13^{TH} ICCRTS: C2 for Complex Endeavors
"Extending Service Oriented Architectures to Edge Networks with Active Metadata and Swarming"
Topics: Collaborative Technologies for Network-Centric Operations, Networks and Networking, Modeling and Simulation

Christopher McCubbin, R. Scott Cost, Markus Dale, Paul Worley, Daniel Bankman [STUDENT]
POC: Christopher McCubbin
Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
443-778-8663
christopher.mccubbin@jhuapl.edu

March 17, 2008

Abstract

Much work has been done on perfecting Service-Oriented Architectures (SOAs) in the case of connected networks and networks which have full-time access to centralized service directories. Networks which have sparse connectivity, mobile nodes, and limited bandwidth cannot use this model for a SOA. Many of today and tomorrow's tactical networks will have these limitations. We have designed and built a SOA which combines the advantages of swarming technology and active metadata, extending SOA and service invocation capability to these tactical edge networks. Autonomous mobile nodes within our swarming architecture also have the capability to reconfigure the edge network topology to optimize service response time, while at the same time completing complementary tasks such as area search.

Our architecture is modeled on the successful Representational State Transfer (REST) architecture for Web Services. Metadata and service invocation are therefore very lightweight and are capable of easily being transported through the limited-capability swarming network. Service responses can either be transmitted via direct link, if one exists, or can be transported via mobile swarming nodes for network delay-tolerance.

We present the results of testing our architecture in a ChemBio scenario during an officially-sponsored autonomy experiment.

1 Introduction

Modern information age warfare places a premium on a force's agility, which Alberts and Hayes [1] define as the ability to develop effective situational awareness and effective decisions *rapidly*. Agile forces will require command and control infrastructures which are themselves agile. Information age command and control infrastructures must rapidly integrate information resources. These resources may exist in relatively static or highly dynamic environments. Transient systems composed of loosely coupled federations of relevant resources provide more flexibility than rigidly organized, tightly coupled persistent systems. This is especially true if the resources themselves may be entering and exiting the environment.

By distributing information about services and service activations throughout a *swarm network*, we gain the capability to generate a SOA even when the underlying network is highly dynamic as will be the case in future agile combat systems.

In this paper, we describe our technical approach to the implementation of a swarm-based metadata distribution and service invocation along with the results of a hardware-in-the-loop validating experiment.

2 Motivation

The issue of creating a robust SOA in an agile environment poses several difficulties. Because information resources available may not be known during design or even deployment of a future agile network, the agile information integration will require run time rather than static integration. Manning costs and delays associated with operator driven integration makes using human operators at best undesirable and probably infeasible for agile command and control infrastructures. We therefore conclude that information resources must be capable of autonomously integrating during the course of an operation. In order to effectively self-integrate resources must be able to recognize and understand

peer resources whose identity was not known at deployment. To support this each resource must be able to determine the range of specific relevant and available resources. That is, resources must recognize the set of resources that can effectively be used, and that can provide the most relevant information or valuable service. One approach is a directory based approach that supports the registration of resource information with a centralized (or distributed) directory service. This directory service can in turn be queried by resources requiring a match. While this provides an effective means of coordination in some situations, there are drawbacks. In a highly dynamic environment, information in the directory may not be updated frequently enough to reflect reality. Also, there is significant overhead associated with maintaining a common directory, especially as the scope of the region or the number of entities increases. Also, this approach assumes that all entities have persistent access to a directory service. In a highly dynamic or hostile environment, this may be too strong an assumption to make. Our emphasis in this work is on environments in which there is a benefit to exploiting significant amounts of autonomy on the part of the framework elements.

An alternative approach assumes that resources share information in a distributed manner. In contrast to the pull-based directory approach, service providers have a responsibility to make themselves known directly to the device network. This puts more of a burden on the service provider, but has the advantage that resource information is distributed in advance, and therefore may be available even if access to resources or directory servers is not.

Our approach, a modified version of a swarming model using push technology, puts the burden on the swarm network by allowing the swarm communications to maintain and distribute metadata. Service activations are also carried out in a similar manner. This provides for a scalable, distributed approach to the advertisement and activation of resource information with a domain that is also tolerant of network delays and failure.

3 Background

In this section we describe some of the technologies that are important for understanding the design of SWARM-AMF.

3.1 Service-Oriented Architecture

Service-oriented architecture (SOA) is an architectural style of building distributed applications by distilling required functionality into reusable services. This allows for maximum reuse of existing infrastructure while accommodating future change. An application can be flexibly composed of services. Services can be reused between applications. Loose coupling, implementation neutrality and flexible configurability are some of the key elements of SOA [10]. Current implementations of SOAs are typically Web Services based using the HTTP communications protocol, SOAP [11] for exchanging messages, the Web Services Description Language (WSDL) [3] for describing web services, and Universal Description, Discovery and Integration (UDDI) for registering services.

3.2 Swarming and Dynamic Co-Fields

We use an internally developed system called Dynamic Co-Fields (DCFs) to generate strong autonomy between our devices. This approach is a form of potential field theory that is extended beyond previous unmanned vehicle potential field applications by incorporating elements of recent advances in swarm behavior optimization. These modifications

to swarm behavior solve the traditional problems with potential field approaches, generating robust, effective behaviors in diverse, complex environments. The central theme of this approach is the use of stigmergy to achieve effects-based control of cooperating unmanned vehicles. Stigmergy is defined as cooperative problem solving by heterarchically organized vehicles that coordinate indirectly by altering the environment and reacting to the environment as they pass through it. We accomplish stigmergy through the use of locally executed control policies based upon potential field formulae. These field formulae are used to coordinate movement, transient acts, and task allocation between cooperating vehicles.

