LEAP into Ad-Hoc Networks

Jamie Lawrence
Media Lab Europe
Sugar House Lane, Bellevue,
Dublin 8, Ireland.
+353 1 474 2868
jamiel@mle.media.mit.edu

ABSTRACT
This paper provides an overview of the work currently underway at Media Lab Europe to enable an existing, open-source, FIPA-compliant agent platform with the ability to operate in an ad-hoc environment. The motivations for using agents in ad-hoc networks and the requirements this places on an agent platform are discussed. A mechanism for discovering instances of an agent platform using current service discovery techniques is presented and we detail how an existing platform will be modified to support this. Finally, modifications to the existing FIPA standards are proposed to support ad-hoc environments.

Categories and Subject Descriptors
I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – Multi-agent systems

General Terms
Standardization, Design, Management

Keywords
Agent platforms, wireless, ad-hoc networks, resource-limited devices, standards, FIPA.

1. INTRODUCTION
Mobile ad-hoc networks (MANETs) are to data communication what the walkie-talkie is to telephony; they allow peer-to-peer communication without the use of an established third-party infrastructure in an asynchronous but relatively unreliable manner. These networks have two interesting properties: their spontaneity creates transient local connections between individual nodes and the use of multi-hop routing allows a short-range radio to span large distances. To date, the majority of research in MANETs has been in the area of routing protocols designed to cope with high-levels of node mobility, route changes and a constantly changing network membership[15]. This use of multi-hop routing allows existing software to operate in new ways but it doesn't create fundamentally new applications. However, we believe the dynamic connections formed between nodes will enable novel applications to be developed.

There is a natural synergy between agents, entities that are capable of complex, dynamic interactions, and mobile ad-hoc networks, environments that inherently require such interactions. In this paper, we describe our initial exploration into the coupling of these two technologies.

As a first step towards exploring agent-enabled applications, an agent platform is required which is capable of operating on mobile devices without a fixed infrastructure. It is clear that a completely new agent platform is not required to fulfil this role, as there are a number of suitable, stable and functional open-source platforms available. One of most advanced platforms is the Lightweight Extensible Agent Platform, JADE-LEAP; a Java-based, FIPA-compliant platform that allows deployment of agents on devices as small as a mobile phone [1,3].

Four main modifications to the JADE-LEAP platform are proposed in order to support an ad-hoc environment: leased directory entries, a notification mechanism for directory changes, the removal of the Directory Facilitator (DF) and Agent Management System (AMS) as mandatory components, and the addition of a Discovery Agent (DA) to handle platform discovery and peer-to-peer agent discovery.

Sections 1.1 and 2 discuss our approach and motivations for this project. Section 3 describes the basic mechanisms and steps for performing discovery and Section 4 goes into further detail regarding the modifications required to the JADE-LEAP platform. Finally, Section 5 compares this work to current agent platforms and FIPA standards.

1.1 Approach
Although other alternatives exist, JADE-LEAP was chosen for a number of reasons. JADE-LEAP is an evolution of the popular JADE agent platform [2] from which it has inherited a lightweight behaviour scheduling mechanism, a container-based deployment mechanism and a set of FIPA-compliant content languages, transport protocols and directory services. The LEAP development team have squeezed the core JADE features onto mobile phones running Java 2 Micro Edition (J2ME) whilst retaining compatibility with existing JADE agents. This ability to deploy agents on small devices will become invaluable when building applications such as sensor networks that require large numbers of simple, inexpensive nodes.

Other agent platforms were also considered but they were rejected due to the dependency on more powerful Java VMs or weak interoperability. Micro-FIPAOS is dependant on PersonalJava which limits its deployment to relatively powerful PDAs such as Compaq iPAQs[13]. The Java Agent Services (JAS) is a J2ME-

compliant API specification based on the FIPA Abstract Architecture, however the reference implementation is targeted at the standard version of Java[11]. As the JAS platform is only concerned with the *abstract* notion of an agent platform, it has considerably weaker interoperability in a heterogeneous environment than platforms which implement the FIPA message transport, agent management and communication specifications.

