MARE: Resource Discovery and Configuration in Ad Hoc Networks

Matt Storey, Gordon Blair and Adrian Friday
Department of Computing
Lancaster University
Lancaster
U.K.

Telephone +44 (0)1524 594532
Fax +44 (0)1524 593608
E-mail: {storeym, gordon, adrian}@comp.lancs.ac.uk

Abstract

The emergence of personal portable devices, such as PDA's and Mobile phones, with considerable processing and communication capabilities, has led to a desire to use various combinations of these devices together to achieve new and as yet unrealised operations. Not only are mobile devices expected to offer 'conventional' facilities like email and web browsing but also more demanding multimedia applications. Attaining these operations within a fixed network environment with high-power workstations is non-trivial; however highly dynamic ad hoc environments further complicate this scenario. In particular, a means of discovering available devices and enabling manipulation of them within a highly active environment is required. In this paper we present a novel architecture called MARE that facilitates the detection and manipulation of resources in ad hoc environments.

Keywords: ad hoc, resource discovery, mobile agents, tuple space

1 Introduction

The emergence and development of technologies such as IrDA, wireless modems, TETRA [1], WaveLAN [2] and Bluetooth [3] is leading to a much wider acceptance and use of wireless networking technologies. In particular, the promise of Bluetooth to provide small form-factor, low-power communications has led to an increased interest in devices equipped with wireless communications facilities. Using such technology, devices can exchange information and share resources forming an *active environment* where not only physical structure but also resource availability can change.

In the near future it is also foreseeable that most computer peripherals will be wireless, utilising short-range communications technologies; the necessity to connect keyboards, mice, printers, cameras, etc., physically to static desktop computers is disappearing. In addition it is foreseeable that many devices that currently are not '*smart*' will soon incorporate microprocessors with communications facilities. Such devices may include refrigerators and cupboards that can monitor levels of household supplies or vehicles that monitor and report their own performance (already a reality in some high-end automobiles). The potential explosion in numbers of devices will require careful consideration of how interaction between them is to take place, such that these devices may interact to achieve a common task - triggering a kettle located in the kitchen to respond to a wake-up signal from an alarm clock in the bedroom, for example.

This paper presents our work in developing a system to support distributed operations in a highly active ad hoc communication capable environment. Such environments are characterised by a highly dynamic ad hoc grouping of devices, changed by the unpredictable availability of computational devices as they arrive and depart from the grouping. Our architecture facilitates such operations by providing efficient mechanisms for the discovery and configuration of resources.

The remainder of this paper is structured as follows. Section 2 presents an illustrative scenario based on a "Mountain Rescue", in which Emergency Service units converge to form an ad hoc grouping of resources, enhanced by our platform technology. From this scenario a set of requirements are formed before presenting an examination of existing discovery mechanisms in section 3. Alternative approaches are discussed outlining the MARE approach in section 4. MARE is then examined in depth in section 5 before presenting an example of MARE in action in section 6. Section 7 provides an analysis of the mare approach to other related systems before offering concluding remarks in section 8.

2 A closer examination of mobile environments

2.1 Example: A mountain rescue

The scenario presented in this section is based on our work in developing an experimental application based on a scenario of a mountain rescue. This work is part of an ongoing initiative investigating mobile support for mountain rescues in the Langdales mountain range, in the Lake District [4]. This is an excellent test case for ad hoc networking in general, as will become clear below.

A typical mountain rescue will start with a report of an injured climber to the Mountain Rescue Service. The rescue service coordinate each rescue from their headquarters at the mountain rescue base, which is wirelessly linked to the away team using TETRA digital private mobile radio equipment. Information and advice is relayed to the rescuers who proceed to search for the stricken climber. As the rescue workers converge upon the climber the base can be kept informed of progress through the TETRA handsets.

Upon arrival at the scene, first aid can be administered before a call for an airlift or drafting in more medically qualified personnel. Typical equipment carried by rescuers will include medical kits, cameras and GPS compasses to pinpoint their location. As some of the equipment is heavy and bulky different people carry it helping to distribute the weight. Team members will have different roles dictated by training, experience and the equipment they carry. For instance, fell runners are sent on ahead with lighter equipment to administer first aid, whereas team members with the heavier equipment arrive later on the scene once the climber has been located. As more rescuers converge at a rescue site more equipment becomes available making the rescue easier and quicker.

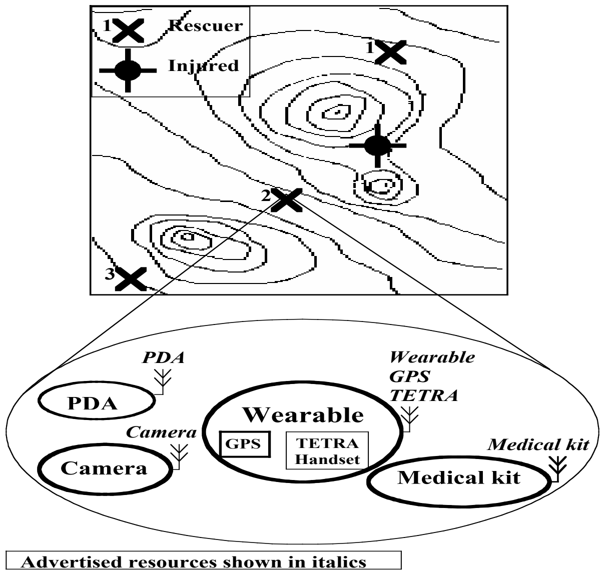

Figure 1 Map showing a rescue in progress with rescuers equipment

As a rescue will typically start from several locations, as depicted in figure 1, the ability to detect the presence of other rescuers and the devices they carry will enable the creation of a working group of resources. With the introduction of wireless communication facilities, an individual rescuer can detect and manipulate another's resources without physical contact, thus saving time and simplifying the equipment usage.

