An experience report on using DAML-S

Marta Sabou
Dept. of Artificial Intelligence
Vrije Universiteit
Amsterdam, The Netherlands
marta@cs.vu.nl

Debbie Richards
Div. of I.C.S.
Macquarie University
Sydney, Australia
richards@ics.mq.edu.au

Sander van Splunter
Dept. of Artificial Intelligence
Vrije Universiteit
Amsterdam, The Netherlands
sander@cs.vu.nl

ABSTRACT

Though DAML-S is growing into a *de facto* standard for semantic web-service markup, we have only found few complete service descriptions and even less papers discussing technical issues about the markup process. We addressed this lack by (1) reporting on our experiences in describing a set of services, (2) concluding several limitations of the latest DAML-S version (v0.7) and (3) making our work accessible to the research community[1].

1. INTRODUCTION

DAML-S is an initiative of the Semantic Web community to facilitate automatic discovery, invocation, composition, interoperation and monitoring of web-services (WSs) through their semantic description[4]. DAML-S is a DAML+OIL ontology conceptually divided into three sub-ontologies for specifying *what a service does?* (Profile[2]), *how the service works?* (Process[3]) and *how the service is implemented?* (Grounding[4]).

We decided to apply the latest release (v0.7) to one of our WS projects. Our preliminary literature study yielded four types of usage scenarios. First, within the DAML-S coalition two complete, fictitious examples (on the DAML-S site) and two concrete applications[7] are provided. Second, several projects use only certain parts of the DAML-S ontology, eg. matchmaking research tends to focus on the Profile ontology [3, 8]. Third, we found work which extends parts of DAML-S: [2] enriches the Process ontology, [9] extends the Profile ontology with bio-informatics related properties, [6] extends the specification of conditions. Finally, some papers mention use of complete DAML-S as is, but the purpose of the papers was to describe other research work. Common to all the above referenced papers is that none of them describe the process of writing the DAML-S markup. We were also concerned that very little of the DAML-S markup we found pointed to actual services.

We sought to fill this gap by providing a set of complete, real web service descriptions and sharing our modelling experiences. For space, this paper only presents the two simplest services we marked up and discusses a set of emerging, noteworthy issues.

A brief description of DAML-S, WSDL and our application scenario (section 2) is followed by our practical experiences in describing increasingly complex services (3), a discussion of emerging global observations (4) and our conclusions (5).

[1]http://www.cs.vu.nl/~marta/services/
[2]http://www.daml.org/services/daml-s/0.7/Profile.daml
[3]http://www.daml.org/services/daml-s/0.7/Process.daml
[4]http://www.daml.org/services/daml-s/0.7/Grounding.daml

WWW2003, May 20–24, 2003, Budapest, Hungary.
ISBN 963-311-355-5.

2. TECHNOLOGY AND CASE STUDY

2.1 DAML-S

The information about a DAML-S Service instance is distributed in three main entities. We will explain these entities and introduce the schematic service representations we use through this paper.

A service presents a *ServiceProfile* which describes the capabilities of the service for discovery purposes. The Profile ontology contains the vocabulary to describe the ServiceProfile. Its central concept, Profile, is a subclass of ServiceProfile and contains the contact information of providers, an extensible set of service characteristics and a functionality description by specifying the inputs, outputs, preconditions and effects of the service (IOPE's). It also points to the described Process. In this paper, for each Profile instance we depict the process it describes (hasProc) and its functional characteristics (IOPE's) together with their type. In the example below, the Bib2Rdf service presents the Pr1 Profile, which describes Process P1, has a BibFile input and an RdfFile output.

```
Service Bib2Rdf:
 *Profile:Pr1 (hasProc P1)(I(BibFile), O(RdfFile))
 *ProcessModel:
    AtomicProcess:P1 (I(BibFile),O(RdfFile))
 *WSDLGrounding:
    WsdlAtomicProcessGrounding: Gr1 (P1->op1)
 *WSDL:
  Service(PortType:Translator (
        op1 (IMsg(url), OMsg(stream))))
```

A *ServiceModel* describes the internal workings of the service and it is further specialized as a ProcessModel in the Process ontology. A ProcessModel has a single Process which can be atomic, simple or composite (composed from atomic processes through various control constructs). Each Process has a set of IOPE's. In our notation, for each service we represent its ProcessModel with its Process. For each Process we depict its type, the involved control constructs, the IOPE's and their types. Bib2Rdf has a single AtomicProcess P1, with a BibFile input and an RdfFile output.