With DCF, a virtual potential field is associated with the all germane entities contained within a locally held model of the vehicle's world. These models typically include peer vehicles and elements that are impacted by peer vehicles. These fields are used to influence vehicle action, most notably movement. In these systems a vehicle is considered a point particle with a fixed position in space at a fixed time. Vehicle movement is determined by the locations of influencing entities that are likewise fixed in Euclidean space at a fixed time. The force associated with a specific influence may be attractive, driving the vehicle towards the influence when greater than zero or the force may be repulsive, driving the vehicle away from the influence when less than zero. Details of the specific equations used to generate interesting behavior using this approach can be found in the literature [4].

The devices in our network also communicate information, known as beliefs, to each other through an ad-hoc networking scheme. A networking layer known as the Knowledge Layer performs two functions related to belief communication. First, the Knowledge Layer accepts beliefs and merges them into the Knowledge Base. Beliefs may be accepted from organic sensors or from transmissions received through the network. In merging the beliefs the Knowledge Layer must deconflict contradictory beliefs and reduce the complexity of the knowledge base (as necessary) either through compression, removal of less important knowledge, or through aggregation. Aggregation is accomplished through the representation of multiple objects as a weighted centroid. This reduction in the complexity of the stored and transmitted knowledge proportionally decreases bandwidth used by the vehicle group.

The ad-hoc propagation network required for this system is an addressless communication mechanism by which vehicles share knowledge using periodic omnidirectional broadcasts. Transmission occurs without knowledge of the recipient or the information's utility. Received information is used to update a vehicle's world model without regard for the information's source. By receiving and transmitting in the blind, knowledge propagates asynchronously across the swarm without the need for a continuous, high quality of service, routed network. Numerous off the shelf link-layer networks are able to accomplish this type of networking effectively.

Both the Dynamic Co-Fields and Propagation Networks have undergone substantial testing in simulation and in actual hardware. A number of specific behaviors have been developed and evaluated in simulation-based experiments. DCF has been employed to provide highly intelligent high-level command and control capabilities on board ONR's Unmanned Sea Surface Vehicle prototype. In these experiments the DCF algorithms are used to generate operational coordination between Unmanned Sea Surface Vehicles (USSVs) and other cooperating vehicles.

3.3 Active Metadata Framework (AMF) version 1.0

The previous AMF framework consisted of a set of nodes running on platforms. The collection of these nodes formed a virtual SOA. Services and consumers connected to this SOA by connecting to their local AMF Node and performing advertisements or queries, respectively.

In order to advertise a service to the AMF, a service first formed a metadata object representing itself. The service then transmitted this metadata object to the local AMF node. To represent the service, the node created a number of autonomous agents known as Active Metadata Agents (AMAs). AMAs also represented an a-priori service need created by a consumer. The AMAs took the place of the directory service by representing the searchable metadata of the service that created it. The intention was that AMAs be pushed to other nodes in an intelligent way, anticipating where queries will be made that require the service [8, 7, 6].

Creating the algorithms to route the AMAs effectively in this manner is quite difficult. Often, AMAs were not arriving at the places that they were needed. Also, deciding when and if to duplicate an AMA proved a very challenging problem. If too many AMAs were created, they would flood the network. If too few were created, there would not be enough to meet demand.

Shifting the paradigm from an agent-based approach to a swarm-based approach solved these problems at the price of replicating a single copy of AMA data at each node, now encoded as lightweight eXtensible HyperText Markup Language (XHTML) instead of a heavyweight agent. We believe this new approach to be a viable one however.

3.4 Tactical Networks

The future of tactical networks is varied and complex. According to the DARPA Strategic Technology Office, tactical network centric operations "must be reliable, available, survivable and capable of distributing large amounts of data quickly and precisely across a wide area. Additionally, these networks must be able to simultaneously and seamlessly meet the needs of both manned and unmanned systems throughout the strategic and tactical battlespace." [9]. Certainly these networks will contain mobile ad-hoc networks, reconfigurable networks, and other networks that are outside the realm of standard wired high-bandwidth networking.

Implementing SOAs on mobile ad-hoc networks presents a unique set of problems. Since the network is highly dynamic, even notions such as point-to-point messaging become an issue. Various routing protocols have been developed [5, 12], but most prominent routing protocols only search for routes that exist at the current time and are not delay-tolerant. Some research has been done in delay-tolerant routing. Depending on the ad-hoc operational concept, a centralized directory of services, popular in pull-style SOA networks, may not be a feasible option. Several alternatives exist, such as on-demand advertising and directory replication.

The swarming network system and belief network that our system uses requires only the most rudimentary capabilities from the underlying tactical network. These requirements include the ability to broadcast to local nodes and a connectionless link layer capable of transporting User Datagram Protocol (UDP).

3.5 Microformats

The purpose of microformats [2] is to provide machine-readable semantics in a web page along with human-readable content. Microformats are designed for humans first and machines second: they exist as additions to the page markup and do not change or modify the display of information. On the internet, microformats serve as standards for semantic markup so that different services and software agents can aggregate and interpret microformatted content. Without microformats, complex artificial intelligence is necessary to gather semantics from plain text. Several microformats are already widely used, such as hCard and hCalendar, and more are being developed every day. hCard is designed for

marking up contact information (for people, places, and things) and organizes data such as name, location, address, phone number, etc. hCalendar is used for marking up events and dates, and is currently used by the new online planning system Eventful.

