12^{TH} ICCRTS
"Adapting C2 to the 21^{st} Century"
Resource Integration and Inference in Vanilla World
Modeling and Simulation, Network-Centric Experimentation and Applications, Technologies and Systems

R. Scott Cost, John Cole, Markus Dale, Chris McCubbin,
Ronald Mitnick, Dave Scheidt
POC: R. Scott Cost
Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Rd.
Laurel, MD 20723
443-778-1137
scott.cost@jhuapl.edu

May 2, 2007

Abstract

In a highly dynamic environment, knowledge of currently available and relevant resources is essential to the development of comprehensive situational awareness and assessment of threat level or status. We present an agent-based framework, the Active Metadata Framework, which supports the intelligent distribution of resource information among diverse assets, in a manner which supports focused awareness of resources and provides some measure of network disruption tolerance. This framework supports the concept of *power to the edge* by rapidly moving key data products directly to front-line warfighters. This accelerated movements of information improves war-fighter effectiveness by shortening the observe-orient-decide-act (OODA) loop. This framework is demonstrated within a simulated complex military engagement in which assets are required to identify information sources (such as other similar assets, sensors, or databases), and utilize the data they provide to infer information about their situation and required actions. Inference in this framework is performed using Markov Logic Networks (MLNs) with temporal extensions. Preliminary results are presented which demonstrate the successful, ad-hoc creation of networks of assets using this framework, the effective distribution of data through this dynamic network of assets, and the ability to infer information of value from the available data.

1 Introduction

In a highly dynamic environment, it is important to be able to connect entities with the information resources most relevant to their immediate need. As the number of distributed services available increases, the amount of data available grows. For large systems the amount of information potentially available at a single node can easily overload the capacity of an individual node. It is therefore important for information to be filtered, limiting information provided to a node to that which is useful. Typically the filtering of data sources is performed by the operator, or a supervisor responsible for operator performance. Reliance on the edge war fighter is costly in terms of manpower and time-criticality as during those times when new information is most important edge war fighters do not have time to pull information from the network. This leads us to the first contribution of this effort, a 'smart' distributed technology that is capable of getting the right information to the right people at the right time. These smart technologies must provide a 'pluggable' framework to dynamically discover and integrate heterogeneous sensor information, and dynamically adapt to ever-evolving environments. Local information must be combined with larger, distributed data systems to create a complete informational picture.

Adversary activities are the most important time-critical information. While stealth and deception have always been used in warfare, their use has never been more prevalent than today's fourth generation conflicts. Adversary activities are only partially observable and the amount of similar non-combatant activity, 'noise', surrounding adversary activity is high. The current analysis-based approach to identifying the precursors to a fourth generation attack is both labor intensive and time-consuming. The latency between a key observation that allows an analyst to "connect the dots" and when an actionable intelligence report reaches the edge warfighter limits the tactical use adversary information, placing our forces in a reactive, rather than proactive position. This motivates our second contribution which is a reasoning system that provides tactical information products for the edge warfighter by rapidly, and autonomously inferring threatening adversarial behavior.

One example of a dynamic environment which requires this type of information awareness service is in-theater military situational awareness. We have developed a generic scenario called *Vanilla World* which represents just such an environment. We have developed an architecture of intelligent software and networking components that will allow for enhanced situational awareness.

In this paper we describe the application of our architecture to the Vanilla World scenario and present our results and analysis.

2 Background and Motivation

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 integrating information resources. These resources may exist in relatively static or highly dynamic environment. 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.

Because the information resources available may not be known during design or even deployment the agile information integration will require run time 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 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 push information to potential consumers. In contrast to the pull-based directory approach, service providers have a responsibility to make themselves known directly to consumers, which may or may not need the information at the time they receive it, but may cache it for later use. 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 push-based model, puts the burden on the resource description, or metadata, itself, by creating and leveraging the use of intelligent resource proxies, or agents. These agents are tasked seeking out likely partners for the resource they represent. In this way, once released, they free the advertising resource from the responsibility of managing awareness. This simplifies the job of the resource, and makes the framework more robust in the face of intermittent network connectivity. This provides for a scalable, distributed approach to the advertisement of resource information with a domain that is also tolerant of network delays and failure.

2.1 Related Technology

Our approach combines several recent advances in agents, service oriented architectures[18], mobile ad-hoc networking[5], and simulation. Combining these technologies is an active area of research. Implementing Service Oriented Architectures (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[4, 11], 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 use of agents within Mobile Ad-Hoc Networks (MANETs) is another current area of research. Several research teams have focused on the use of agents to improve bandwidth and real time communications within MANETs[8, 15]. Peysakhov experimented with this idea in the domain of the "compromised host" problem, in which agents were allowed to reason about the state of the underlying network topology, based on a state description of the MANET[12].

Automated migration of agents to geographical area of interest inside MANETs was explored by Tei[17]. The agents then migrated intelligently to remain in this location despite changes in the underlying MANET.

Hijazi researched the use of agents for intrusion detection within MANETs[6]. His team utilized agents to reduce the strain on communications across the network with lack of infrastructure by migrating code, rather than data, to reduce latency and save battery life.

Moro et al.[9] describe the advantages of using agents in a P2P environment to overcome the P2P limitations of message expressiveness, data models, data integration/transformation and routing. In [16], we see an example of an agent-based P2P system that uses queries (an information "pull" paradigm) to discover resources. That system consists of "ResultAgents" that generate a resource query, "SearchAgents" that distribute the queries to their peers and "Resource SearchAgents" that match the query against ontology-based resource descriptors. The resource descriptors remain stored with the resource that they describe. The Active Metadata agent system in contrast uses an information "push" paradigm to distribute resource metadata. The K-Trek system[3] describes a framework where resource information is also statically stored with the resource in form of a "K-Beacon" which "stores contextual information about a specific location" and periodically broadcasts information about itself. This information source can be discovered by "K-Voyager" mobile devices based on location proximity. Active Metadata agents allow information about a resource to be spread to locations in the network where this information can likely be used beyond location proximity. The TOTA ("Tuples On The Air") system described in [7] provides a P2P middleware which uses tuples with content and propagation rules to distribute location- and content-based information which is similar to the Active Metadata system. However, the Active Metadata system also provides proxy services to resources via the instantiated Active Metadata agents representing a resource. TOTA also used an emulator environment to analyze propagation in a larger scale MANET environment.

