Agent Migration and Communication in WSNs

Conor Muldoon, Gregory M.P. O'Hare, Michael J. O'Grady, Richard Tynan
CLARITY: The Centre for
Sensor Web Technologies,
School of Computer Science and Informatics,
University College Dublin (UCD),
Belfield, Dublin, Ireland.
{conor.muldoon, gregory.ohare, michael.j.ogrady,richard.tynan}@ucd.ie

Abstract

Intelligent agents offer a viable paradigm for enabling AmI applications and services. As WSN technologies are anticipated to provide an indispensable component in many application domains, the need for enabling the agent paradigm to encompass such technologies becomes more urgent. The resource-constrained ad-hoc nature of WSNs poses significant challenges to conventional agent frameworks. In particular, the implications for agent functionality and behaviour in a WSN context demand that issues such as unreliable message delivery and limited power resources, amongst others, be considered. In this paper, the practical issues of agent migration and communication are considered in light of WSN constraints. The discussion is illustrated through a description of approaches adopted by Agent Factory Micro Edition (AFME).

1 Introduction

Ambient Intelligence (AmI) is an emerging field whose focus is on the development of applications that combine the concepts of intelligent systems with those of ubiquitous computing. The goal of ambient intelligence is the creation of intelligent electronic environments that are sensitive and adaptive to people and their perceived requirements. If systems are to be truly sensitive to the context of the user, they must be capable of dynamically altering their behaviour at run-time. In essence, they must be capable of acting in an intelligent and proactive manner.

Considerable research has been invested in the development of proactive intelligent systems by the agent development community and, as such, agent technology has much to offer the ambient domain. This is why ubiquitous computing has been recognised as one of the key areas of research for agent technologies [1] [2]. The ability to migrate enhances an agent's capabilities to adapt and respond to unexpected events. This flexibility makes mobile agents adept at managing the complexity of emergent behaviour synonymous with ambient systems. Traditional agent frameworks have implicitly assumed the availability of fixed network communications and significant computational resources. Ubiquitous computing and its constituent technologies of mobile devices and Wireless Sensor Networks (WSNs) challenge this assumption. One framework that seeks to address the constraints and heterogeneity of AmI environments is Agent Factory Micro Edition (AFME) [3] [4].

AFME is a minimised footprint intelligent agent platform that was originally developed for use with 3G mobile phones. Deploying the platform on sensor motes introduces a number of problems to be addressed. As an example, consider communication. In 3G networks, communication and migration is facilitated through the use of sockets that operate on top of a TCP/IP stack. Sensor nodes do not typically support such a stack; therefore, it was necessary to reengineer the original AFME message transport and migration [5] services. As an initial target platform, the Sun SPOT mote was chosen as it comprised the Squawk JVM [6] and has relatively high processing capabilities when compared to other motes.

2 Some Reflections on Agents

The majority of computer systems currently in operation employ algorithms that are based on the concept of perfect information. In ubiquitous environments, the situation is more complex. Pervasive or ubiquitous systems necessitate a capability to deal with partial information and uncertainty. Such types of systems are highly complex and are intractable using traditional approaches to software develop-

ment. Agent architectures encapsulate a number of mechanisms for dealing with uncertainty and change. One sophisticated agent architecture is based on the mentalistic constructs of Belief, Desire, and Intention (BDI) [7]. AFME is strongly influenced by the BDI paradigm.

Beliefs represent knowledge that an agent has of the world at a given moment in time. Beliefs are necessary for a number of reasons. The world is dynamic, therefore past events must be remembered. Agents have a limited view of the world and only remember events within their sphere of perception. In resource-bounded scenarios, agents cache important information rather than re-compute it from perceptual data.

Desires or goals represent a state of the world that an agent wishes to bring about. Goals provide an agent with the means by which it can identify the purpose of a particular task. Traditional approaches to software development are activity-oriented rather than goal-oriented. With an activity-oriented approach, the system has no memory of why a particular activity was executed. By abstracting this information, agents are endowed with a mechanism to recover from failures and to opportunistically take advantage of unexpected events or possibilities as they become available.

Agents are resource-bounded and will not be capable of achieving all of their desires even if the desires are consistent. An agent must fix upon a subset of its desires and commit resources to achieving them. This subset of desires represents an agent's intentions.

Agents rarely exist in isolation, but usually form a coalition of agents or a Multi-Agent System (MAS). Though endowed with particular responsibilities, each individual agent interacts with other agents in the framework to fulfil the objectives of the system. Fundamental to this cooperation and collaboration is the existence of an Agent Communication Language (ACL) shared and understood by all agents. The necessity to support inter-agent communication has led to the development of an international ACL standard, which has been ratified by the Foundation for Intelligent Physical Agents (FIPA), an autonomous standards committee of the IEEE.