2.2 Experimental System

In the experiment the equipment carried by the mountain rescuer consists of a small pack containing a small wearable computer that acts as a hub, connecting a PDA, a TETRA handset and a number of ancillary devices. These devices may include digital cameras and medical monitoring equipment (e.g. a Propaq unit that is capable of measuring heart rate, blood pressure and oxygen content). As mentioned previously the physically demanding nature of trekking long distances and climbing it is not possible to carry more than a few devices at any given time. Hence devices are shared between rescuers to distribute the weight. The small wearable computer is designed to remain running using a minimal amount of power whilst the PDA is turned on and off as desired to preserve power. Other devices are connected as required. It is possible to see the variety of combinations that may occur by different hardware devices being available at any given time. An example of rescuers equipment can be seen in figure 1.

Applying a wireless communications technology to facilitate communication between devices rather than a physical connection opens up the opportunity to let individual devices interact and form ad hoc collaborations with one another without external interaction. Furthermore, such technologies also allow the possibility of other resources not necessarily physically seen in the environment to be used, for instance, resources carried by the climber being rescued. A rescuer will get notification of these resources and may manipulate them to aid in the rescue.

The current Bluetooth API does not readily lend itself to ad hoc networking, although the continued evolution of Bluetooth and its integration into next-generation mobile end-systems will presumably make Bluetooth the technology of choice for later experimentation. In our prototype system we have used readily available Orinoco 802.11b wireless networking equipment. The Orinoco equipment provides a much larger range and higher bandwidth, although it is clearly physically larger and requires more power than Bluetooth.

2.3 Scenario Analysis

When attempting to carry out a task it is important to have a consistent view of the resources available in order to calculate the best place for an operation to take place. Using a fixed environment such as a wired LAN the resources available such as printers and file stores are relatively static and can be assumed to be available for long periods of time. As environments become more changeable, such as through the introduction of wireless LANs, the availability of any given resource becomes harder to guarantee. More specifically, the resources offered by a device disappear as it is moved out of range or communications fail. Equally devices or new resources may arrive or be created at any time.

As outlined in the introduction, in such networks where it is anticipated that many resources have wireless communication facilities, the formation of ad hoc networks can be anticipated. Such ad hoc networks can be viewed as groupings of hosts wishing to interact with one another. Traditional networked applications tend to assume the availability of servers for specific tasks, such as resolving hostnames to IP addresses (DNS lookups). This same approach is clearly not viable in an active environment, since the availability of a particular host's services and resources can no longer be guaranteed.

The ad hoc wireless communications infrastructures underpinning active environments also suffer from the limitations of mobile networks - namely, intermittent connectivity, increased error rates, reduced bandwidth and an increase in traffic latency. Reducing the load that is placed on the communication infrastructure is thus of paramount importance; a means of discovery of resources utilising the minimum of communication overhead is sought along with suggested operations that can preserve network bandwidth.

Viewing a mobile device as being a PDA or wearable computer brings in other important considerations. These devices are typically resource poor when compared with desktop machines. For instance, such devices will typically have limited battery life and less powerful processors, smaller storage capacities and display facilities. The ability for such devices to share their resources can aid in making these differences less obvious by opportunistically utilising the resources in the surrounding network to maximum potential.

In summary, a platform is required that addresses the following requirements:

- Consistent view of the resources within an ad hoc environment
- No reliance upon the availability of a single host for networked services
- Minimise traffic on the network due to resource discovery
- Aid efficient use of resources – device / processing power, storage, screen size

In the next section we consider existing approaches for the discovery of services examining their suitability for operation in ad hoc networks.

3 Existing Service Discovery Protocols

3.1 Service Location Protocol

SLP is a protocol designed to aid the discovery of resources on a network. SLP [5] assigns a user agent to each client to act on its behalf. This agent then interacts with a directory agent that contains lists of service agents to gain interaction with a service. If the directory agent is unavailable, the user agent can multicast to the service specific multicast address to gain a response from a service. Services have a lifetime such that a directory agent can remove the service from its listing upon its expiry.

Within an active environment the user agent will issue requests to the service specific multicast address. A request for a specific service can lead to all services responding to the request (leading to acknowledgement implosion). Each type of service subscribes to a separate multicast port. This approach requires a potentially large number of multicast ports as many service types become available. If a general port was used for a query request and response, then clients requiring common services could reduce network traffic by snooping the response generated to a given.

3.2 Universal Plug and Play

UPnP [6] is a Microsoft technology built upon plug and play technology found in windows 9x family of operating systems. The principle is extended to encompass services as well as devices. UpnP is aimed at *zero configuration* of attached services or peripherals.

UpnP consists of a suite of sub-protocols for service discovery, interaction and event notification. Simple Service Discovery Protocol (SSDP) [7] is used for the discovery of resources. A service announces its presence upon creation and clients can request (search) for services when required. Both of these operations are carried out using IP multicast. The response to a client's multicast search is sent by unicast back to the client from all services of the correct type. A service can also multicast a state change or the removal of the service from a system.

Applying this approach to a highly active environment, the transmission of information from a service will only be carried out once, typically upon instantiation of a service. The instantiation of a service is likely to take place in isolation such that no clients hear the announcement (the notification event will be lost). Within an active environment every time a user wants to get an up to date view of the surrounding resources it will have to multicast a request for a service it wants. This will generate a large amount of requests to be answered singularly by each service.

Furthermore a user may not know what services interest the user and therefore may not know about new resources of potential interest that are available.

3.3 Jini

Jini from Sun Microsystems [8] provides a system for a client to locate services. For example a client may join a Jini lookup service and request information about a printer. Jini makes a key assumption about the environment it is operating within, i.e. it assumes a low latency network. Jini uses a server structure to hold and disperse information about services. Jini can also have more than one server to advertise services working together as a single service. If a client cannot find a lookup service it can resort to talking to service providers directly.
Clearly the active environment will not currently offer a low latency network as assumed by Jini. Running a copy of Jini on each device wishing to participate in a grouping makes the system distributed. However Jini is a larger system than some devices can currently execute. The size and complexity of Jini, although useful in larger systems, can currently be seen as a restriction upon smaller devices. Furthermore, the fluctuating availability of lookup services would require frequent switching from client to server based service location.

