15th ICCRTS

"The Evolution of C2"

Semantically Enabled QoS Aware Service Discovery and Orchestration for MANETs

Paper ID 077

Topic 9: C2 Architectures and Technologies, Topic 2: Networks and Networking

Trude Hafsøe, Frank T. Johnsen, Marianne Rustad

POC:

Trude Hafsøe

Norwegian Defence Research Establishment (FFI)

P.O. Box 25, NO-2027 Kjeller, Norway

+4763807431

trude.hafsoe@ffi.no

Abstract

In this paper we present our semantically enabled Quality of Service (QoS)-aware service discovery solution for dynamic environments such as Mobile Ad Hoc Networks (MANETs). Through the use of a decentralized service discovery mechanism coupled with liveness properties of services we get the needed resilience to changes that are required in a MANET.

Our novel solution includes an extended service description which, in addition to the standard Web service parameters, also allows for the description of both static and dynamic QoS parameters. The service descriptions have a supplementary description where a semantic layer adds meaning to the elements in the extended service descriptions. This is done using a Semantic Markup for Web Services (OWL-S) based service ontology, thus augmenting the solution with semantic capabilities.

Our experimental service discovery mechanism supports the distribution of extended service description parameters. These parameters are linked to an OWL-S based profile that can be used to perform more accurate service selection at the client side.

Introduction

The Service Oriented Architecture (SOA) concept, with loosely coupled entities and clearly defined interfaces is highly suitable for dynamic networks. In situations where heterogeneous nodes come together to perform a joint operation, such as search and rescue operations or internationally coordinated military operations, having the ability to quickly set up a functional information sharing capability is essential. SOA, implemented using Web services has, despite being designed for use in fixed infrastructure networks, shown promise when being extended to dynamic, low capacity networks such as military tactical networks.

There are, however, a number of challenges related to the use of Web services in disadvantaged grids, one of which is Web services discovery. In an operation the participating nodes will be mobile, and they will be likely to experience both network partitioning and/or loss of connectivity. A squad will need to access locally available services, even when connections to other networks or other partitions of the same network fail. In such a scenario, relying on a centrally located registry for Web services can lead to the loss of the discovery capability if the reach back link goes down. Nodes in the network will thus be prevented from finding local services even if these services are still available. In order to maintain the availability of the discovery capability, and thereby ensure that all currently available services can be located, we need a discovery mechanism that is suitable for mobile ad-hoc networks (MANETs).

Operations vary greatly in complexity, making it difficult to accurately predict which units, how many units and what the capabilities of the units involved in the situation are ahead of time. The units involved are likely to be heterogeneous and their capabilities, both when it comes to connectivity and end system resources, may differ from one client to the next. These differences in capabilities mean that not all clients will be able to use the available services in the same way, even if they need to access the same information. One example of this could be that a client using a handheld device such as a rugged PDA would require images provided by a camera service to be transmitted at a different resolution or using a different file format than a client using a laptop computer. In order to enable each client to find the service that can best satisfy their requirements, the service discovery

and selection mechanism must, in addition to the standard properties of a service, also take the Quality-of-Service (QoS) attributes of the services into account.

The benefit of including QoS properties is that it allows for more advanced service discovery and selection, but it introduces a requirement for an extended service description, and a means of using this new information to select which services to use.

Semantic Web services aim at enabling a more automatic and dynamic service environment. Services are described by their capabilities and properties rather than by syntactic WSDL descriptions of endpoints and data types. In QoS sensitive settings, such as tactical MANETs, non-functional properties need to be considered during service selection. Based on Semantic Markup for Web Services (OWL-S) we use a subset we call *OWL-S Light with QoS*, or OWL-S LiQ for short, for description, distribution and selection of services. The service ontology subset defines the minimum of conceptual elements needed to describe services and non-functional QoS concepts important in a QoS-aware environment. Individual services are described according to the ontology and the descriptions are distributed to clients and reasoned upon during service selection. Service selection has two reasoning parts, first based on service capabilities, then based on QoS parameter values. The semantically enabled QoS aware service discovery solution we suggest in this paper addresses all phases of service discovery and selection: We suggest a tailored service *description*, a mechanism for *distributing* service information, and discuss how QoS attributes can be integrated into the service *selection* phase.

Related work

The term QoS has been used to describe many different aspects of computer systems, with one of the more common usages being as a reference to network resource reservation schemes rather than the quality aspects of an application or a computer system. However, with the advent of multimedia applications, the QoS concept has been extended to cover not only network characteristics, but also those aspects of both middleware and applications that affect the end user's satisfaction with the service or application.

An established and more descriptive definition of QoS is the one given by Vogel et al. [2]: *"Quality-of-Service represents the set of those quantitative and qualitative characteristics of a distributed multimedia system necessary to achieve the required functionality of an application."*

This definition, along with accompanying division of QoS parameters into five categories, was designed to describe multimedia systems, but many aspects of it also apply to other application types. The five categories of QoS defined in that paper are performance-, format-, synchronization-, cost-, and user-oriented parameters, all of which are equally significant in a Web service setting as they are for multimedia applications.

It is important to note that these parameter categories affect each other; for an end user, the user oriented parameters might seem most significant, but these parameters are in turn dependent on the performance-, format-, and synchronization-oriented parameters. For instance, to perceive the subjective image quality of a video as being satisfactory, the video must be coded in a format and resolution that allows a sufficiently high image quality. Next, the system must be able to provide a sufficiently high bandwidth to support the bandwidth requirement of the video, and the video and audio tracks need to be played back in a synchronized fashion.

There are currently no finished standards for describing and handling the quality aspects of Web services, but work is ongoing within OASIS. Their Web Service Quality Definition Language (WS-QDL) [3] is an XML based language for describing QoS attributes of services, but this effort is geared towards Internet and business use of Web services, and thus has a focus on more high level QoS parameters such as cost and billing. QoS support in MANETs requires a more flexible and light-weight solution for QoS handling.

