Agent-based Semantic Web Services

Nicholas Gibbins
Department of Electronics and
Computer Science
University of Southampton
Southampton, United Kingdom
nmg@ecs.soton.ac.uk

Stephen Harris
Department of Electronics and
Computer Science
University of Southampton
Southampton, United Kingdom
swh@ecs.soton.ac.uk

Nigel Shadbolt
Department of Electronics and
Computer Science
University of Southampton
Southampton, United Kingdom
nrs@ecs.soton.ac.uk

ABSTRACT

The Web Services world consists of loosely-coupled distributed systems which adapt to ad-hoc changes by the use of service descriptions that enable opportunistic service discovery. At present, these service descriptions are semantically impoverished, being concerned with describing the functional signature of the services rather than characterising their meaning. In the Semantic Web community, the DAML Services effort attempts to rectify this by providing a more expressive way of describing Web services using ontologies. However, this approach does not separate the domain-neutral communicative intent of a message (considered in terms of speech acts) from its domain-specific content, unlike similar developments from the multi-agent systems community.

In this paper, we describe our experiences of designing and building an ontologically motivated Web Services system for situational awareness and information triage in a simulated humanitarian aid scenario. In particular, we discuss the merits of using techniques from the multi-agent systems community for separating the intentional force of messages from their content, and the implementation of these techniques within the DAML Services model.

1. INTRODUCTION

The world of Web Services may be characterised as a world of heterogeneous and loosely-coupled distributed systems where adaptivity to ad-hoc changes in the services offered by the components of the systems is considered advantageous. By loosely-coupled, we mean that the interactions between system components are not rigidly specified at design time, but that system components may opportunistically make use of new services that become available during their lifetime without having been explicitly told of their existence from the outset.

The task of searching for a system component which can perform some given service, or *service discovery*, is the enabling technique that makes loosely-coupled systems possible, and provides a process by which system components may find out about new services on offer. An essential adjunct to service discovery is *service description*, by which names or descriptive expressions are attached to services, allowing both the advertisement of services by providers and the formulation of queries about services by users. Service discovery services have been seen as an essential component of loosely coupled systems such as (but not limited to) multi-agent systems [10].

A typical service discovery service (also often referred to as a *directory service*) consists of a registry (possibly distributed) which provides two services. The first allows service providers to adver-

tise the services that they offer in the registry, while the second enables service users to query the registry and so determine which service providers can provide relevant services.

One rough characterisation of the technologies used for service discovery in the Web Services world can be made by studying the difference between approaches which could be considered semantically poor and those which are semantically rich. In the former case, services are often referred to by opaque names or function signatures which give little or no indication of the nature of the services being managed. In the latter, however, service descriptions are more complex expressions which are based on terms from agreed vocabularies, and which attempt to describe the meaning of the service, rather than simply ascribing a name to it.

A key component in the semantics-rich approach is the *ontology*, the formal, agreed vocabulary whose terms are used in the construction of service descriptions. An ontology is a conceptualisation of an application domain in a human-understandable and machine-readable form, and typically comprises the classes of entities, relations between entities and the axioms which apply to the entities which exist in that domain. Ontologies are currently a fast-growing research topic, with interest from several communities, not least the agent-based computing, Semantic Web and knowledge management communities, because they offer a more formal basis for characterising the knowledge assets held by software agents, Semantic Web services or organisations [12, 11].

Although such an ontology defines the agreed meaning for the application domain-specific terms used in the content of messages, it does not define the meaning of the message types themselves, or their effects upon the recipient. The current approach in the Semantic Web to Web Services, such as that taken by DAML Services, does not provide a common basis for defining the pragmatics of different message types, as we might expect from a speech act-like treatment of messages[16]. Such a basis would provide a way to ease the introduction of new types of messages, since there would be a common understanding of what was meant by, for example, a directive message (which instructs a system component to perform an action) or an assertive message (which informs a system component of some fact) which was independent of any domain specific meaning.

The technique of factoring out the common attributes of message types and ascribing them to different classes of speech acts is commonly used in the design of *agent communication languages* (ACL) for multiagent systems [9, 13], where there is a clear separation made between the domain-specific and domain-independent aspects of communication. We believe that a similar approach can be applied to Web Services, in which an ACL component is integrated into the semantically rich service descriptions.

In this paper we outline our experiences of building semantically

rich software services based on the integration of ontologically-motivated DAML-S-based Web Services and an agent communications language, and describe our prototype demonstrator, a situational awareness application based on a humanitarian aid scenario.

2. SEMANTIC WEB SERVICES

The aim of this work has been to investigate the integration of the nascent Web Services infrastructure with the richer semantics of the Semantic Web, in particular through the use of more expressive languages for service description. In their implementation of service descriptions, the existing Web Services specifications are more concerned with the signature of services. Such signatures comprise the types of the parameters of the service (typically expressed in terms of XML Schema datatypes), rather than with any form of ontological classification of the services.

