Towards a FIPA Compliant Multiagent based Middleware Architecture for Sensor Networks

Khalid Nawaz
Databases and Dist. Systems Group
Dept. of Computer Science
TU Darmstadt, Germany
khalid@dvs.tu-darmstadt.de

Pablo E. Guerrero
GK Electronic Commerce
Dept. of Computer Science
TU Darmstadt, Germany
guerrero@dvs.tu-darmstadt.de

Alejandro P. Buchmann
Databases and Dist. Systems Group
Dept. of Computer Science
TU Darmstadt, Germany
buchmann@dvs.tu-darmstadt.de

***Abstract*—In this paper we suggest a multiagent based middleware architecture that is particularly suitable for supporting wireless sensor network deployments in industrial environments like mines and chemical processing plants. Our intention is to make the proposed middleware architecture to be as closely aligned to the Foundation for Intelligent Physical Agents standards as possible. The existing agent based solutions for wireless sensor networks lack some important features that these standards mandate and thus reduce their interoperability with other existing or upcoming agent based systems. These standards that we choose for our middleware have the potential of providing a more reliable event reporting mechanism than the one supported by existing agent based middleware.**

I. Introduction

Wireless Sensor Networks (WSNs) are composed of tiny microelectromechanical devices with sensing and wireless communication capabilities. Their use is on the rise in different application scenarios since their introduction less than a decade ago. As is true of any new technology, their rapid acceptability is also occasionally marred by technical difficulties arising due to the specific requirements posed by different application scenarios. For instance, in industrial setups like underground mines, it is very difficult to ensure the connectivity of the deployed WSN due to the geometry of these underground structures. This difficulty in wireless communication underground, coupled with inherent problems of node failures due to running on low battery power, results in partitions in the deployed WSN. Additionally, in almost all industrial setups, it is one of the basic worker safety issues to ensure a safe working environment for them. Coming back to the scenario of an underground mine, safety engineers might want all the miners to know the average temperature and concentration of other hazardous gases in close proximity to them. Similarly, in a chemical processing plant, workers should be aware of the environmental conditions around them and should immediately be informed in case any chemical spills happen around their workplace. Any WSN deployment in such industrial setups should meet this Proximate Environment Monitoring (PEM) [3] requirement of the users.

In addition to PEM, another requirement on such a deployment could be not only to detect hazardous gaseous mixtures (or chemical fumes) but to track them too. This tracking should be done while keeping the nearby miners informed about it. Additionally, some sort of actuation support might also be required in most cases. These actuation decisions should be performed in the network under very tight timing constraints without going through a fixed base station, if there is any. On top of that, one might also want the deployed WSN to be shared by multiple applications performing different tasks like detecting and tracking gas plumes, controlling mine lighting system and monitoring the structural health of the mine. Furthermore, achieving reasonable levels of fault tolerance in WSN applications is almost always desired. However, these high levels of fault tolerance are not always easy to achieve especially in the event of frequent node failures and problems related to wireless communication in the environmentally challenged scenarios under consideration. All these requirements dictate having a middleware that allows dynamic and proactive behavior. We argue that this behavior can best be provided by Multiagent (MA) based middleware coupled with mobile base stations to cope with the sparse WSN topology in the environmentally challenged scenarios under consideration. Since in literature the term agent based middleware and MA based middleware has been used interchangeably, we use both these terms in this paper. The term MA based middleware encompasses the other though. Since the middleware architecture that we suggest in this paper is based on MA paradigm, it is appropriate to present a brief introduction of agents first.

A. Agents and their Benefits in WSNs

According to [8], an agent is anything that can perceive its environment through sensors and act upon that environment through actuators. When such an agent tries to optimize some performance measure it is called a rational agent. A MA system consists of several such agents that can interact with each other to achieve their assigned goal. One important characteristic of an agent is that it can move from one node to another in carrying out its assigned task. Consequently, there are two types of agents in a MA system, namely, static and mobile. Though this agent mobility concept has its roots in mobile code paradigm, there is a slight difference between the two. As opposed to simple mobile code, a mobile agent starts its execution at each node from the same point at which it leaves it in the previous node. For this purpose, mobile agents have to carry their code, data, and state information across migrations.