Existing frameworks that support QoS for Web services are largely limited to experimental solutions. One such framework, WS-QoS [4] supports not only the description of QoS parameters for Web services, but also how to use these QoS descriptions in conjunction with QoS mechanisms in underlying networks. This work addresses not only the service discovery process, but also the invocation of services and monitoring of Web service quality aspects. Its service description and discovery are however closely linked to service registries, which makes it less suitable for use in limited resource networks such as MANETs.

OWL-S [10] is a Semantic Web services initiative that defines a service ontology for describing Web services. The ontology incorporates the possibility to add additional service parameters, but has no specific QoS concepts defined.

WSMO [12] is another initiative that defines a conceptual framework for Semantic Web services through a service ontology. As part of their service descriptions they represent network level QoS and other interesting non-functional aspects of services. WSMO can be quite complex, which is something we want to avoid in our solution.

SAWSDL [13] (Semantic Annotations for WSDL) facilitates service descriptions with semantics, but does not have a service ontology specified.

OWL-Q [5] is as an alternative QoS focused ontology language. OWL-Q is a syntactical separated extension to OWL-S defined as a set of upper ontologies. The solution combines reasoning with the Constraint Satisfaction Problem and SWRL [11] for QoS service description and discovery. The service description defined in that paper allows for the description of QoS parameters of services, but the work focuses mainly on the description and selection phases. Thus, it relies on existing mechanisms (i.e. registries) for making the service descriptions available to clients. Handling dynamic QoS parameters requires a distribution mechanism that is more flexible, without generating a need for sending full semantic service descriptions between nodes whenever an update of the service parameters occur.

QoS in mobile Web services

As previously mentioned, QoS parameters can be categorized according to which aspects of the described application or service a parameter affects. For the use of Web services in a mobile setting, we divide QoS parameters into categories according to how they need to be handled by the service discovery mechanism.

Static QoS parameters are parameters that are known at the time when a service is deployed, and do not change during the lifespan of the service. These parameters are entered into the service description before the service is deployed, and because they remain unchanged, it is sufficient to

distribute these parameters once to each client. Examples of such parameters are the maximum available resolution of a camera connected to an imaging service, or the data formats the images can be provided as.

The *dynamic QoS* attributes of a service instance need to be included in the same service description as the static parameters, but due to them changing over time they cannot be determined before the service is deployed, and will need to be updated whenever a change occurs. Because of the need for frequent updates, the current values of these parameters need to be included in the service announcements. Most dynamic QoS parameters, such as information about the availability of a service (for instance uptime and current server load) are known by the service, or the server hosting the service, and can thus be included in the service descriptions published by the server. Some QoS parameters, particularly those related to the availability of network resources, depend on the network connection between a given client and service, and must be calculated by the client side. These parameters, which include round trip times and available bandwidth, can be included in the same service description as the other parameters. However, the inclusion of such parameters will require the client node to be able to calculate the current values of the parameters and insert them into the service description themselves. Because of the complexity of handling such parameters, we have not included these in our current implementation.

Adding QoS awareness to Web services is complex, and requires QoS support to be integrated in everything from Service Level Agreements and service descriptions to the ability for a client to specify QoS needs during service invocation. Additionally, we need mappings from higher level QoS parameters down to simple networking parameters such a traffic priority and classification.

An important first step in implementing QoS awareness in Web services is to QoS enable the service discovery process, in which the clients gain knowledge about existing services and the QoS these services support. In order to successfully implement such a QoS aware service discovery mechanism there are a number of issues that must be addressed:

First, an *extended service description* that includes information about the QoS attributes of the services must be generated. This extended service description must be designed in such a way that new QoS attributes can be added to the descriptions with little effort. The reason for this requirement is that which QoS attributes must be supported will vary from service to service, and may also change over time. It is important that this service description can handle both static and dynamic QoS parameters.

Second, a mechanism for *distributing service information* about available services to clients is needed. This mechanism must provide the client with all the information the client needs to make a well founded decision on which services that best fits the client's requirement.

Last, the client system needs to be able to make the decision about which services can fulfill the client's needs based on the client's capabilities. This *service selection* can take on many forms, from simple parameter matching in low end clients, to advanced service orchestration by more capable clients. In order to enable these more advanced selection scenarios, the client needs to maintain a QoS profile that specifies both its preferences and abilities.

Semantic Web services with QoS

OWL-S is a Semantic Web services initiative that defines a service ontology through Web Ontology Language (OWL) constructs. OWL gives us the logical foundation we need to reason about services. OWL-S is an extensive service ontology, but in a MANET setting we try to keep the descriptions at a minimum as we have to consider possible limitations on client equipment and network capacity. Possible equipment and network limitations motivated us to use a subset of OWL-S with QoS parameters and in this way make the ontology as small as possible. OWL-S has the ability to express non-functional service parameters, and as QoS is such an important part in a MANET setting we introduce this as a full-fledged part of the service ontology while still keeping the descriptions compliant with other OWL-S descriptions.

From OWL-S we have a service description with profile, process and grounding. The *Profiles* are for advertisements and discovery. The *Process* describes how to use the service and to define orchestrations. The *Grounding* describes the technical information needed for invoking a service. This OWL-S based ontology is extended in the profile with a QoS ontology parameter. The QoS ontology is connected to the service profile through the Service Parameter concept and the QoS concept. A first version of the OWL-S LiQ ontology (without the property names) is as shown in Figure 1. It shows some top level elements defined in OWL-S and also the additional QoS element added in OWL-S LiQ. It defines the conceptual parts of a service. Advertisements are possible through the profile, the process and orchestration enabling process concept, and the physical invocation details listed in the WSDL defined in the grounding.