3 System Design

The overall design of the system can be seen in Figure 1. We have created a generic scenario called "Vanilla World". A detailed description of this scenario can be found in section 3.1. This scenario consists of a set of events of various

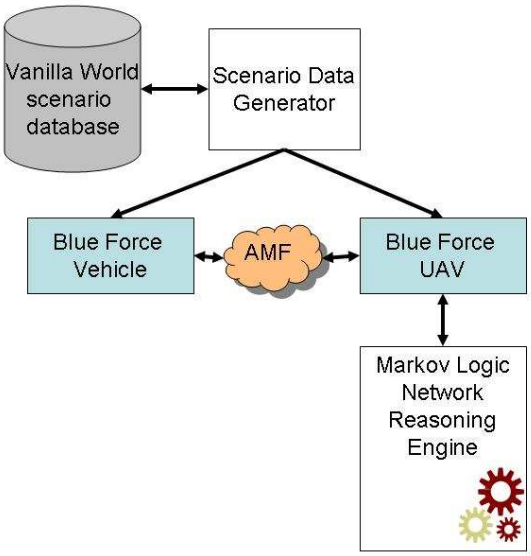

Figure 1: Simulation Inferencing System: Overall Design

types. A piece of software called the *Data Generator* interfaces with this database and publishes events at a specified multiple of real-time. These events are captured by agents representing friendly, also known as blue, forces. These forces are then in turn connected to the Active Metadata Framework (AMF), a prototype architecture for the intelligent dissemination of metadata. This framework is described in section 4.2. Using this framework, agents share organic information with other blue force entities. Some blue force entities have a reasoning engine that attempts to determine several things, such as probable terrorists or threat indexes. This reasoning generates the main metrics of the system. The reasoning engine we use, called MLNs, is described in section 4.3 and the results are detailed in section 5.

3.1 Vanilla World Scenario

We have designed a scenario which we call Vanilla World. This scenario takes place in a hypothetical country, and describes a number of events, some of which are undesirable. We wish to detect these undesirable events from a stream of data that contains not only information leading up to the events, but also a significant quantity of random "normal" events.

The scenario used in the experiment required randomly generated "noisy" background data in which to embed the significant undesirable activity that the system is meant to detect. The following rules were used to create approximately 70 days worth of activity; days 1-39 are considered historical data and days 40-70 are used to generate the simulation.

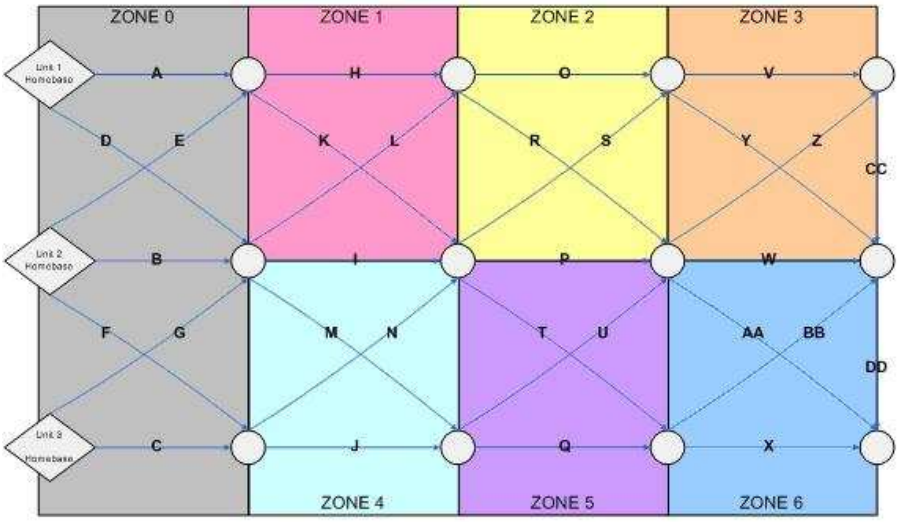

Figure 2: Capitol City Zones/Segments

- Two thousand identities were created randomly from a list of names.
- For locations, a "Capital City" was divided into six zones, as shown in Figure 2. In addition, five external cities were utilized.
- Each person was assigned a home city. Half of the names were assigned to one of the Capitol City districts, whereas the other half were divided evenly among the five external cities.

One of the data streams in the simulation is information about aircraft manifests. In the scenario, there are ten aircraft originating from Capitol City, and one aircraft from each of the 5 external cities. In the A.M. of any given day five aircraft leave Capitol City; one for each of the five external Cities. In the P.M. of any given day one aircraft from each external city returns to Capitol City; the other travels to another external city so that each city ends with the original number of aircraft they started with. To determine the passengers for each flight, two groups of 100 persons were selected at random from Capitol City and divided into groups of 20, with overlap allowed. Next, four groups of 10 persons were selected from each of the five other destinations, with overlap allowed.

Another type of data stream in the scenario is a phone call log. To create the phone call log, at each time frame we created 50 cell phone conversations, selecting sender and receiver randomly. There was no correlation between time frames.

Finally, every day each of the three Units based in Capitol City patrol a path along the "segments" of the city (see Figure 2) that eventually return them to their original base. Patrol information for days 1-39 are found in the historical record. Patrol data for days 41-70 are used by the simulation, but also stored in the historical record as the "planned routes" for the Units.

3.1.1 Significant Activity

Significant-Activity data was added to the noisy data to create the final dataset that would be stored in "historical" databases and distributed by the Vanilla-World simulator.

First, a collection of Human Intelligence (HUMINT) data with names from the "Names" and "Locations" database tables was compiled into a free-text file. Next, eighteen "Significant" names were added to the database. Names from the "Significant" list were added to the passenger lists of certain flights to coincide with the periods preceding events of interest in the scenario. In addition, names from the "Significant" list were added to the phone call list; contacting each other to coincide with the periods preceding events of interest in the scenario. Event reports were added to days 1-39 of the dataset. These events were added to the historical and simulation datasets, but were not distributed to the simulation until seven (simulation) days after the event. Unmanned Aerial Vehicle (UAV) reports were added to days 1-39 of the dataset. The possible UAV reports included significant precursors to the events of interest. Each UAV report is identified by segment or zone (See Figure 2).

It should be noted that only one quarter of the UAV events defined for the scenario were actually added to the dataset to represent activity when a UAV was not present. An additional five percent of false-positive activity was added to the dataset as well.