The focus of the modifications proposed to JADE-LEAP is on providing the platform with the ability to recognise the appearance of other available platforms. This is termed *platform discovery*. Upon discovering another platform, a routine is initiated to perform *agent discovery*. Platform discovery differs from agent discovery since it's performed using existing service discovery technologies whereas agent discovery will use agent-level interactions. This allows us to reuse both existing peer-to-peer (P2P) and service discovery technologies for platform discovery and current agent standards specified by the Foundation for Intelligent Physical Agents (FIPA) for agent discovery. *Service Discovery* is a higher-level concept referring to the dynamic composition of atomic services to fulfil a larger-scale application, but it is beyond the scope of this paper.

As shown in Figure 1, the platform discovery mechanism may be tightly coupled with the transport medium (such as the Service Discovery Protocol in Bluetooth) or a generic system such as JXTA[16] or Jini[18]. Routing algorithms are expected to handle the creation and maintenance of routes between platforms to avoid the inefficient message routing done at the level of the agent platform. Active routing protocols (such as OLSR[5], ZRP[10]) or information gathered directly from the physical layer may be used to aid the platform discovery process.

In contrast to existing agent platforms which are required to host many agents, we assume that a constrained device will host only a few application agents (typically just one) and therefore less emphasis is placed on common services (such as directories). However, the concept of a platform is still valid as it allows the abstraction of common functionality from the agent code and compatibility with larger environments.

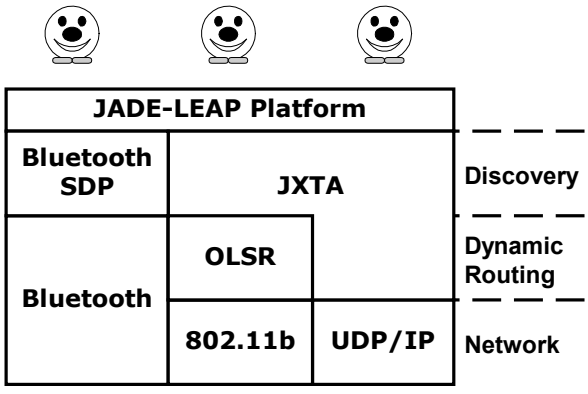

Figure 1: The "Big" Picture. The modified JADE-LEAP platform will utilise existing discovery and routing technologies.

Therefore, the approach taken by this project is to integrate existing P2P, service discovery and ad-hoc networking technologies into the JADE-LEAP platform to provide a robust infrastructure for deploying agents within an ad-hoc network.

2. MOTIVATIONS

We believe that the transient connections, which exist in an ad-hoc network, will enable innovative applications in addition to allowing us to study complex systems and emergent behaviour. By allowing an application to become aware of other agents passing by in the street, analogies can be drawn with the interactions that occur between ants. In ant colonies, these interactions are used to calculate colony density and current task allocations without conveying significant meaning [9]. In a colony of agents, each carried around the city by a user or vehicle, we can envisage applications emerging from the (possibly complex) communication between agents. These applications include networks of autonomous sensors, capable of analysing and processing data "in the field" and the sharing of information between people: file sharing, news filtering and even the construction and visualisation of networks of personal stories.

For this vision to be realised, the current limitations in both existing platforms and standards discussed below must be addressed. In particular, they both assume the stability of the connections and the full connectedness of the whole network, leaving them vulnerable to changes in topology.

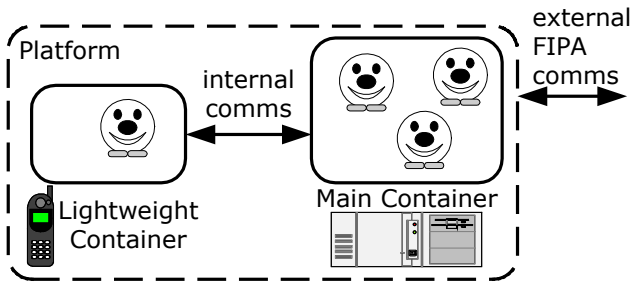

Figure 2: The current JADE-LEAP architecture of multiple containers forming a logical FIPA platform

As shown in Figure 2, the current implementation of JADE-LEAP, like that of its JADE ancestor, has a static, distributed platform structure consisting of one main container providing FIPA interoperability to one or more (possibly lightweight) containers. Each container may be hosted on separate JVMs that are distributed across a network using internal JADE-LEAP protocols for management and communication. From an external perspective, these containers appear as a single FIPA-compliant platform that allows for advanced features (such as agent mobility and management tools) and optimisation within the platform yet retains interoperability with other platforms.