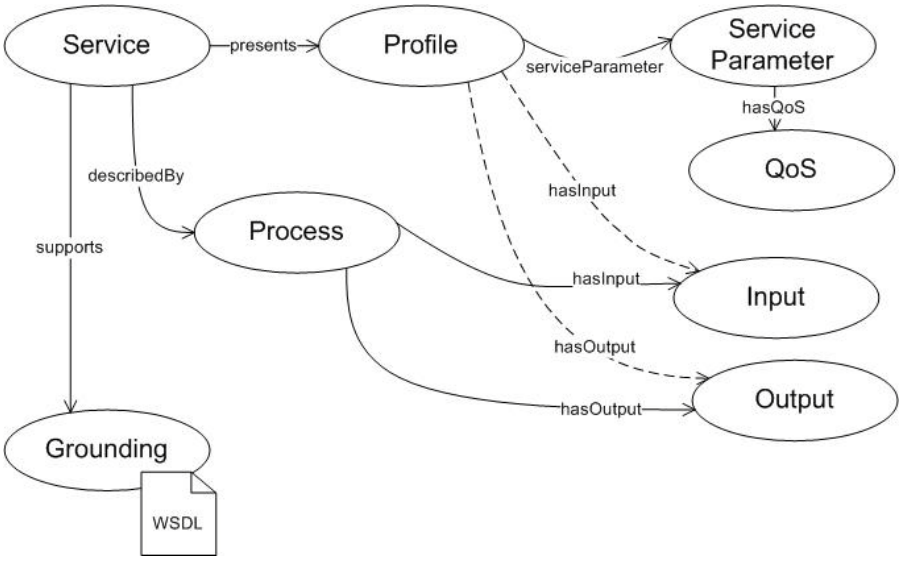

Figure 1 OWL-S Lite with QoS ontology

OWL-S LiQ describes the very simple advertisement setup using the profile part of the ontology. Based on limited processing and network resources often found in MANETs we decided to eliminate as much as possible. The new QoS concept is used in the profile to help find the most suitable service for the client's possibilities and the service abilities.

The process part of the ontology describes how a service invocation flow is to be executed. Both *CompositeProcess* and *AtomicProcess (not shown in the figure)* are subtypes of the process concept and has by this input and output defined.

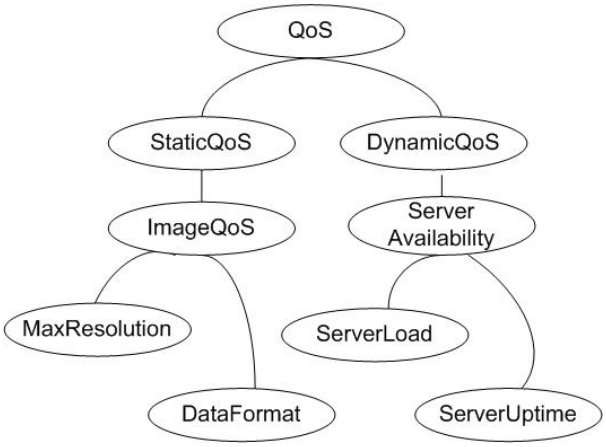

Figure 2 OWL-S Light with QoS, the QoS ontology

Figure 2 shows the first edition of the new QoS ontology. The ontology is limited to our example but will be expanded when needed.

To sum up this service ontology we see that concepts are adopted from OWL-S in order to describe functional aspects of services, these concepts are used for reasoning about the functional aspects of the services enabling clients to find services based on the service capabilities and properties. The QoS ontology used in the profile enables clients to filter discovery results based on QoS parameters. The QoS ontology enables reasoning on QoS the same way as for functional service capabilities.

