In-Motes: An Intelligent Agent Based Middleware for Wireless Sensor Networks

DIMITRIOS GEORGOULAS AND KEITH BLOW
Adaptive Communications Networks Research Group
Aston University
Aston Triangle, Birmingham, B4 7ET
UNITED KINGDOM
http://www.ee.aston.ac.uk/research/acrg/

Abstract: Wireless sensor networks (WSNs) have been identified as a promising technology that will allow people and machines to interact with their environment in a revolutionary way. These networks, however, are facing limitations such as energy constraints of the sensor and difficulties in reprogramming the actual network. To address these limitations we propose a novel agent middleware. Namely In-Motes can be considered as an intelligent network which is deployed with no pre-installed application. Mobile agents are injected into the network, then migrate and clone across it, following specific rules and performing application specific tasks. By doing so, each mote is given a certain degree of perception, cognition and control, forming the basis of its intelligence. Linda-like tuplespaces and federated system architecture are proposed as the means for collaboration and coordination of the agents. In order to make the network more robust, certain behavioural rules are proposed taking inspiration from a community of bacterial strains. These preserve each agent's certain degree of autonomy and identifies a highly coordinated architecture for them.

Key-Words: Wireless Sensor Networks, Middleware, Agents, Federated Systems, Bacterial Strains

1 Introduction

Wireless Sensor Networks (WSNs) have been identified as one of the most important technologies for the 21st century [1]. As technology advances and hardware prices drop, WSNs will find even more prosperous ground to spread in areas where traditional networks are inadequate. WSNs consist of tiny sensors which can be supplied to a specific environment, running applications such as habitat monitoring, microclimate research, medical care and structural monitoring [2]. Each sensor is attached to battery powered microprocessors combined with a set of specific instrumentation for environmental measurements such as sound, light and temperature.

WSN current infrastructure makes applications quite difficult to develop and install. One of the most wide spread platforms for WSNs is TinyOS with a combination of MICA2 motes [3]. TinyOS builds its architecture in a component based model triggered by event-driven calls [4]. Applications that are developed in this platform can not be reprogrammed dynamically and the only flexibility is located in changing various parameters prior to the installation. This means that post-development reprogramming is not feasible as the huge deployment of nodes in an environment makes the manual collection, reprogramming and re-deployment of them impossible. To that extent, an additional complication is also located within the area of energy management of the motes. Similarly, it would be infeasible to change the batteries for every single mote in an environment.

Many middlewares that have been developed in recent years offer a solution to upgrade the flexibility of the application level of WSNs. Various solutions are provided by middlewares such as XPN [5], Deluge [6], Mate [7], SensorWare [8] and Agilla [9]. Both XPN and Deluge architectures are based in network reconfiguration. This is succeeded by flashing the instruction memory of each mote, with Deluge also allowing multi-hopping reprogramming. However, both architectures suffer from long delays produced by the image that must be transferred over the network, increasing in parallel the energy levels a network must use. Mate is based in the generation of a virtual machine that divides each application into a specific number of capsules that are sent into the network by applying flooding, an action that consumes substantial energy resources of the WSN. The Agilla architecture is primarily based upon Mate. However, instead of capsules Agilla allows users to deploy applications by injecting mobile agents into the network, rather than mobile executable scripts that SensorWare uses. More fundamentally, Agilla lacks precise real location information since the virtual grid is identical with the real one and it does not clarify if devices are able to die.

To address the limitations of the previously mentioned middleware solutions, we are proposing In-Motes, an intelligent agent based middleware for WSNs. In-Motes is based on Agilla and Mate by allowing users to inject agents inside the network and provides a high level architecture for the given agent community based on federated systems [10] and behavioral rules produced by a parallelism of bacterial strains [11]. Mobile agents encapsulate a dynamic behavior within the network, adopting certain degrees of perception, cognition and control of the given environment and acting as local virtual machines with dedicated instructions and rules. An agent can migrate or clone from one node to another and communicate/coordinate its actions based on Linda-like tuplespaces [12] and federated system architecture.