JADE-LEAP contains no mechanism to actively discover other containers or platforms (it must be specified in a configuration file) or to start a lightweight container without the existence of a main container. In a mobile ad-hoc network, platform discovery and the lack of reliance on a fixed infrastructure are crucial properties and for this reason a Discovery Agent (DA) is introduced to handle platform discovery. Our modifications will also remove the distributed container support, as it is important

for each ad-hoc node to represent a self-contained platform to enable interoperability in, what is by definition, an open environment.

FIPA has specified two mandatory components of an agent platform: the directory facilitator (DF) and agent management system (AMS) that act as yellow and white page directories, respectively. These directories are necessary to support multiple agents on a single platform and efficient agent discovery but they also represent large, resource-consuming components (some 40 classes in total), not suitable for deployment on embedded devices. It is clear then that both of these components must be removed from small ad-hoc nodes. However, since directories improve the scalability of the network, our modified platform will host directory services if the device is capable (in terms of memory, processor and network resources) and when the surrounding environment makes it necessary to do so (i.e. there exists a high density of small, unregistered nodes).

The current AMS and DF directories contain no mechanisms to actively heal themselves if a client fails to deregister before being disconnected from the network or moving out of range. This will quickly lead to inconsistent directories in an environment of frequent changes to the network topology. The Jini technology has popularised a leasing approach whereby directory entries are leased to individual services for a specified period. It is the responsibility of the service to renew the lease before it expires; when this occurs the entry is removed from the directory. A similar mechanism allows our modified DF and AMS to "self-heal" within the specified leasing period if an agent disappears from the network (or indeed becomes too overloaded to maintain the lease).

The typical request-response interaction requires the agents to actively poll a directory to receive new directory entries. A more efficient publish-subscribe method would allow agents to subscribe to notifications when a new entry is added to the directory.

In summary, the proposed modifications avoid the restrictions imposed by the JADE-LEAP platform and current FIPA standards by:

- adding a Discovery Agent (DA) to handle peer-to-peer platform and agent discovery.
- removing the distributed container concepts currently present in JADE-LEAP to allow an ad-hoc node to be a fully contained platform.
- removing the DF and AMS as mandatory components of a platform, but allowing their activation should a device be capable and an environment require them.
- leasing directory entries within the DF and AMS
- providing a notification mechanism to allow the propagation of directory changes

3. BASIC DISCOVERY EVENTS

The discovery process can be broken down into a number of individual events, each described below and shown in Figure 3. Within these descriptions, the term "fragment" is used to refer to a self-contained instance of our modified JADE-LEAP platform which is not hosting an AMS or DF. These fragments can form "compounds" by registering their agents with a platform (i.e., a FIPA-compliant entity with an AMS and DF). A compound is simply an abstract concept used to group together the fragments registered with a common platform and has no physical or virtual representation. The terms defined above are inspired by the FIPA Ad-hoc technical committee [6].

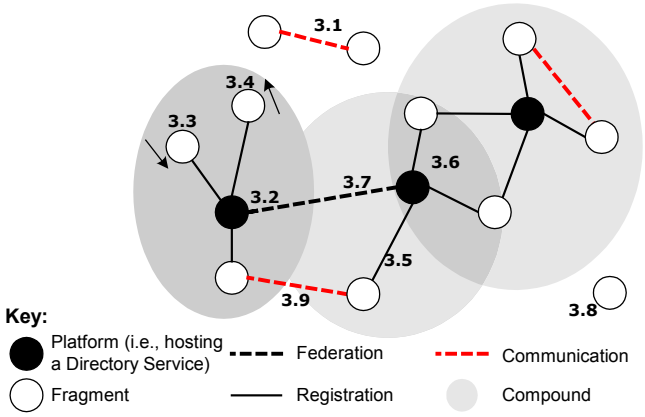

Figure 3: Basic Discovery Events

3.1 Direct Communication between Fragments

When two fragments meet in isolation, a form of peer-to-peer discovery must take place in order to allow communication between them. As the two devices discover each other, the discovery protocol notifies each discovery agent and they exchange their platform descriptions and the descriptions of *all* the agents each fragment is hosting. In addition, each fragment internally notifies each hosted agent of the newly discovered agents in the other fragment. The individual agents are responsible for examining the agent descriptions of the newly discovered agents and deciding if communication is possible, desirable or necessary.