A *ServiceGrounding* links the abstract description of the service to actual implementation details, such as message exchange formats and network protocols so that automatic invocation of the service is possible. Currently support is offered for grounding DAML-S to WSDL service descriptions. The Grounding ontology specializes the ServiceGrounding as WSDLGrounding which contains a set of WsdlAtomicProcessGrounding elements, each grounding one of the atomic processes specified in the ProcessModel. We depict each atomic process grounding by showing the link between the atomic process and the corresponding WSDL element. Bib2Rdf has a single atomic process grounding where Process P1 is grounded to op1, which is a WSDL operation. See the next subsection.

2.2 WSDL

WSDL is an industry standard for describing web-accessible services. As an XML-based language, it is machine processable, being a structured and standardized way to describe web-interfaces of services. However the intended meaning (semantics) of the interface elements is only accessible to human readers.

In WSDL a service is seen as a collection of network endpoints which operate on messages. A WSDL document has two major parts. First, it specifies an abstract interface of the service. Complex data types are defined and used to construct messages. A message has a name and a set of parts of certain type. Messages represent either an input parameter set or the output of some abstract operations, which are conceptually grouped together into a PortType. Second, an implementation part binds the abstract interface to concrete network protocols and message formats (SOAP, HTTP). A service is a collection of ports of various PortTypes.

We depict the WSDL representation by presenting the main PortTypes and the operations which are contained by each port. For each operation we show the input and the output messages, and for each message we enumerate its parts. It follows that Bib2Rdf has a single port (of type Translator) with one operation (op1), where the input and output messages have a single part.

2.3 Case Study - the Semantic University

The goal of the SW@VU[5] project is to use Semantic Web technology within the Vrije Universiteit Amsterdam. The first case study was to share the bibliographic data of the involved researchers through semantic portals.

While building the infrastructure for this experiment, a set of web-accessible software components were created. The task of creating the portal simply became shipping data from one service to another. First, each available BibTex file is converted to RDF(S) using the *Bib2Rdf* service then saved in a web-accessible RDF(S) repository and query engine, *Sesame*[6]. When merging our data, syntactically different resources pointed to the same person. We used `daml:sameIndividualAs` to encode this redundancy and extended Sesame's reasoning capabilities to interpret this new tag. To provide automated support for the task of finding the resources referring to the same person we developed the *SIA(SameIndividualAs)* service. Using machine learning techniques on the RDF(S) source, SIA extracts the resources which might refer to the same person. It returns tuples of similar resources. Therefore the second step is to extract all the data from Sesame, send it to SIA, obtain the redundancy file and save it back to Sesame. Finally, portal creator software creates the portals of publications by querying Sesame.

By augmenting our web-services with a semantic description we allow automatic execution of a set of services in a pre-defined way. However, we envision more "intelligent" service composition that will be able to adapt according to changes in the requirements. For example, the portal creation task could be performed by different sets of services. Or, if RDF(S) source is provided there is no need for translation. Also, a single data source would not require any redundancy checking. We aim to support such adaptive service composition.

3. MODELLING EXPERIENCES

We used DAML-S to describe services with an increasing level of complexity. We present our experiences related to the Bib2Rdf and SIA services. We describe the main issues/pitfalls we encountered. In retrospect, some of what we learned was available in the documentation, but the counterintuitiveness of some aspects of the language meant we had to learn them for ourselves. Therefore, to provide the basis for discussion of some of the features we found difficult in DAML-S, we do not only convey our final solutions but describe the major design decisions we took during modelling.

3.1 Modelling a simple service - Bib2Rdf

Bib2Rdf[7] is a simple service: it transforms a BibTex file into a RDF(S) representation. The service takes as input the URL of a BibTex file and returns the RDF(S) encoding of this data.

Using domain knowledge for service description. It is important to realize that the Profile makes use of already specified domain knowledge (ontology). If multiple services are described using the same ontology, matchmaking and integration will be based on this ontology. We have modelled a set of terms from our application domain in the ProfileTaxonomy[8] ontology and used them in our service descriptions. For example we have declared that a "Translator" specializes the "InformationSystem" Profile.

