LearnLib tool [27], and employs methods based on monitoring and model-based

[5] http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/index.html

testing of the networked systems. It takes the semantic annotations of the interface as input, and returns the system's behaviour description.

The role of the *Synthesis Enabler* is to take the completed networked system models of two systems and then synthesize the emergent middleware that enables the networked systems to coordinate on a given affordance. The emergent middleware specifically implements the needed mediation between the protocols run by the systems to realize the affordance, which are abstractly characterized by the behavioural description. The synthesis of the mediator results from the automated behavioural matching of the two protocols based on the ontological semantics of their actions. In few words, the mediator defines the possible sequences of actions that serve translating semantic actions of one protocol to semantic actions of the other. Obviously, many approaches to behavioural matching and related protocol mediation may be applied considering the state of the art in the area [30, 31]. Basically, the solution to automated protocol mediation shall allow for efficient mediator synthesis, while at the same time enabling interoperability beyond current interoperability middleware solutions. In particular, protocol mediation shall span all the targeted protocol layers, dealing with the semantics of both application and middleware actions [28], as illustrated in the next section. An approach that is particularly promising and that we are investigating lies in ontology-based model checking [32]; this exploits the power of both ontologies to systematically reason about the semantics of actions and model checking to systematically reason about the compatibility of protocols. Still, the more flexible is the compatibility check, the more complex is the reasoning process. The challenge is then to find the appropriate tradeoffs so as to foster interoperability in open networks in a computationally tractable way.

Finally, the *emergent middleware* is deployed, with the resultant connector following the architecture as depicted in Fig. 1, with listeners and actuators providing message interoperability and the synthesized mediator dealing with behavioural differences and translating the message content between heterogeneous message fields. Note the listeners and actuators are automatically generated using the Starlink framework[6].

While this section has focused on the core Enablers toward the generation of emergent middleware, additional enablers are necessary to cope with the uncertainty associated with emergent middleware. Indeed, the learning phase is a continuous process where the knowledge about networked systems is being enriched over time, which implies that emergent middleware possibly needs to adapt as the knowledge evolves. Furthermore, it is important that emergent middleware respects the quality requirements of networked systems regarding their interactions, which requires appropriate dependability and security enablers.

The development, from the supporting theory to concrete prototype implementation, of such enablers is currently ongoing as part of the CONNECT EU project[7]. Despite the tremendous challenges that are raised in unifying and combining the principles of semantic technologies and interoperability middleware to enable emergent middleware, we have been developing experimental enablers to validate this

[6] http://starlink.sourceforge.net/
[7] http://connect-forever.eu/

vision. Our initial experiences with the use of ontologies within this broad solution space are sketched in the next section; these further highlight the important role ontologies have to play in realising our vision of emergent middleware.

4 Experiments

To provide initial insight into the benefits of using ontologies to support interoperability, we now present two experiments that show how semantic technologies can underpin the automatic generation of emergent middleware. The first experiment examines the use of ontologies to address data and behavioural heterogeneity at both application and middleware layers. The second experiment demonstrates how ontologies are used to perform automated matching of message fields to support interoperability at the network layer.

4.1 Reasoning about Interoperability at Application and Middleware Layers

This experiment illustrates the role of ontologies in handling heterogeneity both at application and middleware layers. For this purpose, we consider two travel agency systems that have heterogeneous application interfaces and are implemented using heterogeneous middleware protocols (one is implemented using SOAP and the other with HTTP REST). We use application-specific and middleware ontologies to reason about the matching of both application and middleware behaviour.

The travel agencies example. The first networked system, called *EUTravelAgency*, is developed as an RPC-SOAP web service. Thus, data is transmitted using SOAP request and response envelopes transported using HTTP Post messages. The service allows users to perform the following operations concurrently:

- *Selecting a flight*. The client must specify a destination, a departure and a return date. The service returns a list of eligible flights.
- *Selecting a hotel*. The client indicates the check-in and check-out dates. The service returns a list of rooms.
- *Selecting a car to rent*. The user indicates the period of rental and their preferred model of car. The service then proposes a list of cars.
- *Making a reservation.* Once the user has chosen a flight and/or a hotel room and/or a car, they confirm their reservation. The service returns an acknowledgment.