The notion of an ontology is central to the Semantic Web, which uses languages such as RDF Schema [3] or DAML+OIL [6] (or in future, the ontology Web language OWL, a current work in progress) to describe ontologies. An integration of Web Services with the Semantic Web should involve the use of these languages to describe and characterise services in a manner which the existing Web Services service description languages cannot.

There are two options for the form of this integration. We could choose to layer RDF or DAML+OIL on top of an existing XML-based service description language (such as the Web Services Description Language or WSDL [5], for example), so that the description includes an RDF expression that characterises the service. Alternatively, we could choose to build on a service description language which is itself written in RDF or DAML+OIL, such as the DAML Services ontology [17].

We have chosen the latter approach, and have used DAML Services as the basis for our design because it allows the definition of classes of related services, which makes service reuse more feasible (because agents are better able to reason about the relationships between services) and the system more adaptable as a whole (because rich service descriptions give agents the means to determine whether they can use new types of service).

In addition to reuse and adaptability concerns, DAML Services also allows the types of service parameters to be specified as DAML class expressions, in addition to the XML Schema datatypes [1] that are used by WSDL and other Web Services languages, so the parameter values that are passed when a service is invoked may be objects from a knowledge base as well as literal values.

3. AGENT WEB SERVICES

In the conventional Web Services approach exemplified by WSDL [5] or even by DAML Services, the communicative intent of a message (for example, whether it is a request or an assertion) is not separated from the application domain. This is at odds with the convention from the Multi-Agent Systems world, where there is a clear separation between the intent of a message, which is expressed using an agent communication language, and the application domain of the message, which is expressed in the content of the message by means of domain-specific ontologies.

This separation between intent and domain is beneficial because it reduces the brittleness of a system. If the characterisation of the application domain (the ontology) changes, then only that component which deals with the domain-specific information need change; the agent communication language component remains unchanged.

The division of service descriptions into a profile and a process component, as in DAML Services, provides a means to compartmentalise Web Services in a manner similar to that found in agent
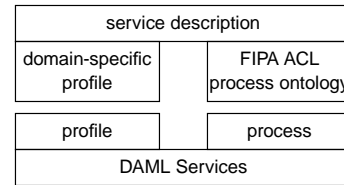

Figure 1: Service Description with ACL Process Ontology

systems. We have therefore determined to describe the pragmatics of message types in the process component, giving an abstract ontology of message types that corresponds to the agent communication language, while the more application-specific details of the abilities of a particular agent (expressed as constraints on the content of messages) are expressed in the profile component, as shown in Figure 1.

To this end, we have designed a simple process ontology of message types based on the FIPA[1] agent communication language [9]. In this ontology, ACL message types are represented as atomic processes (see Figure 2 for a fragment of this ontology containing the `subscribe` performative), with the content of the message as a parameter of the process. The `Subscribe` process has two properties, one a sub-property of `input` which is used to pass the query expression that forms the content of the message, and one a sub-property of `output` which is used to return an indicator of the success or failure of the subscription.

In addition to input and output parameters, DAML Services also provides a facility for specifying the necessary preconditions and the side-effects of a service. While a full description of the FIPA performatives could make use of this facility to fully describe the pragmatics of the messages (as described by FIPA in the appendix to [9]), the facility is not yet fully specified and expressing the FIPA theory of agency in DAML (a necessary prerequisite for expressing the pragmatics of individual performatives) is not the focus of this work.

In FIPA, the query performatives (`query-if` and `query-ref`) are treated as asynchronous messages which form part of the FIPA Request protocol, which is in keeping which the message-passing (as opposed to procedure-calling) idiom used by FIPA. In this idiom, a query does not return an answer directly, but causes the formation of an intention in the recipient to send an `inform` message (containing the answer) to the sender of the query. However, the predominant communication idiom found in the Web Services environment is procedure-calling, as exemplified by the Simple Object Access Protocol [2], not plain message-passing. In our adaptation of the FIPA ACL for a procedure-calling Web Services environment, we have chosen to amend the semantics of the query performatives and make them synchronous messages which return the answer to the query. The advantage of this approach is that we no longer need to track the conversations in which a service is participating (in order to determine which response message corresponds to which query) because a response message cannot be separated from the the query to which it is providing an answer. This has the effect of simplifying the service's implementation of the ACL, and allows us to concentrate instead on the service profiles which are used to determine whether or not a service will be of use to us.