Microformats are not a new language for semantic markup like Resource Description Framework (RDF); they simply add semantics to regular HyperText Markup Language (HTML) code. In most cases, microformats are coded as spans and divs around plain text with standardized names. For example, the contact data for a fictional company called TechCompany could be coded as:

```
<div class="vcard">
  <a class="fn org url" href="http://www.techcompany.net/">TechCompany</a>
  <div class="adr">
    <span class="type">Work</span>:
    <div class="street-address">100 Business Street</div>
    <span class="locality">Columbia</span>,
    <abbr class="region" title="Maryland">MD</abbr>
    <span class="postal-code">21044</span>
    <div class="country-name">USA</div>
  </div>
  <div class="tel">
   <span class="type">Work</span> +1-410-812-4432
  </div>
  <div class="tel">
    <span class="type">Fax</span> +1-410-812-4433
  </div>
  <div>Email:
   <span class="email">info@techcompany.net</span>
  </div>
</div>
```

with all of the hCard microformat keywords bolded. The hCard microformat simply slips around the HTML markup in the form of class names, hidden from human readers but clearly visible to machine interpreters. By microformatting this page content, it is directly parsable by web services.

In the AMF application, microformats serve as embedded semantics in metadata pages that make the services on the tactical network machine accessible. More information about the specific microformat designed for AMF is located in section 4.3. This microformat is designed to provide all of the necessary information that a machine agent needs to construct the same HyperText Transport Protocol (HTTP) request that the metadata pages use to communicate with the services.

4 System Design

4.1 Design Goals

The goals of the SWARM-AMF system are as follows:

1. To encode metadata in such a way that service invocation is simple for humans and machines, and efficiently represented.
2. To distribute metadata about services to potentially interested clients who may be mobile, often over-the-horizon, and often disconnected from the network.
3. To allow clients to invoke a service that they discover even if that service is currently unreachable via the tactical network.

4.2 Overall System Concept

The current design concept can be seen in Figure 1. Our concept is based around notions of the REST architectural principles. All service metadata is encoded as XHTML. Details of the encoding are found below in section 4.3, but the idea is that human clients should be able to invoke services just by loading the appropriate web page and submitting a form found on that web page. Conversely, once a service finds out that it has been invoked (through whatever means), it will generate an XHTML response for the client to interpret, usually by displaying on the same web page that invoked the service.

Now, the question remains, how do clients discover the metadata, and how does it invoke the service? If we were in a fully-connected "strategic" environment this would be a simple matter, and is done all the time using the internet paradigm. A client wishing to find a service uses some kind of search engine to find the metadata, also known as the webpage, of an available service. The client then loads the webpage, which has details on how to invoke the service. For example, the Amazon webpage has a field for searching its book database. The client fills in the appropriate query and executes it, using a HTTP POST operation to send data to the (connected) service. The service then generates an HTML reply and responds to the query, either with a new webpage or by updating the current page if it is using something like Asynchronous JavaScript and XML (AJAX).

This way of doing things breaks down in several places in our target environment. Most importantly, the network is not assumed to be fully connected. This fact alone implies many difficulties. There can be no centralized service search engine since we are not guaranteed to be connected to such a device often, or indeed, ever. Broadcast search, as is done in the case of wireless routing, is equally ineffective. A further difficulty with this environment is that services cannot be assumed to be reached directly. Therefore the service itself cannot host the server that serves metadata web pages nor can the service be invoked directly using HTTP POST.

The best we can hope for is that using mobile nodes, there will eventually be a delay-fraught path from a service to a client and back that must include "bucket brigading" of data through the mobile nodes. Therefore we must use a delay-tolerant networking style if we wish to get data to the right places in this network. We have made a significant paradigm shift from the AMF v1.0 system as described in Section 3.3. The v1.0 system used a federation of collaborating virtual