Migration refers to an agent's capability to transport its state from one environment to another, with its data intact, so that it is capable of continuing to operate at its destination. There have been several uses for migration reported in the literature [8] [9]. For instance, in certain circumstances it will be more efficient for agents to migrate to a shared location and communicate locally, rather than communicate remotely. Mobile agents are more autonomous than static agents, if something goes wrong at a particular location, a mobile agent will move to a new location and from there continue to achieve its objectives. In certain cases, it is better in terms of network load to bring the computation to the data rather bringing the data to the computation.

3 Overview of AFME

Agent Factory Micro Edition (AFME) is an intelligent agent framework for ubiquitous devices. Typically, intelligent agent frameworks are based on a declarative agent programming language, although in practice most of them are used in conjunction with imperative components. This is the case with AFME. AFME agents are imbued with mechanisms that enable them to interact with their environment. Agents perceive and act upon the environment through perceptors and actuators respectively. The word perceptor is used rather than sensor to distinguish the software component from hardware sensors. Perceptors and actuators represent the interface between the agent and the environment and are implemented in Java. This interface acts as an enabling mechanism through which the agents are situated.

AFME uses the Agent Factory Agent Programming Language (AFAPL), which is declarative. AFAPL is based on a logical formalism of belief and commitment [10]. AFME agents follow a sense-deliberate-act cycle.The agents are executed at periodic intervals using a scheduler. Four functions are performed when an agent is executed. First, the perceptors are fired. The perceptors generate beliefs, which are a symbolic representation of information content. The information content is in relation to either the agent's state or to something in the environment. To adopt a belief, the perceptor calls a method that adds the belief to the agent's belief set. Second, the agent's desires are identified using resolution-based reasoning. Resolution-based reasoning is the goal-based querying mechanism commonly employed within Prolog interpreters. Third, the agent's commitments are identified using a knapsack procedure [11]. Fourth, depending on the nature of the commitments adopted, various actuators are fired.

In AFME, rules that define the conditions under which commitments are adopted are used to encode an agent's behaviour. The following is an example of an AFME rule[1]:

```
message(request,?sender,
removeData(?user))>deleteRecord(?user);
```

The truth of a belief sentence (text prior to the $>$ symbol) is evaluated based upon the current beliefs of the agent. The result of the query process is either failure, in which case the belief sentence is evaluated to false or to a set of bindings that cause the belief sentence to be evaluated to true. In AFAPL, the *?* symbol represents a variable. In this example, if the agent has adopted a belief that it has received a message from another agent to remove user data, it adopts

[1]It should be noted that this is a shorthand version of AFAPL, used within AFME to reduce development time.

a commitment to delete the record related to the user. At an imperative level, a perceptor that is written in Java monitors the message transport service, which contains a server thread that receives incoming messages. Once a message is received, it is added to a buffer in the service. Subsequently, the perceptor adds a belief, which is a first order structure Java class, to the agent's belief set. The interpreter periodically evaluates the belief set. If the conditions for a commitment are satisfied (that is, all of the beliefs prior to the $>$ symbol in the rule have been adopted), either a plan is executed to achieve the commitment or a primitive action or actuator is fired. In this paper, we shall only consider primitive actions. When an actuator is created, it is associated with a symbolic trigger. In this case, a delete record actuator, written in Java, is associated with the trigger string *deleteRecord(?user)*. Once the commitment is activated, the *?user* variable is passed to the actuator and the imperative code for deleting the file is executed. Structuring agents in this manner is useful in that it enables their behaviour to be altered at a symbolic level rather that having to modify the imperative code.

In AFME, the commitment to the right of the implication (the $>$ symbol) can take additional arguments. These arguments represent to whom the commitment is made, the time at which the commitment should be executed, the predicate for maintaining the commitment, and the utility values of the commitment. These additional arguments go beyond the scope of this paper and shall not be described here (for a discussion of how these arguments are supported in AFME see [12]).

3.1 Platform Architecture

In AFME, as with the majority of agent frameworks, agents operate on platforms rather than as stand alone processes. This reduces resource usage as separate JVM instantiations or Java Isolates are not required for each individual agent. Additionally, agents share common platform services. Typically, there is one platform per device or computer and multiple agents on the platform. Within a platform there are usually a number of platform services, such as the message transport service. A service is a shared information space between agents on a local agent platform that provides functionality that the agents can avail of through the use of actuators and perceptors.