This scenario is limited in scalability and will only support a few fragments.

3.2 Activation of Directory Services

One or more directory services (typically an AMS and DF) will be activated within a fragment according to a pre-defined strategy. On a constrained device (such as a mobile phone), this strategy may be simply "never host a directory". On a more capable device (that perhaps forms part of a backbone), the strategy would be "always host a directory", mimicking the current JADE-LEAP functionality. A wealth of strategies exists between these two extremes.

A possible strategy would involve monitoring metrics such as the number of discovery requests made and the number of local fragments not registered in a directory. If either of these measures crosses a specified threshold then a directory service is created (with a suitable random back-off time to ensure every fragment doesn't come to the same conclusion).

Once an AMS or DF is created, the discovery agent on that fragment will register the local agents with it (as explained more fully below). In addition, from this point on only the directory services will be advertised, and the individual agents must be discovered by first searching the directory service.

The deactivation of directory services will follow a similar pattern.

3.3 A Fragment Connects

When a fragment moves into range, it can detect another fragment hosting a directory service and subsequently register its hosted agents with this directory. Once the agents are registered with at least one (local or remote) directory, the fragment will only advertise that directory (not all of the hosted agents) unless specifically asked to do so. This mechanism forms a *compound* and allows smaller fragments to reduce their load during discovery by referring all search requests to the fragment with a directory service.

In cases where a fragment is discovered but its associated directory cannot be contacted (see Figure 4), the basic P2P discovery can take place between the two fragments (see section 3.1).

3.4 A Fragment Disconnects

When a fragment shuts down it may intentionally disconnect by deregistering its hosted agents from all directory services. However, more often a fragment is unexpectedly disconnected due to user intervention or network disruption. In these circumstances, the directory should self-heal as the leases on the directory listings begin to expire. The fragment will also recognise the disappearance of a directory through the same method (i.e., it will attempt to renew the lease only to find the directory not contactable). The fragment will continue to attempt discovery of other nearby fragments and the hosted agents will only be able to contact each other.

3.5 Registration of an Agent

Upon start-up, an agent registers its description with the local discovery agent. The discovery agent registers these descriptions with a directory service once one is discovered (covered in section 3.3).

3.6 Multiple Registrations of an Agent

An agent may be registered with multiple directory services at any one time, i.e., it may exist in more than one compound.

3.7 Federation of Directory Services

When a fragment hosting a directory service discovers another directory service, the two may federate together based upon some pre-defined strategy. This strategy may be simple: a timer to ensure the link is stable enough and to prevent spurious and short-lived federations.

3.8 Disconnected Fragments

A fragment may be completely disconnected from all other fragments (see section 3.4).

3.9 Communication between Fragments

Communication between agents happens as usual, with one caveat: due to the nature of wireless networks, it is possible for an agent to be discovered on a remote fragment that it is not possible to directly communicate with (see Figure 4). It seems intuitive that a route should exist between the two agents by using the directory to route the messages and in some cases the underlying transport protocols will provide multi-hop routing to fulfil this. However, it is worth noting that although a valid route might exist it does not guarantee that a routing protocol will have discovered it.

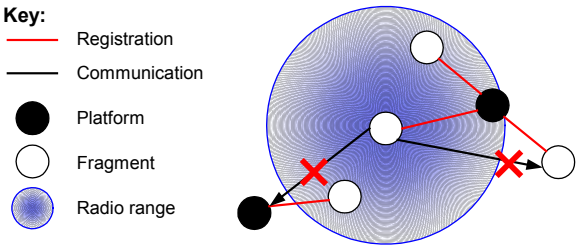

Figure 4: Wireless networks are not fully connected. An agent discovered through a directory may not be contactable due to the limitations of the wireless technology and the routing protocols in use.