978-1-4244-1653-0/08/$25.00 © 2008 IEEE

Agent based middleware for WSNs has the potential of providing proactive reprogramming of the network along with benefits like in-network decision making and actuation [4]. Some of the other most prominent benefits of agent approach in WSNs include better energy usage of the network nodes by doing *in-network data aggregation*; fault tolerance by avoiding agent visits to the nodes that are running low on battery; and suitability with the disconnected nature of WSNs by injecting agents using handheld gadgets. Agent based approach also allows multiple agents to run on a single node performing different functions, thus effectively sharing network nodes for multiple applications [4].

B. Problems with Existing Agent Based Approaches in WSNs

There are some successful implementations of agent based middleware in WSNs like Agilla [4] and TinyLime [6], both of which are based on the tuplespace paradigm [7]. Agilla is an agent based middleware that uses agent mobility feature to support scenarios like detecting and tracking wildfires. However, the current agent based middleware for WSNs, e.g. Agilla, not only lack compliance to *Foundation for Intelligent Physical Agents* (FIPA) standards [1] but also elaborate mechanisms to support reliable communication between agents. Agilla has been developed on top of TinyOS [2] and its agents communicate with each other using a shared memory paradigm called tuplespaces. Though the tuplespace paradigm provides a decoupled way for agents to communicate with each other, it has its problems when it comes to reliable event reporting between agents. Agilla has extended tuplespaces with a mechanism called reactions. Reactions are a way for agents to insert template tuples in the local tuplespace of a node telling middleware to inform them if a matching tuple is inserted by some other agent. However, these reactions work only locally within a single node in Agilla, effectively limiting event reporting to a single node. Agilla also provides remote tuplespace operations that agents can use to coordinate their activities. However, these operations require location parameter to work. These remote tuplespace operations can only work on tuplespaces of one-hop neighbors of a node, since Agilla maintains neighbor-list on each node containing location information of only one-hop neighbors of that node. All this implies that when an agent detects an event its inserted tuple in the local tuplespace can only trigger reactions on that particular node. In case of a fire detection and tracking scenario, the agent detecting the event fire would insert a tuple in the local tuplespace of that node on which it is currently running. The fire tracking agent has to be on the same node in order for it to be informed about this event or both capabilities of fire detection and tracking should be delegated to a single agent. In the latter case, the size of the agent would become very large, thus incurring high communication costs while migrating node to node in the network. In our middleware architecture, we have addressed this uncertainty problem by providing a mechanism for agents to find the location of other agents. Agents detecting an event can query the location of agents interested in that event from Cluster Directory Agent (CDA) maintained at the cluster head(explained in §III-A). This makes them communicate with each other easily and removes the uncertainty involved in event reporting process.

C. What is FIPA Compliance?

We strongly believe in standards compliance for agent based middleware in WSNs, since it brings twofold advantage. First, it provides wealth of knowledge that could be utilized to build better systems. Second, different agent based middleware for WSNs could be made interoperable, if they follow FIPA standards. We believe that the next step in agent based middleware approaches for WSNs is to move towards standards compliance. Our middleware architecture is an effort to achieving the same goal.

FIPA was established in 1996 as a standardization body for developing standards for software agent technology [5]. It has, since then, defined several standards that MA systems can be made compliant to. Some of them are suitable for resource rich environments and, we believe, some are also suitable for resource constrained environments. We are focusing on making our middleware architecture compliant to those FIPA standards that are suitable for resource constrained environments. These standards mandate having an Agent Management System (AMS), a Directory Facilitator (DF), and Agent Communication Language (ACL) standard for communication between agents in a MA system.

The rest of the paper is organized as follows. In Section II, we mention some related work on agent based systems and other related techniques in WSNs. Section III describes our middleware architecture. Section IV describes a simple mechanism of choosing a cluster head in our approach. Section V describes one possible way for event detection using our approach. Finally, Section VI provides some concluding remarks.