Figure 1 illustrates the AFME architecture. An AFME platform comprises a scheduler, a group of agents, and serveral platform services. This paper concerns the development of two such services, namely the wireless message transport and migration services.

To improve reuse and modularity within AFME, actuators, perceptors, and services are prevented from containing direct object references to each other. Rather than passing

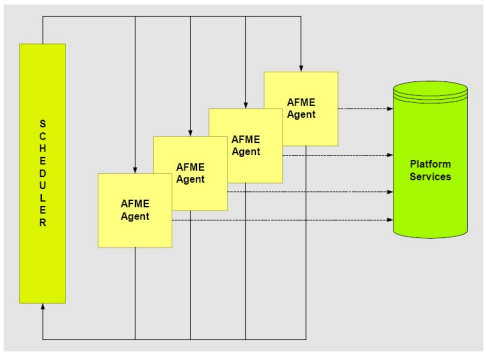

Figure 1. AFME Architecture

messages directly, they interact via perception and affect managers, which are generic AFME classes. This ensures that communication between the components is agnostic. The messages that are passed between the components are in the form of first order structures. First order structures provide a symbolic representation of the information content and ensure that messages passed do not expose internal details of the message senders. Actuators and perceptors developed to interact with a platform service in one application can be used, without making any imperative alterations, to interact with a different service in a different application and vice versa. The implementation of platform services can be completely altered without having to modify or recompile the actuators and perceptors. A completely different class could even be used to provide the functionality. Additionally, the same platform service may be used within two different applications to interact with a different set of actuators and perceptors. This provides a more flexible approach than simply using object polymorphism in that the classes' inheritance structure or implemented interfaces need not be known and can thus be altered without affecting caller or callee constructs[2].

The system components of AFME are interchangeable because they interact without directly referencing one another. They contain dependencies on the first order structure class and the affect and perception managers. They do not contain dependencies on each other. When a service is created, it is associated with a uniquely identifiable name. Actuators and perceptors use this name to indicate the target object for a particular message. They call the appropriate method on the affect and perception managers. The name is resolved to a service instance and the message is forwarded on appropriately.

To illustrate the advantage of decoupling the actuators, perceptors, and services, consider the situation whereby an

[2]The callee is restricted from knowing (referencing) the class structure of the service. The only assumption that can be made is that it extends a particular abstract class. If the abstract class is changed, it will still not affect the callee only the managers.

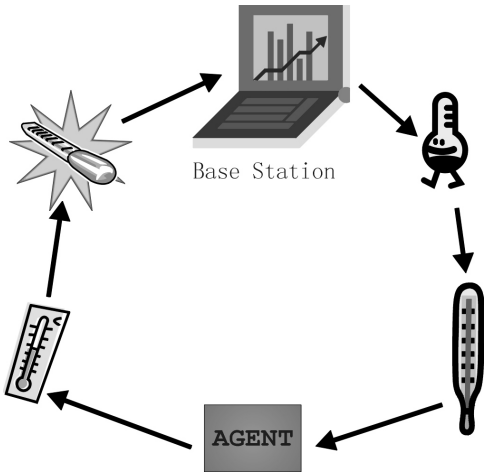

Figure 2. Agent migration path between heterogeneous temperature platforms.

agent must monitor and control a network of heterogeneous temperature sensors (see Figure 2). The imperative code for calibrating and obtaining information from the sensors on each platform is different and has been implemented within a local platform service. In this example, the agent migrates from platform to platform, obtaining information from the service, and calibrating the various sensors accordingly. Since the agent does not obtain a direct object reference to the local temperature service as it's operating on it, the developer need only write a single set of actuators and perceptors for interacting with the services.

Decoupling the system components is useful when agents must migrate between significantly different types of environment, such as a mobile phone and a sensor mote. In such situations, internal services will be implemented differently and have a different class structure.

4 Wireless Message Transport Service

The wireless message transport service discussed here differs significantly from the original message transport service [4], which was developed for mobile phones. The Sun SPOT motes communicate using the IEEE 802.15.4 standard, including the base-station approach to sensor networking. The wireless message transport service facilitates peer to peer communication between agents and is based on the Sun SPOT radiogram protocol rather than TCP/IP. The radiogram protocol uses datagrams to facilitate communication between motes. With the Sun SPOT radiogram protocol, the connections operating over a single hop have different semantics to those operating over multiple hops. This is due to a performance optimisation. When datagrams are sent over more than one hop, there are no guarantees about delivery or ordering. In such cases, datagrams will sometimes be silently lost, be delivered more than once, and out of sequence. When datagrams are sent over a single hop, they will not be silently lost or delivered out of sequence, but sometimes they will be delivered more than once.