Representing QoS with XML

In order to be able to include QoS information in the service advertisements, we need to express the QoS parameters using XML. To do this we use a simple XML schema which defines the metadata in terms of an URI identifying a semantic profile. In addition to this URI, a list of QoS parameters can be expressed. For example, a simple image service providing output in a certain resolution but with a choice of different image formats can be expressed as shown in Figure 3.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<QoS xmlns="http://xml.netbeans.org/schema/QoS_metadata">
  <ProfileURI>http://my.profile.org/semanticwonders</ProfileURI>
  <Parameter value="320" name="X"/>
  <Parameter value="240" name="Y"/>
  <Parameter value="gif" name="format"/>
  <Parameter value="jpg" name="format"/>
  <Parameter value="png" name="format"/>
  <Parameter value="raw" name="format"/>
</QoS>
```

Figure 3 Example metadata for a simple image service.

The service descriptions are pre-distributed together with the service ontology and the other domain specific ontologies. When a server gets ready to advertise its service, it has a service template, which

is used to define the appropriate URI of the service along with a list of dynamic QoS parameters. When the advertisements arrive at the client they are stored in the repository of adverts. At selection time these adverts are used to fill instances of service templates to enable semantic matching based on service functionality and non-functional aspects that might have changed from the first advertisement was received. Further details of the distribution mechanism are discussed in the following section.

Distribution

There are several challenges that arise when attempting to perform service discovery in MANETs. Below we summarize some of the most important challenges as identified by [9]:

- Dynamic environments may lead to frequent change in both service metadata (service descriptions) and the topology of the nodes that are part of the system. Frequent topology change means that both service nodes and registry nodes can come and go.
- A proper service discovery architecture for such an environment would reduce the amount of manual configuration, enable automatic discovery and selection of relevant services, and offer a complete and up-to-date picture of the services available at the given point in time.
- Service discovery should work in environments disconnected from the Internet. Moreover, it should be robust in terms of partial failure.
- The system should allow flexible resource utilization, since capacity (memory, CPU, storage) and connectivity can differ from node to node.
- The infrastructure should support different kinds of service description mechanisms, ranging from simple to rich (i.e. semantic) descriptions. Thus, both normal Web services as well as Semantic Web services should be able to use this infrastructure.

Considering these requirements, we look at each of the existing solutions for Web services in turn: Currently, there are three standards related to service discovery for Web services. All three standards are under OASIS. The registry standards, UDDI [6] and ebXML [7], define central registries for use in wide area networks or local area networks. In order to find a service, the client needs to access the registry, search for an appropriate service, and choose a service from the query reply. However, this solution was developed for business use over the Internet, where connections are fixed, have a high bandwidth, high uptime, and nodes are immobile. In a MANET the network topology is unpredictable, and if the network is partitioned then only clients in the same partition as the registry will be able to discover services, whereas the clients in other partitions will not. This means that a client residing in the same partition as the service it needs may be unable to use it, just because it cannot access the registry to discover it. In contrast, a client that is able to access the registry may discover a service that it is unable to connect to, since the service resides in another network partition. These two aspects (shown in Figures 4 and 5) are important drawbacks with registries if you attempt to use them in a dynamic environment such as a MANET.

WS-Discovery [8] is a standard which attempts to remedy the drawbacks of registries. It is designed for use in local area networks, and is based on multicasting queries and responses. This means that by using WS-Discovery, you can discover services in your partition and use them. There is no single point of failure in WS-Discovery, and the query responses mirror the current network state.

All three discovery mechanisms support simple service descriptions. The registries have third party support for semantics as well. WS-Discovery in its current form has no support for semantic descriptions at all, and is thus not suitable for discovery of services beyond that provided by simple descriptions. As we can see, there is currently no standardized solution for Web services that supports semantic service discovery in MANETs, and there is a need for such a solution. Lacking a standard, we have implemented the necessary functionality in an experimental solution which performs distribution of Service Advertisements in MANETs (SAM). The differences between the existing standards and our experimental solution are described in Table 1.

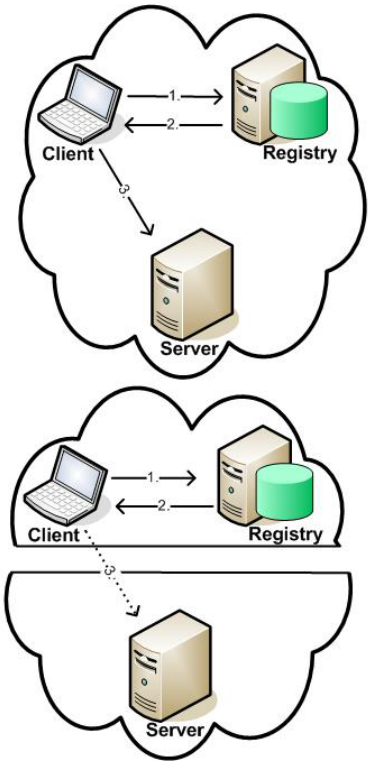

Figure 4 Partitioning of the network leaves the server inaccessible.

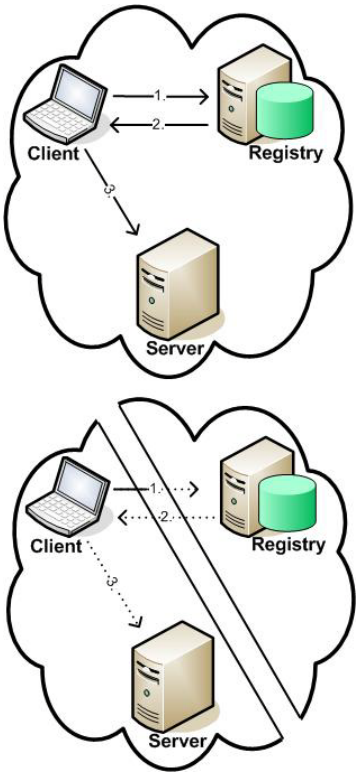

Figure 5 Partitioning of the network leaves the registry inaccessible.

	Registries	WS-Discovery	SAM
Can work disconnected from the Internet?	UDDI: No ebXML: Potentially, since it has a repository which can host important data	Potentially, as long as you only need information from the WSDL and no additional schemas	Yes
Resilient to partial failure?	No	Yes	Yes
Support for user defined metadata?	Yes	No	Yes
Support for semantic descriptions?	Yes, third party support	No	Yes
Gives a current and up-to-date picture of services?	No, requires services to actively deregister before they disappear	Yes	Yes
Distribution method	Unicast-based: Client connects to the registry and executes a query.	Multicast-based: Client queries the network, and gets responses back.	Multicast-based: Periodic dissemination of available services. Client queries local cache.
Gives the location of services?	Perhaps, could be supported through user added metadata	No	Yes
Suitable in MANETs?	No	Yes	Yes

Table 1 Capabilities of Web services service discovery standards compared to our experimental SAM.

Implementation of SAM

Our experimental service distribution mechanism is tailored to the specific requirements for service discovery in MANETs. It can function disconnected from the Internet since it hosts all the necessary data and metadata in a local repository. It is resilient to partial failure since it is a fully decentralized solution. Thus, it will still function if the network becomes partitioned – the clients in each partition will have a fully operational service discovery mechanism at hand. The mechanism is based on periodic dissemination of service advertisements, and the advertisements are cached locally in each recipient. Whenever a new update is received the cache is refreshed, and any outdated services are removed. There is a limited time each service can exist in the cache before it is deleted, thus ensuring that a query in the local cache will give an up-to-date picture of the available services.

An advertisement contains a list of services being provided by the node that sent the advertisement. This list contains entries uniquely identifying the service, and any metadata associated with the service. If the node chooses to report its position (e.g. if it is fitted with a GPS), then the positional data (e.g. latitude and longitude – the field uses the PositionDataType (PDT) from NFFI version 1.3 (STANAG 5527)) is sent along with the list of services.

Basically, an advertisement contains the following data:

- Position (optional)
- Service list
    - Unique service ID (required), endpoint URL (required), metadata (optional)
    - (Possibly more service entries...)

The *endpoint URL* is the invocation address of the service being identified by the Unique service ID at that particular node. In other words, this is the endpoint address which can be found in the service definition of the WSDL. This URL is the only dynamic aspect of a Web service description, as the rest of the WSDL can remain static from deployment to deployment. Based on this observation, we found that by removing the endpoint URL from the WSDL, we would get a WSDL which could uniquely describe all services of the type defined therein. A WSDL is a large XML document, so to reduce the bandwidth requirements of the advertisement distribution mechanism we chose to use a hash over the WSDL without endpoint to uniquely identify a service. Metadata associated with the service is optional, but for the purposes of semantic service discovery, this field must be used. If it is left empty, then no semantic reasoning over QoS can be done by the client (see Figure 6 for an example with only the mandatory fields of the advertisement being used). The metadata field is thus a flexible measure provided by the solution, in that it can be used by regular Web services for discovery (i.e. just ignore the field) or by Semantic Web services for added value and reasoning capabilities (see Figure 7 for an example of all fields of the advertisement in use).