The Profile of a service uses domain knowledge in two situations. First, it can be of a type described in the domain ontology: the Bib2Rdf Profile instance is of type "Translator". Second, we used domain concepts for the functional description of the service. The DAML-S Profile ontology models a service in terms of a set of parameters. Inputs, outputs, preconditions and effects (IOPE's) are subproperties of `profile:parameter`. Each parameter has a ParameterDescription which mandates describing the parameter through the `profile:restrictedTo` property (see DAML-S specification bellow). This property has an unspecified range, therefore it can refer both to a DAML+OIL concept or to a data type. We recommend specifying a domain level concept: this gives the "semantics" of the parameter.

```
<daml:ObjectProperty rdf:ID="parameter">
  <daml:domain rdf:resource="#Profile"/>
  <daml:range rdf:resource="#ParameterDescription"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="restrictedTo">
  <daml:domain rdf:resource="#ParameterDescription"/>
</daml:ObjectProperty>

<daml:Class rdf:about="#ParameterDescription">
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#restrictedTo"/>
    </daml:Restriction>
  </rdfs:subClassOf>
 </daml:Class>
```

The ProcessModel also makes use of domain concepts. For Bib2Rdf we declared the input and the output of the Process P1 in terms concepts defined in the domain ontology: BibFile and RdfFile.

How does a ServiceProfile relate to a ServiceModel?. The ServiceProfile and ServiceModel are two different descriptions of the same service, and naturally links exist between them. Identifying these links and specifying them correctly ensures the consistency of the description.

Firstly, each Profile instance states the Process it describes through the unique property `has_process`. Secondly, the IOPE's of a Profile correspond to the IOPE's of the described Process. Understanding this correspondence is not trivial because the IOPE's play

[5]http://www.cs.vu.nl/~mcaklein/SW@VU/
[6]http://sesame.aidministrator.nl
[7]http://www.cs.vu.nl/~marta/services/Bib2Rdf/Bib2RdfService.daml
[8]http://www.cs.vu.nl/~marta/services/ProfileTaxonomy.daml

different roles for the Profile and the Process. In the Profile ontology they are treated equally as parameters of the Profile (they are subproperties of the `profile:parameter` property). In the Process ontology only IO's are regarded as subproperties of the `process:parameter` property. The PE's are just simple properties of the Process. While technically the IOPE's are properties both for Profile and Process, the fact that they are treated differently at a conceptual level is misleading.

Even more, this leads to a DAML-S model that favors inconsistencies. As suggested by the following DAML-S specification, the Profile ontology enforces that each ParameterDescription `refersTo` exactly one element of type `process:paramater`.

```
<daml:ObjectProperty rdf:ID="refersTo">
  <daml:domain rdf:resource="#ParameterDescription"/>
  <daml:range  rdf:resource="&process;#parameter"/>
</daml:ObjectProperty>


<daml:Class rdf:about="#ParameterDescription">
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#refersTo"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
```

This induces two problematic scenarios. First, one would expect that Profile parameters of a certain type can only refer to Process parameters of the same type. However, this is not enforced: one can easily make a link between parameters of different types (eg. `profile:input` and `process:output`). The DAML-S coalition states that they do allow inconsistencies between Profile and Process and that identification of inconsistencies will occur when something breaks[4]. Since matchmaking is based on the Profile description, the break may occur during usage of the service. The rationale for this design decision is not clear.

Second, because the Process ontology does not model PE's as subproperties of `process:paramater`, it is inconsistent to use entities of this type as values for `profile:refersTo`. Therefore Preconditions and Effects (PE) of a Profile cannot refer to corresponding PE's of a Process.

We conclude that this link between the Profile and Process IOPE's should be corrected and made more explicit.

How does the Process relate to the WSDL grounding?. The first task for realizing grounding is creating a WSDL file. Our service has an HTTP interface. Unfortunately most efforts/examples in WSDL concentrate on describing SOAP interfaces, so we had to expend significant effort in understanding the HTTP binding as it is different in many ways from SOAP bindings.