3.4 Analysis

The previously examined approaches to service discovery are not specifically targeted at ad hoc networks offering less than optimal solutions for such an environment. Further analysis of these approaches is undertaken in section 7. The following section examines novel approaches to discovery and configuration drawing a list of core requirements of such a system for operation within ad hoc networks.

4 Alternative Approaches

4.1 Mobile Distributed Systems Platforms

It is well known that existing distributed systems platforms such as CORBA [9] and DCOM [10] do not offer communication semantics that are well suited to mobile environments. A number of researchers have attempted to improve such platforms, generally by introducing an element of asynchronicity into the designs of the underlying communications mechanisms (e.g. delayed or queued RPC mechanisms). Examples of such platforms include Mobile DCE [11], MOST [12], Odyssey [13] and the Rover toolkit [14]. Such platforms however only partially address the requirements discussed above, in effect, only really dealing with problems relating directly to weak connection or disconnection to the network.

A more radical attempt to providing support for mobility is through the provision of a different, non-remote procedure call based, programming abstraction centred on the concept of a shared tuple space [15]. The tuple space paradigm provides an asynchronous programming API, offering undirected, anonymous and, time and space decoupled communication. We consider the benefits of this approach in more detail in the following section.

Researchers have also postulated that the problems introduced by mobility can be addressed by utilising agent technology to allow computation to be moved to the (resource-rich) fixed portion of the network. Again, agent technology can help to address these problems, but does not offer a complete solution. We consider this approach in more detail in section 4.3.

4.2 Tuple Spaces

A tuple space can be thought of as a pool of shared information (akin to a distributed shared memory architecture). This pool can have information added, read or removed from it atomically. Tuple spaces were developed initially to address the issues of process and data distribution in massively parallel computer architectures [15]. As the paradigm has been adapted for more uses including aiding work in a mobile environment, it has acquired more functionality and specialisation [16][17]. Since communication between processes is only achieved through the generation of typed data structures (called *tuples*) that are inserted and removed from the tuple space, communication is mediated, leading to properties of anonymity and time and space decoupling between processes. More specifically, processes do not need to be able to communicate directly to be able to exchange data, providing they have access to the shared space. Moreover, since tuples exist persistently within the tuple space, beyond the lifetime of the processes that created them; processes do not need to co-exist simultaneously at the same instant in time in order to exchange data. It is these properties that have inspired researchers to apply the tuple space paradigm to mobile environments [17].

Data is exchanged in the shared tuple space *generatively* [18] through the generation and revocation of the tuples themselves. The tuple consists of an arbitrary sequence of typed data fields; each field may be either actual or formal (all fields are typed, but actuals have a specific value, whereas formals act as pattern matching parameters). Tuples are of two basic forms, tuples and anti-tuples; anti-tuples match a tuple with corresponding formal and actual parameters (assigning instance values to the corresponding formal parameters, as appropriate). The base set of operations upon a tuple space consist of in, out, rd and eval. These operations are used for acquiring, adding, reading and evaluating a tuple placed into the tuple space respectfully (eval is distinct from the other operations in that it can take '*active*' tuples containing code for execution; the active tuple is evaluated in the tuple space, resulting in the generation of a passive tuple containing the results of the active calculation). Many extensions have been added including support for multiple tuple spaces [20] and bulk transfer primitives [16] where more than one tuple at a time can be read or withdrawn from a tuple space (addressing the multiple-rd problem).

Several platforms are available that implement the tuple space paradigm for example JavaSpaces, T-Spaces and L^2imbo.

- **JavaSpaces** is the Sun Microsystems tuple space implementation built using Jini [8] (see section 3.3). JavaSpaces has all the features of Jini as it is developed using the Jini system.
- A **T-Space** [19] is an implementation of a tuple space offered by IBM. It incorporates features from the database community to provide a data management layer with a richer set of tuple matching operations. Such techniques include matching by tuple field name or data contained therein to gain a list of tuples not necessarily of the same structure. In the current version, T-Spaces uses centralised servers making it unsuitable for working in an active environment.
- The **L^2imbo** platform [21] developed at Lancaster University is a fully decentralised tuple space implementation specialised for mobile environments. It offers unique withdrawal of a tuple, multiple specialised tuple spaces (including a local space for the sharing of information on the same host and callbacks on resource arrival and departure). The L^2imbo API offers may of the facilities provided by other tuple space implementations, such as the base Linda primitives, non-blocking Linda operations, bulk primitives, multiple tuple spaces and callbacks. An L^2imbo tuple space can also be used to monitor and control the action taken on tuples by utilising bridging of tuples in and out of tuple spaces and the eval operation providing a means of code execution within a tuple space to manipulate its contents. Such an operation can be utilised to control forwarding of agents through a bridge by buffering or transforming tuples.

4.3 Mobile Agents

A mobile agent can be seen as a piece of code and state information that is capable of movement to a host and execution at that host. The agent may travel to more than one host in order to complete its task. Mobile agents are commonly used for completing tasks near to the source of data to save large data transfers through a network. An example of this can be seen in searching a sizeable data source, where it is clearly preferable to do data centric operations as close (in network terms) to the data as possible to avoid unnecessary network utilisation. There has been a substantial quantity of work into the use of agents for use in mobile environments [22] [23] [24] [25]. These systems use agents written in portable code (typically Java and Tcl) to execute the agents on heterogeneous host platforms. For larger tasks, more than one agent may be used collaboratively, leading to a requirement for the agents to be able to communicate with one another.