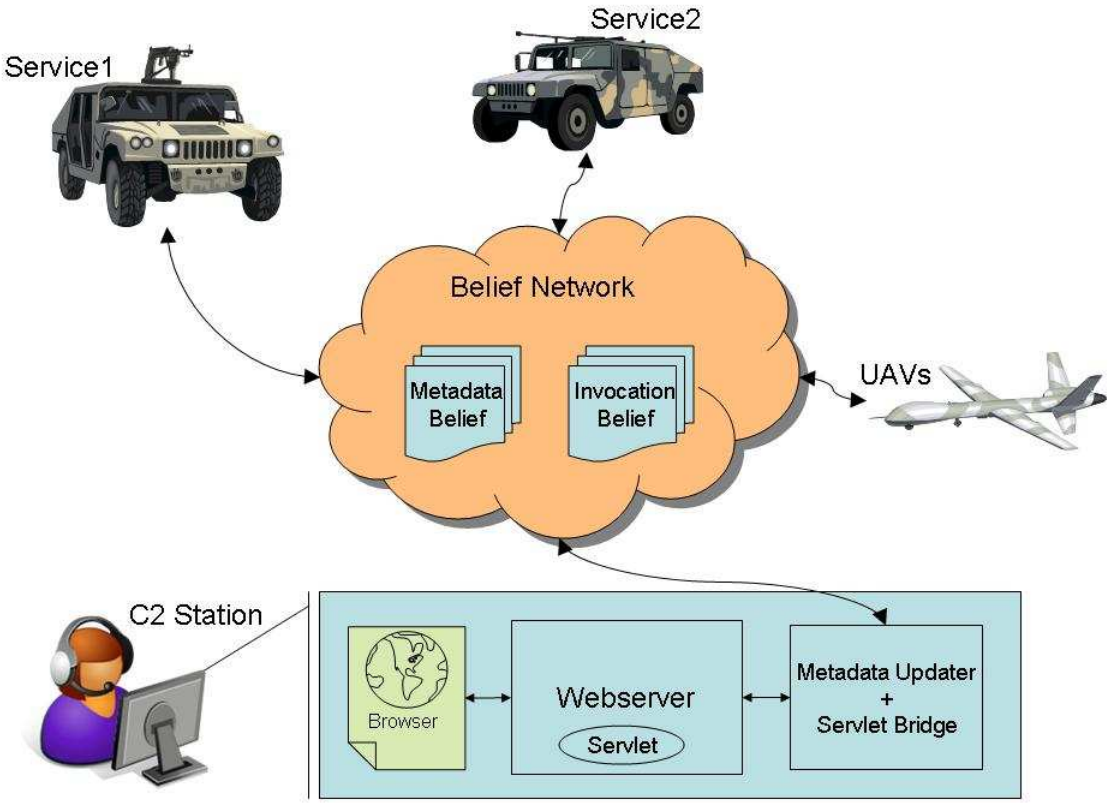

Figure 1: SWARM-AMF Block Diagram System Concept

agents to accomplish the task of metadata distribution. The current system, referred to as SWARM-AMF, leverages the delay-tolerant technology of the APL SWARM and DCF systems instead.

As described in section 3.2, the SWARM system contains a virtual shared blackboard called the *belief network*. This device allows for delay tolerant communication of small pieces of data around the swarm. For details of its operation see for example [4].

Using the belief network, we can create a delay-tolerant version of the internet-style service architecture discussed previously, with some modifications. We use the belief network itself as the ultimate repository of all relevant metadata. When a service comes online, it creates a metadata webpage and deposits it in the belief network.

We also move the webserver that handles human client requests to the client (or a machine always network-connected to the client). On this client-side webserver we have a daemon running that periodically looks at the belief network and receives metadata present there. This daemon also creates an index of all current metadata on the client-side webserver. Details of this process and how a client invokes a service can be found in Section 4.5.1.

Since the metadata also contains embedded machine-readable data, any machine connected to the swarm belief network may also read the metadata and programmatically invoke a service. The details of this transaction can be found in 4.5.2

4.3 Metadata Encoding

The active metadata web pages are designed to describe the services on a tactical network such that they are invocable by human and machine users. For human users, a metadata page may include an HTML form whose inputs will be used to construct a query. When the form is completed, this query will be sent asynchronously via an AJAX request to a Java servlet, and then to the swarm. Because AJAX is used to manage the output of queries, the user's work on the metadata page is not interrupted by a refresh when the query form is submitted.

In the case of the Chem/Bio Data Access Service whose metadata is shown in Appendix B, the user must enter the North West and South East coordinates for the rectangle in which he wants to find chem/bio alarms. The user must also enter start and end times between which to search. When the form is submitted, the JavaScript sendRequest() function takes the information from the form fields and constructs a query string to be sent as post data on the AJAX request. The request is then sent to a Java servlet that manages its dispatch to the swarm. When a response arrives from the Java servlet, the JavaScript callBack() function is executed and the HTML page is udpated with the result.

For machine users, a microformat called hInvocation is located in the head of the metadata page, describing the structure of the AJAX requests so that software agents can construct them. hInvocation was designed explicitly for AMF and provides only the necessary information for requesting data from the services that the metadata pages describe. The hInvocation microformat is structured as follows:

```
hInvocation
    service
        type
        location
    parameter
        type
        name
    parameter
        type
        name
    ...
    returns
        type
        column
            type
            name
        column
            type
            name
        ...
```

All of the hInvocation data must be enclosed by a div with the class “hInvocation”. The first piece of required information is the “service” element, containing two elements called “type” and “location”. The “service” element identifies the service to which requests on the metadata page are sent. Following “service”, each parameter (corresponding to the form fields) must be listed in a “parameter” element, with the two subelements “type” and “name”. “type” can be any of the Java primitive data types, such as double, float, int, etc. The “name” element contains the name the parameter is given in the query string. Finally, hInvocation requires a “returns” element containing a “type” subelement, which is usually “table”. Because the service returns a table, multiple “column” elements are required containing the “type” and “name” for each piece of returned data.

Using the hInvocation microformat, an agent can construct a query to send to the swarm in the same format that the metadata pages would send as an HTTP request. With both AJAX components and microformats, the metadata web pages provide a functional interface for both humans and machines to the services on a tactical network.

4.4 System Component Design

The general pieces of the SWARM-AMF system are detailed in this section. How these components are used in the system is detailed in Section 4.5.

4.4.1 Belief Network

Existing as a virtual blackboard, the belief network is in reality present on all machines connected to the swarm. Visible to each swarm node, the belief network contains the most recent beliefs placed in the network. Some nodes may be out of date if they or their neighbors have not had much contact with the rest of the swarm recently. Details of how beliefs propagate are beyond the scope of this paper but may be found in the literature [4]. For our purposes the belief network contains several beliefs relevant to our interests.

Metadata Belief The Metadata Belief contains the Metadata generated by services. When a service comes on-line, it generates Metadata in the form described in section 4.3 and creates a belief corresponding to this Metadata in the belief network. There is one belief for each service. If the service wishes to update its metadata, it simply overwrites the old version of the belief in the network.

Invocation Belief An invocation belief is used to invoke a service, and to return the response to the client that invoked the service. Each invocation belief has fields for the service name, invocation parameters, the response text in HTML, and flags indicating whether or not the invocation has been serviced and whether it has been seen by the original invoker.

Position Belief The Position Belief holds the position of every mobile node in the swarm. Each node will update its own position in the Belief Network at regular intervals.

4.4.2 Service Components

Services in our network have just a few components. They must have a belief manager to connect to the belief network. On connecting, and anytime their metadata changes, it is the service's job to update its metadata belief entry on the belief network.

