A Middleware Architecture Supporting Native Mobile Agents for Wireless Sensor Networks

Ciarán Lynch and Dirk Pesch

Centre for Adaptive Wireless Systems,
Cork Institute of Technology,
Cork, Ireland
{ciaran.lynch,dirk.pesch}@cit.ie

Abstract. Mobile Software Agents are widely used in telecommunication networks and the Internet, however their application to embedded systems such as Wireless Sensor Networks is immature. We present a novel middleware supporting and enabling Mobile Agent applications to run natively, without any translation layer, on Wireless Sensor Networks. We establish that Mobile Agent systems are beneficial for a wide range of applications – particularly when dealing with complex, dynamic and spatially distributed tasks, and demonstrate their power and certain performance metrics for an example applications. We use an accurate emulation platform to evaluate the system performance in a distributed control application implemented using mobile software agents.

1 Introduction

The challenges of reprogramming Wireless Sensor Networks (WSN) after they have been deployed into the environment are well established [1,2]. Reprogramming an entire sensor node program image is a time- and power-consuming process. Resetting the node also destroys any program state that has been established at the node.

Mobile agent systems have been proposed for WSN, however they either do not execute directly on sensor nodes or use an interpretation layer on the nodes. The system presented here uses native mobile agents, written in standard C, that execute directly on the embedded wireless sensor nodes.

Currently, there are no systems supporting true native mobile software agents on wireless sensor networks (see Related Work, Section 5). This paper presents a possible design of such a system. The proposed system builds on top of SOS [3], an existing modular operating system for WSN. Module support is tightly integrated into SOS and we found it a good base on which to develop such a system. The primary contribution of this paper is the development and evaluation of such a native mobile agent middleware system.

In order for such a system to be useful, it must minimise the cost of operation – primarily the time and energy used to transmit the agent code. The middleware system provides simple routing, reliable migration and remote module fetching, neighbour discovery and inter-agent communication. A weak mobility model is supported.

The architecture of the system is discussed in Section 2, and a simple application to evaluate it in Section 3. Section 4 discusses the results presented in the previous section. Related work is presented in Section 5, while Section 6 concludes the paper.

2 System Architecture

The Mobile Agent System Architecture is shown in Figure 1. It is split into a number of sub-sections. Agents execute as SOS modules, interacting with the middleware when they require the services it provides, running on standard MicaZ sensor nodes.

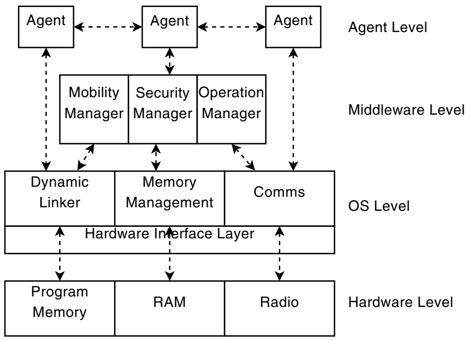

Fig. 1. System Block Diagram

The middleware must provide three types of services – Mobility, Security and Operations. The system is split into these three, essentially independent sections. The mobile agents as well as the middleware components are treated as normal code modules in SOS, with access to all of the operating system functionality.

2.1 Mobility Manager

An agent requires the ability to move. In the simplest case, an agent can move to another node within radio range. More complex movement operations enable more dynamic and powerful agents.

2.1.1 Agent Transfer

The manager controls the transmission of agents to neighbours. This can be either a move or a copy. The agent is suspended in order to capture a consistent snapshot of the agent state. An agent to be moved remains suspended until the transmission completes – if successful, the agent is then terminated, if not it resumes with an error signal.

A request is sent to the transfer node, and if it accepted, the transfer starts. The executable code is transmitted if necessary, followed by the agent state, including any dynamic memory allocated by that agent.

Once the transfer has started, the sending node sends packets regularly. Each packet carries 64 bytes of information. Selective-Repeat ARQ is used to ensure that all packets are received successfully. Since the size of an agent is generally limited to 10–20 packets, a sliding window would not be efficient. SR-ARQ gives better performance than simple ARQ in a lossy link, without overly complex resource requirements at the receiver.