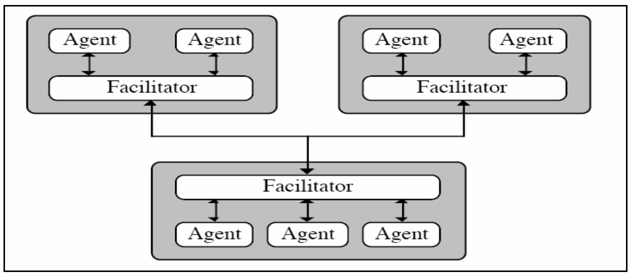


Fig. 1, A Federated System Model representation (Genesereth and Ketchpel from “Software Agents’’ 1994)

Eliminating communication cost and complexity, each facilitator agent will be responsible for a number of agents [10] as shown in Figure 1. Agents will not communicate directly but rather through the facilitators, which will communicate with each other. These facilitators will be governed by rules based on bacterial strains that will allow them to handle network requests in the same way bacterial strains metabolize energy sources. This will establish a high degree of independence and will remove high level network management problems [13]. Inner communication will be obtained by tuplespaces, a shared memory model where a tuple can be defined as an item of factual information which is available for retrieval by pattern matching. This will allow agents to insert or remove a tuple, containing for example a sensor reading, and thus interact without having to maintain knowledge of the current agent population.

Mobile agents can be defined as autonomous programs that can migrate from server to server and do not require continuous communication with the client towards an execution of a job [14]. These attributes contribute to the fact that mobile agents have better efficiency and more robustness than traditional approaches. Lange and Oshima have identified seven good reasons for using mobile agents [15]. The mobile agent paradigm has been used for many years; mainly for internet applications and the benefits are widely known. Some of the most famous platforms that were developed include Java Aglets [16], TACOMA [17], MARS [18] and Agent Tcl [19]. Their success was identified in areas such as data mining [20], e-commerce [21] and network management and coordination [22].

This paper explores whether the use of mobile agents using a federated communication combined with tuplespaces and behavioral instructions based on bacterial strains is technically feasible and beneficial to the application level of WSNs. The remainder of this paper is structured as follows. Section 2 presents In-Motes model and explains how mobile agents are bound with the coordination/communication model we are proposing for WSNs. Section 3 discusses a feasible In-Motes application and identifies how common problems in WSNs can be overcome. Section 4 presents the engineering effort, identifying our implementation tools and the first steps that are taken towards the completion of our middleware, as well as the near future aims of our research. The paper ends with conclusions in Section 5.

2 The In-Motes Model

The In-Motes model is shown in Figure 2. Each node can support multiple agents and maintains a tuplespace where specific reactions, that will be mentioned later, can be stored. This tuplespace will be accessible by all agents and is static as agents cannot carry it throughout the network.

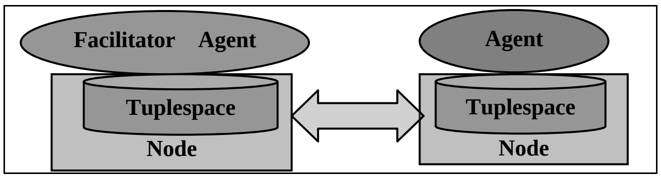


Fig. 2, The In-Motes Model Representation between two nodes. One is hosted by a facilitator agent and the other with an agent following the federated notion

An In-Motes application consists of autonomous agents of different types that are deployed in a given network. Because of this, the generation of a communication model that will allow coordination and cooperation between them is vital. In-Motes, unlike other middlewares provide this scheme with the use of a federated system combined with tuplespaces. Each node will maintain a tuplespace that will be shared by all the agents available at a given time, either remotely by the ones that will be hosted in other nodes or locally by the residing agent.

This shared memory model will be enforced in a federated system where the facilitator agent will generate specific instructions to the analogous node that hosts it. This will be allowed since each tuple will contain a specific set of fields with a type and a value. Types will consist of various sensor readings. Following the Agilla model each tuple will be accessible by any agent whose specific template matches by type. Such a match will be available if they have the same number of fields and each field in the tuple satisfies the analogous field in the template.

The In-Motes model takes inspiration from other well established middlewares [9, 22, 23, 24] and is using specific reactions that can be added in the tuplespaces. Reactions are used in order to ensure that agents operating inside the network are autonomous entities by allowing them to advertise what kind of template they are searching for. When a new tuple is added in the tuplespace of a specific node, the agent will be notified and if necessary will take action upon this notification.