4. MODIFICATIONS TO THE JADE-LEAP PLATFORM

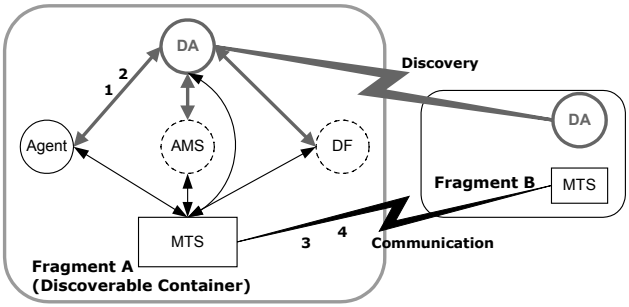

Figure 5: Overview of the modifications to JADE-LEAP. Grey indicates new discovery-related communications or components. Dotted lines indicate an optional component.

As shown in Figure 5, several modifications must be made to enable the JADE-LEAP platform within an ad-hoc environment. We have endeavoured to maintain the current JADE-LEAP API so that existing agents will only require modifications to handle the new functionality. The modifications have the minimum overlap with the existing platform to ensure future compatibility (as both JADE, JADE-LEAP, and our modifications will evolve). In particular, most modifications consist of sub-classing, relaxing access modifiers (to allow sub-classing), creating new classes, and in a few unavoidable cases, modifying the existing source code to allow abstract instantiation of either the existing classes or our modified versions.

4.1 Discoverable Container

JADE-LEAP provides two types of containers that are central to the way the platform operates: the main container for FIPA-compliant communication and the lightweight containers that depend upon this main container. Both of these container types represent a considerable body of code and have a number of dependencies that need to be removed, but to do so would seriously impact existing source code. Our solution is to create our own container type which implements the appropriate JADE-LEAP interfaces to allow it to replace the existing containers. In particular, our "Discoverable Container" will not require either an AMS or DF and will remove the support for intra-platform

communication, distribution, and support for various tools. Some of these features will be reintroduced if needed at a later stage.

4.2 Discovery agent

The largest addition is the discovery agent (DA), which is responsible for advertising and discovering the presence of fragments, agents and directories in addition to controlling the activation of the local directory services. The DA is implemented as an agent with support for one or more discovery protocols and is executed when a discoverable container starts up.

The minimum information advertised by a discovery protocol is the agent identifier (AID) of the discovery agent. Further information may then be requested directly from the DA using standard agent communication. Reducing the amount of information shared by the discovery protocol allows the use of very simple protocols that cannot represent a whole agent description (such as SSDP). The discovery agent will support the following functions: register, deregister, subscribe, unsubscribe, get-advertisement, get-all-agent-descriptions, and get-directories.

4.2.1 Internal Functions

1. *register, deregister*. This function allows an agent to register or remove its description with the discovery agent.
2. *subscribe, unsubscribe*. This allows a local agent to subscribe to the notifications that are broadcast when agents are discovered or disappear.
3. *get-directories*. This function returns all directories, both local and remote, where the discovery agent has registered the hosted agents.

4.2.2 External Functions

4. *get-advertisement*. This function enables a remote discovery agent to retrieve the advertisement for this fragment. If the agents on this fragment are registered with a directory then a reference to this directory is returned, otherwise the descriptions of each agent is sent back to the remote DA. In both cases, the platform description is also returned. This is not a replacement AMS/DF service as there are no methods for querying or searching – the descriptions for all currently registered agents are returned in response to a discovery request. Hence, this discovery mechanism is appropriate for only a small number of agents per fragment.
5. *get-all-agent-descriptions*. This performs exactly the same as the get-advertisement function when the agents are not registered with a directory service. This is used to force P2P discovery in the case where the initiator cannot access the directory service where the agents are registered (see section 3.9).

In response to a notification from the service discovery middleware, the local discovery agent will call the remote fragment's get-advertisement action. When one or more directories are returned two possible actions may occur. If the local agents are not registered with a directory, they will be registered with the directories returned. If a local directory exists then it will be federated with the returned directories (based on some strategy as previously mentioned).

4.3 Directory Services (AMS, DF)

The removal of the AMS and DF as mandatory entities allows for lower network, memory and processor costs, particularly in embedded environments where each fragment only supports a single agent. The costs of these services are related to the storage of the directory entries and the time required to process search requests. It is for these reasons that the discovery agent presented above does not perform any searching and only allows local registrations. On devices that are more capable the AMS and DF can be activated and utilised not only by local agents but also by those on nearby constrained devices.