II. Related Work

There are several successful implementations of multiagent based middleware in the traditional distributed systems like JADE [12] and AGLETS [9]. Java Agent Development Framework (JADE) is the most well known FIPA compliant open source middleware platform in traditional distributed systems. For WSNs, multiagent paradigm is relatively a recent idea that has been attempted by few researchers. One such effort is Agilla, also mentioned in the previous section, which is a mobile agent based middleware developed on top of TinyOS for WSNs. One specific application of this middleware, mentioned by its developers, is in fire fighting applications. They show the flexibility of their approach in detecting and tracking fires along with few other applications. This middleware not only lacks compliance to FIPA defined standards that should be present in a system for it to be interoperable with other FIPA compliant agent based systems but also an unreliable event reporting mechanism as explained earlier in Section I-B.

The use of agents has also been suggested for power management in WSNs in [10]. Authors suggest using interpolation as a technique by the agents to decide the redundant nodes

which could be put to sleep in a WSN. Maté [11] introduces the idea of writing WSN programs in TinyScript which is a scripting language that is compiled into executable bytecode for an application specific virtual machine. This allows a program to be flooded in the network until all the nodes have a copy of it. It is intended to retask a WSN with a single program at a time. This limits its flexibility to some extent. The MA based approach is more flexible than this in the sense that agents in a multiagent system can be assigned to different tasks. In [6], authors suggest an extension to Lime middleware [7] for mobile and ad hoc networks and name it TinyLime. TinyLime makes sensor data available through tuplespace interface, thus, effectively providing an illusion of shared memory between applications. Though it provides notions of proximate and context aware settings for WSN applications, its lack of support for multihop communication and restriction that the mobile data collectors could only collect data from the nodes within one hop range limits its flexibility.

In [13], authors suggest a multilayer cognitive agent framework for hybrid distributed sensor networks. They claim to present a FIPA compliant architecture but their notion of sensors includes devices like palmtops, smart sensors, laptops, single board computers and mobile robots. Region Management Stations (RMS) and Application Agent Platforms (AAP), that are connected to wireless mobile nodes and wired network nodes respectively, are also powerful desktop workstations. The four types of agents, namely, sensor agent, service provider agent, application agent, and interface agent that they suggest all run on either AAP or RMS.

Approaches like Couger [15] and TinyDB [14], that model a WSN as a distributed database system, has the advantage of providing easy to write declarative queries. These queries provide the user with a very easy method of interacting with a WSN without specifying any execution details. These approaches normally develop a semantic routing tree rooted at the base station that is used to disseminate queries in the network as well as to collect the result back at the root. Considering the mobile base station scenario like in an underground mine, the cost of developing a new semantic routing tree each time rooted at the current location of the user and associated communication costs, both in developing the routing tree and subsequently servicing the user's query, would be prohibitively high [3].

III. Middleware Architecture

In our middleware approach, there are two types of agents in the system, namely, management agents and application agents. The management agents are part of the middleware and perform management tasks, whereas the application agents are part of the application layer and perform user assigned tasks. In addition to components modeled as agents in the management layer, there are non-agent components also. These non-agent components, namely, agent mobility manager and communication subsystem (see Figure 1), provide agent mobility and message communication services respectively. Additionally, two other components, namely, Neighbor List Manager (NLM) and Local Sensor Data Acquisition Component (LSDAC) are modeled as non-agent components. The reason behind modeling some of the management components as agents is to make our middleware more flexible. In the event of a cluster head running on low battery, the agent mobility manager can migrate these management agents on any of the neighboring nodes thus assigning it the role of a new cluster head as we explain in Section IV. Therefore, the components that are responsible for managing a cluster are modeled as agents and other components that are present on all nodes are modeled as simple components.

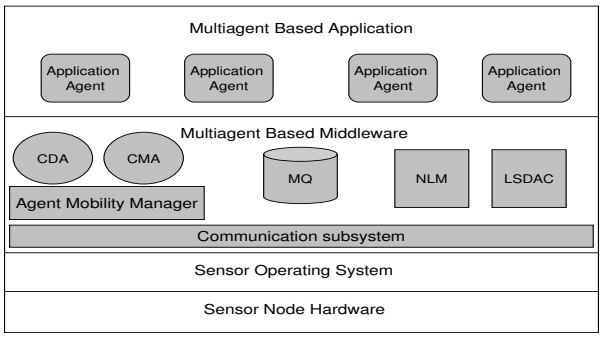

Fig. 1. Middleware Components on Cluster Heads

Our middleware's network topology is based on the concept of clustering, which implies that the WSN is divided into clusters each managed by a cluster head. Consequently, there are some additional management agents on cluster heads that are not present on cluster members. This approach helps save on communication costs by facilitating more localized communication between cluster heads and cluster members.