The interface signature for *EUTravelAgency* (abstracted from WSDL 2.0) is given below, where we provide only the ontology concepts associated with the syntactic terms embedded in the interface:

```
SelectFlight({destination, departureDate, returnDate}, flightList)
SelectHotel({checkIndate, checkOutdate, pref}, roomList)
SelectCar({dateFrom, dateTo, model}, carList)
MakeReservation({flightID, roomID, carID}, Ack)
```

The second system is called *USTravelAgency* and allows users to perform the following two operations:

- *Finding a trip*. The client specifies a destination, departure and return date. The service finds a list of "packages" including a flight and hotel room and car.
- *Making a reservation.* The user selects a trip package and confirms it. The service acknowledges the reception of the selection.

The interface signature, although giving only embedded ontology concepts, is abstracted as follows:

```
FindTrip({destination,departureDate,returnDate,needCar},flightList)
ConfirmTrip(tripID,Ack)
```

The *USTravelAgency* service is implemented as a REST web service over the HTTP protocol. The *findTrip* operation is performed as a HTTP Get and the *confirmTrip* operation is performed using a HTTP Post as shown below (the outputs of both service operations are formatted using JSON[8]):

```
GET http://ustravelagency.com/rest/tripervice/findTrip/{destination}/
{departureDate}/{returnDate}/{needCar}
POST http://ustravelagency.com/rest/tripervice/confirmTrip/{tripID}
```

A client of the *EUTravelAgency* cannot interact with the *USTravelAgency*, and similarly a client developed for the *USTravelAgency* cannot communicate with the *EUTravelAgency* due to the aforementioned heterogeneity dimensions:

- *Application data*. The *EUTravelAgency* refers to the *Flight*, *Hotel* and *Car* concepts, whereas the *USTravelAgency* makes use only of the *Trip* concept. Additionally, the *EUTravelAgency* specifies the departure and the return dates using Greenwich Mean Time (GMT), while the USTravelAgency uses Pacific Standard Time (PST) to describe them.
- *Application behaviour*. In the *EUTravelAgency* implementation, users can independently select a flight, a room and a car, whereas in the *USTravelAgency* implementation all of them are selected through a package.
- *Middleware data format*. The data exchanged in the *EUTravelAgency* implementation are encapsulated in a SOAP message, while the input data of the *USTravelAgency* are passed through a URL and the output data are formatted using JSON.
- *Middleware behaviour*: REST and RPC-SOAP are different architectural styles and induce heterogeneous control and communication models.

The travel agency ontology. The first step of the experiment of interoperability between *EUTravelAgency* and *USTravelAgency* was to create the domain-specific ontology associated with the travel agency scenario (Fig. 4 illustrates an excerpt of this ontology). The ontology shows the relations holding among the various concepts defined in the interfaces of the two travel agencies. Note that the application-specific ontology not only describes the semantics and relationships related to data but also the semantics of the operations performed on data, such as `FindTrip`, `SelectFlight`, `SelectHotel`, and `SelectCar`.

In the general case, the application ontology is not defined by the application developers but by domain experts, to reflect shared knowledge about a specific

[8] http://www.json.org/

domain. Many ontologies have been developed for specific domains, e.g., Sinica BOW[9] (Bilingual Ontological Wordnet) for English-Chinese integration. In addition, work on ontology alignment enables dealing with possible usage of distinct ontologies in the modelling of different networked systems from the same domain, as illustrated by the W3C Linking Open Data project[10].