2.1.2 Conditional Move

An agent may specify a move condition, allowing the agent to move to a point of interest in a network and not a neighbour. Intermediate nodes will still receive a copy of the agent, however they will not excecute it unless it has reached the end of the carrier path (the code is stored at the node for potential execution in the future). The condition has two parts, a move condition and a termination condition. The move condition is followed, until the termination condition is satisfied, or no further movement is possible.

Four carrier move conditions are currently defined – these are sufficient for a simple application, more may be added if required. They are *Move towards node N*, *Move along a gradient of a blackboard value* (see Section 2.3.1), *Move towards a gateway node* and *Move to a random neighbour.*

Five termination conditions are defined – *Execute at node N*, *Execute at the top or bottom of gradient*, *Execute at non-zero blackboard value*, *Execute at gateway node* and *Move once and then execute.*

2.2 Security Manager

A node should not blindly trust every agent received. Malicious users can inject improper agents, and errors or interference can result in an agent transmission becoming corrupted. A native agent does not have the protection of a bounded, virtual execution environment, and a corrupted agent will most likely crash the node. An MD5 signature [4] is used to sign each agent with a secret key, known only to the nodes in the system and to authorised users of the system.

2.2.1 Secret Key

Central to the signature scheme is a secret key known to each sensor node and to authorised users. This key is assumed to have been placed into each sensor node at the time of deployment or through a separate key distribution protocol.

2.2.2 Agent Integrity

The transmission fails if the hash does not match that generated locally. The sending or requesting agent is notified that the transmission has failed due to a security failure, and it is then free to take whatever subsequent action it deems appropriate.

The hash also operates as an integrity check – a corrupted agent that is not detected by the per-packet CRC check, or any misconfiguration in the middleware that causes the received image to be incomplete or incorrect will be rejected.

2.3 Operation Manager

2.3.1 Blackboard

The blackboard is a remotely accessible address space (similiar to Agilla Tuples [5]). A 16-bit key is used to index the space. The blackboard provides a simple, standard mechanism for discontinuous communication between agents in a local one-hop network neighbourhood. It allows agents to deposit information to be used in the future by other agents.

2.3.2 Neighbour Management

The manager maintains a list of neighbours. Agents can retrieve the list of current neighbours, query for new neighbours and select a random neighbour. Nodes are automatically removed from the list after a certain time.

2.3.3 Remote Agent Request

An agent requests the manager to find and execute another agent. If the agent code is available at the node, it is started. If not, a request for that agent is broadcast. Neighbouring nodes will forward this request and respond if the service is available. The agent is transferred by the mobility manager.

3 Evaluation

The middleware system was implemented and tested on MicaZ wireless sensor nodes. The application evaluation presented here is based on the AvroraZ emulation platform, as this provides detailed probes and instrumentation for measurement and testing, and allows repeatable and controllable experiments to be performed.

3.1 AvroraZ

AvroraZ [6] is an extension of Avrora, a wireless sensor network emulator. It uses a detailed model of the AtMega128L microcontroller to execute the same binary image as the physical microcontroller. It reproduces precisely the constraints of an embedded wireless sensor platform, allowing very accurate emulation of complex applications, complete with the timing and memory constraints of the physical system.

AvroraZ adds support for the CC2420 radio used in the MicaZ wireless sensor node platform. The radio model used is based on actual measurements of the performance of MicaZ nodes, and has been shown to correspond well with measurements from real sensor nodes [6].

3.2 Physical Scenario

The simulated scenario is a distributed control application. 25 nodes are placed in 3 rooms, with 3 nodes connected to heaters. The system is self-configuring, self-managing and robust – once one agent is pushed into the network, it operates autonomously, even in the presence of node resets and failures.