A. Middleware Components on Cluster Heads

The following are the management agents that are present on cluster heads.

Cluster Management Agent: Cluster Management Agent (CMA) acts as a controlling authority in a cluster. All the application agents, present in its managed cluster, are registered with it. It controls their life cycle and can create/activate or terminate any of them. It shares its agent termination authority with Node Management Component (NMC) that is present on each member node of a cluster. The inclusion of the CMA makes our middleware compliant to the FIPA standard that mandates having an AMS in a MA system.

Cluster Directory Agent: All the application agents in a cluster, along with the services they offer, are registered with it. If an application agent needs a service offered by some other application agent in the same cluster, it can query CDA. All agents' ids providing the desired service are returned to the querying agent which can communicate with any/all of them. Both CMA and CDA access the same repository maintained on the cluster head. This arrangement is made keeping in mind the constrained resources of wireless sensor nodes. It also eliminates the need of synchronizing two different repositories, if CDA and CMA were maintaining them separately. This

repository has a unique ID for each agent running in the cluster. The uniqueness of this ID is ensured by CMA. Since each application agent in our middleware architecture offers just one service, it is easy to maintain just one record for each agent. Example of a service offered by an agent could be determining average gas concentration, average temperature of a set of nodes, gas detection, gas plume tracking etc. The inclusion of this agent makes our middleware compliant to the FIPA standard that mandates having a DF in a MA system.

B. Middleware Components on Member Nodes

The following types of Components, except for the Node Management Component (NMC), are present on all the nodes including cluster heads.

Local Sensor Data Acquisition Component: The task of the Local Sensor Data Acquisition Component (LSDAC) is to get the local sensor data from the node. If a node has multiple sensors, then it gets data from all the sensors on a periodic basis. This component can be tasked to get data from different sensors in a node at different rates. It makes this sensor data available to application agents by placing it in a repository that contains data freshness parameter also. This component can also maintain a history of readings of sensors and can be configured according to the application requirements.

Node Management Component: Each node, except for the cluster head, has a Node Management Component (NMC) whose task is to register the node with the CMA of the cluster that it is part of. It is also responsible for reporting the arrivals of application agents on a node. As soon as an application agent arrives at a node, NMC registers it with the CMA except for when it migrates from a cluster head node to a member node. When an application agent migrates from a member node of one cluster to a member node of a neighboring cluster, then the NMC of the source node sends a leave message to its cluster head and the NMC of the destination node sends an arrival message to its cluster head. There is no need to re-register application agents with the CDA after each migration, since the service that application agents offer stays the same across migrations. However, when an application agent arrives at a node from a neighboring cluster's member node, then its offered service is also registered with the CDA of the destination node's cluster head. This service registration message is combined with arrival message that is sent to register the application agent with CMA of cluster head. The NMC also shares the authority of removing an agent to free up some resources for high priority agents running on the node. When it does that then it needs to inform the cluster head's CMA and CDA by sending a single message, so that they can remove that application agent's record from the repository that they maintain.

Neighbor List Manager: It keeps an updated list of all the one hop neighbors of a node. As soon as an agent residing on a node wants to migrate to any of the neighboring nodes, it can get the neighboring node's id from this list. The NLM periodically updates this list so as to remove any neighbors that might have died or moved out of one hop distance from the node.

Agent Communication Language: FIPA standards provide an *Agent Communication Language* (ACL) that agents can use to communicate with each other. In our suggested middleware architecture, agents communicate with each other using an asynchronous communication paradigm. Each node has a Message Queue (MQ), maintained by the communication subsystem, containing messages for agents along with their ids. When a message arrives at a MQ of a node, the corresponding agent is informed by the communication subsystem of the middleware. Agent communication can take place between agents residing on a single node or between agents residing on different nodes. Since messages are sent by agents after getting the position of their communication counterpart agents from cluster head's CMA and/or CDA, the communication can always be unicast and more reliable. However, there is also a possibility of broadcasting a message to all one hop neighbors of a node in case of some critical event detection. For the sake of brevity, we don't mention message primitives provided by the FIPA ACL standard here.