The profile component of the DAML Services expression is used to express the service being offered or requested. This profile de-

---

[1]Foundation for Intelligent Physical Agents
`http://www.fipa.org/`

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE rdf:RDF [
 <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
 <!ENTITY daml 'http://www.daml.org/2001/03/daml+oil#'>
 <!ENTITY fipa 'http://www.fipa.org/ontology/acl#'>
 <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
 <!ENTITY proc 'http://www.daml.org/services/daml-s/0.7/Process.daml'>
]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:fipa="&fipa;"
         xmlns:daml="&daml;" xmlns:rdfs="&rdfs;">
 <daml:Class rdf:ID="Subscribe">
  <rdfs:subClassOf ref:resource="&proc;#AtomicProcess"/>
  <rdfs:label></rdfs:label>
  <rdfs:description>The act of requesting a persistent intention to
   notify the sender of the value of a reference, and to notify again
   whenever the object identified by the reference changes. The subscribe
   act is a persistent version of query-ref, such that the agent receiving
   the subscribe will inform the sender of the value of the reference, and
   will continue to send further informs if the object denoted by the
   description changes. A subscription set up by a subscribe act is
   terminated by a cancel act.</rdfs:description>
 </daml:Class>

 <rdf:Property rdf:ID="subscribe-reference">
  <rdfs:subPropertyOf rdf:resource="&proc;input"/>
  <rdfs:domain rdf:resource="&fipa;Subscribe"/>
  <rdfs:range rdf:resource="&daml;Thing"/>
 </rdf:Property>

 <rdf:Property rdf:ID="subscribe-success">
  <rdfs:subPropertyOf rdf:resource="&proc;output"/>
  <rdfs:domain rdf:resource="&fipa;Subscribe"/>
  <rdfs:range rdf:resource="&daml;Thing"/>
 </rdf:Property>
<rdf:RDF>
```

Figure 2: FIPA ACL Process Ontology Fragment

```
<profile:OfferedService rdf:ID="UNHCR-Subscribe-Profile">
 <profile:has_process rdf:resource="&fipa;Subscribe"/>
 <profile:input>
  <profile:ParameterDescription>
   <profile:parameterName rdf:resource="subscribe-reference"/>
   <profile:refersTo rdf:resource="&fipa;subscribe-reference"/>
   <profile:restrictedTo>
    <daml:Class>
     <daml:intersectionOf rdf:parseType="daml:collection">
      <daml:Class rdf:about="&flood;Report"/>
     <daml:Restriction>
      <daml:onProperty rdf:resource="&flood;reportsOn"/>
      <daml:toClass>
       <daml:Class>
        <daml:intersectionOf rdf:parseType="daml:collection">
         <daml:Class rdf:about="&flood;MovementEvent"/>
        <daml:Restriction>
         <daml:onProperty rdf:resource="&flood;actor"/>
         <daml:toClass>
          <daml:Class>
           <daml:intersectionOf rdf:parseType="daml:collection">
            <daml:Class rdf:about="&flood;Vehicle"/>
           <daml:Restriction>
            <daml:onProperty rdf:resource="&flood;memberOf"/>
            <daml:hasValue rdf:resource="&flood;UNHCR"/>
           </daml:Restriction>
          </daml:intersectionOf>
         </daml:Class>
        </daml:toClass>
       </daml:Restriction>
      </daml:intersectionOf>
     </daml:Class>
     </daml:toClass>
    </daml:Restriction>
   </rdfs:intersectionOf>
  </daml:Class>
 </profile:restrictedTo>
 </profile:ParameterDescription>
</profile:input>
<profile:output>
 <profile:ParameterDescription>
  <profile:parameterName rdf:resource="subscribe-success"/>
  <profile:refersTo rdf:resource="&fipa;subscribe-success"/>
  <profile:restrictedTo ref:resource="&daml;Thing"/>
 </profile:ParameterDescription>
</profile:output>
</profile:OfferedService>
```

Figure 3: Sample Profile

scription defines the parameters of the service, cross-referenced to the corresponding parameters of the process of which the service is an instantiation. In conventional DAML Services usage, the parameter type restriction in a service's profile (expressed using the restrictedTo property) should be consistent with the range of the parameter properties on the process on which the service is based, but there is no logical constraint expressed within DAML-S which requires this.

We have adapted this usage so that the range of the process parameter is a superclass of the profile parameter restriction. The process therefore gives an abstract, domain-neutral description of the ACL performative which characterises the service, and the profile gives a more domain-specific description of the service which constrains the parameter type. For example, Figure 3 gives the profile of a service from our situational awareness system for a simulated humanitarian aid scenario (see Section 5 for further details of this system). This service allows an agent to subscribe to reports about UNHCR vehicle movements (terms from the humanitarian aid domain ontology are indicated by the use of the flood namespace). The restriction on the input parameter of this service is the class of reports about the movements of vehicles belonging to the UNHCR, which is a subclass of the range of the corresponding parameter on the abstract Subscribe process (Thing, the most general class) as shown in Figure 2.

The profile in Figure 3 is of type OfferedService, indicating that this is a service advertisement; an agent requesting a service would construct a service profile of type NeededService. This use of offered and needed services allows service brokers to support interactions that are driven both by the clients (those requesting services) and the services (those providing services).

The process and profile components of the service description are referenced together in a top-level service description (see Figure 4) which also includes a reference to the protocol which is to be used to access the service, known as the *service grounding*. In our example, we have chosen to make the service available via SOAP, hence the reference to the W3C Note which defines the protocol by means of the supports property. At the time at which this work was carried out, this area of DAML Services was still largely undefined, and there was no standard vocabulary for grounding DAML-S services using WSDL. This has subsequently been addressed in the most recent version of the DAML-S specification, but these additions to the specifications have not yet been reflected in our software.

In Figure 5, we have constructed a simple SOAP message (using the FIPA inform performative) which contains a report from a UNHCR vehicle (about itself) that is moving with a certain bearing and speed from a certain location. Again, it should be stressed that the manner in which we have written this message is the result of an informed guess as to how one would pass RDF fragments as parameters to Web Services, and as to how a service specified by DAML Services might be grounded in SOAP.

4. QUERY LANGUAGE

When the service in the Subscribe example is invoked, the value of the input parameter should be an instance of the class restriction which is given as the input parameter types in both the profile and the process descriptions. For the various query performatives (query-if, query-ref and subscribe), this input parameter contains the query expression which would be contained in the message content in a conventional agent-based system. However, there is as yet no standard query language for RDF, DAML+OIL or OWL, although there are several under development. One such example is DAML Rules [8], which builds on DAML+OIL and expresses queries as Horn clause-like structures.

Due to this lack of any standard format for expressing queries, we have chosen to express queries as anonymous resources, also known as blank nodes or bNodes. These are instances which are not identified by a URI, but by the values of their properties. As a result, they can be considered to be existentially quantified query

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE uridef[
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY daml "http://www.daml.org/2001/03/daml+oil.daml#">
  <!ENTITY flood 'http://example.org/ontology/flood#'>
  <!ENTITY fipa 'http://www.fipa.org/ontology/acl#'>
  <!ENTITY service "http://www.daml.org/services/daml-s/2001/10/Service#">
]>
<rdf:RDF  xmlns:rdf = "&rdf;" xmlns:rdfs ="&rdfs;"
          xmlns:daml ="&daml;" xmlns:service = "&service;">
 <daml:Ontology rdf:about="">
   <daml:versionInfo>0.1</daml:versionInfo>
   <rdfs:comment>A description of the UNHCR subscribe service</rdfs:comment>
   <daml:imports rdf:resource="&daml;"/>
   <daml:imports rdf:resource="&service;" />
 </daml:Ontology>

 <service:Service rdf:ID="UNHCR-Subscribe">
   <!-- Reference to the UNHCR-Subscribe Profile -->
   <service:presents rdf:resource="&flood;UNHCR-Subscribe-Profile"/>
   <!-- Reference to the FIPA Subscribe Process Model -->
   <service:describedBy rdf:resource="&fipa;#Subscribe"/>
   <!-- Reference to the SOAP service grounding -->
   <service:supports rdf:resource="http://www.w3.org/TR/SOAP/"/>
 </service:Service>
</rdf:RDF>
```