As previously mentioned, In-Motes is based on deploying a network without any application needing to be installed. Agents that implement application behaviour will be injected later, setting up the communication model at the first stage and then handling user requests. The first stage is tailored alone with facilitator agents. The life cycle of a facilitator agent is shown in Figure 3.

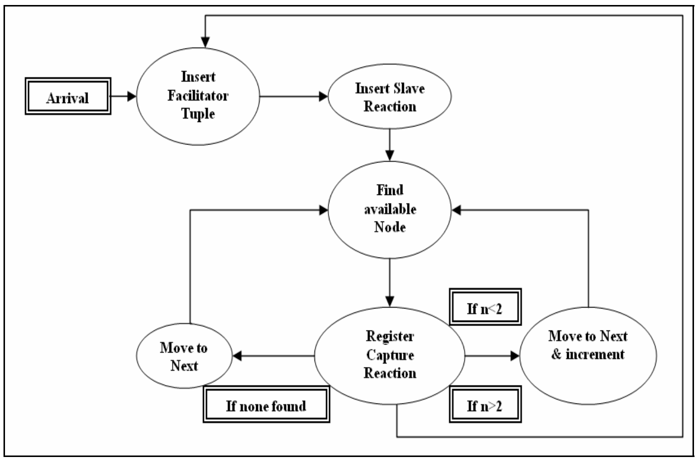


Fig. 3, The representation of a Life Cycle of a facilitator agent inside a wireless sensor network

The facilitator agent works by continuously checking whether any of the nodes are available for capture. The capturing procedure takes places when a facilitator agent during its migration registers a capture or a slave reaction to the analogous node. A counter will be incremented every time a capture reaction takes place; here we restrict the registration of two agents under each facilitator purely based on the motes we had available. When the counter reaches two, the facilitator agent will migrate again to the next available node assigning this time around a new facilitator tuple and slave reaction and the capturing procedure will repeat. Therefore the federated system communication model will be established covering all the available nodes in the network, as is shown in Figure 4.

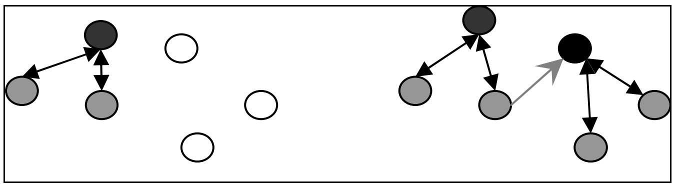


Fig. 4, Establishing the Federated System Model with the use of a facilitator agent, black colour, that is being injected in the network

The second stage of the In-Motes communication model deals with user requests and the execution of application specific jobs. As soon as the federated system is established, the user will be able to forward specific queries to the network. This will be allowed through the injection in the network of a job agent that will contain a specific query. The job agent will visit the facilitator agents of the network and, if allowed, will insert a job reaction. Each facilitator agent is governed by rules, as described in the work of Roadknight and Marshall [13] which make each node in a network responsible for its own behavior. As a result, their network was modelled as a community of cellular automata and each node envisaged as a bacterium and each request for service as food.

Inspired from the above ideas, the facilitators of the network will be governed by the following rules that will be stored in an acquaintance list:

- Each facilitator evaluates the items that arrives in its input queue on a FIFO principle
- Only four requests can be processed in a measurement period (epoch)
- The more time a facilitator spends processing a request, the busier it will appear to be. This busyness is calculated by calculating the pre and post busyness for the current epoch in a 0.8 to 0.6 ratio.
- If a facilitator's busyness is higher than 50% then the job is forwarded to the next available facilitator whose busyness is less than or equal to 50%

Optimization of the rules for a particular application scenario is the subject of ongoing work.

In-Motes middleware has to overcome specific limitations that are produced by the nature of WSNs. The MICA2 motes that we are using have only a 128kB of memory available for instructions and 4kB of data memory while the microprocessor is an 8MHz Atmel 128. Another limitation derives from the fact that TinyOS does not provide any dynamic memory management and as a result all the data memory must be allocated statically. Also, the small size batteries result in a low bandwidth wireless link of 38.4kbaud which can be considered quite unreliable due to his high error rate. To address these challenges, In-Motes adapts Agilla's memory management for its agent instructions and tuplespaces [8]. Also, each agent is divided into tiny packets that are migrated and can be retransmitted, minimizing the impact of message loss which is quite common in mobile agents.

