

Adapting Agent Communication Languages for Web Service to Web Service Communication

Steven Willmott¹, Félix Oscar Fernández Peña¹, Carlos Merida-Campos¹, Ion Constantinescu²

¹ Llenguatges i Sistemes Informàtics,
Universitat Politècnica de Catalunya,
Modul C5 / C6 Campus Nord, UPC
Barcelona, Spain, E-08034
 {steve, ffernandez, dmerida}@lsi.upc.es

² Ecole Polytechnique Fédérale de Lausanne,
Lausanne, Switzerland, CH-1015
 Ion.constantinescu@epfl.ch

Abstract. The underlying goal of the development of semantic representations for Web Services is to provide increased explicit representations of meaning in Web Services applications. One challenging area where this might be applied is in the message exchanged between Web Services - in other words, semantic definitions of the meanings of data / commands exchanged in the execution of a Web Services based application. While current approaches such as OWL-S tackle these elements in service groundings by mapping processes to function calls with specific arguments, Agent Communication Languages could provide a potentially richer alternative. The work presented here and on the associated web page aims to explore how this could be done by showing how an existing Agent Communication Language (FIPA-ACL, FIPA-SL and associated standards developed by the Foundation for Intelligent Physical Agents<a> could be mapped into representations which could be readily used in a Web Services environment.

1 Introduction

The underlying goal of the development of semantic representations for Web Services (and indeed the Semantic Web in general) can be characterised as *making the implicit explicit*. In other words codifying knowledge about how services work, what they do and what inputs/outputs/processes *mean* in order that developers (or other Web Services) might reason about them. The primary focus of this effort to date has been on the use of such semantic descriptions in order to facilitate service discovery and composition – to allow human developers (and potentially other Web Services) to identify services to compose and subsequently sequence their execution.

Related to the service composition however is the problem of how the composed Web Services *actually communicate with one another* (sometimes referred to as their

grounding) in order to carry out their common tasks: the sequence of messages required, the meaning of these messages and their encoding. Whilst Semantic Web Services markup languages such as OWL-S provide structures to express service groundings they remain relatively simple compared to the increasing sophistication of process models and profiles of the services themselves.

The objective of this paper is to outline how commonly used language structures from the Agent research community might be adapted to enrich Web Service to web Service communication semantics by:

- Re-using a well established structure for messages and long running structured conversations.
- Capturing interactions in terms of speech-act style semantics commonly used in Agent Communication Languages research and linguistics.
- Adapting existing languages to encodings compatible with Semantic Web Languages such as RDF and OWL as well as Web Services transport layers such as SOAP.

The paper presents an OWL/RDF over XML encoding of the well known Foundation for Intelligent Agents (FIPA) (www.fipa.org) language structures and demonstrates how these could subsequently be used over SOAP messaging as well as simply specified in WSDL bindings. All ontologies and definitions referenced in this paper are publicly available under an open source LPGL license at the following URL http://www.lsi.upc.es/~steve/download/SWS_FIPA_ACL_04.

2 A Case for Generic Web Service to Web Service Communication Languages

OWL-S [OWLS04] (and the new WSMO* technology stack [WSMO04]) are the two leading approaches towards increasing the explicit semantics associated with Web Services – in particular they focus on providing formalisms for the description of the behaviour of the services, pre-conditions, post-conditions, possible effects, its domain of execution (captured in a domain ontology). In addition they provide structures for expressing how the Services can subsequently be invoked by defining a set of functions/methods for each service as well as inputs, outputs which are bound to domain objects defined in domain ontologies.

Whilst these definitions already provide useful information they arguably miss out on a number of areas where more explicit semantics could be useful. These are:

- The semantics of the method calls themselves (the objective of the call) – in particular the intent of the entity calling the method is captured only in the name of the method and an indirect association with the process it might trigger.

- Semantics of a lengthy sequence of messages as part of a single interaction with a Service – currently groundings are narrowed down to atomic request-response interactions.
- The specification of relationships between the arguments – which can only be inferred from the processes the message triggers and also can only be re-defined by defining a new method/function.

In particular, given a particular method call between two Web Services the interpretation of the meaning of the message would depend heavily on the specification of the process models invoked by the call (which may not always be available).