Figure 4: Sample Service

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope">
 <env:Body>
  <fipa:inform-proposition xmlns:fipa="http://www.fipa.org/ontology/acl#">
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:flood="http://www.qinetiq.com/ontology/flood#">
    <flood:Report rdf:about="">
     <flood:reportsOn>
      <flood:MovementEvent>
       <flood:actor>
        <flood:Vehicle rdf:about="&flood;UNHCR-2323">
         <flood:memberOf rdf:resource="&flood;UNHCR"/>
        </flood:Vehicle>
       </flood:actor>
       <flood:headingTowards>
        <flood:Direction>
         <flood:bearing>284.5</flood:bearing>
         <flood:velocity>43</flood:velocity>
        </flood:Direction>
       </flood:headingTowards>
       <flood:locatedAt>
        <flood:Location>
         <flood:longditude>32.23427</flood:longditude>
         <flood:latitude>16.33871</flood:latitude>
        </flood:Location>
       </flood:locatedAt>
       <flood:occursAt>2002-04-12T12:23:48</flood:occursAt>
      </flood:MovementEvent>
     </flood:reportsOn>
     <flood:reporter rdf:resource="&flood;UNHCR-2323"/>
     <flood:certainty>1.0</flood:certainty>
     <flood:occursAt>2002-04-12T12:23:48</flood:occursAt>
    </flood:Report>
   </rdf:RDF>
  </fipa:subscribe-reference>
 </env:Body>
</env:Envelope>
```

Figure 5: Sample SOAP message

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE rdf:RDF [
 <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
 <!ENTITY flood 'http://example.org/ontology/flood#'>
]>
<rdf:RDF xmlns:rdf="&rdf;" xmlns:flood="&flood;">
 <flood:Report>
  <flood:reportsOn>
   <flood:MovementEvent>
    <flood:actor>
     <flood:Vehicle>
      <flood:memberOf rdf:resource="&flood;UNHCR"/>
     </flood:Vehicle>
    </flood:actor>
   </flood:MovementEvent>
  </flood:reportsOn>
 </flood:Report>
</rdf:RDF>
```

Figure 6: Sample Query

expressions which denote objects of interest. In effect, the query is a template subgraph which is matched against the RDF graph, with the query solutions being the locations where the template matches. The intention behind this approach is to allow us to express the capabilities of a service by describing the class of queries which may be asked of that service. The class expression given as the input parameter type in the profile should contain as instances all the possible queries (expressed using the bNode technique) that may be asked of the service.