3.1.2 Accessing Historical Data

The historical data in the system is stored in a set of disparate data sources distributed across the network. Each set of data offering a different piece of information that can be correlated to identify possible threats. For the purposes of this experiment all of the historical data is stored in two repositories that can be accessed by any agent as-if it were heterogeneous data sources. The first is a Microsoft Access database file that contains tables storing the following information: Names, Cities, Zones, Home cities, Aircraft, Flights, Passengers, Phone-calls, Patrols, event reports, and UAV Reports.

The second data source is the text file of HUMINT data in free-text format. Scattered throughout this intelligence-data are keywords that can be matched against data in the historical database and data from the real-time simulation.

4 Component Descriptions

4.1 Vanilla World Simulation System

The Vanilla World simulation software was designed to present the ground-truth information defining the target scenario. Rather than have every software agent in the experiment simulate its role in the scenario independently, it was decided that a single coordinator process would schedule all the ground-truth data. The coordinator would then communicate the relevant event data directly to the agent producing/observing that data in the simulation. Various agents then communicate event among themselves as appropriate. For example, the simulation software would inform a sensor of an event that it should have 'observed' in the context of the scenario. That simulated sensor would then communicate event detection information directly to other data consumers in the simulation.

The Vanilla World simulator consists of two parts; A Microsoft Access database that mirrors the format of the one used by the historical database and a Java process that reads the database and distributes the message to the agents. The database contains a single table per type of agent that will distribute the data as its own. Event types contained in the database include: Phone calls intercepted by listening posts, event reports, patrols status reports, and UAV status reports.

In addition to event type, the database also records such relevant information as event time, and associated entity (e.g. the simulated entity for which the event is directly relevant.) This information is used to distribute event data to the correct entities and in the correct sequence in the Vanilla World simulation.

4.2 Active Metadata Framework

4.2.1 AMF Architecture

Our implementation utilizes an agent-based framework to implement the push network architecture. Each component of an AMF node is, in fact, an agent following a set of behaviors, which are based on conversations/interactions with other agents. The mechanism for the implementation of the push network is the migration of these agents from node to node. As these agents travel through the network, specific nodes may query them in order to obtain metadata regarding the resource they represent. In order to ensure that this metadata is current, the migrated agents must periodically update their metadata from their source node. In this section, we briefly highlight the functionality of each agent which makes up an AMF node, as well as the process of migration and updating within the network. A high-level diagram of a typical AMF node can be seen in Figure 3.

Components The Active Metadata Agent (AMA) is the building block of an AMF node. AMAs carry the node's metadata around the network as they migrate between nodes. These are the agents to be queried by other nodes to find sources or consumers of data within the network.

A variety of other service agents implement the functionality of the AMF framework itself. The *Manager Agent* serves as a proxy to outside requests to the AMF node. It handles all requests to populate the node with AMAs, as well as queries as to which AMAs currently reside on the node. With all requests, the Manager begins and facilitates conversations with the corresponding agents on the node to produce the desired results. The *Instantiator* and *Deinstantiator Agents* are responsible for low-level creation and destruction of AMAs, respectively, and transition between nodes. The *Atlas Agent* holds information on all AMAs which have lived on the node, as well as other nodes connected to the local node. It maintains a database entry for each AMA this node as seen containing its local name, who it was created by, where it came from, and its last known address. Network connectivity between nodes is managed by a *Network Bridge Agent*.

The *Detailer Agent* is a major hub of activity in the AMF node. It handles requests for migration from the AMAs, as well as registering and de-registering AMAs by utilizing the Instantiator and Deinstantiator. In addition, this agent handles updated metadata provided by its node. The Detailer will pass this updated information through the network to all living AMAs that originally came from the Detailer's node.

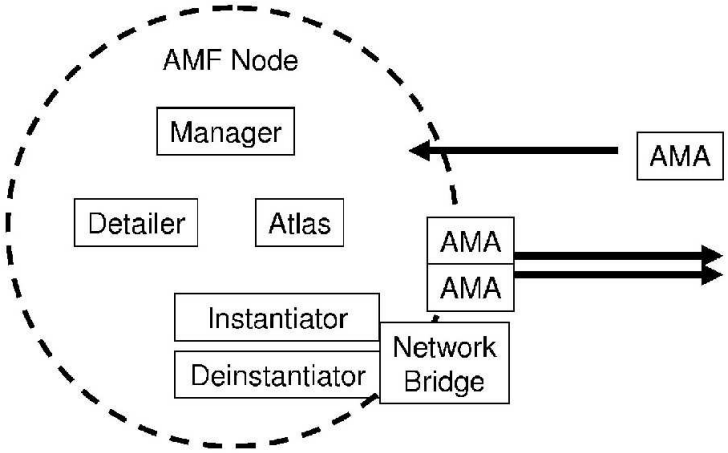

Figure 3: An example AMF node, with all agent-based components listed.

Migration The mechanism by which the push network architecture accomplishes propagation of metadata is via migration of the AMAs through the network. Through this process, AMAs find their way to nodes where their metadata can be best utilized, and remain resident for some period of time. Nodes can query locally resident AMAs for any desired metadata. For example, AMAs originating at a node which produces a certain type of data (e.g. imagery for a specific geographic region) will migrate to nodes where such data is likely to be needed, and vice versa.

As AMAs migrate throughout the network, they lose contact with their home node, and therefore do not see all changes and updates to their node's metadata. It thus becomes necessary for these AMAs to periodically attempt to update their metadata from their home node, in order to obtain current accurate metadata. A detailed description of this process, which is complicated by multiple migration steps and potential gaps in connectivity, is beyond the scope of this paper. It is important to note, however, that the framework will provide maximally up-to-date resource information when connectivity is available, and best-guess capability when nodes are isolated from peers in the network. As with migration, the process of AMA/originating host connectivity, which is managed by the *Detailer Agent*, is facilitated by the *Atlas*. Updates may be 'pulled' on demand by remote AMAs, or may be distributed via push from the originating resource.

We have described our agent-based implementation of the push network architecture, where each component is a rule-based agent. These agent-based components communicate with each other using behaviors based on the messages received. The final implementation is based on the ability to migrate AMAs around the network, each with the ability to periodically update its own metadata.