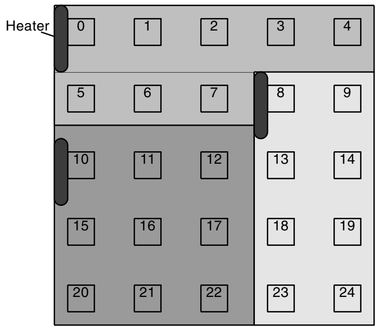

Fig. 2. Physical Scenario

The nodes are arranged as shown in Figure 2. Each node knows which room it is in (from 1 to 3) and whether or not it is attached to a heater. Every node has a temperature sensor. Each heater can be in one of 4 states – Off, Low, Medium or High. Each simulation is run for 6 minutes in total.

3.3 Agents

The agent sizes are shown in Table 4.

3.3.1 Management Agent

The management agent moves around the network, discovering nodes as it moves. On discovering a node connected to a heater, it summons a control agent if none exists. The control agent is given the list of nodes in the current room. This is updated each time the management agent revisits the control agent. The management agent keeps a list of nodes that were previously reachable but that are currently unreachable.

3.3.2 Control Agent (Static)

Every 10 seconds, it summons a data gathering agent to take readings from around the room. Once started, the control agent expects to get regular visits from the management agent. If one minute passes without any visit, the control agent will summon a new management agent, which will begin rediscovering the network from the current node.

3.3.3 Data Gather Agent

This agent moves around each node in the room and takes a sensor reading. It returns these readings to the control agent.

3.3.4 Reporting Agent

The reporting agent remains at an externally connected node. It receives the list of failed nodes from the management agent. Only a test agent is implemented.

3.4 Agent Middleware

The agent middleware, including the complete SOS operating system but without any agents occupies 77.8kB of the 128kB FLASH memory on the MicaZ node. This leaves 50.2kB for agent code. This is comfortably more than has been required in any application tested.

3.3kB of the 4kB of available RAM is used, including 2kB reserved for the dynamic memory heap – this is available to be used by agents and applications. The remaining 705 bytes is used for the program stack.

3.5 Results

Results of the evaluation are shown in Table 1. *Startup Time* is the time for the management agent to discover and visit every node in the system. *Management Period* is the average time between visits of the management agent to each node. *Gathered Values* is the average number of nodes visited by the data gather agent. Ideally this would be 8.3, however only 6.6 values are in fact being used per reading. Some moves fail due to congestion or due to incomplete network discovery at that point.

The performance of the middleware itself is given in Table 3. *Move Time* is the average total time for an agent move between nodes (per hop), including routing and transfer. *Routing Time* is the average time to find a particular node. The average time spent executing at each node for the datagather and management agents are shown as *Execution Time DG* and *Execution Time Man.*

The node setup was altered for some simulation runs, to test the resilience and self-management capability of the system (Table 2). Nodes 6, 17 and 19 are not started until 3, 4 and 5 minutes respectively. The time taken to incorporate them into the system is *Late Start. Reset Test* switches off nodes 9 and 10 after 140 and 250 seconds respectively, and reset and switches them on 50 seconds later. The time taken to reincorporate them into the system is almost the same as the late start test, as the mechanism is very similiar.

Table 1. Normal Operation

Operation	Value
Startup Time	58.12s
Management Period	18.72s
Gathered Values	6.59

Table 2. Failure Recovery

Operation	Value
Late Start	15.3s
Reset Test	15.9s

Table 3. Middleware Performance

Operation	Value
Move Time (per hop)	364.1ms
Routing Time	44.3ms
Execution Time (DG)	8.1ms
Execution Time (Man)	2.2ms

Table 4. Agent Sizes

Agent	Size (bytes)
Management	1220
Control	644
Data Gather	880

4 Significance and Discussion

4.1 Robustness and Self-management

The system uses interlocking agents to guarantee reliability. The loss of any agent or node in the network will not lead to complete system failure. If a node resets, the next time the management agent visits, the control agent will be reestablished. A new data gathering agent is generated for each operation, loss will simply limit the current control operation. The most damaging failure is the failure of the management agent. Assuming that at least one of the control agents survives, it will summon a new management agent, which will rediscover the network.

Once all of the agents have been established, the system is entirely self-managing. New nodes can be introduced into the network by simply placing them within the transmission radius of existing nodes and node failures are detected and resolved. Cooperating systems of mobile agents bring a new level of robustness to mobile code systems. A suitably designed system can work around or recover from complete node failures without any intervention from outside the network.