Also, when a service sees an unfulfilled Invocation Belief that belongs to it, it needs to invoke the native service and fill the response into the Invocation Belief.

4.4.3 Human Client Components

The human-usable client has several components for proper functioning.

Local Webserver and Servlet Once the metadata reaches the human-usable client side of the system (via the belief network), it is managed by an Apache Tomcat webserver. A webserver was required to allow the asynchronous request and response functionality that the metadata pages provide. The Tomcat webserver also allows multiple human users to access the metadata pages through their browsers at once. Tomcat also controls the system for processing the asynchronous requests. The cycle begins when a metadata page produces a query based on a user's input on the page. The query is sent asynchronously and addressed to a servlet on the Tomcat webserver, which causes Tomcat to create a new instance of the servlet and begin its execution. The servlet then forwards the query (via TCP/IP socket) directly to a separate class called the Servlet Bridge that runs as part of the main client swarm agent.

Metadata Updater The metadata updater component periodically checks the belief network for new metadata. If it finds any new metadata web pages present, it uses an HTTP PUT command to place the new web pages in a well-known location on the local webserver. The Metadata updater also creates an HTML index page of all of the known metadata and uses another HTTP PUT to install the index on the local webserver.

Servlet Bridge The purpose of the servlet bridge is to manage the client agent's sending of queries to the swarm. Every time a servlet forwards a query to the servlet bridge, the servlet bridge spawns a new thread to send the query to the swarm and deliver its response back to the servlet. The servlet bridge then packages the servlet's query up as an invocation belief and enters it into the client agent's belief manager to be distributed throughout the swarm. When the invocation belief reaches the service that it was addressed to, the service adds a response component to the belief (containing the requested data) and redistributes across the belief network. Finally, the servlet bridge receives the updated invocation belief and the thread that was assigned to that request continues with its execution by sending the response back to the servlet. The servlet then responds to the asynchronous request from the metadata page, and the page updates with new data.

4.4.4 Machine Client Components

The machine-readable client has several components for proper functioning.

Metadata Parser The Metadata parser uses the Java DOM Model to find and parse the hInvocation Microformat present in the service metadata. After parsing, the program can then determine the service type, its name, the number and type of parameters that are required to invoke the service, and the type of data returned by an invocation of the service.

Response Parser The response parser is created as part of the metadata microformat processing step. Given the information about returned values when invoking the specified service, we are able to create an object which will handle the response XHTML produced by invoking the service. Using this parser we may then programmatically analyze the data that is returned by a service invocation.

4.5 Use Cases

There are two important use cases of the system: one where a human discovers and invokes a service and a second where a program discovers and invokes a service. The following sections describe the steps involved in these use cases.

4.5.1 Human Use Case

The human use case proceeds as follows:

- A number of services come on-line, generate metadata, and add the metadata to the belief network.
- Through the belief network propagation, the client machine learns of the service metadata.
- The metadata updater on the client machine deposits the metadata pages on a local webserver and creates an index of the metadata on the local webserver.
- A user uses a web browser to read the index and find an appropriate service. The user then follows the link to get to the service's webpage (residing on the client's webserver).
- The user fills out the request form on the web page and presses the submit button. This causes the AJAX script on the page to begin waiting for a response.
- A servlet on the local webserver processes the request by forwarding it to the servlet bridge.
- The servlet bridge creates an invocation belief representing the request and forwards it to the belief network.
- The service sees that it has an unfulfilled request, processes it, and writes the response into the invocation belief.
- The servlet bridge sees that a response was written, records the response, and marks the invocation belief as read by the client.
- The servlet bridge forwards the response back to the servlet, who then forwards it back to the user's browser.
- The AJAX script gets the response and displays it on the web page.

4.5.2 Machine Use Case

The machine use case proceeds as follows:

- A number of services come on-line, generate metadata, and add the metadata to the belief network.
- Through the belief network propagation, the client machine learns of the service metadata.
- The client program analyzes the microformats in each metadata and searches for one that matches its requirements.
- Using the microformat information, the client program generates the parameter object that will be used to invoke the service.
- The client machine creates an invocation belief representing the request and forwards it to the belief network.
- The service sees that it has an unfulfilled request, processes it, and writes the response into the invocation belief.
- The client machine sees that a response was written. It then parses the response using the response parser generated by parsing the microformat.
- The client program uses the data parsed by the response parser.

5 Experimental Design

5.1 Scenario

We use a chemical and biological agent monitoring scenario. Figure 2 shows a notional scenario with ground vehicles and Unmanned Aerial Vehicles (UAVs) in a tactical edge network environment. The ground vehicles are equipped with biological and chemical sensors to detect and alert on bio-chemical threats. All mobile vehicles use wireless communications and the circle around each vehicle indicates the communication range. In the tactical edge network one role of the UAVs is to act as communications links. The AMF is distributed over all vehicles and proxy services. The UAVs can transport metadata between the ground vehicles (edge-to-edge communications), between the ground vehicles and strategic services (edge-to-strategic communications) and from strategic services to the ground vehicles (strategic-to-edge communications). The Metadata could also be used within the strategic network as information retrieval proxies (strategic-to-strategic communications). Service invocation is also performed by transport over the UAV network, when necessary. The tactical network communicates via the SWARM software belief network. Using SWARMing, the UAV network will reconfigure itself to provide for more optimal metadata distribution and service invocation.