Considering the characteristics of an ad hoc network there are several architectures to aid agent execution. Approaches requiring a central server for the downloading of additional components for execution of an agent are unsuitable due to the fluctuating availability of a host. Such approaches include Java-To-Go [26], Agent Tcl [24] and Mole [22] where pieces required by an agent are downloaded upon execution from a central server. Fallback strategies, such as downloading classes from the source of the agent can be imposed, but this suffers from the possibility of the named source host becoming unavailable. Reliance on a single host as a meeting point for communication can be seen in TACOMA [23] and ARA [27]. Concordia [25] relies upon an itinerary that directs the agent to the next point of execution, as the availability of a host cannot be guaranteed, the agent may be stuck awaiting the transport to an unavailable host when another host could execute the agent instead.

Inter-agent communication may also be required. The option offered by Mole is a unique identifier in the form of a badge that allows the wearer to receive messages directed at that badge id. Concordia and Aglets [28] make use of proxies for the buffering of communication, to aid communication error recovery although this requires the execution of the proxy on a host causing reliance.

LIME [29] is an architecture that is based upon the use of Linda [15]; it does not suffer from directed communication weaknesses through the use of a tuple space. However with the introduction of directed semantics into the Linda API means that an operation where data is shared between multiple hosts becomes difficult (negating many of the stated benefits of the tuple space approach). An agent being directed to another tuple space will carry all the valid information directed at that agent or as yet undelivered information from it when it moves. The carrying of information will generate large agents potentially holding key information required by other agents. The alternative is to share the data and move a smaller agent, allowing work to continue with the associated data by other agents.

4.4 Analysis

By applying both mobile agents and the tuple space paradigm together, we believe the resulting platform offers several key features towards attaining the requirements already stated in section 2.

- The ability to distribute resource information to all listening hosts allowing acquisition of information about known resources upon a request.
- Avoidance of reliance upon any single host when carrying out operations such as moving an agent. For example the movement of key components with an agent such as code modules removes the need for a code server. Agents communicating can use the distributed tuple space to gain anonymous communications such that one agent can communicate with another despite its movement between hosts.
- The use of a tuple space allowing for less traffic caused by query requests and responses. It can be anticipated that with a users demand for a consistent view of a changing environment they will make update requests regularly. Hosts will already have information about the local environment contained within the tuple space so the request will not need to propagate beyond the local host.
- Distributing agents so that each node can decide whether to execute the agent rather than having the agent thrust upon it. This enables a weak host to refuse the agent to let a more capable host execute it.

These benefits will offer developers of applications for small low power devices operating in active environments the potential to utilise more resources such as processing power, communications and storage located in the surrounding environment.

We thus present an architecture that advocates the use of agent technology, supplemented by a communications model based on the tuple space paradigm [15]. It is our belief that this combination includes the appropriate properties to address the challenges of our active mobile environment. The MARE approach utilises the L^2imbo fully distributed tuple space implementation, however an agent system has been developed to address the weaknesses highlighted previously in existing approaches, we describe our approach in detail in section 5.

5 Mobile Agent Runtime Environment

5.1 Overview

The MARE system is aimed at supporting operations within a mobile ad hoc environment, as mentioned previously exploits both agent technology and tuple spaces. More specifically, MARE uses agents to reduce bandwidth requirements by moving operations to the source or point of the operation required rather than transmitting information across a network. This approach also allows for the coordinated manipulation of several resources to best carry out an operation, for example, using a data source to gain raw information and a high powered node to process this information into a more meaningful format before returning it to a user. The tuple space then provides time and space decoupling allowing anonymous communication within a network. The removal of host targeted communication allows an agent to roam freely and still gain messages targeted for it by listening for messages of interest to the agent. Using a distributed tuple space implementation removes the reliance upon a central host that as stated earlier is highly desirable in an ad hoc environment. Essentially, MARE uses tuple spaces to provide a communication medium for transmitting resource information, messages and agents.
The MARE architecture also offers the concept of a *resource*. A MARE resource can be deemed as any advertised service that could be interacted with, such as the a physical device (e.g. a camera) or the data held in a file store. The advertisement and interaction with resources will be covered in more depth in section 5.3.

5.2 System Architecture

The overall MARE architecture is shown in figure 2 with the system being implemented in Java. We look at each component in turn.

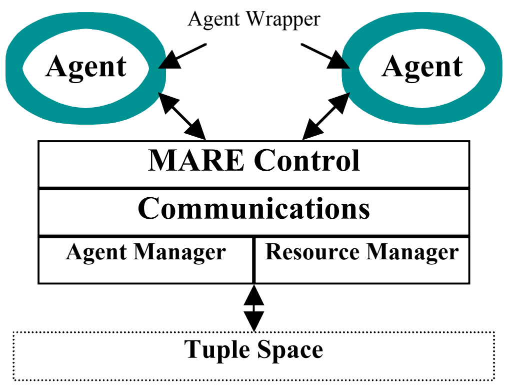

Figure 2 The Mare Architecture

- The **tuple space** is used for passing configuration data within the system containing resource descriptions, agents and messages. Directed bulk communications such as a multimedia streams are passed between agents out-of-band (tuple spaces have been shown to be inappropriate for such applications [17]). Multiple threads can access the tuple space stub on a machine at any time.
- In MARE, resources and agents are handled by dedicated manager components. **Resource managers** receive and transmit information regarding resources, both to the agents that run upon the MARE instance and to other resource managers. More specifically, the manager transmits a beacon from the MARE instance periodically. This beacon contains all the resources residing upon the MARE instance. If more than one resource is present on the MARE instance, the resource definitions are grouped into a *bundle* to reduce the number of network connections and bandwidth required to advertise the resources.
- The **agent manager** constantly listens for agents entering the tuple space. Upon the detection of an agent, its requirements are checked against the constraints of the MARE instance. If the agent requirements are acceptable, the manager acquires it and passes it to the execution environment. The agent manager can be set to not receive agents when carrying out insertion of resources, agents or shutting down. The agent manager also handles the forwarding of agent messages between instances of the execution environment.