The main limitation of this approach is that it is only applicable to queries of a certain structure. The item about which the query is phrased (the anonymous resource) must be the subject of the RDF triples, not the object, if more than one property of the resource is to be specified (this is largely a limitation brought about by the RDF syntax [14]). This is a significant limitation, but also one which can be mitigated against by a suitable design of the domain ontology in which the objects which are most likely to be the subject of queries appear as the subjects of RDF triples rather than as the objects (see Section 5.2 and Figures 8 and 9).

As an example, the domain ontology which we have designed for this application is centred around events and reports of events. We have taken the approach that communication in the system will be about these events and reports (rather than about any persistent world state which the reports might suggest), so the queries can be expressed using the anonymous resource technique by specifying the properties that the report (and the event it contains) must possess. It should be noted, however, that we did not specifically design the ontology in this report to circumvent the expressive limitations of our chosen query language, but rather that the query language was chosen because it was appropriate for use with the domain ontology that we had already designed.

In Figure 6 we show an RDF fragment which expresses the notion that there exists some report which reports on the movements of vehicles owned by the UNHCR; this may be interpreted as a query about reports with those properties.

An additional limitation of this approach to query construction, which is unfortunately also shared with several of the other query languages currently under development, is that it is not possible to specify literal ranges in queries. An example of such a literal range might be a query of movement reports about entities that were north of a particular point, or to put it a different way, whose latitude was greater than a certain value. This limitation arises because the RDF and DAML+OIL models have no notion of how different literal datatypes behave (particularly with respect to ordering and inequalities). However, the W3C Web Ontology Language working group has identified this as a desirable feature for the language that they are currently developing; although it is still to early to tell conclusively, this work may produce a solution to this problem. The

likely solution to this problem is unlikely to involve changes to the model theories of DAML+OIL and the other ontology languages. It will most probably be based on the promotion of *oracles*, entities which have specific knowledge of the behaviour of different literal types (integers, latitude/longitude pairs, dates, etc) and which may be used by inference and query engines to evaluate tests based on those types.

At present, the development and standardisation of query languages for the Semantic Web is largely immature. As Semantic Web development in general becomes more mature, we expect that the current Precambrian period-like diversity of query languages will come to a close with standardisation on a small number of languages. The investigation and design of suitable query languages for use with the style of agentified Web Services that we discuss in this paper therefore remains an open direction for future research.

5. A PROTOTYPE AGENT WEB SERVICES SYSTEM

As a proof of concept of the technologies discussed above, we have designed a system which demonstrates the use of Agent Web Services in the application domain of situational awareness in a humanitarian relief scenario.

The scenario for this study is set in a river delta region which has experienced flooding due to unseasonally heavy rainfall. The people who have been displaced from their homes by the flooding are being sheltered in relief camps. The timeline for the scenario includes a rapid flooding event which forces the creation of new relief camps, and a hostile event upon a relief convoy which requires military intervention and support.

The system contains a number of agents which generate reports on the state of the world (eg. refugee movements, meteorological reports and forecasts) with differing degrees of certainty. A feature of this scenario is that it includes a number of different types of user, each of which has different information needs, and so each of which should be sent a different subset of the reports generated by the entities in the system. The aim of this system is the provision of filtered report streams to these users in a timely manner, a process often referred to as *information triage*.

In addition, the sets of agents which produce and consume reports are not static; agents may join and leave the system while it is running. The requirement for the system both to adapt to the loss of agents, and to opportunistically integrate new agents provides a motivation for the semantically richer service descriptions that were discussed earlier in this paper.

5.1 System Architecture

In our system architecture, illustrated in Figure 7, the flow of information is from left to right. On the left are various data sources which correspond to entities in the domain environment and generate streams of reports about different types of events, while on the right are consumers which take the reports and present them to the user (the map panel), or which perform some further processing on the reports.

The key component of the system is the central broker which mediates the interaction between the other system components. Data consumers use the broker to find sources which can satisfy their information needs, which typically depend on the user view which is being presented, by registering a service requirement (expressed as a `NeededService` in the DAML Services profile model) with the broker. Similarly, the data sources register a service advertisement describing their capabilities with the broker (expressed as a `OfferedService` in the DAML Services profile model). The

broker compares the service requirement to the service advertisements that it has received from the data sources and responds with the matching services. The consumers then communicate directly with the data sources, typically by formulating a subscription to some subset of the reports that the source offers, as was illustrated in Figure 3.