4.2 Structure and Flexibility

The primary benefit of the system is its flexibility. The dynamic distributed control application trades off node coverage and availability for the most up-to-date sensor data, requiring an autonomous and self-managing system. This tradeoff is a design decision and not a limitation of the middleware. The application designer is free to position the application at any point within this tradeoff, and has complete control over the performance.

The agent carrier system allows multihop transfers to take place seamlessly, and takes common mobility operations out of the agent, while the blackboard system allows inter-agent communication.

The native mobile software agent introduces a new level of performance for mobile applications, beyond what is currently possible with existing mobile agent systems for wireless sensor networks. This allows true autonomous multi-agent systems. It is only with networks of cooperating agents that the true benefits of mobile software agents are realised.

4.3 Alternatives

Applications such as those presented here, with dynamic discovery and network analysis, fault monitoring and control decisions are simply too complex to be implemented in a VM. The agent size benefit of a VM-based system is only realised the first time an agent visits each node – after this, only the operating state is transferred. The actual control operation could be implemented using TinyDB or some similiar data gathering tool. This requires an external agent to generate the queries and interpret the results – the sensor network is simply functioning as a data gathering tool.

5 Related Work

Existing mobile and dynamic code mechanisms for wireless sensor networks can be broadly separated at the highest level into application and presentation middleware [7] (Figure 3). Application middleware assists the application in executing on the wireless sensor network. They can be either native or interpreted. Native applications execute directly on the hardware of the wireless sensor nodes, while interpreted applications go through an interpretation or translation layer. Presentation middleware presents the network as a whole to the user, abstracting away the details of the hardware platform and topology. The work presented here is an example of an application middleware.

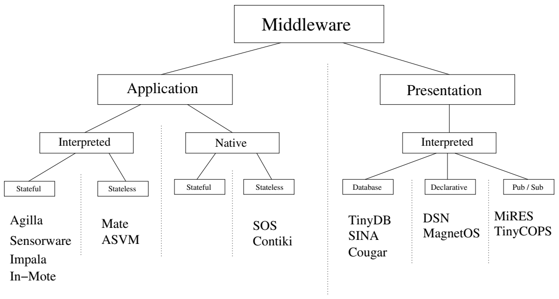

Fig. 3. Classification of existing middleware

All of these middlewares are capable of moving application code around the network. They can be divided into stateful middlewares, which transfer the application state, and stateless middlewares which do not.

5.1 Stateless Interpreted

ASVM [8] is a mobile agent systems for WSNs using a virtual machine (VM) to execute the agent code. ASVM allows extension and application customisation of the byte codes used in the VM, however the agent is constrained by the relatively simple VM in which it executes. TinyOS is required to execute the VM itself. Stateless systems essentially function as dynamic code distribution protocols. The work described here also contains such functionality but extends previous work by introducing stateful operation.

5.2 Stateless Native

Stateless native distribution systems are built into the various WSN operating systems, to send code updates through the network. Our middleware system is based on SOS [3] a dynamic operating system for WSN. This is the most suitable operating system at this time, however the middleware architecture is portable to other similar operating systems.

Contiki [9] also supports dynamic loading of modules, however much of the kernel is still compiled into a large static image. Much of the development of Contiki is focused on providing a large, highly capable networking stack.

The functionality of such distribution systems is duplicated in our work. We build on this by also providing state transfer and selective transfers, advancing the programming model from a simple static module to mobile and dynamic agents.

5.3 Stateful Interpreted

A stateful middleware is a form of mobile software agent [10]. Existing mobile agent systems for WSNs are either implemented on more capable devices such as PDAs or based on a Virtual Machine architecture. Impala [11] and Sensorware [12] are implemented on a PDA, written in Agent TCL. They provide support for complex dynamic applications.Scripts are injected into the network and can migrate, along with their state, to neighbouring nodes.