The radiogram protocol operates in a client server manner. When the message transport service is created, a server thread is created to receive incoming messages. When a message is received it is added to an internal buffer within the service. An agent will subsequently perceive messages through the use of a perceptor.

When an agent is sending a message, it attempts to open a datagram client connection. The datagram server connection must be open at the destination. With datagrams a connection opened with a particular address can only be used to send to that address. The wireless message transport service only allows agents to send messages of a maximum size. If the contents of the message is greater than the limit, it is first split into a number of sub messages within an actuator and each sub message is then sent using the message transport service. When all sub messages have been received, the entire message is reconstructed within a perceptor and then added to the belief set of the agent.

5 Wireless Migration Service

Migration refers to the process of transferring an agent from one platform to another. Agent migration is often classified as either *strong* or *weak*. This classification is related to the amount of information transferred when an agent moves. The more information transferred the stronger the mobility. The strongest form of migration possible requires the transfer of the entire internal stack of the process in question. With this type of migration, execution is interrupted midway through a method or operation invocation at the source and started again at the destination at exactly the same point. This is not possible in heterogeneous environments, where the internal stacks are implemented differently. In such environments, weaker forms of mobility are necessary.

Truly strong migration is not possible in Java, since much of an application's state is under the control of the JVM. Within Java the internal implementation details of the JVM remain hidden. This facilitates platform independent development. The consequences of this, however, are that a Java application is prevented from transferring a process midway through execution in a transparent manner. For the majority of mobile applications though it is not necessary to transfer the internal stack of the JVM. It is usually sufficient to transfer the requisite code and the internal states of the objects within the system.

Within AFME, support is only provided for the trans-

fer of the agent's mental state. Any classes required by the agent must already be present at the destination. This is because the Constrained Limited Device Configuration (CLDC)[3] does not contain an API for dynamically loading foreign objects.

In the Squawk JVM, it is possible to migrate an application to another Squawk enabled device. Squawk implements an isolate mechanism, which can be used can for a type of code migration [6]. Isolate migration is not used in AFME. The reason for this is that isolate migration is dependent on internal details of the JVM and is therefore not really platform independent in the sense that an isolate can only be transferred to another Squawk JVM. It could not be used to transfer an application to a CLDC JVM written in C or C++, such as those used on the majority of mobile phone JVMs, for example. Additionally, with isolates, it would be necessary to migrate the entire application or platform, rather than just a single agent, which is not what we want.

The AFME wireless migration service uses both the Sun SPOT radiogram protocol and radiostream protocol. The radiostream protocol operates in a similar manner to TCP/IP sockets. It provides reliable, buffered, stream-based communication between motes. This, however, comes at a cost in terms of power usage. The reason this approach is adopted for agent migration is that we wish to ensure that agent does not become corrupt or lost due to the migration process. If a message is lost or corrupt, the system can recover by resending the message. If an agent is lost or corrupt, it can not be recovered without duplication or redundancy, which would also use up resources and would become complex to manage as agent artefacts would be scattered throughout the network.

The problem with the radiostream protocol, however, is that both the target platform and the source platform must know each others MAC address before a connection can be established. That is, it does not adopt a client server approach or operate in a similar manner to the radiogram protocol. In a dynamic mobile agent setting, it is unlikely that the addresses of the platforms of all source agents will be known a priori at compile time. To get around this problem, when an agent wishes to migrate to a particular platform, initial communication is facilitated through the use of datagrams. Using datagrams, the platforms exchange address and port information and subsequently construct a radiostream. Once the radiostream is established, the agent is transferred through the reliable connection and then terminiated at the source. Subsequently, the stream connection is closed. At the destination, the platform creates and starts the agent.

[3]CLDC is the Java platform specification for resource constrained devices.

Service	Jar Size	NCSS	McCabe Complexity
Migration	7k	151	2.08
MTS	9k	249	4.25

Table 1. Results

6 Evaluation

The following metrics were used to evaluate the footprint and maintainability of the wireless migration and message transport services: McCabe Cyclomatic Complexity, Non-Commenting Source Statements (NCSS), and independent Jar size[4]. The results are presented in Table 1.

7 Related Work

There have been a number of alternative agent frameworks developed for mobile devices [13] [14] [15] [16], such as mobile phones and PDAs, but at present there are no intelligent agent frameworks for sensor nodes. To deploy such frameworks in a WSN setting, significant changes would have to be made to their networking platform services. Indeed, some of these frameworks do not include a networking capability. Agilla [17] is an agent platform for embedded devices, but it does not contain reasoning capabilities and therefore does not conform to the same definition of agency as AFME.