The mapping between the elements of the DAML-S semantic description and those of the WSDL file is bi-directional: the WSDL file specifies the link back to Process elements, while the WsdlGrounding builds a bridge between the elements of the Process and the WSDL interface. The specification syntax is rather heavy and the multiple interdependencies make the grounding process complicated and frustrating.

There are a few rules which define the mapping between these two models[5]. First, each DAML-S AtomicProcess corresponds to a WSDL operation. Consequently, each input of the DAML-S AtomicProcess is mapped to a message-part in the input message of the corresponding WSDL operation. Similarly, each output of the DAML-S AtomicProcess is mapped to a corresponding message-part in the output message of the WSDL operation.

3.2 Modelling a service with multiple interfaces - SIA

The SIA (SameIndividualAs) service is essentially as simple as the Bib2Rdf service: it acts upon an RDF file, determines resources that possibly point to the same physical person and returns an RDF file with tuples of identical resources. The only element of complexity is that this service can acquire the RDF source in multiple ways: (1) by reading it from a URL, (2) by accepting the data itself as a string and (3) by reading the data from a Sesame repository, given the name of the repository and the log-in information.

Intuitively, this situation is similar to the well-known parametric polymorphism mechanism: a certain method allows different signatures, but essentially it executes the same function. The available documentation about DAML-S provided little guidance on how to model this situation.

[1] have considered the problem of supporting multiple interfaces and offer a solution for the situation where the number of arguments are the same and in the same order but where the data types may differ, i.e. type polymorphism. The solution offered is to define a higher level concept that covers all possible alternative data types. This approach does not address our problem where we have a different number of arguments. This solution also has problems at the grounding level, as we show in our presentation of the alternative descriptions we developed.

One very simple solution to our problem would be to treat each alternative interface as a separate service. We rejected this solution for conceptual, practical and reuse reasons. At a conceptual level, we are describing one and the same service. If we made them separate services, DAML-S did not provide any way of identifying that they were in fact the same service. Knowing that a service and/or its description is related or in fact identical may be important when it comes to choosing services. From a practical point of view, marking up a service is time-consuming enough without having to perform the activity for every possible interface. From a reuse point of view, we wanted to share and reuse as much as possible between these alternative ways of accessing the service.

In order to provide the semantics that would allow more intelligent matchmaking and to handle interfaces with different data types and number of parameters, we have tried a number of alternatives. The first variant (SIA1) was based on a top-down design starting with a service model and working down to the service grounding. Due to the problems we faced in SIA1, the second variant (SIA2) used a bottom-up approach starting with the WSDL definition. SIA2 clarified the DAML-S view of a service as being primarily defined by its IOs, rather than its effects. We developed SIA3 to support the new view of our service as a composite process, involving data readers and translators, rather than an atomic process. The first three descriptions were not valid solutions as they were either conceptually wrong or impossible to specify using DAML-S and WSDL. Our final design (SIA4) was a compromise that provided a valid solution but which did not completely represent our conceptual model of the service. It also involved significant repetition of descriptions. These variants are presented next in more detail, along with discussion of our rationale, choices and conclusions during our design. A schematic description of each variant is provided to clarify the discussion.

(SIA1[9]) Because the effect of the service is not altered by the way in which the RDF file is provided, our first intuition was to model SIA as a single service and to make the differentiation between the three ways of accessing it at the grounding level. At the Profile (Pr1) / Process (P1) level we described the service as

[9] http://www.cs.vu.nl/~marta/services/sia/Sia1Service.daml

accepting an RdfFile and producing another RdfFile. The WSDL representation of the service consists of a port with three operations (op1, op2, op3) which differ through their input messages. The first expects a url, the second a string and the third receives four Sesame related parameters (server url, password, login, repository name).

```
*Service SIA1:
 *Profile:Pr1 (hasProc=P1)(I(RdfFile), O(RdfFile))
 *ProcessModel:
    AtomicProcess:P1(I(RdfFile),O(RdfFile))
 *WSDLGrounding:
    WsdlAtomicProcessGrounding: Gr1 (P1->op1)
    WsdlAtomicProcessGrounding: Gr2 (P1->op2)
    WsdlAtomicProcessGrounding: Gr3 (P1->op3)   !!!
 *WSDL:
  Service(PortType:SIA(
    op1 (IMsg(url), OMsg(stream))
    op2 (IMsg(string), OMsg(stream))
    op3 (IMsg(url,pse,ln,rep), OMsg(stream))))
```

Specifying the first two groundings was easy. However for the third grounding we realized that it was impossible to build a one-to-one mapping between the single input of process P1 and the four parameters of the WSDL operation, op3. Grounding an input to a whole message is not possible.