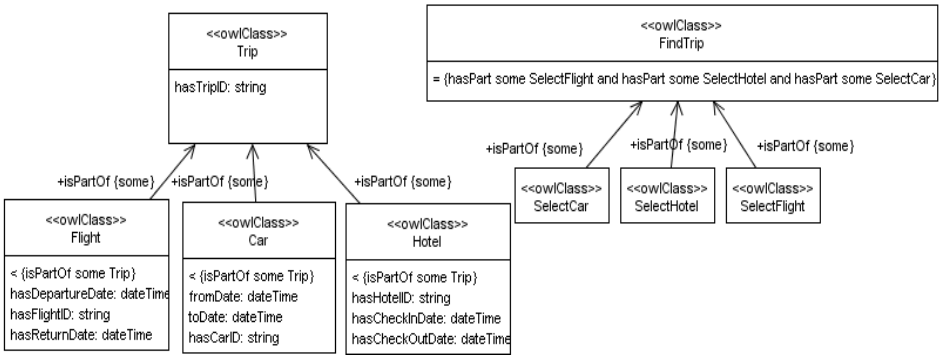

Fig. 4. The travel agency ontology

Dealing with application-level heterogeneity. The travel agency ontology indicates how the *Flight*, *Hotel* and *Car* concepts are related to the *Trip* concept, including their individual attributes. Moreover, we can also use standard ontologies for translation, e.g., OWL-Time[11] can be used to resolve the time difference between GMT and PST.

Solving the application data mismatches is not sufficient. We also need to coordinate the actions of the networked systems in order to make them interoperate. Ontologies help establishing the correspondence between actions. As illustrated in Fig. 4, *FindTrip* is defined as equivalent to the composition of the three operations *SelectFlight*, *SelectHotel*, and *SelectCar*. A mediator that ensures the coordination between the above operations can then be synthesized based on the semantic (subsumption) relations between them and the behaviour of the two networked systems. Moreover, since the *SelectFlight*, *SelectHotel*, and *SelectCar* can be executed concurrently, we need to check all possible executions. Therefore, we rely on model checking further extended with ontology reasoning capabilities, in order to exhaustively explore all the state space and systematically guarantee the correctness of the synthesized mapping rules. As illustrated in Fig. 5, the mediator translates the *FindTrip* action to the concurrent execution of the *SelectFlight*, *SelectHotel* and *SelectCar* actions, and the *MakeReservation* action to the *ConfirmTrip* action. This translation is further refined according to the underlying middleware of each networked system as illustrated next.

Dealing with middleware-level heterogeneity. To reason about the behavioural matching of middleware, we have defined a middleware ontology that identifies where sequences of protocol messages execute similar functionality. For example, the

[9] http://BOW.sinica.edu.tw/
[10] http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
[11] http://www.w3.org/TR/owl-time/

request-response message sequence of CORBA is clearly equivalent to that of SOAP. Yet, there may be cases where the relationship is semantically deeper, e.g., subscription in a publish-subscribe protocol may be equivalent to a RPC invocation (but only when they are performing similar application behaviour) [28].

In the travel agency scenario, the operations are implemented atop SOAP and HTTP REST. The ontology specifies SOAP as a request followed by a synchronous response. Similarly, REST is specified as four alternative synchronous message sends and responses (Get, Post, Put, Delete). The ontology defines that a SOAP operation in the general case is semantically equivalent to all four REST behaviours. Therefore, to reason about interoperability, the application matching must be considered in tandem. For example, in the *FindTrip* operation case, the protocol mediation is from SOAP to Get, whereas in the *ConfirmTrip* case the protocol mediation is from SOAP to Post.