- Acting as an interface between the MARE control above and the communication managers below is the **communication** layer. This is specifically aimed at allowing different communications architectures to be used or integrated beside the tuple space communication managers. Agents and resources are passed through this layer for dispersal through the control layer.
- The **MARE control** layer examines agent requirements including the resources that the instance of MARE has, evaluating if an agent can be executed. The control layer also delivers messages to the correct agents executing in the MARE instance. The control of the migration of agents is carried out at this point in the case of an event such as a system shutdown.
- The **agent wrapper** is used to control the agents' operations and act as an interface to the MARE control. An instance of this wrapper is passed to the agent upon being executed. This wrapper controls registering for receipt of messages such that more than one agent can receive the same message by addressing it to a shared identifier.
- An **agent** implements a set of methods to allow the restarting and movement of the agent. In addition, the agent implements the Java serialize interface to allow the agent to be moved. Allowing agents to have access to system libraries provides the agent programmer with freedom as to the operations of the agent restricted by the agent's permissions. Agents may be grouped to carry out operations such as the filtering of acquired information and then the transformation of the acquired information into a format better understood by a consuming application or agent. In addition this information can be cached at appropriate points within a network. Agents can migrate to valid hosts as they seek resources or are on a host incapable of supporting their needs such as in the event of a host being shut down.

Every host wishing to participate in the environment executes a copy of MARE. The instance may be specified as accepting or deaf to incoming agents. Resources are detected through advertisement by the MARE instance the resource resides upon.

5.3 Implementation Details

5.3.1 Overview

As previously mentioned, MARE is implemented in Java providing a portable byte code supported by many platforms. Java offers the ability to dynamically insert classes into a class structure aiding for maintenance and development of new applications. L^{2}imbo interaction is currently done through a Java API interacting with an ANSI C stub. Although MARE is implemented in Java, a resource or agent may be implemented in a separate language providing the target host can execute the agent; in this case a Java proxy can be used to control interaction. An agent may be formed of multiple Java classes or modules that are transported with the agent to reduce reliance upon a specific host for the retrieval of sub-components. With the development of Java for embedded, personal and smartcard systems, Java is becoming available to small footprint end-systems such as PDA's and mobile phones albeit currently with reduced functionality.

Each agent or resource is given a *unique identifier* to distinguish them and address them individually. Unique identifiers can be attained from the system allowing the assignment of more than one unique identifier to each agent. This enables a group to communicate by addressing data to a shared identifier.

The MARE system can request and, if required, force an agent to migrate. The MARE instance is thus able to remove undesirable or broken agents.

5.3.2 Resources

Resources can be generated using a call to the MARE instance or from an executing agent, the resource is then advertised using the underlying tuple space. Each resource is described as a set of predefined and user-defined key-value pairs. A resource is advertised using the instance of MARE in the format shown in figure 3.

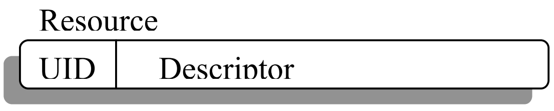

Figure 3 Resource structure

The descriptor is a series of key-value pairs; the keys are separated by a semicolon and passed as a string.

```
Descriptor    -> Pair | Pair;Descriptor
Pair          -> Key = value
Key           -> DESCRIPTION | TYPE | IP | PORT | user-defined
```

If more than one resource resides upon any instance of MARE the resources are gathered together to form a resource bundle. The aim of bundling the resources together is to reduce the number of separate network transmissions and bandwidth required to advertise the resources. Upon the transmission of a resource or *resource bundle* the previous tuple describing the resource is removed, maintaining consistency within the tuple space and ensuring data is available at all times for listening hosts (unlike the transient service announcements in service discovery protocols such as SSDP [7], the resource descriptor remains persistently within the tuple space until the next refresh interval – hosts are thus able to discover resources immediately upon joining the network without having to wait for the next round of advertisements).

Each resource in the resource manager has an associated *lease time* based on time of arrival of a resource. The lease time is set to be slightly longer than the expected next beacon of information. This allows for the beacon to refresh the timer. If a beacon is not received during this period the resource will timeout and be removed from the active list passing a notification to agents running in the MARE instance. The resource manager then notifies the MARE control of the change in resource availability and this will in turn alert interested agents.

A resource is deemed to be potentially available when notification is received from MARE. Ultimately a programmer must allow for a resource being reported as present by the system and being unavailable at the time of use. Such availability issues are due to the nature of the changing active environment; a system cannot guarantee the availability of a resource within such an environment without keeping an open channel to the resource and getting immediate notification of communication failure. This would consume valuable resources, which is clearly undesirable.

5.3.3 Agents

Agents inherit from a Java class that defines key methods (see appendix A). Agents are placed within the tuple space and every instance of MARE attempts to acquire the agent. The agent runtime assesses its ability to execute the agent by the list of requirements carried by the agent. If the agent is appropriate the host will attempt to acquire the agent. Once an agent is accepted, it is registered with the system so as to allow for the receipt of directed information as defined by its unique identifiers. The agent may also register for a call-back upon the arrival or departure of resources or QoS changes.

Agent tuples consist of a UID and a resources required string with a serialised agent containing a Java instance and classes required for the instantiation of the class as shown in figure 4.

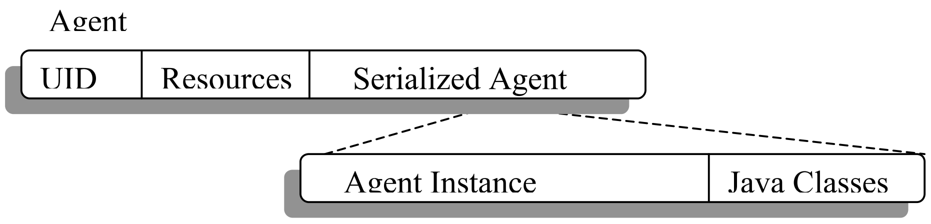

Figure 4 Agent Format

The agent's resource dependencies are expressed as a set of UIDs corresponding to the resources or agents it requires. This field may also be left blank if the agent does not require any specific resources.