3 An In-Motes Application Example

In-Motes middleware can be proved quite beneficial in areas such as energy and resource consumption as well as in information gathering and processing. Due to its hierarchical and structured communication model combined with the mobility and the benefits deriving from the agent infrastructure, In-Motes stands as a promising way of developing and writing applications for WSNs. In this section we are going to describe a simple novel application that deals with information gathering and processing. The application is shown in Figure 5.

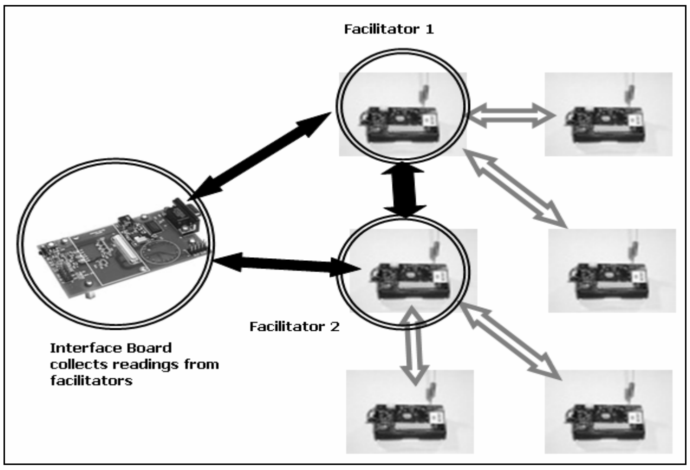


Fig. 5, A representation of an In-Motes application for information gathering and processing,

We assume that our nodes, originally with no preinstalled application, have been structured in a federated notion as described in Section 2. The user wants to gather sensor information regarding photo readings from the nodes. In-Motes will allow the injection of one or more job agents inside the network containing the request(s). At first, the job agent will try to find the federated agents by identifying which tuplespaces contain a facilitator tuple. Control will then be passed to the facilitator upon the arrival of the job agent to check if its behavioral rules allow it to forward the request to the assigned agents. If they do, then the job agent will be able to insert a job reaction containing the user's query. Hence an epoch will be created allowing only three more requests to be processed by the same facilitator for the current epoch providing an efficient energy framework for all the facilitator agents of our network, by eliminating excessive use of specific nodes. Once the results are gathered, the job reaction will be terminated and the job agent will report back to the user. If a facilitator busyness is higher than 50% it will direct the job agent to the next available facilitator of the network.

It is expected that during the lifetime of a WSN some nodes will eventually die and information will be lost. In-Motes can adapt and dynamically take actions upon unexpected scenarios like the once mentioned above. Going back to our previous example if a facilitator node goes down the network will dynamically adapt since the lifecycle of the facilitator agent that we described in Section 2 never terminates and a new capturing procedure will take place.

4 Engineering Effort and Future Work

This section presents the engineering effort and the near future work behind the In-Motes middleware. Currently we are working with a MICA2/DOT Professional Kit (MOTE-KIT 5x4) [25] as shown in Figure 6.

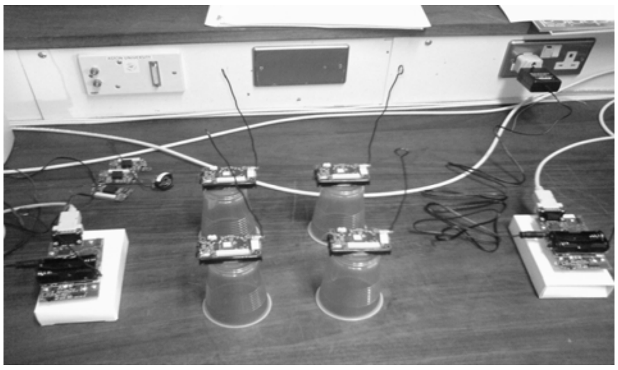
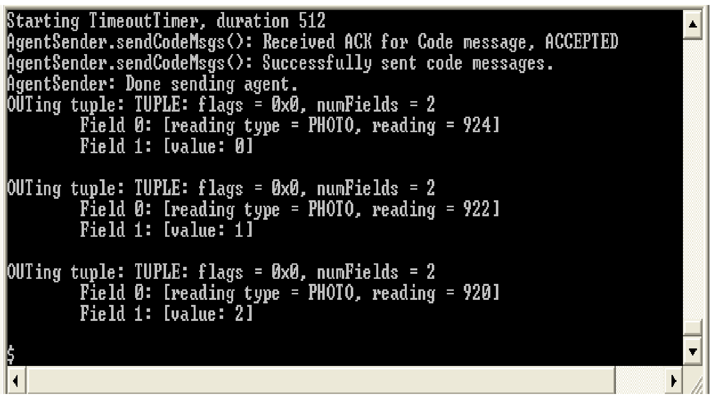