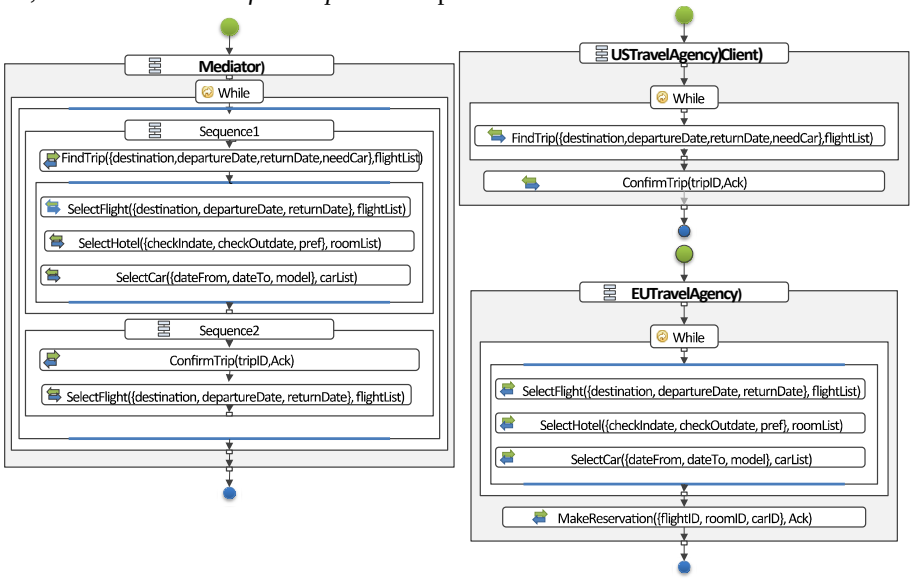

Fig. 5. Behavioural specification of the two travel agencies and the mediator

Another fundamental difference at the middleware level is in the heterogeneity of messages, i.e., the complexity of the translation of SOAP data content to REST data content while message formats are different. We investigate the use of ontologies to reason about this important problem in the second experiment.

4.2 Reasoning about Interoperability at the Network Layer

Devising solutions at the application and middleware level to enable any two systems to interoperate does not suffice if they cannot properly exchange network messages. It is imperative to understand and reason about the heterogeneous message formats of protocols in such a way that message-level interoperability can be achieved on a broader scale. We need systematic ways to dynamically capture the underlying differences of network packets to then generate the mapping between them.

Ontologies provide the methodology to identify these semantic similarities and differences in order to automatically identify the translation between messages.

This experiment focuses on using ontologies to map between heterogeneous Vehicular Ad Hoc Network (VANET) protocols; this domain was chosen because the protocol behaviour is common (i.e., routing of messages to a destination), but there is a high-level of heterogeneity at the packet level. A number of VANET protocols exist that fall into different routing strategies: broadcast, position-based forwarding, trajectory-based forwarding, restricted directional flooding, content-based forwarding, cluster-based forwarding, and opportunistic forwarding. Hence, these exhibit highly heterogeneous message formats owing to the vast array of routing strategies.

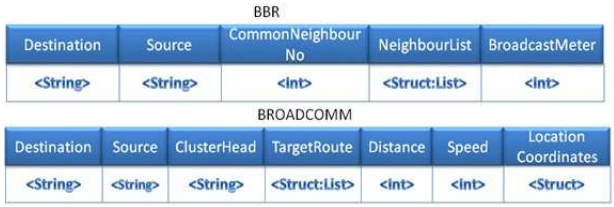

Fig. 6. Packet formats of BBR and Broadcomm packets

Interoperability between BBR and Broadcom. The BBR protocol [34] is a broadcast routing protocol that keeps track of neighbouring nodes and broadcasts the packet at a set rate. The node, lying on the border of the transmission range, is designated to forward the packet further away in the network. This is determined by the number of common neighbours this node has with the source node. This value is represented as a *CommonNeighbourNo* field in the packet. Fig. 6 shows the format of BBR and Broadcomm packets. Broadcomm [35] is a position-based routing protocol, which keeps track of nodes through their geographical locations. This protocol divides the network into clusters and allocates one node in each cluster to be the cluster head. The latter is responsible to forward messages to the cluster members and forward them to the nearest neighbour found outside the cluster. We can say that the behaviour of Broadcomm matches with that of BBR to a certain degree in the sense that both designate a node to disseminate messages further into the network. But both differ in the way their messages are formulated, especially with the use of geographical coordinates in one protocol and not in the other.