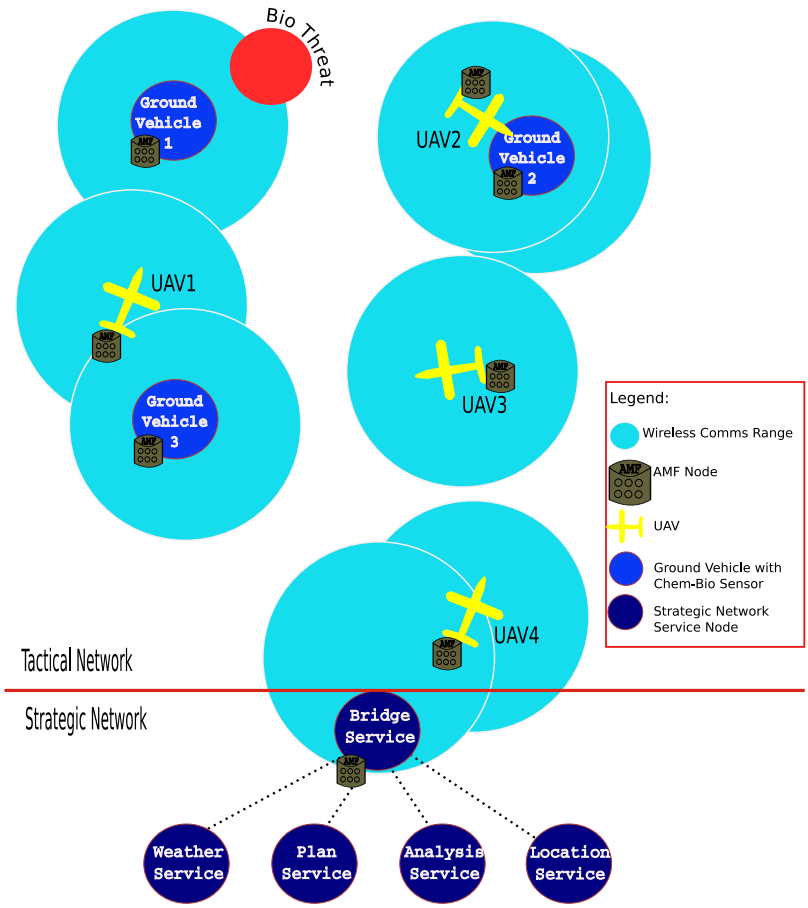

Figure 2: AMF-based UAV and ground vehicle nodes in a tactical network with reachback capability to strategic network and services

6 Results

In testing the SWARM-AMF system, our primary objectives were threefold: to have multiple UAVs self-coordinate to provide wide area communications coverage based on Unattended Ground Sensors (UGSs), Command Post location, and operator queries; to have multiple UAVs receive requests for information from an operator control station and change their flight paths to respond to those requests, and to acquire telemetry and Received Signal Strength Intensity (RSSI) data on each platform to analyze performance of autonomous vehicles and mesh network.

The Johns Hopkins University Applied Physics Laboratory (APL) successfully demonstrated the SWARM-AMF system at the Tactical Network Topology (TNT) experiment in Camp Roberts, California in February 2008. A picture of the setup area with distances involved can be seen in Figure 3. Our UAVs were flying at an approximate airspeed of 15 m/s. The communications range imposed on the devices was 150m. During the demonstration, a user at a Command

and Control (C2) station using a laptop or handheld device was able to request information via a dynamically discovered web page, have the requests routed across UAVs to two ground-based services, and have the responses routed via the mesh network back to the C2 station so the human operator viewed the query results. In doing so, we met the first two objectives above. We also captured the following data from the experiment:

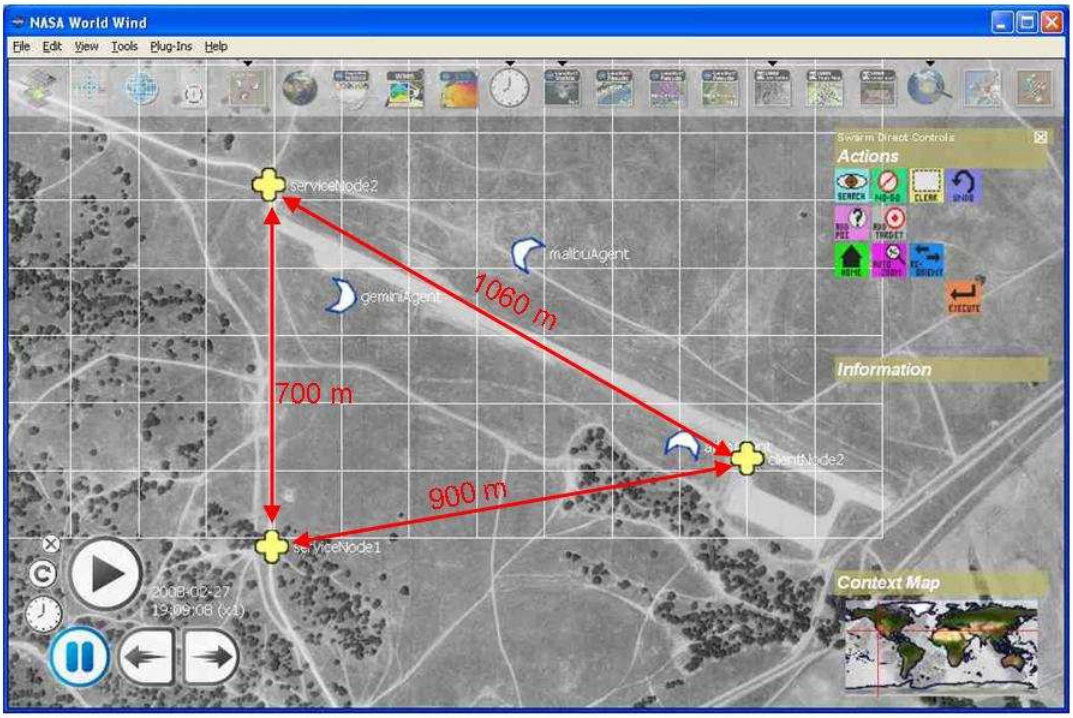

Figure 3: TNT In-Experiment display

- Time stamped positions of all APL UAVs
- Time stamped RSSI information for all network nodes
- Time stamped sensor readings from Chemical Biological Radiological Nuclear (CBRN) UGS