As stated in [4] the second assumption of the DAML-S/WSDL mapping is that "each atomic process input and output corresponds to a WSDL message part". The DAML-S coalition acknowledges that this could be a possible limitation, but do not give a concrete example of a problematic scenario. This example shows that the assumption prohibits modelling of parametric polymorphism. Therefore we encourage its revision.

(SIA2[10]) After SIA1, we changed our strategy to bottom-up modelling. Based on the structure of the WSDL file, we modelled two AtomicProcesses: one with a single input (P1), just like before, and one with four parameters (P2) needed when a Sesame repository is used as the data source. With this model we excluded any grounding problem, but new issues emerged.

The first issue relates to whether the distribution of these two Processes should be within one or two ServiceModel instances. Previously we decided to model a single service, and a service can have at most one ServiceModel (maxCardinality(`describedBy`)=1). Therefore both our processes have to be part of a single ServiceModel. However, the ServiceModel can only accommodate a single Process (of type Atomic, Simple or Composite) because the `process:has_process` property has an exact cardinality of "1". This requires combination of the two atomic processes into a composite one. The `process:Choice` control construct is closest to our needs: it expresses that only one process can be chosen for execution. We modelled a composite process(CP) as a "Choice" between the two atomic processes and included it in a single ServiceModel entity.

The second issue relates to the Profile. Keeping the Profile as in the previous example is technically correct: we can link the input and the output of the Profile to the IO's of the Process P1, while the parameters of P2 are not referenced from the Profile. However, this means that when advertised, the service does not expose its ability to read data directly from Sesame. Adding the other four parameters to the profile is not a solution either because we cannot specify how these parameters relate. This would be misleading at matchmaking. We decided to create two Profile instances: Pr1 generically describes the IO's and maps to P1, while Pr2 reflects the parameters of P2. The service has three groundings, two for the generic Process P1 and one for the Sesame Process P2.

[10]http://www.cs.vu.nl/~marta/services/sia/Sia2Service.daml

```
*Service SIA2:
 *Profile:Pr1 (hasProc=P1)(I(RdfFile),O(RdfFile))
 *Profile:Pr2 (hasProc=P2)(I(server),I(url),I(psw),
                          I(ln),O(RdfFile))
 *ProcessModel:
    CompositeProcess: CP:Choice
    {   AtomicProcess:P1(I(RdfFile),O(RdfFile))
        AtomicProcess:P2(I(server),I(url),I(psw),
                         I(ln),O(RdfFile))}
 *WSDLGrounding:
    WsdlAtomicProcessGrounding: Gr1 (P1->op1)
    WsdlAtomicProcessGrounding: Gr2 (P1->op2)
    WsdlAtomicProcessGrounding: Gr3 (P2->op3)
 *WSDL: same as SIA1
```

This design was based on the constraints imposed by the DAML-S model. The constraints that had the most impact on our decisions were the following:

- There should be a one-to-one correspondence between the IO(PE)'s of all modelling levels (even if directly not imposed but for the sake of consistency)
- What defines a services is not its effect but its signature. Therefore our approach for SIA1 (and P1 for SIA2) was conceptually inconsistent with the DAML-S view: we should provide a different Profile for each of the different ways to access the service. These Profiles provide the meaning of the IOPE's.

With this new view of what it means to define a service we considered another model based on the observation that our service is a combination of four processes: a translator process and three different data acquisition processes.