Applying Ontologies. Given the differences in their message formats, any sort of interoperation does not seem to be valid if Broadcomm and BBR try to interoperate. However, if we can interpret both message formats and deduce their meaning, it is possible to find a basis for comparison. As a result, we create a vehicular ontology for the various routing strategies used in VANETs together with a definition of known packet formats. The main idea is to use this ontology to classify packet formats under the appropriate routing scheme and deduce how to enable this packet to interoperate with another packet, i.e., construct the mappings that are part of the synthesized mediator in the emergent middleware architecture. The presence of a reasoner engine enables us to infer the meaning of a packet (as we discover middleware knowledge of the networked system). As a result, the packet gets 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 displayed in Fig. 7, where the BBR packet (BBRPacket) is ranked under *IdentifiedPacket* and *MFRBroadcastPacket* classes. The requirements for *MFRBroadcastPacket* are the fields: *CommonNeighbourNo* and *NeighbourList*. Since these fields form part of *BBRPacket*, the reasoner is able to classify the latter under *MFRBroadcastPacket*. The *IdentifiedPacket* class denotes that the packet contains known fields. In this way, incoming packets can be classified by the ontology and be compared with existing packet formats. For example, assume the incoming packet to be *Broadcomm* and the existing packet to be BBR.

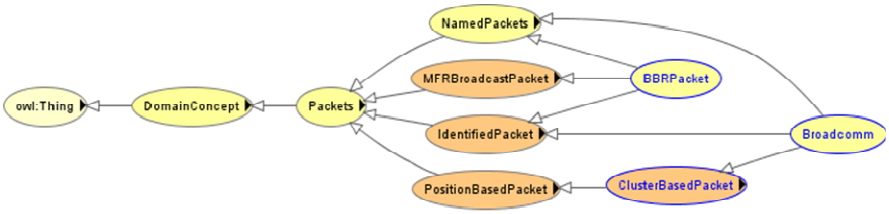

Fig. 7. Inferred Vehicular Ontology

Field Matching. Once both packets are classified, they can be compared to each other through an intuitive mechanism embedded in the ontology, which is the use of SWRL rules and SQWRL[12] query rules. These mechanisms add further reasoning to the classification process to enable field matching. As an example, the following SQWRL rule retrieves the fields from BBR and Broadcomm packets and tries to find the differences between them. To do so, it creates a collection of the fields of each packet using the SQWRL *makeBag* function and identifies the differences with the SQWRL *difference* function. The SQWRL clause is introduced within the SWRL rule by a separator character °. The SQWRL clause handles construction and manipulation operators required to execute SQWRL-based rules. As can be seen in the example below, the ° separator character enables a SWRL rule to include a SQWRL query.

BBRPacket(?b) ∧ hasFields(?b, ?f) ∧ Broadcomm(?p) ∧ hasFields(?p, ?pf) ˚ sqwrl:makeBag(?bag, ?f) ∧ sqwrl:makeBag(?bagt, ?pf) ˚ sqwrl:difference(?diff, ?bagt, ?bag) ∧ sqwrl:element(?e, ?diff) □ sqwrl:selectDistinct(?p, ?e)

The result of this query gives the fields required for BBR to function as Broadcomm and vice versa; the fields lacking in BBR would be *LocationCoordinates*, *TargetRoute* and *ClusterHead*. Moreover, further classification is also possible through the use of SWRL rules to reason about the data types of the fields. As an example, suppose we have a field x in BBR packet of type <int> and a corresponding field y in Broadcomm of type <String>. In this case, we can make use of a SWRL rule to suggest a mapping between these two fields:

[12] http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTab and …/wiki.pl?SQWRL

hasFields(BBR, ?x) ^ hasType(?x, <int>) ^ hasFields(Broadcomm, ?y) ^ hasType(?y,<String>)
 swrlb:MapIntToString(?x, ?y)

The OWL language enhanced with the use of SWRL and SQWRL enables comparison of two packets. The ontology can hence interpret the packet formats through matching and suggest a possible mapping between them. For example, the ontology can suggest that BBR lacks geographical coordinates in order to operate as Broadcomm. This information is fundamental in determining how to enable mapping between these two different types of packets. This is in itself a step forward towards interoperability between different network packets; however, further research is required into how ontologies can be used to generally identify mapping solutions that resolve the differences between packets. Further details about the use of ontologies within the domain of message-level heterogeneity are presented in [7].