From the data we can conclude the average round-trip time to obtain a response to a query was 166 seconds, compared to the theoretical minimum time of about 140 seconds (if a UAV were already within comms range of the command station when the request was made). No queries were lost and all queries were eventually answered. In addition,

the system ran quite well during a later test when there was significant additional traffic on the wireless connections. Overall, the system worked as designed and there were no significant problems.

7 Discussion

In "Power to the Edge"[1], Alberts and Hayes state that the ability of a force to conduct net-centric operations and to self synchronize is linked to force agility and effects mission effectiveness.

The tenets of Net-centric Warfare state that:

- a robustly networked force improves information sharing,
- information sharing and collaboration enhances quality of information and shared situational awareness,
- shared situational awareness enables collaboration and self-synchronization,
- these in turn dramatically increase mission effectiveness.

The edge concept further asserts that to fully achieve the capability gains from net-centric warfare requires both interoperability and agility.

Conditions for interoperability requires that all force entities need the ability to exchange information, e.g. be connected to the net. In this experiment, force entities not only included the human users at the command post, it also included the mobile entities (UAVs) and the fixed sites simulating ground based Chem-bio sensors. Entities in the network, whether humans or machines need to be able to find, retrieve, and understand information available, and they need to the ability to participate in one or more virtual collaboration environments or processes. The demonstrated capabilities of the active metadata and swarming network technology support interoperability for mobile, edge organized, maneuver forces supported by an array of distributed, netted sensors.

The services oriented architecture used in the AMF-swarming demonstration eliminates the need for predetermined Information Exchange Requirements (IERs) or complicated Interface Design Documents (IDDs). Combining the need for information to be widely shared, along with the need for agility in the types of operations and C2 structures being supported, networks with predetermined exchange requirements and fixed structure are bound to limit interoperability and thus fail to support the mission. In order to be truly interoperable, the services need to support recognition of evolving needs to exchange information and discovery of information sources that are not predetermined.

The AMF-swarming technologies support discovery of services, as well as information requests from a variety of sources, whether it is autonomous exchange of information between sensors and unmanned vehicles, the support of information for distributed warfighters needing to share the information made available by a network of distributed sensors, or both simultaneously. This networking technology, while not demonstrated in the limited format TNT experiment, allows for the sharing of sensor fields for multiple simultaneous or sequential users and local or Over-The-Horizon (OTH) monitoring of distributed sensors.

Joint, coalition, and inter-agency missions require peer-to-peer interoperability at the tactical level in order to achieve a shared situational awareness. Jointness is too often thought of only at operational and above levels of command and

control. The ability to share information using the AMF-swarming SOA has the potential to extend the tactical edge network to the platforms, sensors, weapons, and individual warfighters operating as a unified force through its inherent interoperability and agility, the two key characteristics necessary to support a edge force. By linking distributed information services and service activations throughout a swarm network we gain the capability to generate a SOA even when the underlying network is highly dynamic.

Agility is further defined in terms of: robustness, resilience, responsiveness, flexibility, innovation, and adaptation. The AMF-swarming SOA developed at APL and demonstrated in the TNT experiment in February 2008 enables the network communications needed at the tactical edge to support C2 for distributed forces utilizing netted sensors and platforms, both manned and unmanned.

The combination of Active Metadata and Swarming network supports the synergistic combination of the key attributes of agility -

- Robustness - the ability to maintain effectiveness across a range of tasks, situations, and conditions.
- Resilience - the ability to recover from or adjust to misfortune, damage, or a destabilizing perturbation in the environment
- Responsiveness - the ability to react to a change in the environment in a timely manner
- Flexibility - the ability to employ multiple ways to succeed and the capacity to move seamlessly between them
- Innovation - the ability to do new things and the ability to do old things in new ways
- Adaptation - the ability to change work processes and the ability to change the organization

In addition to the improved situational awareness, the self-governing behavior of the unmanned vehicles allows for operations without dedicated operators. Vehicles acting on information requests from warfighters, unattended sensors, and other vehicles, all within the context of a functional SOA network was demonstrated. This demonstrated concept is poised to greatly enhance shared situational awareness and allow the information exchange needed to achieve the tenets of Net-centric Warfare.

References

[1] David S. Alberts and Richard E. Hayes. *Power to the Edge*. CCRP, 2003.

[2] John Allsopp. *Microformats: Empowering Your Markup for Web 2.0*. friends of ED, 2007.

[3] David Booth and Canyang Kevin Liu. Web services description language (WSDL) version 2.0 part 0: Primer. `http://www.w3.org/TR/wsdl20-primer/`, June 2007.

[4] R. Chalmers, D. Scheidt, T. Neighoff, S. Witwicki, and R. Bamberger. Cooperating unmanned vehicles. In *AIAA 1st Intelligent Systems Technical Conference*, 2004.

[5] T. Clausen and P. Jacquet. Optimized link state routing protocol (OLSR), October 2003. RFC 3626 (Experimental).