Fig 6, Our experimental motes kit with six MICA2 motes and two MIB510 Programming and Serial Interface Boards

The radio has an available range of 38 kbps over a range of 100m; however, the exact values are dependent upon the environment [26]. The microprocessor gives us 128kB of instructions and 4kB of data memory. MICA2 motes are widely used in many research projects around the globe. As we have described before, building applications for them is quite challenging mainly due to the limited amount of data memory, the unreliable low-bandwidth radio and the lack of substantial documentation that explores their features. Our test platform uses a desktop which monitors the WSN by obtaining feedback from the MIB510 Programming and Serial Interface Board.

MICA2 motes run under a specific platform named TinyOS [5] whose applications are divided into components that follow a specific hierarchy. The platform does not allow any dynamic memory management and as a result all application and system variables must be declared statically. Given that a program behavior is installed and deployed, the modification of it is very challenging since the motes must be retrieved and reprogrammed in the programming board. A middleware can hide or even overcome the above limitation. In order this to be applicable it is necessary for the middleware to provide high level communication architecture and allow quick implementation, testing and deployment of programmer's applications.

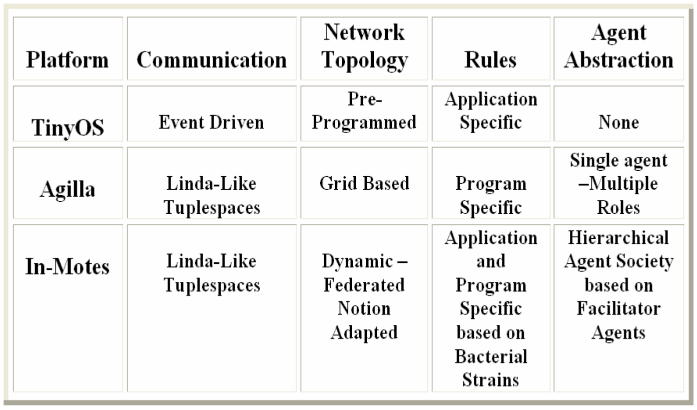
Currently we are using our agent architecture to modify, program and deploy the TinyOS applications that the platform offers, such as the Blink and the Oscilloscope [27], under our middleware with the use of agents and so far we have been successful as shown in Figure 7.



```
Starting TimeoutTimer, duration 512
AgentSender.sendCodeMsgs(): Received ACK for Code message, ACCEPTED
AgentSender.sendCodeMsgs(): Successfully sent code messages.
AgentSender: Done sending agent.
OUTing tuple: TUPLE: flags = 0x0, numFields = 2
        Field 0: [reading type = PHOTO, reading = 924]
        Field 1: [value: 0]

OUTing tuple: TUPLE: flags = 0x0, numFields = 2
        Field 0: [reading type = PHOTO, reading = 922]
        Field 1: [value: 1]

OUTing tuple: TUPLE: flags = 0x0, numFields = 2
        Field 0: [reading type = PHOTO, reading = 920]
        Field 1: [value: 2]

$
```

Fig. 7, In-Motes results after running the TinyOS oscilloscope application in our middleware

Also we have created an application, namely SenseLight, which we are going to deploy it in TinyOS, Agilla and In-Motes respectively generating some comparison graphs. In Figure 8 we present the main features of each platform/middleware.