5 Overall Reflections

Interoperability remains a fundamental problem for distributed systems due to the increasing level of heterogeneity and dynamism of the networking environment. In this paper, we have argued for a new approach to interoperability, i.e., *emergent middleware* that is synthesized on the fly according to the behaviour of the associated networked systems. A central element of our approach is the use of ontologies in the middleware design so that middleware may dynamically emerge based on semantic knowledge about the environment. Hence, while interoperability in the past has been about making concessions, e.g., pre-defined standards and design decisions, emergent middleware builds on the ability of machines to themselves reason about and tackle the heterogeneity they encounter. Further, acknowledging that interoperability is, as with many features of distributed systems, an end-to-end problem [5], emergent middleware emphasizes that interoperability can only be achieved through a coordinated approach involving application, middleware and network levels.

This paper has introduced the core elements of the emergent middleware vision, i.e., ontologies and related Enablers to reason about and implement interoperability on the fly. The architecture of Enablers outlined in Section 3 has provided a view of how emergent middleware can be realised, where associated technologies becoming available through the CONNECT project. This architecture illustrates the important roles of discovery, learning and synthesis in achieving our goals. The most notable feature of the architecture is that ontologies have a cross-cutting role. The experimental work reported in Section 4 has further illustrated the central role of ontologies in supporting meaning and reasoning in distributed systems, not just at the application level but also in the underlying distributed systems substrate, for achieving interoperability in the highly heterogeneous and dynamic style of today's distributed systems. However, despite the latest advances in Enablers for emergent middleware, significant challenges remain ahead as discussed below.

While emergent middleware relieves the burden of interoperability from the middleware designers and developers, and fosters future-proof interoperability, its general applicability is dependent upon the effectiveness of the supporting Enablers.

The latest results of CONNECT are encouraging in that they introduce base building blocks for the Enablers, spanning automated support for discovery, learning and synthesis. Small-scale experiments further demonstrate that Enablers may adequately be combined. Still, applicability to real-scale experiments is area for future work.

Realizing the central role of ontologies to allow machines to tackle interoperability across time raises the issue of how large, comprehensive ontologies may be deployed for interoperability in practice. At first sight, this basically depends on the development of supporting ontologies by domain experts and hence on the requirements of a given domain in terms of interoperability. For instance, it is expected that the Internet of Things will lead to major ontology development. Another consideration is the cost of processing large ontologies and, more specifically, the efficacy of semantic tools, which keep improving over time given research in the area. There is also considerable potential for core research on ontologies concerning the role of fuzziness in supporting richer forms of reasoning [21], the possibility of learning new ontological information and merging it with existing information as it becomes available, and also dealing with heterogeneity in the ontologies themselves.

We have so far concentrated on the synthesis of mediators from scratch, while the construction of mediators by composing existing ones would enable more efficient synthesis and support self-adaptive emergent middleware. Ongoing CONNECT research on an algebra for mediators will provide us the required foundations [4].

The inherent openness and probability of failure in emergent middleware solutions raise important challenges. If the solution is to be deployed at Internet scale, then it must be reliably able to produce correct mediators and also be secure against malicious threats. Hence, dependability is a central research question; this has to overcome the partial knowledge about systems as well as security concerns arising. A related concern is that of dealing with interoperability between fault tolerant systems and in general with the heterogeneity of non-functional properties across systems. Dedicated solutions are being investigated within CONNECT.

Furthermore, failing to generate emergent middleware in a specific context is not only dependent on the reliability of our solution, but also, most importantly in the open target environments, on the degree of incompatibility between candidate systems. For example, semantic matching may indicate that the semantic distance between the application features of two systems is too great to be bridged. Precisely evaluating the limitations of our approach in producing a result is an area of future work; we are already studying aspects of this important issue within CONNECT.

Another interesting research direction for emergent middleware is that of involving end-users in the synthesis process to inform the automated approach. For example, end-users can assist semantic matching where ontology heterogeneity may lead automated reasoning to ambiguous results. This raises various challenges, including how to provide user-friendly interfaces to the emergent middleware internals.