[6] R. Scott Cost, John Cole, Markus Dale, Christopher McCubbin, Ron Mitnick, and David Scheidt. Resource integration and inference in vanilla world. In *12th International Command and Control Research and Technology Symposium*, Newport, RI, June 2007.

[7] R. Scott Cost, Christopher McCubbin, and John Cole. The active metadata simulation framework. In *Agent-Based Ubiquitous Systems*, Honolulu, HI, May 2007.

[8] R. Scott Cost, Christopher McCubbin, John Cole, Markus Dale, and David Scheidt. Active metadata framework and emergent metadata distribution with scout agents. In *2nd International Workshop on Engineering Emergence in Decentralised Autonomic Systems*, Jacksonville, FL, June 2007.

[9] Darpa strategic technology office. `http://www.darpa.mil/STO/index.html`.

[10] M. Huhns and M.P. Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, 2005.

[11] Nilo Mitra and Yves Lafon. Soap version 1.2 part 0: Primer (second edition). `http://www.w3.org/TR/soap12-part0/`, April 2007.

[12] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing, July 2003. RFC 3561 (Experimental).

Appendix A: List of Acronyms

AJAX Asynchronous JavaScript and XML

APL The Johns Hopkins University Applied Physics Laboratory

AMA Active Metadata Agent

AMF Active Metadata Framework

C2 Command and Control

CBRN Chemical Biological Radiological Nuclear

DCF Dynamic Co-Field

GPS Global Positioning System

HTML HyperText Markup Language

HTTP HyperText Transport Protocol

IDD Interface Design Document

IER Information Exchange Requirement

OTH Over-The-Horizon

RDF Resource Description Framework

REST Representational State Transfer

RSSI Received Signal Strength Intensity

SOA Service-Oriented Architecture

TNT Tactical Network Topology

UAV Unmanned Aerial Vehicle

UDP User Datagram Protocol

UGS Unattended Ground Sensor

USSV Unmanned Sea Surface Vehicle

XHTML eXtensible HyperText Markup Language

Appendix B: Metadata Example

The following file is an example of the metadata that we have created for our field test.

```
<html>
<head>
<title> Chem/Bio Data Access Service </title>

<div class="hInvocation" style="display: none">
  <span class="service">
   <span class="type">chembiodb</span>
   <span class="location">serviceNode2</span>
  </span>
  <span class="parameter">
   <span class="type">double</span>
   <span class="name">ULLatitude</span>
  </span>
  <span class="parameter">
   <span class="type">double</span>
   <span class="name">ULLongitude</span>
  </span>
  <span class="parameter">
   <span class="type">double</span>
   <span class="name">LRLatitude</span>
  </span>
  <span class="parameter">
   <span class="type">double</span>
   <span class="name">LRLongitude</span>
```

```
  </span>
  <span class="parameter">
   <span class="type">datetime</span>
   <span class="name">startTime</span>
  </span>
  <span class="parameter">
   <span class="type">datetime</span>
   <span class="name">endTime</span>
  </span>
  <span class="returns">
   <span class="type">table
     <span class="column">
       <span class="type">integer</span>
   <span class="name">detectindex</span>
     </span>
     <span class="column">
       <span class="type">double</span>
   <span class="name">latitude</span>
     </span>
     <span class="column">
       <span class="type">double</span>
   <span class="name">longitude</span>
     </span>
     <span class="column">
       <span class="type">datetime</span>
   <span class="name">time</span>
     </span>
     <span class="column">
       <span class="type">string</span>
   <span class="name">originator</span>
     </span>
   </span>
  </span>
</div>



<script type="text/javascript">
    // <![CDATA[
                    function getxmlhttprequest()
                       {
                       var xmlHttp;
                       try
                         {
                         // Firefox, Opera 8.0+, Safari
                         xmlHttp=new XMLHttpRequest();
```

```
                        }
                      catch (e)
                        {
                        // Internet Explorer
                        try
                         {
                         xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
                         }
                       catch (e)
                         {
                         try
                           {
                           xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
                           }
                         catch (e)
                           {
                           alert("Your browser does not support AJAX!");
                           return false;
                           }
                         }
                        }
return xmlHttp;
                      }

     var ajaxRequest=getxmlhttprequest();

function callBack()
{
if(ajaxRequest.readyState == 4)
{
if(ajaxRequest.status == 200)
{
document.getElementById('response').innerHTML = ajaxRequest.responseText;
}
}
else
{
document.getElementById('response').innerHTML = "<img src = '/amf/images/throbber.gif' alt
}
}

function sendRequest()
{
var url = "/amf/SwarmAMFServlet";
var query = "ULLatitude=" + document.queryForm.ULLatitude.value + "&ULLongitude=" + documer
ajaxRequest.open("POST", url, true);
```

```
ajaxRequest.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
ajaxRequest.onreadystatechange = callBack;
     ajaxRequest.send(query);
}
     // ]]>
</script>

</head>

<body>

<center> <h1> Chem/Bio Data Access Service </h1> </center>

<br />

<b> North West Coordinates: </b>
<form name="queryForm">
<input type="text" name="ULLatitude"/> Latitude <br /><br />
<input type="text" name="ULLongitude"/> Longitude

<br />
<br />

<b> South East Coordinates: </b> <br />
<input type="text" name="LRLatitude"/> Latitude <br /><br />
<input type="text" name="LRLongitude"/> Longitude

<br />
<br />

<b> Time: </b>YYYY-MM-DD hh:mm:ss (military) <br />
<input type="text" name="startTime"/> Start <br /><br />
<input type="text" name="endTime"/> End

<br /> <br />
<input type="button" onclick="sendRequest()" value="Search"/>

</form>

<div id="response"></div>

</body>

</html>
```