Agilla [5] is a more sophisticated version of ASVM, simpler than Sensorware but capable of operating on sensor nodes. Agents can move, copy and terminate themselves. Information is propagated among agents and one-hop neighbours using a tuple-space [13]. Nodes must have knowledge of their physical location and addresses are a function of the predetermined network topology.

The work described in this paper draws on much of the functionality of Agilla, but extends it to direct execution on the sensor node. The blackboard system is based on Agilla tuples, and the the simple agent mobility functions among one-hop neighbours are similiar. Our work however is more flexible and powerful in its execution model, and incorporates more powerful agent conditional transfers.

6 Conclusion

We have presented a native mobile software agent system for wireless sensor networks. It allows a level of complexity and performance in mobile software agents above that of existing systems. While the application presented here is only a preliminary examination of the system, it demonstrates its power and flexibility. Many design choices are available to the application designer, and the implementation can be tailored to suit the needs of the designer, rather than being constrained by the system in which it is implemented.

This flexibility does not come for free – agents must be carefully designed and the system will suffer in terms of application size. The safety of a virtual machine is not available, and agents must carefully manage their memory and resource usage. Simple applications will probably benefit little from moving to a native mobile agent system, however more complex applications can benefit greatly from the power and flexibility of the system, and the robustness and self-management that it achieves.

References

1. Jong, J., Culler, D.: Incremental network programming for wireless sensors. In: Proceedings of the First IEEE International Conference on Sensor and Ad Hoc Communications and Networks (October 2004)
2. Koshy, J., Pandey, R.: Remote incremental linking for energy-efficient reprogramming of sensor networks. In: Proceedings of the second European Workshop on Sensor Networks (EWSN 2005), Istanbul, Turkey (January 2005)
3. Han, C.-C., Kumar, R., Shea, R., Kohler, E., Srivastava, M.: A dynamic operating system for sensor nodes. In: Proceedings of the 3rd international Conference on Mobile Systems, Applications, and Services MobiSys 2005, Seattle, Washington, June 06 - 08, 2005, pp. 163–176. ACM Press, New York (2005)
4. Rivest, R.: The MD5 message-digest algorithm, Internet Engineering Task Force (IETF) RFC 1321 (1992)
5. Fok, C.-L., Roman, G.-C., Lu, C.: Rapid development and flexible deployment of adaptive wireless sensor network applications. In: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS 2005), Columbus, Ohio, June 6-10, pp. 653–662 (2005)
6. de Paz Alberola, R., Pesch, D.: AvroraZ: Extending Avrora with an IEEE 802.15.4 compliant radio chip model. In: Proc. 3rd ACM International Workshop on Performance Monitoring, Measurement, and Evaluation of Heterogeneous Wireless and Wired Networks, Vancouver, BC, Canada (October 2008)
7. Sugihara, R., Gupta, R.K.: Programming models for sensor networks: A survey. ACM Transactions on Sensor Networks 4(2) (March 2008)
8. Levis, P., Gay, D., Culler, D.: Active sensor networks. In: Proceedings of the Second USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2005), May 2–4 (2005)
9. Dunkels, A., Grönvall, B., Voigt, T.: Contiki – a lightweight and flexible operating system for tiny networked sensors. In: Proceedings of the First IEEE Workshop on Embedded Networked Sensors 2004 (IEEE EmNetS-I), Tampa, Florida, USA (November 2004)
10. Tripath, A.R., Ahmeda, T., Karnik, N.M.: Experiences and future challenges in mobile agent programming. Microprocessors and Microsystems 25(2), 121–129 (2001)
11. Liu, T., Martonosi, M.: Impala: a middleware system for managing autonomic, parallel sensor systems. In: PPoPP 2003: Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming, pp. 107–118. ACM Press, New York (2003)
12. Boulis, A., Han, C.-C., Srivastava, M.B.: Design and implementation of a framework for efficient and programmable sensor networks. In: MobiSys 2003: Proceedings of the 1st international conference on Mobile systems, applications and services, pp. 187–200. ACM Press, New York (2003)
13. Cabri, G., Leonardi, L., Zambonelli, F.: Mobile-agent coordination models for internet applications. IEEE Computer 33(2), 82–89 (2000)