Platform	Communication	Network Topology	Rules	Agent Abstraction
TinyOS	Event Driven	Pre-Programmed	Application Specific	None
Agilla	Linda-Like Tuplespaces	Grid Based	Program Specific	Single agent –Multiple Roles
In-Motes	Linda-Like Tuplespaces	Dynamic – Federated Notion Adapted	Application and Program Specific based on Bacterial Strains	Hierarchical Agent Society based on Facilitator Agents

Fig. 8, Comparison Table of the three testing platform/middlewares

SenseLight, is a multi-hop application that that uses mobile agents to collect light readings and report back to the end user. There are two types of agents in our society that will be generated in the WSN those of a Facilitator Agent (2) and Slave Agent (4).

Each slave agent will transmit 1 light reading per message to the analogous facilitator agent. Each facilitator agent was instructed to transmit back to the end user 2 light readings per message.

Running the Agilla engine in our motes, we deployed the same application with the difference that this time all the agents were reporting back to the end user 1 light reading per message based on the engine specification.

As can be seen from the below comparison graph, Figure 9, In-Motes produced better moving average of 2 light reading to the end-user over a period of 2 minutes.

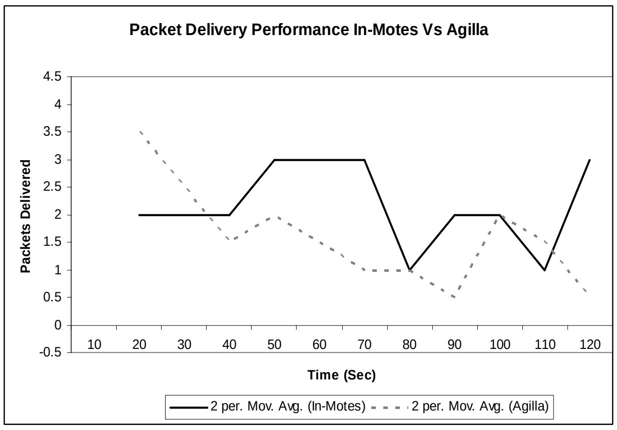


Fig. 9, Packet Delivery Performance of In-Motes and Agilla over a period of 2 minutes.

These modifications will allow us to understand better the TinyOS platform and also to specify and highlight certain behaviors that the agents of our middleware must have. We are currently developing a full version of In-Motes middleware which will be able to deploy all the TinyOS applications.

Future work will also include the creation of a powerful front end for the user, based on a blackboard scheme [28] where all the submitted queries will be handled automatically by the system and satisfied without the user having to monitor continuously the progress of the job agents.

5 Conclusion

In this paper, we have presented and analytically described In-Motes, an intelligent agent based middleware for WSNs. Having identified certain limitations in currently existing middlewares and in the application platform where motes are widely used, TinyOS, we are proposing an agent based middleware that tries to resolve these problems and create a flexible platform for programming, testing and deploying applications for wireless sensor networks. Our proposition brings together promising technologies, such as mobile agents, tuplespaces and federated systems under a common framework and provides a substantial communication and coordination architecture for the needs of a WSN. Our MICA2 implementation is already starting to demonstrate the feasibility of using mobile agents and tuplespaces under a federated notion where each agent is working based upon what is best for it and for the group that it belongs to. We envisage that In-Motes will serve as a foundation for rapidly building applications for WSN.

Acknowledgements
The authors would like to thank Dr. D. J. Holding, of Electronic Engineering Department of Aston University, for the immense help provided during the preparation of this paper. D. Georgoulas would like to acknowledge the support of Assistant Prof. M. Welsh of Harvard University, for his interest in In-Motes middleware, Mr C.L Fok of Washington University in St. Louis, for helping us understand the Agilla programming core, M. Smith, technical support engineer of Crossbow Technology, Inc for the technical advises he provided during the installation of the platform, and P. Trevis, technician of Aston University for the installation of the software packages.