The data sources in the system are grouped into three rough categories. The first category consists of entities which have GPS devices and so can produce high-certainty reports of their own positions and movements. The second category consists of entities which are able to observe their immediate environment, and so are able to generate moderate certainty reports on the movements of other entities, on hostile, support and relief events or on changes to the infrastructure present in the environment (damage to roads and bridges, for example). The final category consists of meteorological sensors which provide reports on the level of the flood waters.

5.2 Domain Ontology Design

As a demonstration of our approach to Agent Web Services, we have designed an ontology to describe the application domain and scenario that we have outlined earlier. This domain has a number of features which are interesting from an information management point of view. An agent in such a system is unlikely to have direct knowledge of the status of entities in the domain environment, since almost all knowledge is mediated through reports of events (entity state changes) which are issued by other entities in the system. For this reason, the provenance and certainty of the reports become of prime importance, and the role of the agents through which users interact with the system becomes one of information triage and filtering.

Therefore, the queries which agents ask of the system are less likely to be about domain entities directly, and are more likely to be about reports about those entities. In a scenario where there may be many conflicting and partial reports, the query idiom would be to ask only for high certainty reports from trusted sources about those entities which are of interest, but one has the ability to configure or change this assumption. To this end, the ontology that we have designed comprises two main parts. The first part consists of the entities in the domain environment and their invariant properties, as shown in Figure 8

The other part of the ontology consists of the events which describe changes to the state of the entities, as shown in Figure 9. The key class in this hierarchy is the `Report`, which represents information which has been gleaned from some source (newsfeed, satellite image, etc) about some event which has occurred in the environment. For example, if the movement of an entity (a relief convoy) has been observed by a third party (a journalist from CNN), this datapoint is represented by a report about a movement event by the convoy, which has been reported by the journalist. This approach captures the provenance of the report (as the entity which reported it), as well as the degree of certainty that the reporter has in the report. We have adopted the Stanford Certainty Factor Algebra [4] for dealing with certainty measures; although this has some shortcomings, it is a well-understood formalism and provides a general representation of confidence.

The separation of reports from events makes it possible to separate the time at which the event occurred from the time at which the report was made, which allows us to represent both the timeliness of reported events (yet another report facet which can be used to filter the stream of incoming reports) and also to represent event predictions as reports about future events. Finally, by making `Report` a type of `Event`, we make possible secondary reporting (reports about reports).
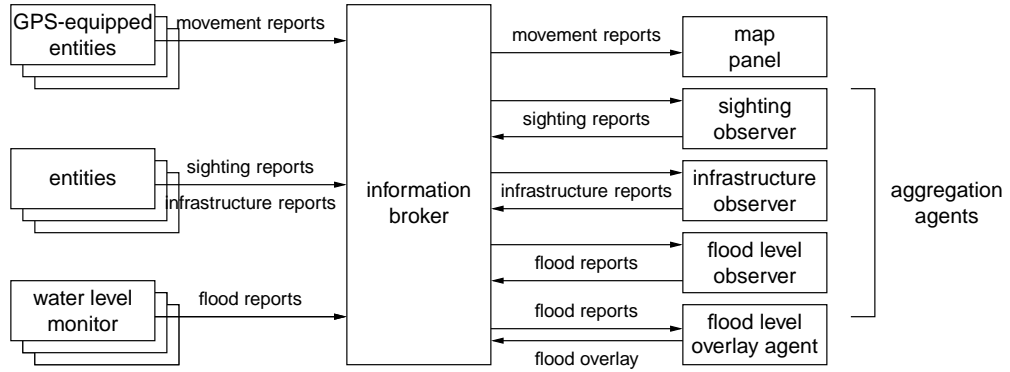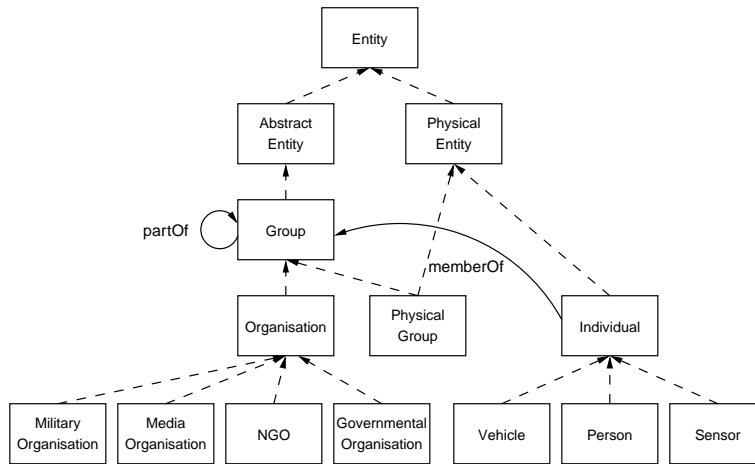
Figure 7: System architecture