There have been several frameworks developed to facilitate agent migration. At present, however, most of these frameworks are based on protocols that are not supported by WSN motes. For such frameworks to be deployed in a WSN setting, significant modifications would be required

8 Conclusion

This paper detailed the wireless message transport and migration services of AFME, an open source intelligent agent framework for ubiquitous devices. The original migration and message transport services of the framework had to be reengineered due to the different communications mechanisms available on the Sun SPOT when compared to mobile phones. Due to the manner in which AFME is structured, the consequences of this alteration remained hidden. When AFME agents wish to interact with platform services, they do so through the use of first order structures. That is, information is transferred through the use of a symbolic construct rather than directly through the use of object references. The consequences of this are that if a new service

[4]Indpendent Jar size refers to the individual Jar size of the service. In deploying the software to a device, only a single Jar file, containing AFME and requisite services, would be used.

is created or is replaced, no alterations are required to the agents that use the service.

At present, there are no other intelligent agent frameworks designed to operate on Sun SPOT motes. The reason for this is likely that, until recently, the processing power of WSN motes was insufficient to deploy such frameworks. Agent frameworks do exist for low specification devices, such as Berkeley motes, but those frameworks do not contain reasoning capabilities. With more powerful motes, such as the Sun SPOT, on the market, it is feasible to implement more intelligent applications; such developments are a prerequisite to the attainment of the AmI vision.

Acknowledgment

This material is based upon works supported by the Science Foundation Ireland and the Irish Research Council for Science, Engineering, and Technology.

References

[1] S. Creese, "Future challenges in pervasive computing environments," *SC Infosec article*, Mar. 5, 2003.

[2] J. Pearce, "Pervasive computing is the future," *ZD Net article*, Jan. 30, 2003.

[3] C. Muldoon, "An agent framework for ubiquitous services," Ph.D. dissertation, School of Computer Science and Informatics, Dublin, Ireland, 2007.

[4] C. Muldoon, G. M. P. O Hare, R. W. Collier, and M. J. O Grady, "Agent Factory Micro Edition: A Framework for Ambient Applications," in *Intelligent Agents in Computing Systems*, ser. Lecture Notes in Computer Science, vol. 3993. Reading, UK: Springer, 28-31 May 2006, pp. 727–734.

[5] C. Muldoon, G. M. P. O Hare, and J. F. Bradley, "Towards Reflective Mobile Agents for Resource Constrained Mobile Devices." in *AAMAS 07: Proceedings of the Sixth International Joint conference on Autonomous Agents and Multiagent Systems*. Honolulu, Hawai'i: ACM, May 14-18 2007.

[6] D. Simon and C. Cifuentes, "The squawk virtual machine: Java^{tm} on the bare metal," in *OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. New York, NY, USA: ACM, 2005, pp. 150–151.

[7] A. S. Rao and M. P. Georgeff, "BDI Agents: from theory to practice," *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, pp. 312–319, June 1995.

[8] T. Schlegel, P. Braun, and R. Kowalczyk, "Towards autonomous mobile agents with emergent migration behaviour," in *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM Press, 2006, pp. 585–592.

[9] A. E. Fallah-Seghrouchni and A. Suna, "An unified framework for programming autonomous, intelligent and mobile agents." in *CEEMAS*, 2003, pp. 353–362.

[10] R. W. Collier, "Agent Factory: A Framework for the Engineering of Agent-Oriented Applications," *Ph.D. Thesis*, 2001.

[11] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[12] C. Muldoon, G. M. P. O Hare, and M. J. O Grady, "Managing Resources in Constrained Environments with Autonomous Agents."

[13] M. Berger, S. Rusitschka, D. Toropov, M. Watzke, and M. Schlichte, "The Development of the Lightweight Extensible Agent Platform," *EXP in Search of Innovation*, vol. 3, no. 3, pp. 32–41, 2003.

[14] F. Koch, J.-J. Meyer, F. Dignum, and I. Rahwan, "Programming Deliberative Agents for Mobile Services: the 3APL-M Platform," *AAMAS'05 Workshop on Programming Multi-Agent Systems (ProMAS05)*, 2005.

[15] S. Khalique, S. Farooq, H. F. Ahmad, H. Suguri, and A. Ali, "Sage-lite: An architecture and implementation of light weight multiagent system," *ISADS*, vol. 0, pp. 239–244, 2007.

[16] W. Wright and D. Moore, "Design considerations for multiagent systems on very small platforms," in *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM Press, 2003, pp. 1160–1161.

[17] C.-L. Fok, G.-C. Roman, and C. Lu, "Rapid development and flexible deployment of adaptive wireless sensor network applications," in *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'05)*. IEEE, June 2005, pp. 653–662.