Lastly, a significant amount of redundancy is likely across many applications since at a high level of abstraction many functions/methods in Service descriptions from different domains are likely to have similar intuitive meanings – requests, informational statements, acknowledgements and so forth. These additional semantic elements are likely to become increasingly critical:

- As the level of automation of Semantic Web Service composition and execution increases.
- As the level of Semantic Web Services reuse and interoperability is required to increase.

An extract from the OWL-S Bravo Air example illustrates this in Figure 1. Whilst the model provides a mapping from inputs to a service down to groundings in terms of WSDL functions:

- It does not directly define the intent of the entity invoking the service – the meaning is only defined in terms of the possible actions of the recipient of the method call.
- The interactions are atomic request-responses, mapping from a set of inputs to a set of outputs.
- The function definitions are necessarily domain / application specific – tied to the specific process models they relate to – and thereby difficult to re-use across applications.

From 1.1B/BravoAirProfile.owl

```
<profile:hasInput rdf:resource="&ba_process;#DepartureAirport"/>
<profile:hasInput rdf:resource="&ba_process;#ArrivalAirport"/>
...
<profile:hasResult rdf:resource="&ba_process;#HaveSeatResult"/>
```

From 1.1B/BravoAirProcess.owl

```
<process:Input rdf:ID="DepartureAirport">
<process:parameterType rdf:datatype="&xsd;#anyURI">
  &concepts;#Airport
</process:parameterType>
</process:Input>
```

From 1.1B/concepts.owl

```
<owl:Class rdf:ID="Airport">
```

...
</owl:Class>

From 1.1B/1.1B/BravoAirGrounding.owl

```
<grounding:wSDLInput>  
  <grounding:WSDLInputMessageMap>  
    <grounding:owlsParameter rdf:resource="&ba_process;#DepartureAirport"/>  
    <grounding:wSDLMessagePart rdf:datatype="&xsd;#anyURI">  
      &BravoAirGroundingWSDL;#departureAirport  
    </grounding:wSDLMessagePart>  
  </grounding:WSDLInputMessageMap>  
</grounding:wSDLInput>
```

From 1.1B/BravoAirGrounding.wsdl

```
<message name="GetDesiredFlightDetails_Input">  
  <part name="departureAirport" owl-s-parameter="BravoAir:#departureAirport_In"/>  
  <part name="arrivalAirport" owl-s-parameter="BravoAir:#arrivalAirport_In"/>  
  <part name="outboundDate" owl-s-parameter="BravoAir:#outboundDate_In"/>  
  <part name="inboundDate" owl-s-parameter="BravoAir:#inboundDate_In"/>  
  <part name="roundTrip" owl-s-parameter="BravoAir:#roundTrip_In"/>  
</message>
```

Figure 1: Sequence of interface related definitions from DAML-S coalition Bravo Air Model (<http://www.daml.org/services/owl-s/>), version 1.1 Beta.

In order to increase the explicit semantics associated with the inter Web Services messages themselves however requires a significant effort in language design. While for any given domain it may be possible to design a language (set of objects, function calls, statements) which capture the needs of that domain there are strong motivations for seeking generic languages for some aspects of the task which cover a least several different domains. Chief amongst these are:

- **Language re-use:** many of the higher level elements of communication – the different types of attitude expressed in a message (request, query, inform, ...) and management of sequences of messages are common across many applications.
- **Interoperability:** applications using the same generic communication structures instantiated for different domains are more likely to be able to interoperate at some future date – the semantics of some aspects of their communication is shared.
- **Engineering Considerations & Tooling:** engineers working on multiple projects are likely to favour using similar tools and techniques in each of them – improving their modelling skills rather than having a new framework to apply in each case.

Lastly, generic languages for communication also make it more likely domain ontologies for specific applications will be re-usable: encouraging designers to separate out abstract, general communication elements from true domain specific semantic knowledge.

3 Adapting Agent Communication Languages

Although there is a large range of languages that might be used between web services (languages such as SWRL [SWRL04], ebXML Messages or raw XML/RDF are often mentioned) an important class of languages to be considered are those developed in the course of Agent Communication Languages which aim to solve an exactly analogous problem – arbitrary, semantically well founded, communication between autonomous systems.¹ In particular we consider here speech act [Austin62] based paradigms and those approaches which have a well developed structure such as FIPA-ACL and KQML with KIF. The main motivations for building on these languages are:

1. They are specifically designed to structure arbitrary communication and break up the challenge into several levels, often: communication context, protocols (sequences of messages), speech acts, content expressions and domain references.
2. Each level is associated with clear language semantics.
3. The languages often provide links with higher levels of agent coordination patterns (such as contract nets or patterns as in COOL [COOL95]) which might subsequently be exploited by Web Service application designers.
4. The properties of these systems (both positive and negative) are well documented in an extensive literature.