4.2.2 AMF Validation Experiments

We conducted several experiments of AMF at the site of The Johns Hopkins Applied Physics Lab (Figure 4). In the context of this experiment, we created three AMF nodes, two of which served as ground stations, while the third traveled in and out of connectivity along the roads surrounding the test site. As the moving vehicle moved in and out of

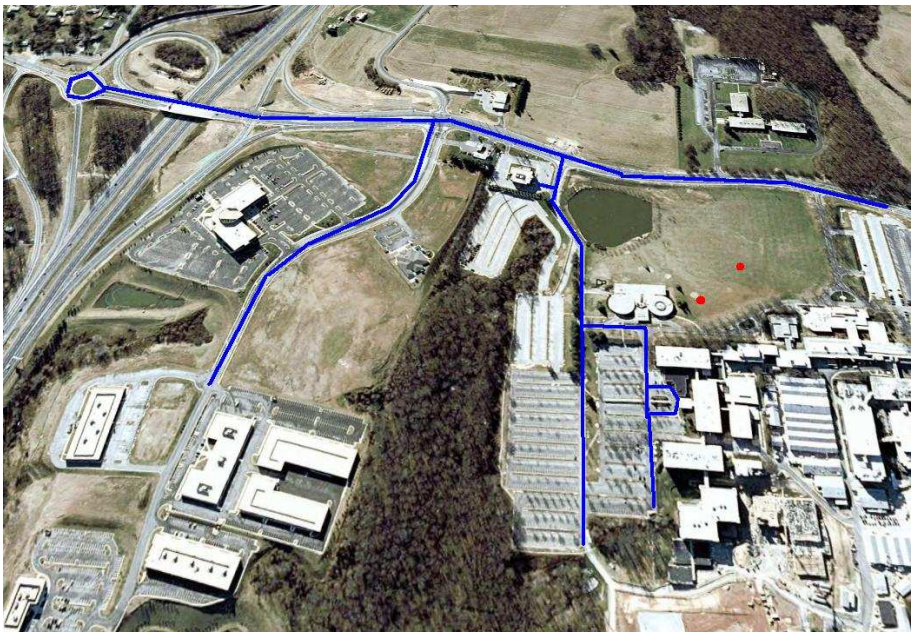

Figure 4: Satellite image of APL testing area. Two ground stations are shown in red, while the traversable area of the moving vehicle is shown in blue.

range of the ground stations, AMAs traveled among the three nodes, dispersing metadata throughout the network. The intent of these tests was to examine the moving vehicle's ability to gather current metadata produced by the ground stations in real time through its AMF node by examining the produced data using a client application.

Physical Setup Our experiment was conducted on an 802.11g ad-hoc network. Each AMF node consisted of a Dell laptop connected to a Lynksis wireless ethernet bridge, which in turn was amplified by a G network Hyperlink antenna amplifier.

The ground stations were given fixed Global Positioning System (GPS) coordinates by a hand held Garmin GPS device. However, the moving AMF node's laptop was connected to a Garmin GPS device to obtain current GPS locations. This moving AMF node was placed in a vehicle and driven around the surrounding areas of the test site.

Services Each ground station was able to continually supply images from an infrared camera. These images could be produced periodically, or upon a detection of movement in the image. The ground station's AMF node would be updated with metadata on these new images as they became available. Additionally, the ground station supplied its static GPS location.

In contrast, the moving vehicle was a consumer of these camera images. Specifically, the test was designed for the moving vehicle to gather the image from the closest ground station within a configurable range, and be able to view that image on demand.

Figure 5: A screenshot of the moving vehicle's client application GUI. The top line shows filtered query results from the AMF node, while the bottom line shows the current GPS location of the vehicle.

Client Applications The purpose of the ground station client application was simply to provide updates of the infrared image metadata. Upon arrival of any new data, the client application would relay this update to its AMF node, which in turn would update its AMAs. A screenshot of the client application can be seen in Figure 5.

The moving vehicle's client application was in fact a Graphical User Interface, providing the driver of the vehicle updated metadata on the closest ground station's infrared images. The Graphical User Interface (GUI) did so by querying the AMF node for any AMAs which came from a ground station. Of these, the GUI pulled the GPS location of the ground stations, and displayed the GPS location of the closest ground station to the vehicle within a range chosen by the user of the GUI. In addition, when that ground station contained an infrared image to consume, the GUI provided the user a button to view the image, using the metadata provided by its AMF node.

4.2.3 AMF Simulation Environment

Following lessons learned in field simulations, we have created an AMF simulation environment. This environment allows us to more easily experiment with various protocols and approaches to metadata distribution than the full-blown implementation. As the simulation environment itself was not used in the overall architecture described in section 3, a detailed discussion of the implementation is beyond the scope of this paper. However, it has proven invaluable to rapidly evaluate AMF algorithms. We describe it briefly here.

Our AMF simulation framework is designed in Java with certain specific goals in mind; primarily, the ability for rapid prototyping and testing of different AMA behavior strategies as well as the different actions from users of the AMF network. In addition, a flexible, abstract framework is needed for the porting of the simulation to field tests. We would like to make the framework flexible, so that as much code as possible from a simulation can be ported directly into field tests. These goals are accomplished by placing an abstract framework in place, which will apply to all strategies and underlying network representations.

Simulation frameworks for ubiquitous computing are currently being researched in several projects. Reynolds et al. [13] have laid out a set of requirements for a generic tool to simulate many types of ubiquitous computing situations. Hewlett Packard's Ubiwise[2] is another simulation environment that combines a network simulation with a 3-D visualization environment based on the Quake3 engine. A similar simulator is the Tatus simulator[10].

Framework Structure Each AMF node is represented as a class which holds a user proxy implementation, as well as a listing of all AMAs currently occupying the node. In addition, the node maintains a database of objects that AMAs can query or post to while they inhabit the node. The abstract user proxy class allows for varying implementations of user actions throughout execution. An implementing proxy must handle an initialization call upon the start of execution which will create and return a list of the user's AMAs. Furthermore, at each timestep, the user proxy will be allowed to execute any additional user action, such as updating the metadata for its AMAs. Each AMA is represented as a class which holds its assigned metadata. The metadata class will be abstract, as to allow several possible types of implementing metadata. In order to implement its behavior strategy, the AMA is assigned two abstract classes, *behaviors* and *abilities*. The behaviors class implementation will allow for the implementation of different migration and updating strategies by the AMA, whereas the abilities class implementation provides the basic network abilities to the behaviors class in order to perform its strategies.

Behaviors The abstract behaviors class is implemented by two functions: an initialization function, and a method to execute the AMA strategy. The initialize function is invoked when the AMA is first created by its originating node's user proxy. From then on, during each timestep the execute strategy method is invoked on the AMA. When calling this function, the behavior is additionally provided with whether or not this is the first execution timestep since a successful migration in case any additional processing is required.