In summary, this paper has argued that, given the increasing complexity of contemporary distributed systems, both in terms of increasing heterogeneity and dynamism, there is a need for a fundamental rethink of approaches to even the most basic of problems, that is, interoperability. We advocate a new approach to middleware, that of emergent middleware. This paper has looked at one key aspect of emergent middleware, namely, the role of ontologies in supporting core underlying middleware functions related to achieving interoperability. This leads to a fascinating

set of research challenges both in terms of understanding a given deployment environment and also dynamically creating appropriate connectivity solutions. We hope this paper has given a flavour of the potential of this approach and also some real experimental evidence that the approach can work in selected aspects of distributed systems. As a final comment, while CONNECT is addressing a number of the ongoing challenges, this is a vast and largely unchartered territory and we invite other researchers to join in the quest for suitable solutions for emergent middleware.

Acknowledgements

This work is carried out in the CONNECT project, a European collaboration funded under the Framework 7 Future and Emerging Technologies Programme (Proactive Theme on ICT Forever Yours): http://www.connect-forever.eu.

References

1. M.W. Maier., "Architecting Principles for System of Systems," Systems Engineering, Vol. 1, No. 4, pp. 267-284, 1998.
2. M. Van Steen and A. Tanenbaum, *Distributed Systems: Principles and Paradigms*.: Prentice-Hall, 2001.
3. A. Bennaceur, G. Blair, F. Chauvel, G. Huang, N. Georgantas, P. Grace, F. Howar, P. Inverardi, V. Issarny, M. Paolucci, A. Pathak, R. Spalazzese, B. Steffen, B. Souville, "Towards an Architecture for Runtime Interoperability". In ISoLA 2010: 206-220
4. M. Autili, C. Chilton, P. Inverardi, M. Kwiatkowska, and M. Tivoli. Towards a connector algebra. In ISoLA 2010: 278-292.
5. H. Saltzer, D. P. Reed, and D. D. Clark. 1984. End-to-end arguments in system design. ACM Trans. Comput. Syst. 2, 4 (November 1984), 277-288
6. Object Management Group, "COM/CORBA Interworking Spec. Part A & B," 1997.
7. V. Nundloll, P. Grace, G. Blair, “The Role of Ontology in Enabling Dynamic Interoperability”, In Proceedings of the 11th IFIP International Conference on Distributed Applications and Interoperable Systems, Reykjavik, Iceland, June 2011
8. Y. Bromberg and V. Issarny, "INDISS: Interoperable Discovery System for Networked Services," in *Proceedings of the IFIP/ACM/Usenix International Middleware Conference*, Grenoble, France, 2005, pp. 164-183.
9. J. Nakazawa, H. Tokuda, W. Edwards, and U. Ramachandran, "A Bridging Framework for Universal Interoperability in Pervasive Systems," in *Proceedings of 26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006)*, Lisbon, Portuga, 2006.
10. C. Cortes, P. Grace, G. Blair, "SeDiM: A Middleware Framework for Interoperable Service Discovery in Heterogeneous Networks", ACM Transactions on Autonomous and Adaptive Systems, Volume 6, Number 1, Article 6:1-8, February2011.
11. P. Grace, G. Blair, and S. Samuel, "A Reflective Framework for Discovery and Interaction in Heterogeneous Mobile Environments," ACM SIGMOBILE Mobile Computing and Communications Review, vol. 9, no. 1, pp. 2-14, January 2005.
12. M. Duftler, N. Mukhi, S. Slominski, and S. Weerawarana, "Web Services Invocation Framework (WSIF)," in Proceedings of OOPSLA 2001 Workshop on Object Oriented Web Services, Tampa, Florida, 2001.
13. R. Masuoka, B. Parsia, and Y. Labrou, "Task Computing – the Semantic Web Meets Pervasive Computing..," in *Proceedings of the 2nd International Semantic Web*