Furthermore in the long run work such as [Colombetti00] and [Wan03] on the nature of commitments, public statements of belief / fact and other social properties are likely to become increasingly important for cross-organisational Web Services applications.

4 Web Services Communication Framework

In order to demonstrate how existing Agent Communication Languages (ACLs) could be used as generic languages for Web Services, this section outlines a simple communication framework based on FIPA language specifications, which are some of the best known ACL specifications. In overview the architecture is constructed as follows:

- A number of OWL language ontologies which capture the grammar of the languages FIPA-ACL, FIPA-SL and a number of supporting elements.
- XML message representations for messages in these languages from the OWL specifications.
- A logical message structure specification for messages to be carried over SOAP transport mechanisms.
- A simple WSDL specification for SOAP endpoints supporting the message interface.

¹ Note that this does not exclude the previously mentioned languages since they may be recast in a new role.

Implicit in this model are mechanisms for interaction protocol management (sequences of messages) through the FIPA Message structure and bindings to domain ontologies (through the use of content languages such as FIPA-SL). It should also be noted that the model is *different* to the existing FIPA specifications for an XML syntax for FIPA-ACL [FIPA00071]² and RDF content language [FIPA00011], since neither of these are currently directly Web Services or OWL compatible.

4.1 FIPA-ACL and FIPA-SL as OWL ontologies

For this illustration we focus solely on FIPA-ACL and FIPA-SL, the IOP and HTTP transport layers are not treated here since they are replaced by the use of SOAP (see next section) and the UML/AUML protocols are not covered since they are designed to describe protocols which can be enacted using FIPA-ACL messages (rather than being explicitly used in communication).

The method by which the languages were repurposed was to describe each in its own language ontology, alongside a number of utility/intermediate ontologies - creating the hierarchy of ontologies, by importing some into others, as represented in the figure 2.

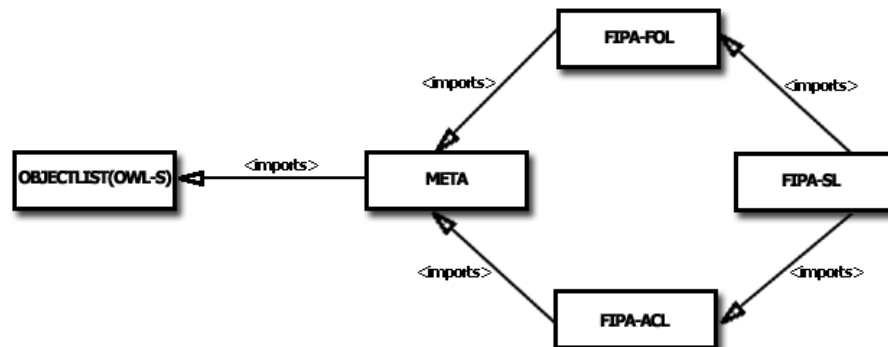


Figure 2: Ontology, the META ontology includes a number of basic concepts such as Action and Result (which could eventually be taken from something like OWL-S), FOL represents a simple first order logic language, the FIPA-ACL and SL ontologies extend these. Substituting OWL lists with OWL-S ObjectList.owl made it possible for all ontologies to OWL-Lite.

The ontologies together include 120 classes, in a 7 deep hierarchy. Many of the concepts are operators (such as +, - etc.), others are grammatical constructs such as “functional term” which play a particular role in statements. A complete graphical overview generated using the OWL plugin of Protégé is available on the web page associated with the paper. An example of messages created in this way is shown below; the message is a query for a Car in FIPA-ACL and FIPA-SL:

² Note – throughout FIPA Specifications are referenced by their sequence number – they can all be retrieved on the basis of this number from <http://www.fipa.org>.

```
(query-ref
 :sender (agent-identifier :name B)
 :receiver (set (agent-identifier :name A))
 :content "(iota ?x (Car ?x))"
 :language fipa-sl
 :reply-with query1)
```