```
<?xml version="1.0" encoding="UTF-8"?>
<ns0:ServiceDiscovery  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   xmlns:ns0='urn:no:ffi:servicead'
   xsi:schemaLocation='urn:no:ffi:servicead FServAd.xsd'>
    <ns0:service address="http://my.host.org/service1"
hash="F82F0CFAD205B9A88907D24DF1ADFF94BF5BD70D"> </ns0:service>
    <ns0:service address="http://my.host.org/service2"
hash="47FC70C359DED9E80BE5A476C9E2DEEF0EB12638"> </ns0:service>
</ns0:ServiceDiscovery>
```

Figure 6 An example advertisement using only the mandatory fields.

The idea is to pre-distribute service and domain ontologies to the clients with a set of service instance templates. A service instance template is a definition of a specific kind of service, but without the values of the dynamic QoS parameters. The grounding points to a WSDL without a specified endpoint.

In addition to the slots for WSDL hash and endpoint, the advertisements also have the metadata slot which we use for a semantic description of the service and a list of dynamic QoS parameters defined in a list with names and values (e.g. the metadata presented in Section 4.2).

See [15] for further details about SAM.

```
<?xml version="1.0" encoding="UTF-8"?>
<ns0:ServiceDiscovery  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
   xmlns:ns1='urn:nato:fft:protocols:nffi13'
   xmlns:ns0='urn:no:ffi:servicead'
   xsi:schemaLocation='urn:no:ffi:servicead FServAd.xsd'>
    <ns0:position >
        <ns1:dateTime>20090327111334</ns1:dateTime>
        <ns1:coordinates>
            <ns1:latitude >35.5310533333333</ns1:latitude>
            <ns1:longitude >67.4055644444445</ns1:longitude>
        </ns1:coordinates>
    </ns0:position>
    <ns0:service address="http://my.host.org/service1"
hash="F82F0CFAD205B9A88907D24DF1ADFF94BF5BD70D"
metadata=”myMetadata”></ns0:service>
    <ns0:service address=" http://my.host.org/service2"
hash="47FC70C359DED9E80BE5A476C9E2DEEF0EB12638"
metadata=”myMetadata2”></ns0:service>
</ns0:ServiceDiscovery>
```

Figure 7 An example advertisement with position and metadata for each service.

Service selection

Clients receiving these announcements handle the advertisements as previously described, ready to populate service instances during selection. If a new announcement arrives from the same provider about the same service the old announcement is overwritten.

Service selection is the process where the client discovers, orchestrates and invokes services. Non-semantic Web Service technology has limitations in its static nature. WSDLs define where the service is located and the data type of the messages. BPEL orchestrations are defined before runtime. Adding semantics to Web services could enable dynamicity at runtime, where services are found based on their ability and orchestrations are done on-the-fly to satisfy client requests.

The pre-distributed ontologies are used by the client to formulate a service request. Requests are matched against service profiles and by this the request has the same format as the service templates. The client defines the desired service based on conceptual descriptions of input, output and QoS parameters. Requests are matched and the service advertised to the client.

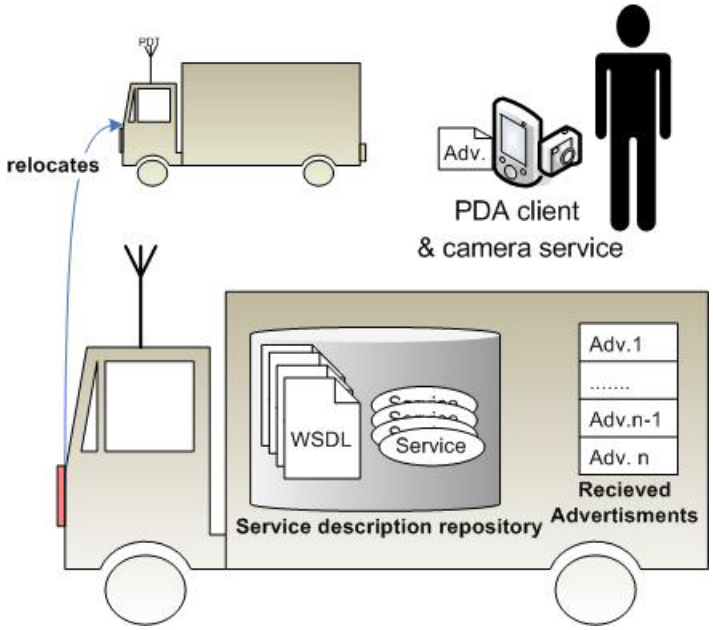

Figure 8 Mobile service provider and consumer

Figure 8 shows how a vehicle can have a service repository and a list of received advertisements ready to be searched if services are needed. To define the difference of equipment in a MANET, Figure 8 also shows how a soldier with a PDA and a camera can be a client as well as a provider.

Figure 9 shows a camera ontology. The model shows that *Camera* subsumes *WebCamera.* This means that the *WebCamera* concept has inherited all its properties from the *Camera* concept. In addition, the *WebCamera* also has its own specific properties. This means that a *WebCamera* is in fact a camera, and a *Camera* could be a web camera.

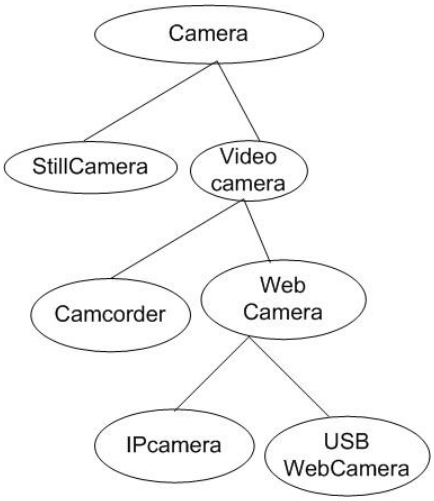