Abilities An abilities implementation will be the interface with which the behaviors class interacts with the framework. Functions such as *tryToUpdate* and *tryToMigrate* will be implemented here according to the underlying framework implementation. For example, these functions may be much more complex if the framework is implemented for field tests, whereas they might be fairly simple for simulations. This flexibility allows for rapid prototyping and portability of our code across different platforms.

Graphical Interface In order to test our AMA migration strategies, a graphical user interface was created to view the migration and logs of the AMAs as they traverse the network. An example network is shown in Figure 6. Nodes are shown in orange, while AMAs are shown as red dots inside the nodes. At any point in the simulation, the user can click on the node to see a listing of the AMAs currently on it, as well as their current metadata. In addition, the user can click on the node's database, shown in green, to see a listing of the logged objects by that node.

To begin the simulation, a user first loads a configuration file, which describes how the user proxy and AMA behaviors and abilities will be implemented, as well as the structure of the underlying network. The user also has the ability to change the underlying network at given timesteps through the configuration file. Once this file is loaded, the user clicks initialize which displays the network, and invokes the initialize method on each node's user proxy as well as each AMA created by the user proxies.

The graphical interface now allows the user to step, play, and pause the simulation by using the provided VCR controls. Upon each step, the simulation invokes the execute behavior methods in each node's user proxy, as well as on each AMA.

Underlying Network Network connections between nodes in this simulated interface are represented by two values, the probability of failure and the type of failure. Once a link has been determined to have failed by drawing random

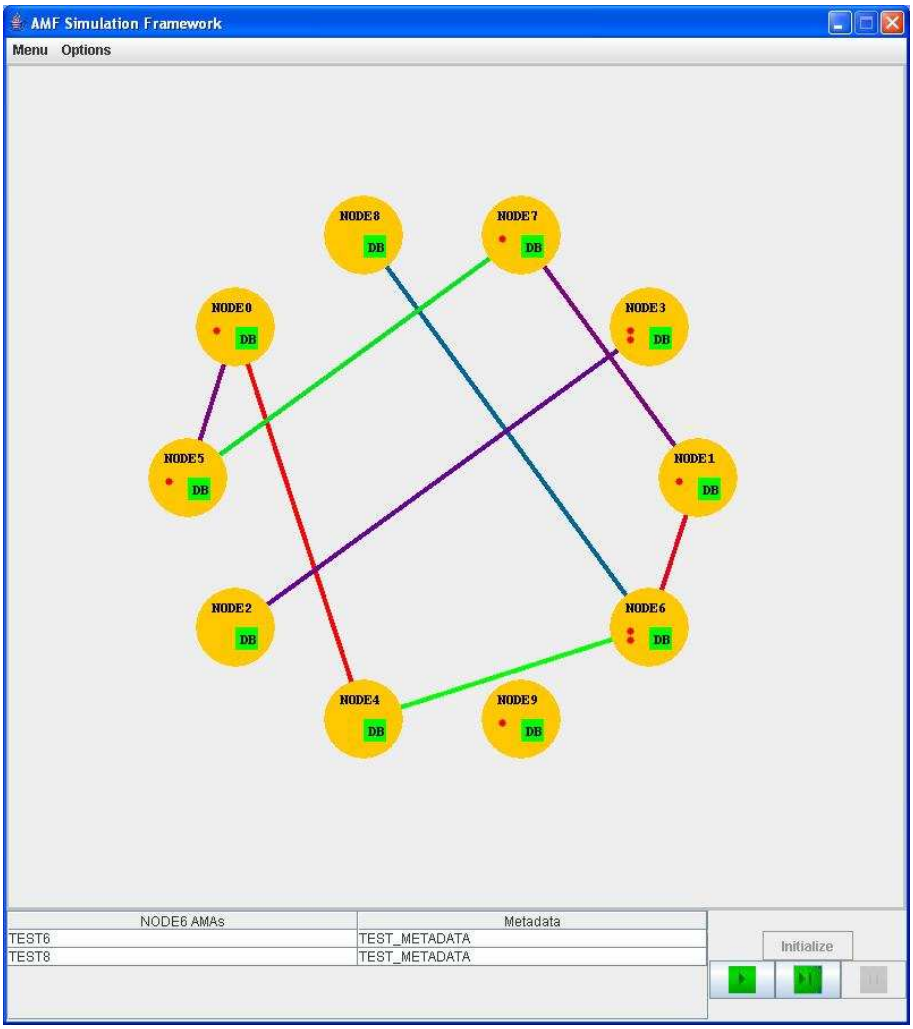

Figure 6: Graphical User Interface screenshot, showing an example AMF network.

numbers according to its probability of failure, the link can fail in two ways. First, the link can simply bounce all communications, in which case the sending node is aware of the loss of communications. For example, a migrating AMA will be aware of the failure, and its sending node will be notified of the loss, and keep the AMA there. However, the other type of failure forces all communications across a link to be lost, in which case the sending node is not made aware of the loss. In the case of migration, the sending node believes the communication to have worked, and removes its local copy of the AMA, whereas the receiving node never receives the AMA, causing it to be dropped from the simulation.

4.3 Markov Logic Network

Markov Logic Networks (MLNs) are a tool combining first order logic with probabilistic reasoning. A MLN can be described as a collection of statements in first-order logic. A full description of MLNs can be found in[14]. First order logic statements are made up of conjunctions, disjunctions, and existential operators applied to a set of *ground atoms*, variables that can either be true or false. An example of ground atoms are "rained()" which is true if it rained, and "wetGrass(lawn)" which is true if the grass is wet on the specified lawn. A first order logic statement would take the form "If it rained and the season is spring, then there exists a lawn where the grass is wet". Each first order logic statement in a MLN is given a weight which is approximately equivalent to the probability of that statement holding. Once we have a network like this in place, we may *assert* a set of true ground atoms, and ask the system questions like "given that the ground atoms I have listed are true, what is the probability that this other ground atom is true?". We are using the 'alchemy' implementation of MLNs.

In our studies we are also using a variant on the idea of *fluents*, that is, some of our ground atoms also have a discretized notion of time as part of their variable tuple. For example, a fluent "wet(lawn, day)" is true if the lawn specified is wet on the specified day. We utilize alchemy's ability to define predicate functions that can be specified in C code to implement functions like "before(day1,day2)" which is true when day1 falls before day2. In the following sections we describe the specific MLN that we used for our experiments.