The algorithm for transforming this message matches each term of the FIPA message to an automatically generated instance of the corresponding class of the language ontologies. For instance, query-ref is translated to:

```
<acl:QueryRef rdf:ID="AG001">
<acl:replyWith rdf:datatype="http://www.w3.org/2001/XMLSchema#string">query1</acl:replyWith>
<acl:receiver rdf:resource="#AG004"/>
<acl:language rdf:datatype="http://www.w3.org/2001/XMLSchema#string">fipa-sl</acl:language>
<acl:sender rdf:resource="#AG002"/>
<acl:content rdf:resource="#AG011"/>
</acl:QueryRef>
```

So, the complete XML message, to be sent in the BODY of the SOAP message, is composed of classes instances and properties. As showed in the example, each property value of FIPA message elements is assigned to the corresponding property of the instantiated class. Meanwhile, instances identifiers are generated automatically and used to establish relations between instances. For instance,

```
<acl:receiver rdf:resource="#AG004"/>
```

...states that the instance identified as 'AG004' represents the receiver of the 'AG001' 'Query-Ref'.

Analyzing another part of the FIPA message been processed, FIPA-SL establishes that 'Iota' identifying expression has two parameters, a variable -x, in this case- and a Well Formed Formula (WFF). This is expressed in the language ontology, and is the reason to generate the instance of the class Iota as:

```
<Iota rdf:ID="AG005">
<firstIdentExpr rdf:resource="#x"/>
<secondIdentExpr rdf:resource="#AG006">
</Iota>
```

'secondIdentExpr', as property of the instance 'AG005', of type Iota, establishes the link to the instance 'AG006'. This link, as restricted by the FIPA-SL ontology should be established to an instance of WFF class:

```
<WFF rdf:ID="AG006">
<domainRef rdf:ID="AG014"/>
</WFF>
```

Generated instance of WFF shows how 'domainRef' property is used for establishing the link to a domain ontology class instance -in this case, an instance of the class Car, identified by the ID 'AG014'-, that is expressed in the message as:

```
<car:Car rdf:ID="AG014"/>
```

Including the 'domainRef' property in this instance of WFF class expresses that the

domain ontology class 'Car' should be considered as a WFF, in the evaluation of this FIPA message. This makes possible to change the meaning of domain classes for the processing language, as needed without changes in the implementation and ontologies' definition. So, with an iterative process, the whole FIPA message is parsed and an XML well formed document is generated –the complete XML document associated to this example can be found on the web page associated with the paper.

4.2 Agent messages over SOAP

The messages are subsequently transported over standard SOAP (1.0 or 1.1) transport mechanisms. The structure is illustrated by a possible response to the query message in the previous section which in FIPA-ACL/FIPA-SL would be:

```
(inform
:sender (agent-identifier :name A)
:receiver (set (agent-identifier :name B)
:content
"((= (iota ?x (Car ?x)) (Car :colour red))) "
:language fipa-sl
:in-reply-to query1)
```

Over SOAP the resulting message would be:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
<rdf:RDF
xmlns:fol="http://cerebro.lsi.upc.es/ontology/fipaFOL.owl#"
xmlns:acl="http://cerebro.lsi.upc.es/ontology/fipaACL.owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:meta="http://cerebro.lsi.upc.es/ontology/meta.owl#"
xmlns:car="http://cerebro.lsi.upc.es/ontology/Car.owl#"
xmlns="http://cerebro.lsi.upc.es/ontology/fipaSL.owl#"
xmlns:ObjectList="http://www.daml.org/services/owl-s/1.0DL/generic/ObjectList.owl#"
xml:base="http://cerebro.lsi.upc.es/ontology/fipaSL.owl">
<acl:Inform rdf:ID="AG007">
<acl:language rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
fipa-sl</acl:language>
<acl:receiver rdf:resource="#AG008"/>
<acl:inReplyTo rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
query1</acl:inReplyTo>
<acl:sender rdf:resource="#AG003"/>
<acl:content rdf:resource="#AG012"/>
</acl:Inform>
<acl:AgentIdentifierSet rdf:ID="AG008">
<ObjectList:first rdf:resource="AG002"/></acl:AgentIdentifierSet>
<Term rdf:ID="AG010">
<domainRef rdf:resource="#AG013"/>
</Term>
<ContentExpressionSet rdf:ID="AG012">
<ObjectList:first rdf:resource="#AG009"/>
```