Conference (ISWC2003), 2003.
14. S. Singh, S. Puradkar, and Y. Lee, "Ubiquitous Computing: Connecting Pervasive Computing Through Semantic Web," *Information Systems and e-Business Management Journal*, 2005.
15. S. Ben Mokhtar, D. Preuveneers, N. Georgantas, V. Issarny, and Y. Berbers, "EASY: Efficient SemAntic Service DiscoverY in Pervasive Computing Environments with QoS and Context Support," Journal of Systems and Software, 8:5(785-808), 2008.
16. Y. Bromberg, P. Grace and L. Reveillere, "Starlink: runtime intereoperability between heterogeneous middleware protocols," in *Proceedings of the 31st IEEE International Conference on Distributed Computing Systems*, Minneapolis, USA, June 2011.
17. D. Martin, M. Burstein, D. Mcdermott, S. Mcilraith, M. Paolucci, K. Sycara, D. L. Mcguinness, E. Sirin, and N. Srinivasan. Bringing semantics to web services with OWL-S. *In World Wide Web Journal*, 10:243-277, September 2007
18. D. Athanasopoulos, A. Zarras. "Fine-Grained Metrics of Cohesion Lack for Service Interfaces," *In Proc. of ICWS 2011* (to appear).
19. A. Bennaceur, R. Johansson, A. Moschitti, R. Spalazzese, D. Sykes, R. Saadi, and V. Issarny, "Inferring affordances using learning techniques," in *International Workshop on Eternal Systems (EternalS'11)*, 2011.
20. S. B. Mokhtar, P.-G. Raverdy, A. Urbieta, and R. S. Cardoso. Interoperable semantic and syntactic service discovery for ambient computing environments. *IJACI*, 2(4):13-32, 2010
21. U. Straccia. A Fuzzy Description Logic for the Semantic Web. In E. Sanchez, editor, Fuzzy Logic and the Semantic Web, Capturing Intelligence, chapter 4, pages 73–90. Elsevier, 2006
22. A. Heß and N. Kushmerick. Learning to attach semantic metadata to web services. *In International Semantic Web Conference*, pages 258-273, 2003
23. I. Krka, Y. Brun, D. Popescu, J. Garcia, and N. Medvidovic. Using dynamic execution traces and program invariants to enhance behavioral model inference. *In ICSE (2)*, pages 179-182, 2010
24. A. Bertolino, P. Inverardi, P. Pelliccione, and M. Tivoli. Automatic synthesis of behavior protocols for composable web-services. *In ESEC/SIGSOFT FSE*, pages 141-150, 2009
25. M. Caporuscio, P.-G. Raverdy, H. Moungla, and V. Issarny. ubisoap: A service oriented middleware for seamless networking. In ICSOC, pages 195-209, 2008
26. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003
27. M. Merten, B. Steffen, F. Howar, and T. Margaria. Next generation learnlib. *In TACAS*, pages 220-223, 2011
28. V. Issarny, A. Bennaceur, and Y.-D. Bromberg. Middleware-layer connector synthesis: Beyond state of the art in middleware interoperability. *In SFM-11*, 2011
29. N. Drummond, A. L. Rector, R. Stevens, G. Moulton, M. Horridge, H. Wang, and J. Seidenberg. Putting OWL in order: Patterns for sequences in OWL. *In OWLED*, 2006
30. R. Vaculin and K. P. Sycara. Towards automatic mediation of OWL-S process models. *In Proceedings of ICWS*, 2007
31. S. K. Williams, S. A. Battle, and J. E. Cuadrado. Protocol mediation for adaptation in semantic web services. *In ESWC*, pages 635-649, 2006
32. E. M. Clarke. Jr., O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999
33. F. Howar, B. Jonsson, M. Merten, B. Steffen, and S. Cassel. On handling data in automata learning - considerations from the connect perspective. In ISoLA (2), pages 221-235, 2010
34. Zhang, M and Wolf, R. "Border Node Based Routing Protocol for VANETs in Sparse and Rural Areas". IEEE Globecom Autonet Workshop (Washington, Nov. 2007). 1-7.
35. Durresi, M., Durresi, A., and Barolli, L. 2005. Emergency Broadcast Protocol for Inter-Vehicle Communications. In Proc. 11th International ICPADS Conference Workshops, 402-406.