5 Results

The motivation for this effort is to develop an autonomous resource integration system that supports agile forces. The quality of the system must be judged by the correctness, utility and timeliness of the information provided to the operator. To understand the correctness and utility of the system we go to the three tasks the system is being asked to perform:

1. Identify patterns of behavior that indicate a threat
2. Match existing behavior to established threat patterns
3. Transmit information on current threats to threatened friendly forces

To date our efforts have focused upon hardware in-the-loop experiments with the active meta data infrastructure and simulation-based experiments of the Markov Logic Network. The AMF experiments focused upon (3), the ability

to transmit information on current threats to threatened friendly forces. Markov Logic Network experiments have focused on (2), matching current observations to hidden patterns of threat behavior.

Each of these tasks can be evaluated by three metrics:

1. The probability that a result was correct (P_D)
2. The probability that a result was incorrect (P_F)
3. The speed at which the task was performed

For our AMF experiments P_D is the percentage of available germane facts that were provided to each member of the blue force and is P_F is the percentage of unimportant facts that were provided to each member of the blue force. For MLN experiments P_D is the percentage of attacks in the vicinity of a member of the blue force that were predicted and transmitted to the blue force and P_F is the percentage of non-threatening behaviors that were transmitted to members of the blue force as "threatening".

5.1 Active Meta-Data Results

A series of hardware in the loop experiments with as many as three blue force vehicles, three unmanned air vehicles, two unmanned ground sensors and intelligence databases were connected through AMF. The blue force vehicles conducted a mock war game against red team forces that included vehicles and dismounts. The small size of the opposing forces limited the experiment's complexity and scope. In addition, the fitness criteria for determining for "threatedness" used by AMF to filter information and the evaluation criteria (P_D and P_F) were the same. Over the course of the experiments P_D was 100

5.2 Reasoning Engine Performance

The objective of the MLN testing two-fold: first we sought to establish that the speed at which threatening behavior could be identified could be reduced sufficiently to be used tactically rather than strategically or forensically; second, we needed to infer information on behavior that was tactically useful.

In our testing of the MLNs application to the Vanilla World data, we must create a mechanism for inserting events into the MLN as they occur in simulation time from the Vanilla World. To accomplish this, a data generating simulation agent was created. This "ground truth" agent steps through the events of the Virtual World at a configurable simulation rate. These events, such as phone calls and precursor observations, will then be packaged by the ground truth agent, and broadcast to another intermediate agent. This intermediate agent parses and collects these events, appends the *currentday* predicate, and sends them to the MLN inference engine. The engine will run the current collection of ground truth observations through its MLN, and return the computed results for the Persons of Interest (POIs) and *threatened* predicates for each person and location to the intermediate agent for analysis.

Baseline Results Our first experiment was set up as previously described, with the ground truth agent supplying observations at a simulated rate of five seconds for each thirty minutes of the Vanilla World. These observations were packaged by the intermediate agent, and sent to the MLN inference engine at every instance of a non-call event, and at the end of each simulated day. So any time the intermediate agent was presented with a non-call observation, such as a precursor event, or an observation for a new day, the agent would collect all previous observations, append the *currentday* predicate, and send the set of observations to the MLN inference engine. The results from this experiment are shown in Figures 7, 8, and 9. In each of these graphs one can see the threat level determined by the MLN for each simulated day. The red lines indicate when significant events take place. Threat level rises in concert with significant events in these three zones. It is important to note that no zone other than the three shown experienced significant events in the Vanilla World, and the MLN threat level for those zones shows any indication of a imminent threat.

A-priori Information By analyzing the output of the threat levels, we noticed a failure to identify the early events in the Vanilla World due to the lack of *a-priori* information. Without a previous knowledge of any known POIs, the MLN engine is unable to identify the patterns leading to the first significant events. However, once these events happen, POIs become identified, as well as patterns leading to the event. So future events can be identified.

To examine the value of even a small amount of *a-priori* information, we first examined the baseline results to compile a list of possible POIs. We chose the nine people identified by the MLN engine as POIs for the longest period of time. We classified an identification of a Person of Interest (POI) as that person's computed probability of being a POI to be above ninety percent. Next, we added this list of "known" POIs to each collection of observations sent to the MLN engine in successive simulations. An obvious improvement was seen in the detection of the first significant event. As shown in Figure 10, the first event was predicted a full three days before the event was reported. As the simulation progresses, the simulation with no a-priori information "catches up" and discovers the POIs itself. This is the reason why the advance warning only occurs for the first event.

Obtaining Social Networks Another interesting analysis of output from the MLN inference engine is the construction of social networks. By looking at the *friends* predicate as an output from the MLN, as well as the *POI* predicate, we can construct social networks graphs such as Figure 11, where the circle size relates to the probability a person is a POI, and line width related to the probability two people are friends. We have filtered the "friend links" to only appear in cases where the probability of the two people being friends was computed as greater than fifty percent.

6 Future Work

There are a number of avenues that we would like to pursue further. Clearly, if such an approach is to succeed, there must be a means of driving the intelligent distribution of AMAs. This is very rudimentary in the current instantiation. However, it is intended that migration patterns of AMAs should be guided by information collected about the environment, local needs and capabilities, and sophisticated algorithms and protocols. There is a balance that needs to be struck; relying on/requiring too much information will result in slower performance and a reliance on broader connectivity. On the other hand, the use of minimal information would likely result in inefficient or ineffective distributions of metadata. There are a number of different aspects of the work which are relevant to this, including the compilation and representation of knowledge about the world, the algorithms which make use of this information, and the mechanisms of interaction employed (e.g. AMA to home-resource interaction, AMA-AMA coordination, etc.) Our current focus

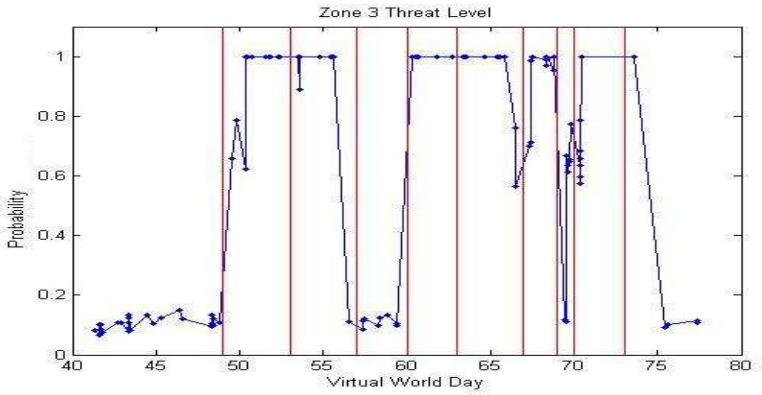

Figure 7: Predicted threat levels for the zone 3 of the Vanilla World throughout the simulation.