References:

[1]- [2] D. Culler, D. Estrin, and M. Srivastava. Overview of sensor networks. *IEEE Computer*, 37(8):41–49, 2004.
[3] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, “System architecture directions for networked sensors,” in Architectural Support for Programming Languages and Operating Systems, 2000, pp. 93–104.
[4] P. Levis and D. Culler, “Mate: A tiny virtual machine for sensor networks,” in International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA, Oct. 2002.
[5] http://www.tinyos.net/tinyos-1.x/doc/Xnp.pdf. Last visited on 24/08/2005
[6] J. Hui and D. Culler, “The dynamic behavior of a data dissemination protocol for network programming at scale,” in Proceedings of the 2nd international conference on embedded networked sensor systems. 2004, pp. 81–94, ACM Press.
[7] P. Levis and D. Culler, “Mate: A tiny virtual machine for sensor networks,” in International Conference on Architectural Support for

Programming Languages and Operating Systems, San Jose, CA, USA, Oct. 2002.
[8] A. Boulis, C.-C. Han, and M. Srivastava, “Design and implementation of a framework for efficient and programmable sensor networks,” in Proc. of MobiSys, 2003.
[9] C.-L. Fok, G.-C. Roman, and C. Lu, “Rapid development and flexible deployment of adaptive wireless sensor network applications,” Tech. Rep. WUCSE-04-59, Washington University in St. Louis Department of Computer Science and Engineering, 2004.
[10] Genesereth, M, R and Ketckpel, S, P. “Software Agents”, In Communications of the ACM 37(7), pp. 48-53, 1994.
[11] C. Roadknight and I. Marshall, “Future Network Management – A bacterium Inspired Solution”, In Proceedings of Second International Symposium Engineering of Intelligent Systems, June 27-30, 2000
[12] D. Gelernter, “Generative communication in Linda,” ACM Trans. on Prog. Languages and Systems, vol. 7, no. 1, pp. 80–112, 1985.
[13] C. Roadknight and I. Marshall, “Management of Future Data Networks: An approach based on Bacterial Colony Behaviour”, Artificial Life, December 2001.
[14] Bradshaw, M, Jeffrey (1997). “An introduction in software agents”, In (Eds.) Bradshaw, M, Jeffrey in Software Agents, The MIT Press, 1997.
[15] D. B. Lange and M. Oshima. “Seven good reasons for mobile agents”, Commun. ACM, 42(3):88–89, 1999.
[16] P.E.Clements, T. Papaioannou, and J. Edwards. “Aglets: Enabling the virtual enterprise”, In Proc. of the Int. Conf. on Managing Enterprises - Stakeholders, Engineering, Logistics and Achievement, 1997.
[17] D. Johansen, R. van Renesse, and F. B. Schneider. “An introduction to the TACOMA distributed system”, version 1.0. Technical Report 95-23, University of Tromso, Tromso, Norway, June 1995.
[18] G. Cabri, L. Leonardi, and F. Zambonelli. “MARS: A programmable coordination architecture for mobile agents”, Internet Computing, 4(4):26–35, 2000.
[19] R. Gray.Agent Tcl. Dr. Dobb’s Journal of Software Tools, 22(3):18–71, 1997.
[20] D. B. Lange and M. Oshima. “Seven good reasons for mobile agents”, Commun. ACM, 42(3):88–89, 1999.
[21] M. Baldi and G. P. Picco. “Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Application”. n R. Kemmerer, editor, Proceedings of the 20th International Conference on Software Engineering, pages 146–155. IEEE Computer Society Press, 1998.
[22] P. Maes, R. H. Guttman, and A. G. Moukas. “Agents that buy and sell” Communications of the ACM, 42(3):81–91, 1999.
[23] C. Julien and G.-C. Roman, “Egocentric Context-Aware Programming in Ad hoc Mobile Environments”, In Pro. of the 10th Int. Symp. on the Foundations of Software Engineering, pages 21–30, Nov. 2002.
[24] A. L. Murphy, G. P. Picco, and G.-C. Roman. “LIME: A Middleware for Physical and Logical Mobility”, In Proceedings of the 21st International Conference on Distributed Computing Systems, pages 524–533, April 2001.
[25] http://www.xbow.com/Products/ Last visited on 24/08/2005
[26] J. Zhao and R. Govindan. “Understanding packet delivery performance in dense wireless sensor networks”, In Proc. Of the ACM SenSys, 2003.
[27] http://www.tinyos.net/. Last visited on 24/08/2005
[28] Englemore R, Morgan T, “Blackboard systems”. AddisonWesley, Reading, MA, 1988