```

        </ContentExpressionSet>
        <Equal rdf:ID="AG009">
        <secondBinaryTerm rdf:resource="#AG010"/>
        <firstBinaryTerm rdf:resource="#AG005"/>
        </Equal>
        <car:Car rdf:ID="AG013">
        <car:colour rdf:datatype="http://www.w3.org/2001/XMLSchema#string">red<car:/colour>
        </car:Car>
    </rdf:RDF>

</soapenv:Body>
</soapenv:Envelope>

```

With a separation between the SOAP envelope, the namespace definitions (which serve to link to related ontologies) and the message content again encoded using the OWL ontologies introduced in the previous section.

4.3 WSDL encodings

The framework uses non RPC, document processing web services methods, establishes a synchronous messaging model between the requester and the replier. This way, web services implement methods matching with an AXIS method pattern such as:

```
public void method(SOAPEnvelope req, SOAPEnvelope resp)
```

This implies the inclusion of an input parameter (responseMRequest) and an output parameter (responseMResponse) in the WSDL, for conformity with the AXIS architecture. To retain the asynchronous nature of Agent communication, the output parameter is used just as an acknowledgement of receipt of the message (and does not contain application relevant content).

Meanwhile, a new thread is created in the web service method for the actual processing of the received message. This child thread stays alive, as long as necessary, until the end of the message's processing, when it delivers the response to the web service that made the request, or until other thread take control of the processing. (Note that in high throughput situations thread pooling or request queuing could be implemented – the essential notion here is that the Web Service processes the input in its own time – rather than in the interval between method call and response.)

An example WSDL file with full details can be found on the paper website.

5 Proof of concept implementation using AXIS and JENA

The specifications provided in the previous section were verified using a simple implementation of Web-Service to Web-Service communication using the freely available AXIS and JENA toolkits:

- AXIS provides an API for SOAP messaging. It allows the development of Java Classes as web services and provides tools for writing client Java programs that use web services.
- JENA is a Java framework for building Semantic Web applications. It provides a programmatic environment for OWL, including a rule-based inference engine.

These tools were combined within a simple communicating web service implementation in Java as shown in Figure 3. Each web service is able to exchange messages with others using the JENA API to manipulate message structures. It is important to note in this figure that the services created fit the definition of a web services – they are simply web services *which use a specific language to communicate* rather than arbitrary XML encoded document.

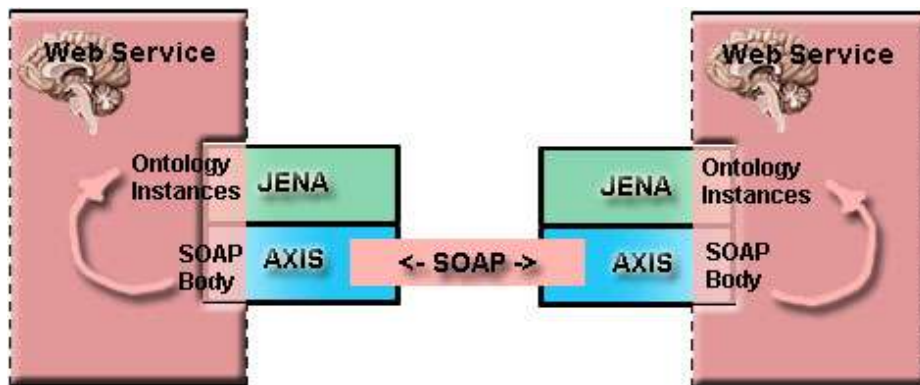


Figure 3: FIPA-ACL speaking Web Services using JENA over AXIS.

Using the prototype, test messages have been exchanged between several Web-Services instances including the sample messages in the previous section and those available on the web page.

Since the communication is fully standards based, the system should be able to communicate in any case where SOAP can currently be used; a similar endpoint could also be implemented using another SOAP toolkit such as Java Web Services Developer Pack from SUN (JWSDP) or Systinet deployment platform for Web Services, by implementing the same WSDL described end point. At the RDF level generalisation should also be straight forward – since all communication is based on shared language and domain ontologies, in principle, any OWL (OWL-DL) aware RDF processor could be used to manipulate messages. Message manipulation done with simpler tools such as RDF or XML processors is possible– in these cases extra layers of processing would be needed in the web-services core itself to ensure the OWL/RDF etc. constraints are met.