IV. Choosing a Cluster Head

In our middleware architecture, cluster heads perform some additional management functions than the member nodes of a cluster. Therefore, the choice of cluster head is important to ensure the longevity of the network. Initially, the role of a cluster head is assigned at the time of deployment. During operation of the network if the cluster head runs low on battery at some point in time, it transfers the management agents and associated repositories to one of its neighboring nodes that still has more battery power. Since all the management agents are modeled as agents, they can be migrated to other neighboring nodes by the agent mobility manager. If all the nodes in the neighborhood of the cluster head also run low on battery, then the search for a node to act as a new cluster head is extended to two hop neighbors and this process continues until an alternate node for the role of cluster head is found or the current cluster head dies due to low battery. If it dies due to low battery, then the member nodes of that cluster have no other option but to join other clusters. Their NMCs can broadcast a fresh cluster joining request with progressively increasing radio power. If they are unsuccessful, then they have no other option but to serve the rest of their lives as standalone nodes serving application agents, if they receive any.

V. Event Detection Process

We elaborate the event detection process by giving an example from an underground mining scenario (Figure 2). Each mining team has at least one handheld device that acts as a mobile base station. This mobile base station could be used to inject Gas Detection (application) Agents (GDAs) in the deployed WSN. Note that the use of mobile base stations can help tackle the problem of connectivity of nodes with a fixed base station, if there is one. The injected application agents could be tasked to move proactively to the nodes observing higher than usual gas concentration or to perform PEM. As

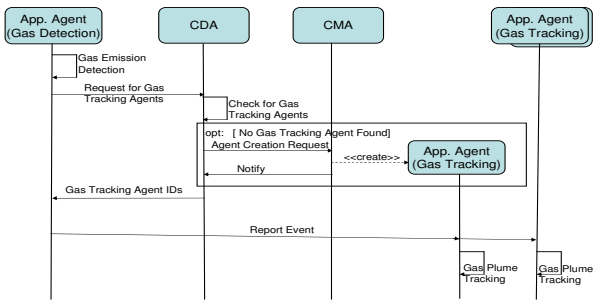

Fig. 2. Gas Plume Detection and Tracking it

soon as an agent is injected in the network on a node, NMC of that node registers it with the cluster head's CMA and it's offered service with the CDA by sending a single message. If the application agent needs some local data of the sensors of that node, it can directly query the LSDAC and migrate to any of the neighboring nodes after consulting the neighbor list maintained on the node by the NLM. If an application agent decides to search for some service that is offered by some other agent in that cluster, it can query the CDA of the cluster head remotely. Since each node is already registered with a cluster head, agents present on either of them can easily communicate with the other. Application agents can be terminated either by the CMA or by NMC. In that case they are unregistered from CMA and CDA both by sending a single message. This agent termination feature is useful in those situations where resources for high priority agents are needed.

Once emission of a gas is detected by any of the GDAs, it requests the CDA for a list of all the Gas Tracking Agents (GTAs) that are present in that particular cluster. If there are any GTAs present in the cluster, the CDA provides this information to the requesting agent. If there are not any GTAs present in the cluster, the CDA requests CMA to create/activate a GTA. After creation/activation of a GTA, the CMA notifies the CDA about it. The CDA in turn communicates the ID of the created agent to the requesting GDA that can communicate the detected event to it. Then GTA can follow the body of the detected gas by migrating to neighboring nodes and by cloning itself. These GTAs can migrate to other nodes by querying the neighboring nodes' LSDACs. GDAs and/or GTAs on the node(s) that have observed increased gas concentration follow the same process to look for an actuation agent in the cluster, for performing some action like turning on the ventilation fan in the mine, by querying the CDA. Since it is very likely that more than one GDA detect higher gas concentration and all of them inform the corresponding actuation agent, it is very likely that this redundancy will ensure a more reliable event reporting to the actuation agent that can cause the desired actuation. Note that having this cluster based strategy, along with management agents like CMA and CDA present on the cluster heads, brings twofold benefits in event detection process. First, it makes it easier for an agent that has detected an event to communicate it to the corresponding actuation agent. Second, it keeps the communication among agents localized inside a cluster as opposed to the case where agents have to communicate with a fixed base station to get location information of other agents.