Communication tuples consist of UID and data fields, the UID identifies the recipient or recipients the message is directed at as shown in figure 5.

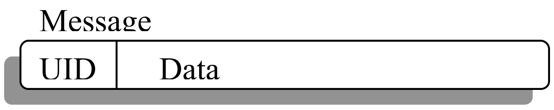

Figure 5 Message Format

The following example is of a small agent that prints out its status on the host it is running on.

```
import MARE.*;
class CollectionAgent implements MARE.Agent, java.io.Serializable {
  private AgentRuntime  agentEnvironment = null;

  public void agentEnvironment(AgentRuntime agentRuntime) {
        agentEnvironment = agentRuntime;
  }
  public void close() {
    System.err.println("Request to go");
  }
  public void receive(String destination, byte data[]) {
    System.err.println("data arrived");
  }
  public void run() {
    System.err.println("I have arrived");
  }
}
```

The design of the message format of MARE allows a dynamic approach to resource description, data and unique identity of resources and agents.

6 Mare in action

In this section we revisit the scenario from section 2.1, highlighting the contribution of MARE. The discovery and use of agents can be demonstrated by considering the convergence of rescuers on a given point as shown in figure 1. As the first rescuer arrives at the scene, they may only have limited equipment. As stated earlier, such equipment may consist of a small wearable computer with PDA, communications facilities for short range using a wireless technology such as Bluetooth and a TETRA handset for low bandwidth long distance communication. As other rescuers come into range of the first rescuer the collection agent of the first rescuer will detect the arrival of new resources. Notification will then be provided to the first rescuer with information about the resources that are now available. Without cabling, the first rescuer can make use of resources carried by the second rescuer such as a head mounted video camera to examine the area further. Further agents may be used to transform or filter the data into a more suitable representation for example video with audio to just audio. Through the course of the rescue, as rescuers arrive and depart, they will implicitly offer a dynamic set of facilities to the other rescuers. Furthermore, as they change location, resources embedded in their environment will become available. For example, it is advantageous for the rescuers to make use of a stationary-wired communication point, the high power transmitter in the rescue vehicle or a positional marker. It is also possible to communicate the current status of the rescuers or injured climbers at this point. For this use an agent would be dispatched that can place itself upon a fixed point or travel towards a base transmitting information on behalf of the generator for example images, medical information or expected arrival time at hospital.

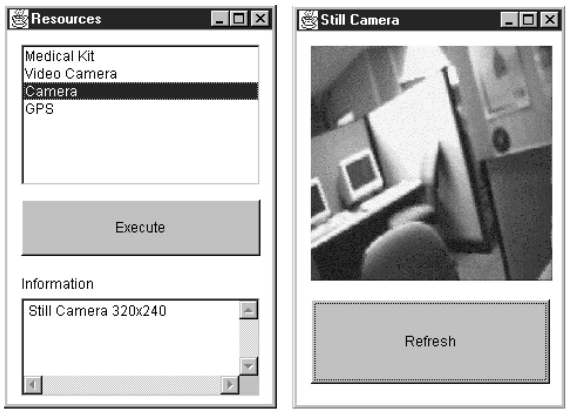

Figure 6 Agent views

Examining the software used in more detail in the previously outlined scenario shows the use of agents and the discovery of resources. Each rescuer creates a collection agent on their PDA to collect tailored information for the user about available resources (see figure 6). The agent can be tuned with preference to use specific resources to reduce resource discovery traffic transmitted to the PDA. The collection agent will act as the rescuer '*view*' on the available resources. When the PDA powers down, the agent will be migrated to a host with available resources, typically to the nearest host. If the agent migrates too far, the agent is assumed to be lost and times out and a new instance of the agent is generated from the persistent copy in the tuple space.

7 Analysis of approach

To gain an impression of the viability of the MARE approach, we compare our architecture's performance against key service discovery technologies. In more detail, we have examined SSDP, the resource discovery mechanism of UPnP and SLP, compared them against the discovery mechanism employed in MARE when operating within an ad hoc environment. We examine the number of messages passed between hosts whilst performing a discovery of surrounding services.

7.1 Test Scenario

Four hosts, each providing a service relating to a physical device (e.g. a digital image capturing service relating to a digital camera), will each be advertising and trying to acquire a consistent view of services surrounding them. We take the straightforward case in this scenario, in which all four hosts are powered on simultaneously and require a complete image of surrounding resources.

This example demonstrates a worst-case scenario where all resources need to be discovered. However, this scenario illustrates a similar case where a host moves into an existing ad hoc grouping and requesting an update of all available resources.

A summary of the results can be seen in figure 7 below highlighting a need for consideration of the number of discovery messages transmitted in an ad hoc environment. This figure is derived from examination of the currently released specifications in detail of the previously discussed platforms [5] [7] [30].

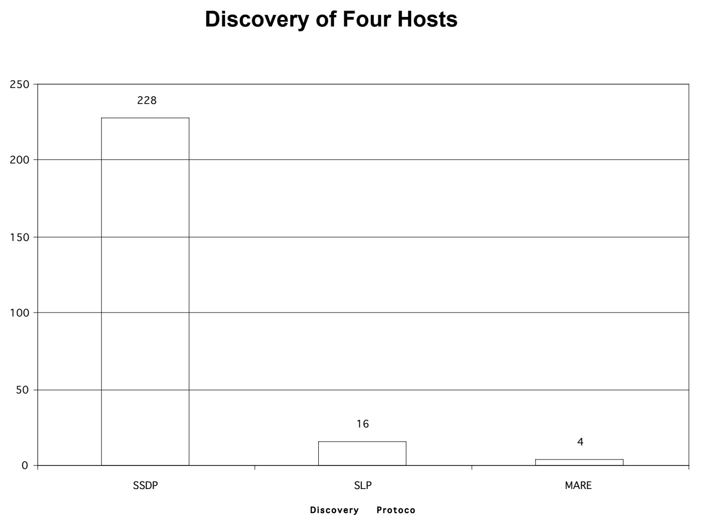

Figure 7 Messgaes produced while achieving a consistent view (4 hosts, 1 service per host)