(SIA3[11]) We modelled the service as having a complex ProcessModel. We consider that the service offers three CompositeProcesses (CP1, CP2 and CP3) combined in a global CompositeProcess (CP). Each of these processes is a sequence of a DataReader (DR) and the Translator (T1) process itself. The service has three Profiles (Pr1, Pr2, Pr3) each describing one of the three composite processes. The same WSDL file can be used.

```
Service SIA3:
 *Profile:Pr1 (hasProc=CP1)(I(url),O(RdfFile))
 *Profile:Pr2 (hasProc=CP2)(I(RdfStream),O(RdfFile))
 *Profile:Pr3 (hasProc=CP3)(I(server),I(url),I(psw),
                            I(ln),O(RdfFile))
 *ProcessModel
    CompositeProcess: CP:Choice
    {
     CompositeProcess:CP1: Sequence
         {
          AtomicProcess:DR1(I(url),O(RdfFile))
          AtomicProcess:T1(I(RdfFile),O(RdfFile))}
     CompositeProcess:CP2: Sequence
         {
          AtomicProcess:DR2(I(RdfStream),O(RdfFile))
          AtomicProcess:T1(I(RdfFile),O(RdfFile))}
     CompositeProcess:CP3: Sequence
         {
          AtomicProcess:DR3(I(server),I(url),I(psw),
                            I(ln),O(RdfFile))
          AtomicProcess:T1(I(RdfFile),O(RdfFile))}
     }
 *WSDLGrounding:
    WsdlAtomicProcessGrounding: Gr1 (CP1->op1)   !!!
    WsdlAtomicProcessGrounding: Gr2 (CP2->op2)   !!!
    WsdlAtomicProcessGrounding: Gr3 (CP3->op3)   !!!
 *WSDL: same as SIA1
```

[11]http://www.cs.vu.nl/~marta/services/sia/Sia3Service.daml

We have encountered a new grounding problem. Conceptually each composite process corresponds to a WSDL operation: the inputs of the DataReader are the same as the inputs of the WSDL operation, and the output of the Translator corresponds to the output of the WSDL operation. However the first assumption of the existing mapping states that "a single atomic process corresponds to a single WSDL operation"[4]. Therefore we cannot perform this mapping. The DAML-S coalition acknowledge that this could be a possible limitation, but they are skeptic about "the importance of relaxing this assumption"[4]. This example shows that the assumption prohibits modelling of a complex internal structure if it is not directly reflected in the web interface of the service. We see this as a serious limitation to conceptual modelling.

(SIA4[12]) Based on all of these experiences and aware of the limitations of the model our final model is reflected in SIA4. We model a single service with three different profiles (Pr1, Pr2, Pr3) each describing one of the three different functionalities of the service. The service has a single ProcessModel containing a composite process (CP). This indicates that the service can perform one of three possible atomic processes (P1, P2, P3). We then mapped each of these atomic processes to one WSDL operation. This is essentially the same as SIA3, however we must give up the conceptual complex internal model of processes so that we are able to map them to WSDL operations. This description is also similar to SIA2 but reflects that we are dealing with three different signatures, not just two.

```
Service SIA4:
 *Profile:Pr1 (hasProc=P1)(I(url),O(RdfFile))
 *Profile:Pr2 (hasProc=P2)(I(RdfStream),O(RdfFile))
 *Profile:Pr3 (hasProc=P3)(I(server),I(url),I(psw),
                          I(ln),O(RdfFile))
 *ProcessModel:
    CompositeProcess: CP:Choice
    {
    AtomicProcess:P1(I(RdfStream),O(RdfFile))}
    AtomicProcess:P2(I(RdfFile),O(RdfFile))}
    AtomicProcess:P3(I(server),I(url),I(psw),
                     I(ln),O(RdfFile))}
    }
 *WSDLGrounding:
    WsdlAtomicProcessGrounding: Gr1 (P1->op1)
    WsdlAtomicProcessGrounding: Gr2 (P2->op2)
    WsdlAtomicProcessGrounding: Gr3 (P3->op3)
 *WSDL: same as SIA1
```

We are not completely satisfied with this model as we are unable to model the P1, P2 and P3 processes as composite processes which we feel better reflects the structure of the service. Also, we are concerned with the amount of effort involved in providing (and maintaining) this rather large set of descriptions. While at the Service superclass level we only have one definition, which is appropriate conceptually, this document is the smallest of all. The amount of repetition that exists in the three Profile and Grounding documents is considerable. Of course, cut and pasting will reduce the initial effort and defining concepts in one document and pointing to them in the other two can reduce maintenance overheads. However, the verbosity of this approach seems somewhat excessive. Our greatest concern with this final compromise is that we are unsure whether this model would be consistent with the model developed by another team to represent this same service. We feel that if semantics are to be added in a meaningful and useful way, greater direction and precision should be provided by the modelling language so that this uncertainty is minimal.