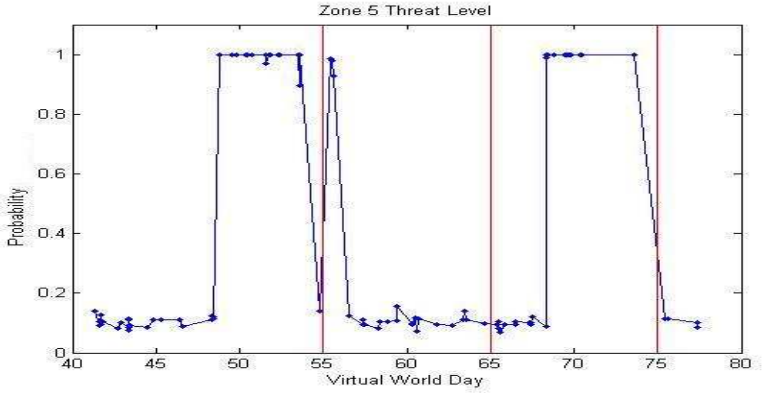

Figure 8: Predicted threat levels for the zone 5.

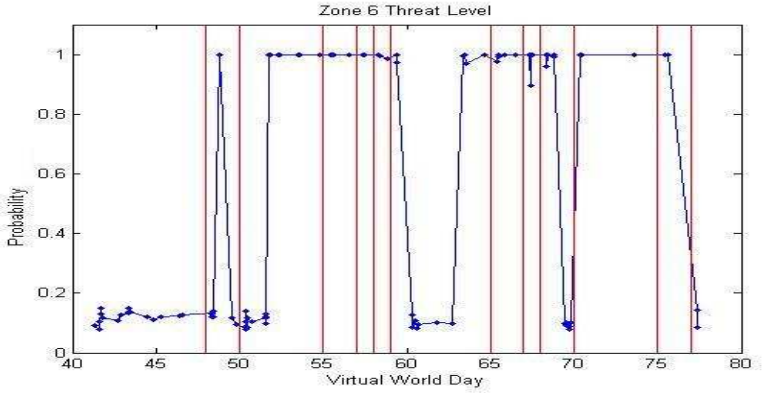

Figure 9: Predicted threat levels for the zone 6.

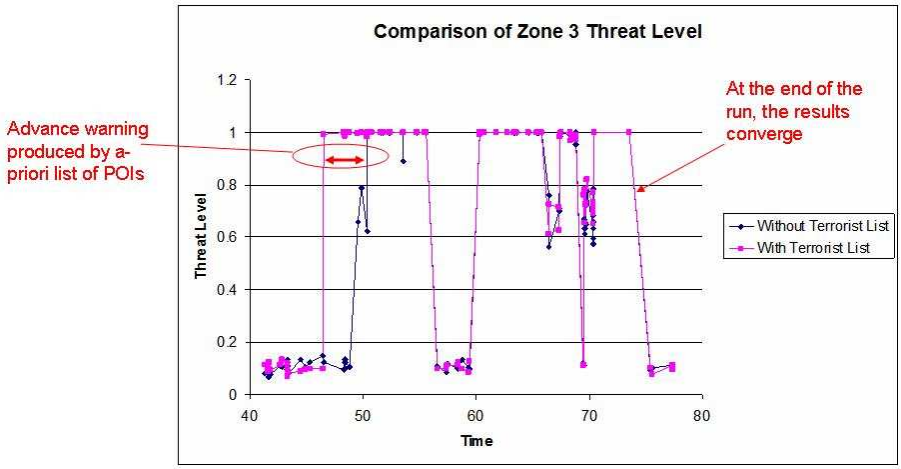

Figure 10: An example of the impact of *a-priori* information to the MLN.

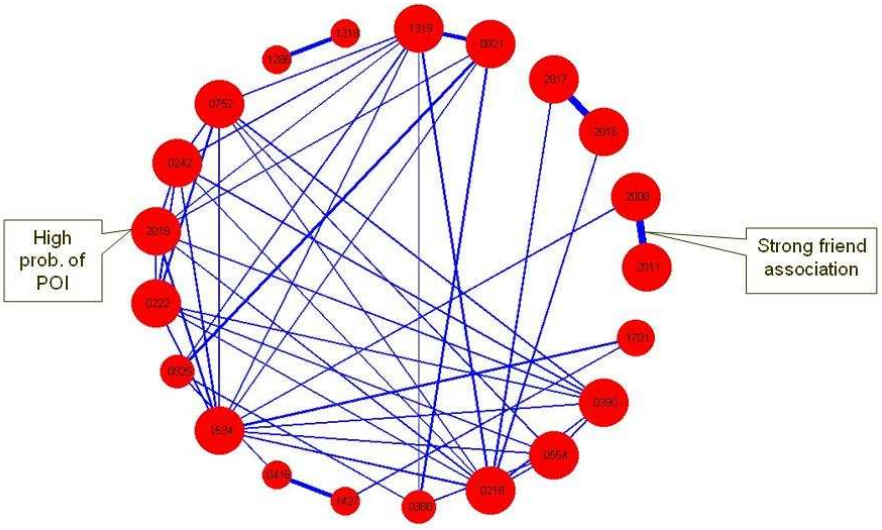

Figure 11: An example social network extracted from MLN results. Red circle size represents how likely a person is a POI, and line width indicates the probability two people are friends. Links weighted less than 50% have been removed.

is on the first aspect: creating richer representations of the domain via interaction with AMAs and other nodes in the network.

Another aspect of the work that needs to be explored further is the integration of security. This also is relevant in a number of ways. While AMF provides a fairly safe and constrained environment for the participating hosts, it is still essential that they be protected from any malicious or uncontrolled behavior on the part of the AMAs or the remainder of the network. The framework must also support authentication, to protect against the introduction of rogue agents, and encryption, to protect information being carried/represented by AMAs.

A related point is the implementation of proxy features which enhance overall security (rather than leveraging it). For example, it may be desirable to make the services of a sensitive resource available within a specific environment. As such, it may be preferable to distribute as minimal a description of that resource as possible. An AMA could be designed to interact with its home resource to field queries about its capabilities, rather than carrying complete information with it, in this way minimizing the amount of information placed on the wire.