7.2 Analysis of Discovery

- An **SSDP** query in an ad hoc grouping will provide three multicast requests, each responded to by three unicast responses consisting of three root device advertisements, two embedded device advertisements (for the camera) and one service advertisement. Relating to the test scenario, a single host requesting a view of all services held by its surrounding three neighbours will generate fifty seven messages. Thus, for all hosts to achieve a consistent view of the services offered by its neighbours, a total of 228 messages are needed.
- Providing a much simpler approach than SSDP, **SLP** can be assumed to not have the availability of points of directory agents in an ad hoc grouping requiring all hosts to perform a multicast discovery followed by a unicast response from every host. This approach generates an RPC style interaction whereby each host multicasts a request for services and all other hosts respond with a unicast response. This approach generates one request and one response from each host requiring 4 messages for each host and a total of 16 messages for all four hosts to gain a view of each other's services.
- **Jini** describes a mechanism for operation without a lookup service that is unlikely to be found in an ad hoc grouping due to its operational requirements being in excess of many mobile devices. Making a comparison could be done when forcing each host to act as if it were a lookup service performing *peer lookup* forcing direct interaction between client and server. This approach is mentioned in the Jini specification in limited amount of detail making an educated calculation of figures currently infeasible.

- **MARE** adopts a much simpler approach using periodic announcements of services thus generating only four messages transmitted through a tuple space. Multiple services on a host are combined into a single message containing multiple service descriptions reducing the number of transmitted messages, message size permitting. The L^2imbo tuple space also acts as a cache whereby local requests can be satisfied locally saving external communications.

Whilst these figures only examine start up costs, the possibility of frequent power cycles being performed by mobile hosts is likely and the ability to know about new resources that may not fall into specific type groups that are used to restrict discovery.

Clearly service discovery protocols can be optimised to improve service discovery performance (e.g. SLP can use a directory agent). However in ad hoc environments such techniques typically lead to increased startup costs for example when a system must detect the absence of central resources. UPnP supports announcement messages from services however to be effective these messages must be transmitted frequently forming an approach similar to the approach taken by MARE without message combining.

The number of messages produced in a running system can be seen as related to the number of hosts in the surrounding environment. Assuming each host requires a view of surrounding resources at all times, MARE and UPnP will generate announcements for all services; MARE combines announcements from the same machine whilst UPnP will produce one per service also producing larger numbers of messages upon receipt of a request for services, whilst SLP relies on requests and responses.

From the performance figures taken thus far, we can see that MARE offers a highly effective and bandwidth efficient resource discovery mechanism, at least comparable with existing service discovery approaches. In addition MARE offers functionality beyond that of service discovery covered in this section for example configuration of the discovered resources using agent technologies.

8 Conclusions

MARE addresses resource configuration in ad hoc environments using a novel combination of agents and tuple spaces. More specifically, MARE:

- Demonstrates a means of achieving a near consistent view of a constantly changing ad hoc environment.
- Has *no* reliance upon a single host for services central to the system such as resource discovery and naming.

- Provides a means to help reduce the amount of communications by the use of mobile agents.
- Aids the efficient use of resources through use of agents and the knowledge of resources available at a given time.

A distributed test bed has been implemented providing a number of ad hoc resources including the taking of camera stills, streaming medical telemetry and GPS location information. The initial implementation maintains a view of resources available in an environment, despite the random removal and insertion of new resources into the environment. Note that the current prototype of MARE has not been developed with performance as a primary goal, although MARE has been shown to perform well in the small systems developed so far.

Future work is required into security both of the system and the agents executing upon it. The Java security model holds many keys to this including for example a *per class author* security policy. The use of user permissions upon a host can also enable a protection of the system by executing MARE in a user mode with restricted permissions. Protection of data itself can be carried out utilising strongly encrypted tuple spaces with user authentication (cf. the username/ password required to access IBM's T-Spaces). We plan further development and evaluation of MARE though the introduction of more resources and enlarging of the existing prototype test bed.

To conclude, mobility is commonplace and users expect device interaction. The weaknesses highlighted in existing systems show a need for a different approach to handling the new era of collaboration between mobile wireless devices. MARE is a step towards enabling operations in the increasingly more diverse active environment being generated by the relentless development and deployment of mobile wireless technologies.

9 References

[1] Radio Equipement System (RES) Trans-European Trunked Radio (TETRA); Voice plus Data (V+D), Designer's Guide, ETSI Work Programme DE/RES – 0601, Subtechnical Committee (STC) RES 06, May 1995

[2] Development of WaveLAN, an ISM band wireless LAN, AT & T Technical Journal, pp. 27-37, July/August 1993

[3] Bluetooth Specifications, ongoing work, http://www.bluetooth.com/

[4] Ed. Candy et al, Emergency Multimedia, Simoco Europe Ltd., the Langdale Ambleside Mountain Rescue Team, and HW Communications Limited, Multimedia Demonstrator Programme funded by the Department of Trade and Industry.