In contrast with the existing DF/AMS all directory entries are leased and must be renewed prior to expiration, in a similar manner to Jini[18]. The additional functions subscribe and unsubscribe are also required to allow the propagation of directory changes to individual agents.

4.4 Agent

The core Agent class requires only a few modifications that include default registration with the local discovery agent rather than with the AMS and modifications to the DF and AMS communicators to hide the possible absence of these directories.

5. COMPATIBILITY ISSUES

5.1 Target Environment

The eventual target environment for these modifications will be smaller-than-phone embedded devices. Therefore, the platform should be able to comfortably operate with a single agent in (much) less than 512KB of RAM.

5.2 FIPA Compatibility

FIPA is aware of the potential benefits and problems with using agents in ad-hoc networks and they have recently formed a technical committee with the task of creating standards in this area[6]. Although this project is taking a pragmatic approach, it is intended that the results will be applicable to the standardisation efforts of FIPA.

With regards to FIPA compatibility, it is debatable whether this platform can comply with the current standards. On the surface, the removal of the AMS and DF as mandatory platform components fails to comply with both the FIPA Agent Management[8] and Abstract Architecture specifications[7]. However, the DA can be viewed as an inefficient directory service, which in response to a query performs no filtering and returns all registered entries. Viewed in this way, each DA fulfils the role of a directory and our fragments can therefore comply with the *abstract* notion of an agent system but not with the current Agent Management specification.

The removal of the AMS has significant side effects that have not been mentioned previously; in addition to providing a white pages directory, the AMS is responsible for platform and agent lifecycle management. In our implementation, the discoverable container will actually perform these functions since they are only notionally under the control of the present AMS. A more amenable modification to the design presented here would be to leave the AMS as a mandatory component of a fragment and integrate the DA functionality with that of the current AMS. We have not taken this route because the AMS in JADE-LEAP already has a large number of responsibilities other than those specified by FIPA, including the support for add-on tools, intra-platform notifications, debugging, "sniffing", and transport protocol management – most of which will not be required in our platform.

5.3 Expected Benefits

The purpose of making the DF an optional component on small devices is to conserve the limited memory and processing resources available. The DF does not provide any discovery functionality and therefore cannot replace the DA but, if present, the DF would exist *in addition* to the DA.

Currently, a JADE-LEAP lightweight container will occupy about 100KB of memory but this relies on a main container to be available elsewhere in the network to provide FIPA compatibility (DF, AMS and transport protocols). It is possible for a modified JADE-LEAP main container to run in roughly 700KB memory using PersonalJava[17]. Although further optimisations may be possible, sufficient free memory is also required to prevent excessive garbage collection and to allow for memory fluctuations due to temporary objects such as message buffers. For example, the memory usage of the current DF will change over time as it allows registrations by external agents and the searching of these directories entries. In addition, any directory service will need some form of self-healing mechanism (such as the leasing discussed in Section 3.4) that further increases the processing burden on the device. By contrast, the DA only allows local agents to register and simply returns a pre-built advertisement in response to *get-advertisement* requests.

Memory is not the only limited resource on mobile devices; in some embedded Java products (such as the TINI[14]), the number of threads is restricted and there is a high processing cost associated with switching threads. In practice, as each agent consumes a single thread this translates into minimising the number of mandatory agents on the platform.

Obviously, minimising the resources that are consumed by the platform increases the resources available to the agents and provides greater flexibility to the application developer.

5.4 Related Work

Related work has been performed by Langley et al [12] on using the Simple Service Discovery Protocol (SSDP) and the Gnutella P2P network to discover agent platforms . Their RETSINA agent platform utilised SSDP to broadcast discovery announcements and requests within the local network. The Gnutella protocols that were used extend the reach of these messages, thereby enabling the discovery of services both locally and from across the Internet.

Other agent platforms such as the Ronin Agent Framework [4] use Jini as a basis for discovering agents but these are not suitable for ad-hoc networks due to the high resource requirements of Jini and centralised lookup servers.