Other uses/extensions have been envisioned. It is conceivable that AMAs could be distributed which represent potential resources. In this way, proxies could be used in conjunction with resource servers to invoke the instantiation of resources not currently available within the current network environment.

While this framework provides considerable advantages in a network compromised environment, there are certain advantages that can be gained from a pull-based approach when network availability/reliability is high. In order to reap the maximum benefit, we would like to explore ways in which this approach can be seamlessly integrated with existing pull-based frameworks in order to provide a comprehensive and optimal solution.

The AMF simulation framework will provide a rich simulation environment where we will test these ideas. Future improvements on the simulation framework will allow for a wide range of testing. We would like to more fully develop the simulation components used to derive, represent, and use information about the local and global environment. This component transfers easily into fielded implementations, and is a major focus of this effort; advances here benefit both the simulation framework, and the AMF project in general. A significant portion of this work involves decisions regarding which information should be stored, and how it should be represented. Also important are the means by which this information is collected. Similarly, this relates to protocols which this framework is designed to help explore, but also make for a richer simulation environment. A necessary element in the simulation is the expression of information to be recorded. That is, if nodes are collecting information about the presence of, and activity at, other nodes, the simulation must support the creation and collection of such information at a simulated node level.

References

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

[2] J. Barton and V. Vijayaraghavan. Ubiwise: A ubiquitous wireless infrastructure simulation environment, 2002. note: tech. report HPL-2002-303, HP Labs.

[3] P. Busetta, P. Bouquet, G. Adami, M. Bonifacio, F. Palmieri, G. Moro, C. Sartori, and M. P. Singh. *K-Trek: a peer-to-peer approach to distribute knowledge in large environments*. Agents and Peer-to-Peer Computing. Second International Workshop, AP2PC 2003. Revised and Invited Papers (Lecture Notes in Artificial Intelligence

Vol.2872). Springer-Verlag; UNI.TU.RIM S.p.A., Fondazione Cassa di Risparmio di Rimini, Germany; Berlin, 20040101.

[4] T. Clausen and P. Jacquet. Optimized link state routing protocol (olsr), oct 2003. howpublished: RFC 3626 (Experimental).

[5] S. Corson and J. Macker. Rfc 2501: Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations, jan 1999.

[6] A. Hijazi and N. Nasser. Using mobile agents for intrusion detection in wireless ad hoc networks. In *Wireless and Optical Communications Networks, 2005. WOCN 2005. Second IFIP International Conference on*, pages 362–366, 6-8 March 2005.

[7] M. Mamei, F. Zambonelli, G. Moro, C. Sartori, and M. P. Singh. *Location-based and content-based information access in mobile peer-to-peer computing: the TOTA approach*. Agents and Peer-to-Peer Computing. Second International Workshop, AP2PC 2003. Revised and Invited Papers (Lecture Notes in Artificial Intelligence Vol.2872). Springer-Verlag; UNI.TU.RIM S.p.A., Fondazione Cassa di Risparmio di Rimini, Germany; Berlin, 20040101.

[8] S. S. Manvi and V. Telsang. An agent based approach to qos routing in mobile ad-hoc networks. In *Signal Processing and Communications, 2004. SPCOM '04. 2004 International Conference on*, pages 86–90, 11-14 Dec. 2004.

[9] G. Moro, C. Sartori, and M. P. Singh. *Agents and Peer-to-Peer Computing. Second International Workshop, AP2PC 2003. Revised and Invited Papers (Lecture Notes in Artificial Intelligence Vol.2872)*. Springer-Verlag; UNI.TU.RIM S.p.A., Fondazione Cassa di Risparmio di Rimini, Germany; Berlin, 20040101.

[10] Eleanor O'Neill, Martin Klepal, David Lewis, Tony O'Donnell, Declan O'Sullivan, and Dirk Pesch. A testbed for evaluating human interaction with ubiquitous computing environments. In *TRIDENTCOM '05: Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities (TRIDENTCOM'05)*, pages 60–69, Washington, DC, USA, 2005. IEEE Computer Society.

[11] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (aodv) routing, jul 2003. howpublished: RFC 3561 (Experimental).

[12] M. Peysakhov, D. Artz, E. Sultanik, and W. Regli. Network awareness for mobile agents on ad hoc networks. In *Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. Proceedings of the Third International Joint Conference on*, pages 368–375, 2004.

[13] Vinny Reynolds, Vinny Cahill, and Aline Senart. Requirements for an ubiquitous computing simulation and emulation environment. In *First International Conference on Integrated Internet Ad hoc and Sensor Networks (InterSense 2006)*. OCP Science, may 2006. note: invited paper.

[14] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 0201.

[15] R. RoyChoudhury, K. Paul, and S. Bandyopadhyay. An agent-based protocol to support multimedia communication in ad hoc wireless networks. In *Parallel and Distributed Processing Symposium., Proceedings 15th International*, pages 2026–2033, 23-27 April 2001.

[16] A. Smithson, L. Moreau, G. Moro, and M. Koubarakis. *Engineering an agent-based peer-to-peer resource discovery system*. Agents and Peer-to-Peer Computing. First International Workshop, AP2PC 2002. Revised and Invited Papers (Lecture Notes in Artificial Intelligence Vol.2530). Springer-Verlag, Germany; Berlin, 20030101.

[17] K. Tei, N. Yoshioka, Y. Fukazawa, and S. Honiden. Geographically bound mobile agent in manet. In *Mobile and Ubiquitous Systems: Networking and Services, 2005. MobiQuitous 2005. The Second Annual International Conference on*, pages 516–518, 17-21 July 2005.

[18] Do Van Thanh and I. Jorstad. A service-oriented architecture framework for mobile services. In *Telecommunications, 2005. Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/ E-Learning on Telecommunications Workshop. AICT/SAPIR/ELETE 2005. Proceedings*, pages 65–70, 17-20 July 2005.

Appendix A: List of Acronyms

OODA observe-orient-decide-act

C2 Command and Control

AMF Active Metadata Framework

MLN Markov Logic Network

HUMINT Human Intelligence

UAV Unmanned Aerial Vehicle

UDP User Datagram Protocol

AMA Active Metadata Agent

APL The Johns Hopkins University Applied Physics Laboratory

GPS Global Positioning System

GUI Graphical User Interface

POIs Persons of Interest

POI Person of Interest

SOA Service Oriented Architecture

MANET Mobile Ad-Hoc Network