Figure 9 Camera ontology

In our solution we define service matching as a stepwise process. The idea is to divide the matching into two general steps. First we match based on functional aspects of the service through input and output matching, then we match based on QoS parameters.

In our solution the second step is to match on the QoS parameters. QoS matching is based on the same principles as input and output matching. Based on the result of both matching processes there could be the need for orchestration of two or more services. For example, a service delivers observation data in raw image format. The service takes position as input and that matches the client's requested input parameter, but the output of the service does not match as the client application uses JPEG format. A conversion service available takes raw image data as input and produces JPEG data as output. By orchestration of these services the client gets the requested service. An example of reasoning is if there are other available services that would satisfy the client, but this is not obvious without reasoning. Reasoning on a conceptual level could lead the client to an alternative service. If the client wants surveillance footage from a specific location it could be that there are no available services delivering video from that location, but there are several soldiers near the location with cameras, based on the fact that pictures also could be surveillance footage this service is proposed.

The next step is to find the service advertisements with the same profile hash value from the list of service advertisements received. Instances of the template are populated with QoS parameter values, and a matching of client request QoS parameters and service profile QoS parameters is performed. If there is a service with satisfying QoS the service can be invoked. If there are no QoS matches a new service discovery is launched in order to try to find a service that can be used in an orchestration to deliver a result to the client.

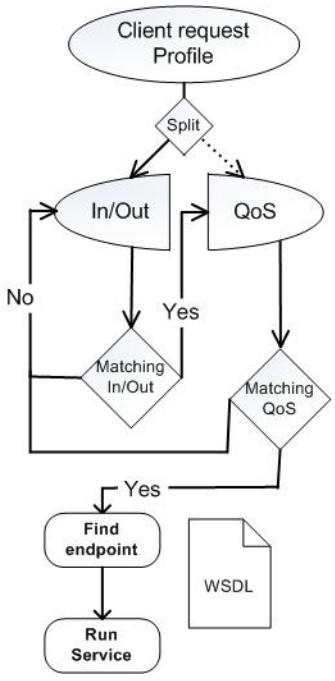

Figure 10 Conceptual service matching process

Figure 10 shows the idea of a conceptual matching process as described earlier. The different services available are instantiated and we are to ready search for services. A client's request is split in

two halves. One part has the input and output parameters and the other has the QoS parameters. Before matching on QoS we search using the functional input and output parameters. If we get a match, then we match the QoS of the service to the user's QoS needs. If there is no match, an attempt to orchestrate is initiated. Orchestration uses the client request input as input to the orchestration matching, and if there are any matches the orchestration process tries to match services with the first orchestration service output as input and the clients service request output as output. If this results in a match the QoS has to be. Let's take a few steps back and assume that the first match was OK, and that a QoS matching now is initiated. If this matching does not return any satisfying results, we need to start again and try to orchestrate services based on QoS parameters. For example, a client requests a service with location as input and picture as output with the QoS resolution to the value pair (X, Y) = (320, 240). This corresponds to QVGA, which is a common resolution for handheld devices such as PDAs. In our template we find a service match, but in the QoS matching we find that the service has a QoS resolution set to (1024, 768). This corresponds to XGA resolution, which is common in larger devices such as laptops. This means that a new service search is started, but this time for a service that has picture as input, picture as output and QoS resolution parameter set to (320, 240). By performing an orchestration with the originally found picture service and this rescaling service, an execution chain consisting of the picture service and the rescaling service would yield the desired output. It would match both service type and the QoS requirements. By orchestration of these services we now have a solution for the client request and the services can be invoked.