VI. Conclusion

We have suggested a MA based middleware architecture with potentially a more reliable event reporting mechanism than the one supported by existing agent based middleware. We elaborated the use of our middleware with an example scenario from a WSN deployment in an underground mine. A subset of those FIPA standards that are suitable for the resource constrained environment of WSNs is included in the suggested middleware. It is an ongoing work and we are working on developing the first prototype of this system.

Acknowledgment

The authors would like to thank Daniel Jacobi, Arthur Herzog, Ilia Petrov and Bettina Kemme for the fruitful discussions and for their valuable feedback on this work. We would also like to thank DAAD, HEC Pakistan, and DFG Graduiertenkolleg GRK 1362 at TU Darmstadt for providing financial support for this work.

References

[1] FIPA Abstract Architecture Specifications at www.fipa.org
[2] www.tinyos.net
[3] K. Nawaz and A.P. Buchmann. Building a Case for FIPA Compliant Multi-agent Based Approaches for Wireless Sensor Networks. *In Proceedings of 1st ICST Int. Conf. on Ambient Media and Systems. Software Organization and Monitoring of Ambient Systems Workshop. Quebec City, Canada, Feb. 2008.*
[4] C.L. Fok, G.C. Roman, C. Lu, Mobile Agent Middleware for Sensor Networks - An Application Case Study, *In Proceedings of the 4th Int. Conf. on IPSN, Los Angeles, California, April 2005. pp. 382-387.*
[5] F. Bellifemine, G. Caire and D. Greenwood. Developing Multiagent Systems with JADE. John Willy and Sons Ltd., Jan. 2008.
[6] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. L. Murphy and G.P. Picco. Mobile Data Collection in Sensor Networks: The TinyLime Middleware. *In Journal of Pervasive and Mobile Computing, vol. 4, no. 1, pp. 446-469, Elsevier, Dec. 2005.*
[7] A. L. Murphy, G. P. Picco and G -C. Roman. LIME: A Middleware for Physical and Logical Mobility. *In Proceedings of the 21st IEEE Conf. on Distributed Computing Systems, April 2001.*
[8] S. Russell and P. Norvig. Artifcial Intelligence: A Modern Approach. Prentice Hall, 2003.
[9] D. B. Lange and M. Oshima. Programming and Deploying Java Mobile Agents with Aglets. Addison-Wesley Publishing Company, 1998.
[10] R. Tynan, D. Marsh, D. O'Kane, and G. O' Hare. Agents for Wireless Sensor Network Power Management. *In Proceedings of 2005 Int. Conf. on Parallel Processing Workshops, June 2005.*
[11] P. Levis and D. Culler. Maté: A Tiny Virtual Machine for Sensor Networks. *In ASPLOS, San Jose, CA, Oct. 2002.*
[12] F. Bellifemine, A. Poggi and G. Rimassa. JADE: Java Agent Development Framework. *Proceedings of the 4th Int. Conf. on Practical Applications of Intelligent Agents and Multiagent Technology, London, pp. 97-108, April 1999.*
[13] P. K. Biswas. Architecting Multi-Agent Systems with Distributed Sensor Networks. *IEEE Int. Conf. on Integration of Knowledge Intensive Multi-Agent Systems, Waltham, MA, April 2005.*
[14] S. R. Madden, M. J. Franklin, J. M. Hellerstein and W. Hong. TinyDB: An Acquisitional Query Processing System for Sensor Networks. *In Journal of ACM Transactions on Database Systems, vol. 30, no. 1, pp. 122-173, ACM Press, New York, USA. March 2005.*
[15] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *In Journal of ACM SIGMOD Record, vol. 31, no. 3, pp. 9-18, ACM Press, New York, USA, Sep. 2002.*