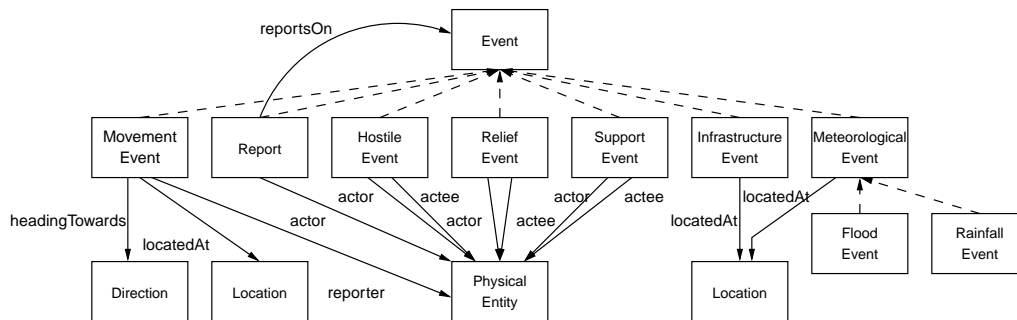**Figure 8: Domain ontology - entity**

Figure 9: Domain ontology - event

We have designed this ontology using the Protege ontology editor [15] which uses RDF Schema as its output format, but in the examples that follow in this report, we use DAML+OIL constructs to define new (unnamed) classes based on the primitive classes in the RDF Schema ontology. An example of these unnamed classes can be found in the parameter restriction in Figure 3. These unnamed classes are especially useful because they allow us to specify classes which were not explicitly created by the ontology designer by describing the necessary and sufficient conditions for class membership. The ability to specify unnamed classes simplifies ontologies by removing the need to state

, and is a particular strength of description logic characterisations of ontologies.

5.3 Simulator Software

Our proof of concept implementation is a Java application which simulates a day of events for the situational awareness system. The events and reports themselves are predetermined and are served according to a script, but the behaviour of the information agents (requesting services and responding to service requests) is not fixed beforehand.

The application consists of two parts, a map panel which provides an overview of the entities in the simulated environment and the current whereabouts, and a control panel which provides a more detailed view of the flow of reports in the system and contains a number of report consumers (users).

The map panel shown on the right of the diagram in Figure 7 is a canonical example of a consumer; it registers its service requirement with the broker, and then subscribes to movement reports from the relevant sources (see the map panel screenshot in Figure 10). The next three consumers provide an aggregation service to the system (and so are not 'pure' consumers) by subscribing to certain types of report, cross-correlating those reports which deal with the same event, and then generating new composite reports which describe those events (often with greater certainty, due to the combination of knowledge from different sources). Finally, the flood level overlay agent takes flood reports and generates an overlay for the map panel which indicates the areas of the map which are under flood waters (again, see the screenshot in Figure 10).

The control panel for the simulator, pictured in Figure 11, provides an overview of the system. For this simulation, we assume a context in which users have an interest in supply, an interest in keeping the bridges up and a requirement to understand. Consequently, the three panes in the middle of the panel show the filtered streams of reports which are being delivered to the various users in the system (respectively, logistics, civil engineering and meterological). The scrolling panel at the bottom shows an aggregated feed of all the reports that have been made in the system (effectively a user who subscribes to all reports).

As a proof of concept and illustrative example, our demonstration implementation of this system differs from the description above in several important ways. We have chosen not to implement the interactions between system components using Web Services technologies such as SOAP because the combination of these with DAML Services and other Semantic Web technologies was still largely undecided; at the time at which this part of the system was written, there was no way to specify a SOAP binding for a service in DAML-S. This has since been rectified in the most recent version of DAML-S, as has been noted earlier in this paper. In our implementation, the agents communicate using standard Java method invocation. This choice is largely unimportant because the process of transferring the system to use SOAP as a message transport technology instead of Java is straightforward, given the procedure-call
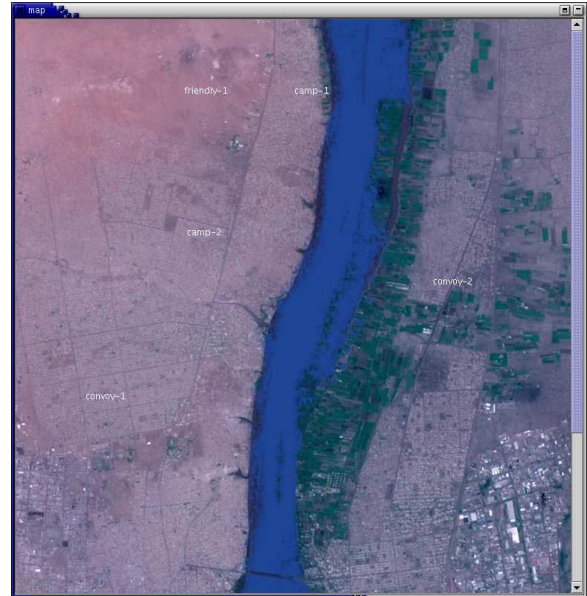