5.1 Discussion

The framework described in the previous section represents one way of porting the FIPA Communication specifications into a Web Services environment, however it is certainly not the only one and a number of the design choices made along the way are potentially interesting. A number of these are discussed in this section. In the long run if there is significant interest discussion could take place in related fora such as FIPA, SWSI or elsewhere.

Specifying Grammar versus Semantics

When attempting to map existing agent languages to other representations one major choice is *which elements of the language to capture explicitly*. In particular choices were available – focusing on capturing the actual semantic description of elements of the language (e.g. using OWL to specify the meaning of something like “inform” or “request”) or focusing on the grammatical structure of the language (i.e. using OWL to capture that an “inform” is a type of speech act, the its content must always be a proposition and so forth).

In the long run both elements may be of interest; however the model presented in this paper focuses solely on the grammatical structure for the following major reasons:

- Modelling the grammar is necessary as a first step in order to create a well defined structure for messages – without this, developers would be unable to create well formed messages in the language.
- Since there remains disagreement on how best to capture language semantics, a grammatical approach provides structure and intuitive guidelines to developers without committing to a precise Semantic Model.
- The same approach could relatively easily be adapted to other, similar languages.

OWL / RDF as a grammar specification

Making the above choice immediately raises the question as to whether OWL / RDF are in fact suitable frameworks for expressing grammatical / parsing constraints. Analysis of the ontologies produced clearly shows that they are certainly not fully adequate for this task:

- Grammatical constraints are split up into arbitrary sequences in the OWL file.
- Certain EBNF grammar constructs used in the original FIPA S-expression grammar (such as multiple expansions with different structures) result in clumsy definitions and possible ambiguities.
- RDF processors remain significantly less efficient than XML processors due to the reduction in structure in most RDF documents.

Despite these drawbacks there are some positive elements – modelling the grammar at this level provides an immediate link between language elements (terms, operators, performatives etc.) and elements in domain ontologies. Secondly, receiving Web Services receive a parse tree which treats all elements equally – annotating them as to which ontology (language or domain) they are referenced in. This provides a powerful

model for subsequent semantic processing – in which the same data structure can be manipulated using the different semantic models from each language.

Hence although the option of mapping messages into (for example) XMLS could be considered, it would be important to retain the direct relationship with the language and domain ontologies.

Linkage between Language and Domain Ontologies

A major issue in language design was how to mix elements from different ontologies in order to preserve class relationships. An example of this problem would be statement (FIPA-SL) such as:

```
(car
  :colour      "red"
  :registration "X123 ABD"
)
```

which in FIPA-SL grammatical terms is treated as a *functional term*, but also represents an instance of class defined in a domain ontology about *cars*. The primary possibilities for making this link were:

1. To insist that all domain ontologies used with the language ontologies be marked up with additional statements as to the language structures they may represent (in-lining the link in the domain ontology) or vice versa insisting language ontologies reference elements in domains they may be used with (in-lining in the language ontology).
2. To create a separate set of “linking ontologies” which link a specific domain ontology to a language. Less restrictive than the previous option, but also very complex to manage.
3. In-lining linking declarations in the messages delivered as they are sent by adding additional class statements valid only for the scope of the message to make such grammatical links.
4. In-lining additional statements which replace a grammatical construct with an instance reference to an element from a domain ontology.

The first two options quickly become unmanageable in realistic context – since they require additional elements of class knowledge to be generated and managed. The third approach is also complex since it requires class knowledge and instance knowledge to be mixed in the same message. In the current framework the choice was to circumvent the issue by allowing any grammatical element to be replaced by a domain reference to an instance of an object from a domain ontology.

As can be seen from the examples in the previous section where the grammar might find a functional term a link to the instance of a car has been inserted. This allows processing tools to construct coherent trees from messages which bottom out in domain terms. However it also ducks the issue of type checking and whether or not a particular domain object would be a valid statement at that point in the message. Communicating agents would need to employ pre- and post-processing on messages to ensure language statements were correctly applies to object instances. The overall reasoning

impact of this choice has not been fully studied, however it does provide a straight forward manner in which to communicate without resorting to additional class knowledge manipulation.

5.2 Longer Term Issues

In addition to the discussion items in the previous section there are a number of longer term issues which would need to be taken into account for any wider usage. In the design of language the following issues often arise:

- How general should language(s) be?
- What level of expressivity should they have / not have?
- Is it one language that is needed? Or a structured set? Or should “many bloom” for the market to decide?

In addition to these more general questions however there are a number of more specific issues which must be carefully considered in creating generic languages for Web-Service to Web-Service languages.

- Integration of other languages (particularly content languages): it is inevitable that most applications will need to pick and mix a number of languages for applications – however semantic models are often wildly distinct. How can a generic framework cope with different elements of the same communication framed in different ways?
- Distinction between grammar and semantics: language grammars often reflect semantics (adding constraints on statements which prohibit some of the incoherent statements sometimes possible), however they rarely capture it fully. Often semantics themselves are captured in another, separate formalism. How far should we go in modelling one, the other or the intersection between them both?
- Convergence on a semantic model: for applications which span multiple organisations it would be of very significant benefit to agree on the semantics of arbitrary or at least some possible statements in the domain – so that post-hoc participants can pinpoint the responsibilities of all parties involved. How can reach this kind of consensus given the large number of approaches to semantic models.
- Integration with WSDL / OWL-S groundings: one potential benefit of using generic languages may be that service descriptions can be significantly reduced – instead of specifying precise method calls statements can be made on which languages are support, which ontologies and which protocols – leaving agents free to generate messages which fit within these constraints to communicate with one another. However this presupposes agents have a common interpretation of semantics of arbitrary statements in the languages in question (i.e. it may be possible to phrase the same request several times) – how can we structure languages to avoid computational intractability and allow practical use?

7 Conclusions

The paper discussed how the explicit semantics of messages used in communication between Web Services could be enriched by moving from a method-call + arguments model to one which used models based on current Agent Communication Languages to which explicitly specify the meaning of all elements of a message. Also included is approach to creating candidate languages for such communication based on the FIPA Agent standard is presented with a proof of concept. In most applications a balance between the current method-based approach and the more declarative ACL approach may need to be struck and there appears to be scope for interesting discussion as to how this might be achieved.

The language ontologies and service descriptions referred to in this paper can be found online at URL http://www.lsi.upc.es/~steve/download/SWS_FIPA_ACL_04 and are reusable under LPGL open source licenses (versions referred to here are the 1.0 versions). The JENA / AXIS implementation is available on request. Ongoing work currently focuses on:

- Extending the for use in the Agentcities / openNet network testbed (<http://www.x-opennet.net/>) deployment in order to further test it and generate discussion within the community on potential further development.
- Development of a simple application involving in the order of 10-20 services to demonstrate the language structures which would emerge in realistic domain.

Furthermore, if there is significant interest the specifications could be provided as inputs for standardization in appropriate bodies and/or user groups.

Acknowledgements

This work was partly supported by the European Projects ASPIC (www.argumentation.org), @lis technology net (www.alis-technet.org) and owes much to discussions with colleagues in the Agentcities and openNet Initiatives. Notwithstanding this the opinions expressed in the paper are those of the authors and do not necessarily reflect those of other project participants. Particular thanks go to Jonathan Dale, Luigi Cecaroni and David Cabanillas for their helpful comments.

References

- [OWLS04] “OWL / DAML Services Homepage”, <http://www.daml.org/services/owl-s/>, Darpa Agent Markup Language Consortium, 2004
- [WSMO04] “Web Services Modelling Ontology”, <http://www.wsmo.org/>, SDK WSMO Working Group, 2004

- [SWRL04] “A Semantic Web Rule Language Combining OWL and RuleML”, <http://www.daml.org/2003/11/swrl/>, Darpa Agent Markup Language Consortium, 2004
- [Austin62] “How to do things with words”, Austin, J. Clarendon Press, Oxford, 1962.
- [COOL95] “Cool: {A} language for describing coordination in multiagent systems”, M. Barbuceanu and M. S. Fox, Proceedings of the First International Conference on Multi-Agent Systems, AAAI Press, 1995.
- [Colombetti00] “Semantic, Normative and practical aspects of agent communication”, Colombetti, M. In Dignum, F and Greaves. M. (eds), Issues in Agent Communication, LNAI vol. 1916, pages 17-30, 2000.
- [Wan03] Feng Wan and Munindar P. Singh, “Commitments and Causality for Multiagent Design”, Proceedings of the 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS), Melbourne, ACM Press, July 2003