[12]http://www.cs.vu.nl/~marta/services/sia/Sia4Service.daml

4. MAJOR OBSERVATIONS

Based on our experiences with the above mentioned problems we have distilled some general observations about DAML-S.

The strength of DAML-S is that it goes beyond syntactic description of a service by providing a semantic description. Semantics allows reasoning about a service and moves us towards the ultimate goal of dynamic service discovery and usage. The DAML-S upper ontologies provide semantics for high level concepts concerning web services. Further meaningful description can be achieved by describing the service in terms of domain concepts contained in a domain ontology. This is particularly valuable for matchmaking where the requestor may use alternative terms. The domain ontology can use the SameClassAs relation to identify synonyms and the SubClassOf relation to identify hypernyms and hyponyms. This supports matchmaking where different levels of abstraction are used between the requestor and provider.

The other key strength of DAML-S is that it links to an industry standard, namely WSDL. In this way, it indeed fulfills its role as a link between the Semantic Web community and industry.

However, we also encountered a set of shortcomings.

A) Imprecise conceptual model. While it is commendable that DAML-S seeks to provide flexibility and thus has not fully defined a number of its concepts, this flexibility comes at the expense of clarity. The result of this imprecision is that DAML-S has an imprecise underlying conceptual model.

We base this on the following facts:

- *Different models within DAML-S*. The parts of DAML-S employ different metaphors to describe services. At the Profile level a service has four types of parameters: IOPE's. At the Process level IO's and PE's are treated conceptually differently as they emerge from two different views of a service. A service viewed as a program is defined by its IO's, while when seen as an action the PE's are important. The conceptual gap is even wider when one tries to align the DAML-S model to the WSDL model which defines services as collections of ports. These alternative conceptual models make specification of services difficult and mapping between models almost impossible. Even within DAML-S the different models can lead to inconsistencies in the specification.

- *Unclear links between models*. Several links exist between the conceptual models however they are often unclear. Insufficient descriptions are provided in the DAML-S documentation to discover the intended meaning of certain properties and in particular which properties are related to properties in one or more of the other models. The lack of precise specification of the interconnections between models and the possibility of inconsistent models is admitted by the DAML-S Coalition.

- *No clear correspondence of DAML-S concepts with software engineering (SE) concepts.* Many, if not the majority, of intended users of DAML-S are software engineers. We feel that reference to and support for SE concepts, perhaps in the form of concept mappings, would ease the understanding of DAML-S. While WSDL intuitively models different interfaces as PortTypes and allows grouping operations in ports (as the methods of an interface), it seems that DAML-S only considers the very simple function metaphor (methods). More complex concepts such as parametric polymorphism, re-use are not supported. We think that the SE model would both disambiguate some of the concepts and give a shared framework for DAML-S and WSDL.

The imprecise underlying conceptual model induces some effects:

- *Multiple modelling possibilities.* While simple services present small problems, modelling of more complex situations is not as straightforward. Because there is no clear view of what services are one can produce a variety of models. Unfortunately, most of them cannot be expressed because of various language constraints/inconsistencies.

- *Parametric polymorphism* is not just a term used in software engineering, but a mechanism equally valid in an e-commerce scenario. For, example any e-commerce site would optimally allow multiple ways of paying: by credit card or by bank transfer. Basically, the same effect would be achieved (i.e. paying for the item) even if the money would be obtained from different sources. From a SE point of view this is only a method so it should be modelled as a single service. Then inherent unclarities arise about how to model these different facets of the same service in DAML-S. It is not only a question to conform to the limitations of the ontology but also to know if our model is conceptually correct.

B) Mapping to WSDL limits DAML-S expressivity. During our work we have experienced that the chosen mapping to WSDL often limits the expressivity of DAML-S. Just by modelling a simple service we conflicted with two out of three basic assumptions that underlie the mapping. This forced us to revise our models so that a grounding was possible at the expense of giving up parametric polymorphism (SIA1) or an accurate specification of a complex internal structure (SIA3). We feel that DAML-S is influenced too much by the actual grounding details.