[5] Object Management Group, CORBA/IIOP 2.3 Specification, OMG document formal/98-12-01, Object management group, http://www.omg.org/

[6] Microsoft Corporation, Distributed Component Object Model Protocol DCOM/1.0 Specification, Microsoft Developer Network Library Document, http://msdn.microsoft.com/

[7] A. Schill and S. Kümmel, Design and Implementation of a Support Platform for Distributed Mobile Computing, Distributed Systems Engineering Journal, 2(3), pp. 128-141, 1995

[8] Nigel Davies, Adrian Friday, Gordon Blair and Keith Cheverst, Distributed Systems Support for Adaptive Mobile Applications, ACM Mobile Networks and Applications, Special Issue on Mobile Computing - System Services, Volume 1, Number 4, 1996

[9] M. Satyanarayanan, B. Noble, P. Kumar and M. Price, Application-Aware Adaptation for Mobile Computing, Proceedings of the 6th ACM SIGOPS European Workshop (Dagstuhl, Germany), 1994

[10] A.D. Joseph, A. F. deLespinasse, J.A. Tauber, D.K. Gifford and M.F. Kaashoek, Rover: A toolkit for Mobile Information Access, Proceedings of the 15th symposium on Operating Systems Principles (SOSP'95), Copper Mountain Resort, Colorado, U.S., pp. 156-171, 1995

[11] Nicholas Carriero and David Gelernter, Linda in Context, Communications of the ACM, Volume 32 Number 4, pp 444 – 458, 1989

[12] Antony Rowstron, Bulk Primitives in Linda Run-Time Systems, University of York, Thesis, 1996

[13] Stephen Wade, An Investigation into the use of the Tuple Space Paradigm in Mobile Computing Environments, Ph.D. Thesis, Lancaster University, 1999.

[14] D. Gelernter, Generative Communication in Linda, ACM Transactions on Programming Languages and Systems, Volume 7, Number 1, pp 255-263, 1985

[15] Susanne C. Hupfer, Melinda: Linda with Multiple Tuple Spaces, Research Report YaleU/DCS/RR-766, February 1990

[16] Jini, JavaSpaces Specification, http://www.sun.com/jini/specs/

[17] TSpaces: The Next Wave, Hawaii International Conference on System Sciences (HICSS-32), 1999

[18] Nigel Davies, Adrian Friday, Stephen Wade and Gordon Blair, L^2imbo: A Distributed Systems Platform for Mobile Computing, ACM Mobile Networks and Applications (MONET), Special Issue on Protocols and Software Paradigms of Mobile Networks, Volume 3, Number 2, pp143-156, 1998

[19] , J. Baumann, F. Hohl, K. Rothermel, M. Schwehm, M. Straßer, Mole 3.0: A Middleware for Java-Based Mobile Software AgentsMiddleware'98 IFIP International conference on Distributed Systems Platforms and Open Distributed Processing, pp. 355-370, 1998

[20] Dag Johansen, Robbert van Renesse, and Fred B. Schneider, An Introduction to the TACOMA Distributed System, Department of Computer Science, University of Tromsø, Norway, Technical Report 95-23, 1995

[21] Robert S. Gray, Agent Tcl: A flexible and secure mobile-agent system, Proceedings of the 1996 Tcl/Tk Workshop, pages 9-23, 1996

[22] David Wong, Noemi Paciorek, Tom Walsh, Joe DiCelie, Mike Young, Bill Peet, Concordia: An Infrastructure for Collaborating Mobile Agents, Mitsubishi Electric ITA, First International Workshop on mobile agents, Berlin Germany, 1997

[23] Weiyi (William) Li, David G. Messerschmitt, Java-To-Go: Itinerative Computing Using Java, Department of Electrical Engineering and Computer Sciences University of California at Berkeley 1996

[24] H, Peine, T. Stolpmann, In Kurt Rothermel, Radu Popescu-Zeletin, The Architecture of the Ara Platform for Mobile Agents, (Eds.): Proc. of the First International Workshop on Mobile Agents MA'97 (Berlin, Germany), 1997

[25] Aglets Specification, IBM Research, http://www.trl.ibm.co.jp/aglets/

[26] Picco, G., Murphy, A., and Roman, Lime: Linda Meets Mobility, G.-C., Technical report no. 98-21, 1998

[27] J. Verizades, E. Guttman, C. Perkins, S. Kaplan, Service Location Protocol, rfc2165, June 1997

[28] Universal Plug and Play Device Architecture Reference Specification, Version 0.90 - November 10, 1999

[29] Y. Goland, T. Cai, P. Leach., Y. Gu, S. Albright, Simple Service Discovery Protocol

[30] M. Storey, G. Blair, Resource Configuration in Ad Hoc Networks: The MARE Approach, Third IEEE Workshop on Mobile Computing Systems and Applications (WMCSA), Monterey California, 2000

Matthew Storey graduated from Lancaster University in 1997 and in the same year joined the mobile computing group at Lancaster University. Matthew is currently a research student in the final stages of his PhD. His research is currently concerned with addressing issues of mobility using mobile agents and the tuple space paradigm. He is also interested in wearable, ubiquitous and ad-hoc computing.
E-mail: matt@comp.lancs.ac.uk

Prof. Gordon Blair is a founding member of the Distributed Multimedia Research Group at Lancaster University. Following the completion of his PhD at Strathclyde University, he moved to Lancaster in 1983. He currently holds a chair in distributed systems at this university, and is also an Adjunct Professor at the University of Tromsø in Norway. He has published over 180 papers in his field and in on the programme committees of many major international conferences in distributed systems, reflection and multimedia. He is on the steering committee of the Middleware series of conferences, was General Chair for this conference in 1998, and co-organised a workshop on Reflective Middleware (with Roy Campbell, University of Illinois at Urbana-Champaign) held in conjunction with Middleware'2000. He is also Programme Co-Chair of DOA'01 to be held in Rome in September 2001. He has been primarily responsible for a number of research projects at Lancaster including Sumo, Adapt and Open ORB.
E-mail: gordon@comp.lancs.ac.uk

Adrian Friday graduated from the University of London in 1991. In 1992 he helped form the mobile computing group at Lancaster University, and has since completed his PhD entitled "Infrastructure Support for Adaptive Mobile Applications". His early work focused on adaptive distributed systems support for mobile computing applications. In 1998 he was appointed as a Lecturer in the department of Computer Science and is an active member of the Distributed Multimedia Research Group. He is now an investigator in a number of key research initiatives at Lancaster, including the Equator IRC and GUIDE projects. His current research interests include: distributed system support for mobile, context-sensitive and sentient computing environments. Adrian is a member of IEEE Computer Society, the ACM and BCS.
E-mail: adrian@comp.lancs.ac.uk