In contrast, the solution presented here is specifically designed for use on resource-constrained devices in an ad-hoc network. We have emphatically avoided defining the routing protocols or actual discovery protocols in use, but have described a mechanism which abstracts the platform discovery from the actual protocols used. This will allow greater flexibility for future projects, experiments and demonstrations.

6. CONCLUSIONS

The design presented here will allow the deployment of agents in an ad-hoc network. A number of modifications have been proposed to an existing platform that will allow it to discover other platforms, thereby allowing the agents to begin interacting with each other without the existence of a fixed infrastructure. These modifications are specifically targeted to embedded devices and are independent of the discovery or routing protocols employed. The work on applying these modifications and analysing the performance gains has already begun.

Future work beyond these modifications will focus on developing emergent applications in the areas of sensor networks and information sharing. Further work is also required to ensure our agents can handle the unreliable nature of an ad-hoc network, in terms of both individual message loss and the premature termination of whole conversations.

7. ACKNOWLEDGMENTS

This work would not be possible without the availability of the JADE and JADE-LEAP agent platforms in open-source and the commitment by their respective development teams to continually improve and support them.

Thanks to the Dynamic Interactions group at Media Lab Europe for supporting this work.

8. REFERENCES

[1] Adorni, G., Bergenti, F., Poggi, A., and Rimassa, G. Enabling FIPA agents on small devices. Cooperative Information Agents, 2001 (Modena, Italy). 2001

[2] Bellifemine, F., Poggi, A., Rimassa, G., and Turci, P. An Object-Oriented Framework to Realize Agent Systems. WOA, 2001 (Parma, Italy). 2001

[3] Berger, M., Bauer, B., and Watzke, M. A Scalable Agent Infrastructure. Workshop on Infrastructure for Agents, MAS and Scalable MAS at Autonomous Agents'01 (Montreal). 2001

[4] Chen, H. L. Developing a Dynamic Distributed Intelligent Agent Framework Based on the Jini Architecture. Masters thesis, University of Maryland Baltimore County.1999.

[5] Clausen, T., Jacquet, P., Laouiti, A., Minet, P., Muhlethaler, P., Qayyum, A., and Viennot, L. Optimized Link State Routing Protocol. http://www.ietf.org/internet-drafts/draft-ietf-manet-olsr-06.txt.

[6] Foundation for Intelligent Physical Agents. [f-out-00105] FIPA TC Ad-hoc First Call For Technology.

[7] Foundation for Intelligent Physical Agents. [FIPA00001] FIPA Abstract Architecture Specification.

[8] Foundation for Intelligent Physical Agents. [FIPA00023] FIPA Agent Management Specification.

[9] Gordon, D. Ants at Work - How an Insect Society is Organised. Norton, London, 1999.

[10] Haas, Z. J. and Pearlman, M. R. ZRP - A Hybrid Framework for Routing in Ad Hoc Networks. Ad Hoc Networking, Perkins, C., 2001, 221-253.

[11] JSR-87 Expert Group. Java Agent Services. http://www.java-agent.org.

[12] Langley, B., Paolucci, M., and Sycara, K. Discovery Infrastructure in Multi-Agent Systems. Workshop on Infrastructure for Agents, MAS, and Scalable MAS at Autonomous Agents'01 (Montreal). 2001

[13] Laukkanen, M., Tarkoma, S., and Leinonen, J. FIPA-OS Agent Platform for Small-footprint Devices. ATAL, 2001 (Seattle, USA). 2001

[14] Loomis, D. The TINI Specification and Developer's Guide. Addison-Wesley, 2001.

[15] Perkins, C. Ad Hoc Networking. Addison Wesley, 2001.

[16] Project JXTA. JXTA v1.0 Protocol Specification. http://www.jxta.org.

[17] Ratsimor, O., Chakraborty, D., Tolia, S., Khushraj, D., Gupta, G., Kunjithapatham, A., Joshi, A., and Finin, T. Allia: Policy-based Alliance Formation for Agents in Ad hoc Environments. http://gentoo.cs.umbc.edu/~oratsi2/FIPA-Ad-Hoc/FIPA-Paper/AlliaFipaAhHocUMBC.pdf.

[18] Waldo, J., Arnold, K., and The Jini Team. The Jini Specifications. Addison-Wesley, 2000.