To invoke the services we need the physical address of the service. The semantic service descriptions have a grounding instance. When the individual service instances are populated the WSDL in the advertisement are also populated with the appropriate endpoint and the grounding is set to point to this WSDL.

Implementation

As a first step towards QoS aware Semantic Web Services in MANETs we have implemented a demonstrator based on the OWL-S API [16] that selects, orchestrates and invokes services based on standard Semantic Web Services input and output.

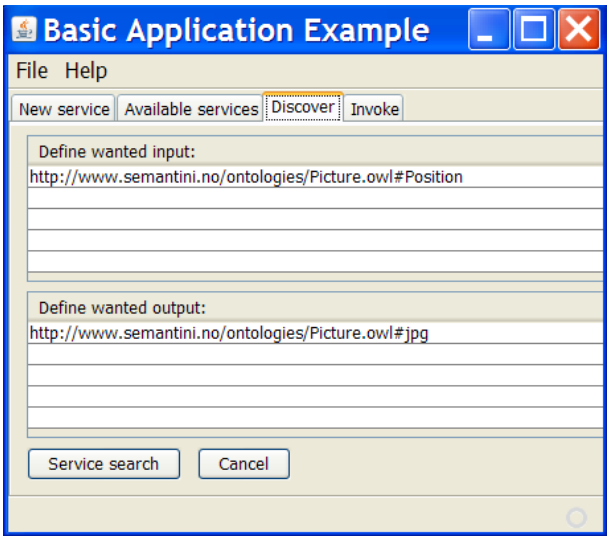

Figure 11 Service selection

Figure 11 displays the demonstrator system selection pane. For searching and selecting services a request is defined with a definition of input and output. The user requests surveillance footage of a given area and the wanted service is by this described to take position as input and JPG picture format as output.

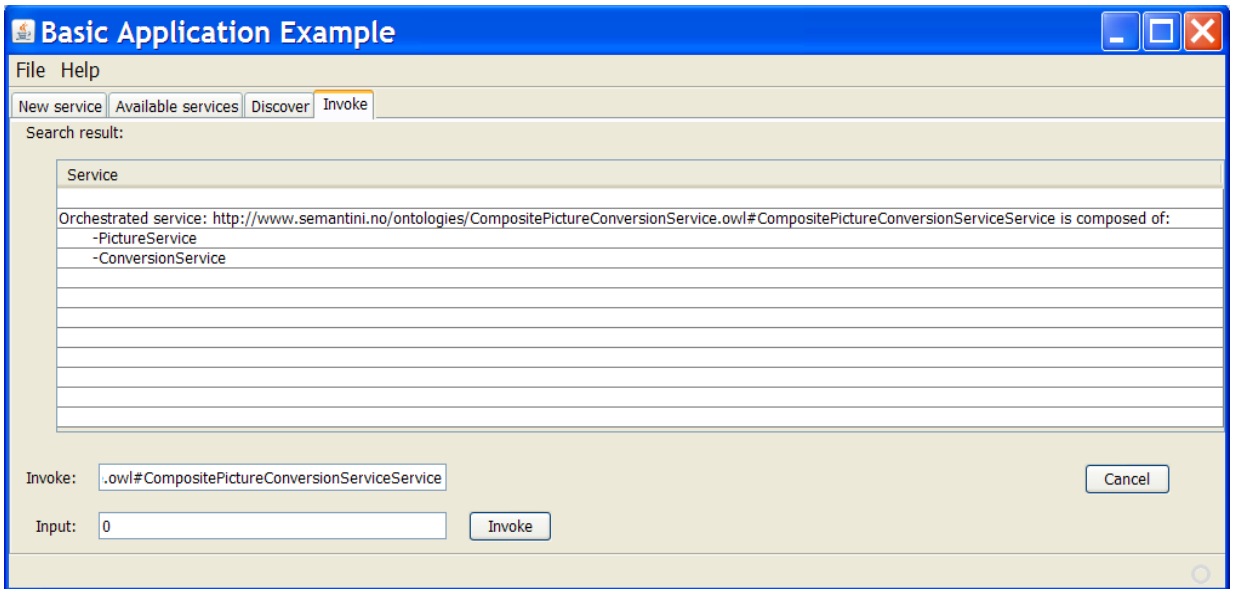

Figure 12 Service search result

Figure 12 shows the services found during search for services. Performing the search the system does not find a single service that match the user's requirement. Performing a search for an alternative the system finds that an orchestration of the Picture service and the Conversion service would fulfill the user's requirements. The picture service takes a position as input and delivers GIF formatted pictures and, as the conversion service accepts GIF as input and returns a converted picture in the JPG format, a combination of the two would serve the user. A new service description is defined automatically and is ready for invocation.

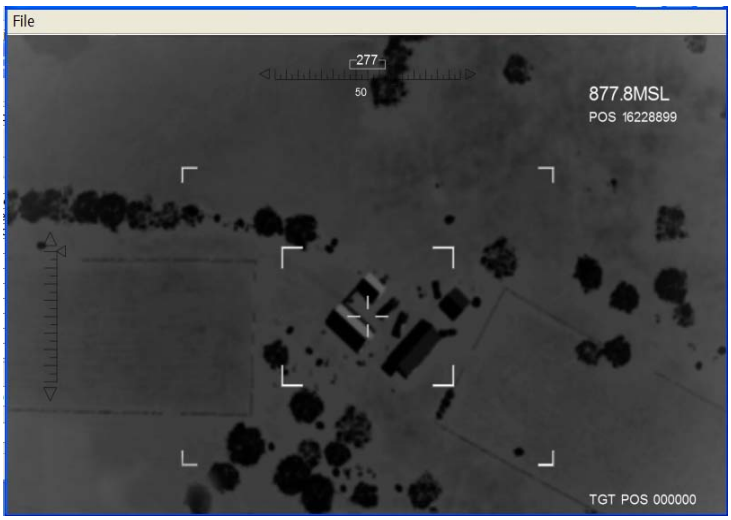

Figure 13 Result picture

Figure 13 shows the end result of the service invocation. After invoking the new service description the user does not participate in the steps of invoking the two services as the system uses the new description to do this automatically. The orchestrated service returns to the user the wanted picture in the correct format.

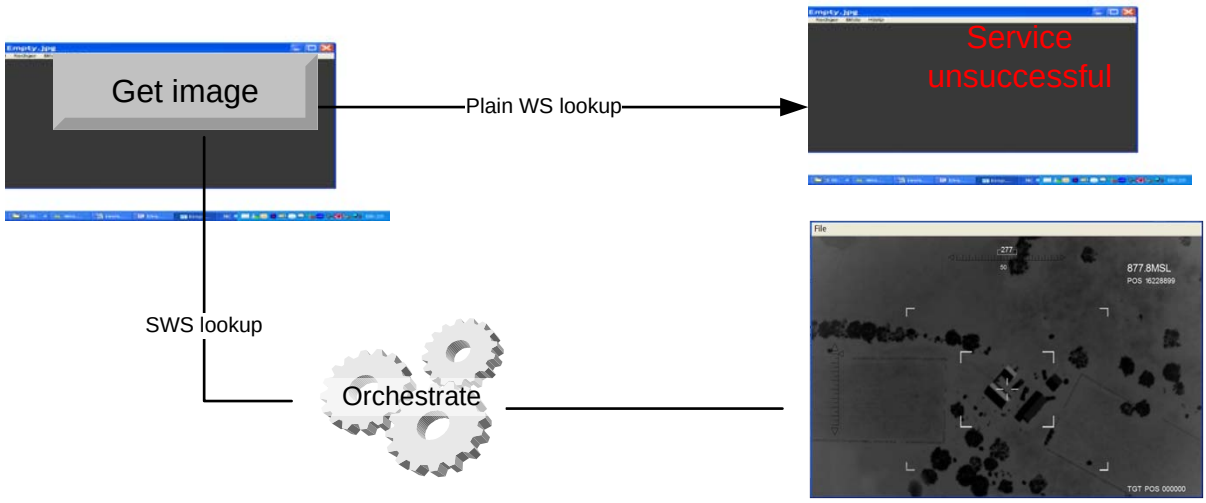

Figure 14 Service execution

Figure 14 illustrates the improvement of using Semantic Web services in contrast to just Web services technology for applications depending on available services. The user interfaces a simple example application, getting surveillance footage by simply pressing the "Get image" button. If plain Web services are used in the background the service lookup would be unsuccessful, as there are no services able to respond successfully to the request. If, on the other hand, Semantic Web services are working in the background we get the runtime semantically orchestrated "Composite Picture Conversion Service" (see Figure 12) which returns a picture with correct formatting shown to the user. Using Semantic Web services in contrast to plain Web services would in such cases have a binary outcome, meaning that plain Web services do not complete the task while Semantic Web services complete the task using runtime orchestration. The user of the surveillance footage application is none the wiser of the complex actions carried out behind the scenes, happy to get the surveillance footage requested. For further details regarding the expressive power of Semantic Web services vs Web services, and the benefit of using semantic technologies in military networks, see our paper [1].

Conclusion and future work

In this paper, we have presented our experimental prototype which brings QoS awareness to Semantic Web services discovery and orchestration. Our contribution is twofold: First, we have implemented a novel mechanism, SAM, for distributing service advertisements in dynamic networks. Second, we have defined an OWL-S based service ontology which we call OWL-S LiQ. By using SAM to provide an up-to-date view of available services and corresponding QoS parameters in a MANET,