C) Difficult to learn. One of our major comments (and worries) is that it was quite difficult to get started with writing DAML-S. The previously mentioned lack of conceptual model played a fair role in this. Other inhibiting factors were:

- *Limited tool support.* DAML-S consists of three different ontologies with some shared links. When one writes a DAML-S description it is critical to be sure that these links are properly made. This implies working with multiple ontologies and descriptions at the same time. The only tool support for now are simple text editors making the task complex, error-prone and frustrating. An exception is the DAML-S Matchmaker[13] which allows a Profile description to be developed using form filling but offers no support for specifying the Process or Grounding. Since only the Profile can be entered using the tool, we get no assistance with specifying and verifying the mappings between models.

- *Few and limited examples.* The DAML-S site provides two complex examples. Our complaint is that these examples were artificially created to fit the language rather then being real-life examples. As a consequence they ignore situations that arise in the case of real life services.

- *Knowledge of DAML/WSDL/SOAP required.* The pre-requisites to start writing complete DAML-S descriptions is rather high: one has to know DAML and WSDL/SOAP. For users who are only partly familiar with these techniques it is quite a burden to learn all of them in sufficient depth to ensure that they are consistent with one another and convey the intended meaning.

[13]http://www.damlsmm.ri.cmu.edu

5. CONCLUSIONS AND FUTURE WORK

DAML-S has generated a lot of interest through its promise to add semantics to web service descriptions. Despite that interest we were only able to find a small number of DAML-S descriptions of web services and most of these did not point to real services and were from the DAML-S community. To fill in this gap we have used DAML-S to mark up some of our own web services.

We have concluded that DAML-S is superior to existing WS languages as it allows use of formally defined domain knowledge. However it also presents a series of weak points which surface when confronting the language with real life situations. We hope that our observations would guide the DAML-S coalition in its work towards version 1.0.

The work reported in this paper provides a set of examples based on actual services that augments the existing available DAML-S descriptions. We are currently marking up a more complex service which is part of this application. We have already raised a considerable number of issues just from the two simple services we have described. We encourage other work similar to ours, i.e. marking up different services with DAML-S, to further contribute to the development of DAML-S and to extend the services available for semantic discovery. The development of a tool to allow creation of descriptions based on multiple ontologies would greatly assist such work.

We hope that both our observations and actual examples will contribute to the development and large scale usage of DAML-S and move our community closer to realizing the Semantic Web.

6. REFERENCES

[1] A. Ankolenkar, F. Hutch, and K. Sycara. Concurrent Semantics for the Web Services Specification Language DAML-S. In *Proc. of The Fifth International Conference on Coordination Models and Languages)*, 2002.

[2] J. Brison, D. Martin, S.I. McIlraith, and L. A. Stein. Agent-Based Composite Services in DAML-S: The Behavior-Oriented Design of an Intelligent Semantic Web. http://www.cs.bath.ac.uk/ jjb/ftp/springer-daml.pdf.

[3] J. Cardoso and A. Sheth. Semantic e-Workflow Composition. Technical report, LSDIS Lab, Computer Science, University of Georgia, July 2002.

[4] DAML Services Coalition. DAML-S: Semantic Markup for Web Services. DAML-S v. 0.7 White Paper, October 2002.

[5] DAML-S Coalition. Describing Web Services using DAML-S and WSDL. DAML-S Coalition working document, http://www.daml.org/services/daml-s/0.7/daml-s-wsdl.html, August 2002.

[6] A. Lopes, S. Gaio, and L.M. Botelho. From DAML-S to Executable Code. In *Proc. of the Workshop Challenges in Open Agent Systems AAMAS 2002*.

[7] M. Paolucci, K. Sycara, and T. Kawamura. Delivering Semantic Web Services. Technical report cmu-ri-tr-02-28, Robotics Institute, Carnegie Mellon University, 2002.

[8] M. Somacher, M. Tomaiuolo, and P. Turci. Goal Delegation in Multiagent System. In *In Proc. Tecniche di Intelligenza Artificiale per la ricerca di informazione sul Web*, 2002.

[9] C. Wroe, R. Stevens, C. Goble, A. Roberts, and M. Greenwood. A Suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data . *Journal of Cooperative Information Science*, 2003.