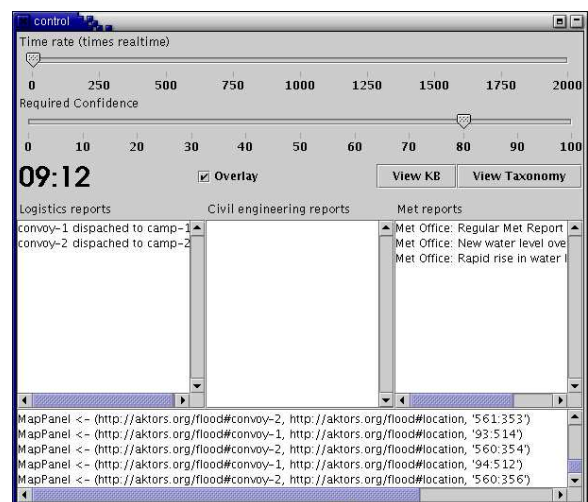**Figure 10: Simulator Map Panel**

Figure 11: Simulator Control Panel

approach that we have taken in the FIPA ACL process ontology.

The hybrid WSDL-based service grounding in the most recent version of DAML-S is sufficiently expressive for us to be able to use SOAP instead of Java method invocation for communication; our future plans for this system include a migration to a SOAP-based message transport layer.

Even though we do not currently use Web Services technologies for message transport, our implementation is still ontologically informed; the reports and events are described using our domain ontology, and expressions from the domain ontology are used as service requests, demonstrating that a formal ontology of a domain can be effectively used to filter and aggregate knowledge and services.

6. CONCLUSION

In this paper, we have described our experience of building a flexible agent-based Web Services system which performs information triage on heterogeneous streams of data in order to provide a situational awareness capability in a simulated humanitarian aid scenario.

A key aspect of the system design is the separation of the intentional force of the messages from their application domain-specific content, and the embodiment of this separation in the process and profile components (respectively) of a DAML Services service description. The resulting rich service descriptions provide a pwerful way of assembling information resources in contexts that require the agile construction of virtual organisations. This agent-based perspective on Web Services is very consistent with the views on the construction of distributed information systems to be found in the Semantic Web and also the Semantic Grid[7].

This work has highlighted the need for expressive query languages which fit well with existing Web Services and Semantic Web technologies. In addition to the migration of the message transport layer to SOAP as mentioned in the previous section, our plans for future work on this system include the investigation of such query languages.

7. ACKNOWLEDGMENTS

8. REFERENCES

[1] P. V. Biron and A. Malhotra. XML Schema Part 2: Datatypes. W3C Recommendation, World Wide Web Consortium, May 2001.

[2] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. W3C note, World Wide Web Consortium, May 2000.

[3] D. Brickley and R. Guha. Resource Description Framework (RDF) Schema Specification 1.0. Technical Report CR-rdf-schema-20000327, World Wide Web Consortium, Mar. 2000.

[4] B. Buchanan and E. Shortliff, editors. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, 1984.

[5] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. W3C note, World Wide Web Consortium, Mar. 2001.

[6] D. Connolly, F. van Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. DAML+OIL (March 2001) Reference Description. W3C Note, World Wide Web Consortium, Dec. 2001.

[7] D. de Roure, N. Jennings, and N. Shadbolt. The semantic grid: A future e-science infrastructure. *Int. J. of Concurrency and Computation*, 2002. (to appear).

[8] S. Decker. DAML Rules - An RDF Query, Inference and Transformation Language. Draft available online at `http://www-db.stanford.edu/~stefan/daml/2001/07/03/rules/damlrules.ps`, 2001.

[9] FIPA. FIPA Communicative Act Library Specification. Technical Report XC00037I, Foundation for Intelligent Physical Agents, Oct. 2002.

[10] L. Gasser. MAS infrastructure definitions, needs and prospects. In *Proceedings of the First Workshop on Infrastructure for Agents, MAS and Scalable MAS at the Fourth International Conference on Autonomous Agents (ICMAS2000)*, 2000.

[11] A. Gómez-Pérez and O. Corcho. Ontology languages for the semantic web. *IEEE Intelligent Systems*, Jan.–Feb. 2002.

[12] N. Guarino and P. Giaretta. Ontologies and knowledge bases: Towards a terminological clarification. In N. Mars, editor, *Towards Very Large Knowledge Bases*. IOS Press, 1995.

[13] KAG. An overview of KQML: A knowledge query and manipulation language. Technical report, KQML Advisory Group, Mar. 1992.

[14] O. Lassila and R. Swick. Resource Description Framework (RDF) model and syntax specification. Technical Report REC-rdf-syntax, World Wide Web Consortium, Feb. 1999.

[15] N. Noy, M. Sintek, S. Decker, M. Crubezy, R. Fergerson, and M. Musen. Creating Semantic Web Contents with Protege-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2002.

[16] J. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.

[17] The DAML Services Coalition. DAML-S: Semantic Markup for Web Services, Dec. 2001.