we were able to employ OWL-S LiQ to perform selection and orchestration of the available Web services based on both service types and specific QoS requirements. The system is capable of having both high end clients such as laptops and low end clients such as PDAs present, and accommodates their specific needs based on the service description, service QoS parameters, and the QoS profile of the specific device. Such a system is valuable in a military operation, where devices with different capabilities take part in the operation and need to access various services and tailor them to their available resources. Semantic Web services can provide the necessary agility and interoperability to heterogeneous systems that need to be interconnected in joint and/or combined operations.

Future work consists of expanding the QoS support of our OWL-S LiQ ontology, so that it can describe a wider array of QoS parameters. Also, the device profile and matching capabilities need to be expanded from the simplistic high/low end client profile of today to a more flexible and configurable type that can support any kind of mobile device.

In addition, our current service matching is done based on a simple fail/success criterion. Due to the hierarchical nature of ontologies, several different *degrees of matching* [14] can be defined, and we will extend our service matching algorithm with support for this.

References

[1] F. T. Johnsen, M. Rustad, T. Hafsøe, A. Eggen, and T. Gagnes, *"Semantic Service Discovery for Interoperability in Tactical Military Networks"*, The International C2 Journal, Volume 4, Number 1, 2010, Special issue: Agility and Interoperability for 21st Century Command and Control. http://www.dodccrp.org/files/IC2J_v4n1_02_Johnsen.pdf

[2] A. Vogel, B. Kerherve, G. von Bockmann, and J. Gecsei, *"Distributed Multimedia and QoS: A Survey"*, IEEE Multimedia, Volume 2, No 2, pp 10-19, 1995

[3] WS-QDLv1.0, OASIS Web service Quality Model TC ,http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsqm

[4] M. Tian, A. Gramm, H. Ritter, and J. Schiller, "*Efficient Selection and Monitoring of QoS-aware Web services with the WS-QoS Framework*", IEEE International Conference on Web Intelligence, Beijing, China, September 2004

[5] K. Kritikos, and D.Plexouakis, *"OWL-Q for Semantic QoS-based Web Service Description and Discovery"*, First International Joint Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web, Busan, South Korea, November 2007

[6] OASIS, UDDI v3.0, standard, http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3

[7] OASIS, ebXML v.3.0.1 official OASIS sponsored site, http://ebxml.xml.org/

[8] OASIS, OASIS Web Services Dynamic Discovery (WS-Discovery), OASIS standard, http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01

[9] Tommy Gagnes, *"Assessing Dynamic Service Discovery in the Network Centric Battlefield"*, IEEE *Military Communications Conference (MILCOM 2007)*, vol., no., pp.1-7, 29-31 Oct. 2007

[10] W3C, OWL-S: Semantic Markup for Web Services, http://www.w3.org/Submission/OWL-S

[11] W3C, SWRL: A Semantic Web Rule Language Combining OWL and RuleML, http://www.w3.org/Submission/SWRL

[12] W3C, Web Service Modeling Ontlogy (WSMO), http://www.w3.org/Submission/WSMO

[13] W3C, Semantic Annotations for WSDL Working Group, http://www.w3.org/2002/ws/sawsdl

[14] OWL-S/UDDI Matchmaker Details, http://www.daml.ri.cmu.edu/matchmaker

[15] Frank T. Johnsen, *"An NFFI-based Web service discovery mechanism for tactical networks"*, Military Communications and Information Systems Conference (MCC 2009), Prague, Czech Republic, September 2009.

[16] OWL-S API, Maryland Information and Network Dynamics Lab Semantic Web Agents Project, http://www.mindswap.org/2004/owl-s/api/