Simulation of Agilla Middleware on TOSSIM

S. Özarslan
Informatics Institute
Middle East Technical University
06531 Ankara, TURKEY.
+90 312 2103747
ozarslan@metu.edu.tr

Y. M. Erten
Computer Engineering Department
University of Economics and Technology
06560 Ankara, TURKEY.
+90 312 2924071
erten@etu.edu.tr

ABSTRACT
In this study, we performed a simulation of mobile agents running on Agilla middleware designed for sensor networks. The simulations are performed using TOSSIM assuming that Agilla middleware is installed on the sensor nodes which are running TinyOS operating system. We simulated different agents corresponding to various functions and measured the time it takes for these agent software to run in the simulated environment and compared these results with those obtained using actual sensor nodes. The results presented in the study show that simulations produce results comparable to real life experiments.

Categories and Subject Descriptors
C.2.1 [Network Architecture and Design]: Wireless Communication
I.6.0 [Simulation and Modelling]: General

General Terms
Performance, Experimentation

Keywords
Middleware, simulation, mobile agents, wireless sensor networks, Agilla.

1. INTRODUCTION
As sensor networks find new application areas and become more and more popular, new techniques are being developed to improve their performance. One such technique is the use of middleware to hide the details of hardware and the operating system (OS) from the applications. There are different middleware developed for different purposes but they all aim at offering flexibility to the wireless sensor networks (WSN) [1] .

Middleware can be classified according to different criteria and one such classification is based on their functionality. Cougar [2], DSWare [3], TinyDB [4], and SINA [5] are members of a category which may be classified as database approach. In this approach, middleware abstract the WSN as a virtual database and the sensors focus on data rather than communication. SQL-like queries which are injected from a base station are used. These middleware have easy-to-use interfaces. In spite of this advantage, however, database approach is not suitable for real-time applications and only approximate results are provided by the middleware. Moreover, security and QoS are not considered in this approach.

Maté [6] and Magnet [7] are considered as instances of virtual machine approach. Virtual machine middleware are based on code interpreters and they are implemented on top of operating systems. Applications, which consist of small modules, run on the middleware which inserts them into WSN for the virtual machine to interpret. Programmability and flexibility are main advantages of this approach. Virtual machine based systems reduce energy consumption and network usage.

Adaptive approaches such as Milan [8] and, AutoSec [9] are special purpose middleware which have adaptability as the main characteristic. This approach is not suitable for general purpose applications.

Finally there are agent based approaches such as Agilla [10] and Smart Messages [11]. Agilla and Smart Messages have many common characteristics and the term “smart message” is used in the latter instead of the term “mobile agent” which is adopted in Agilla. Both of them use mobile agents (only names are different), support migration and use a local shared memory (tuple space in Agilla and tag space in Smart Messages) to provide local communication. However, Agilla have several advantages over Smart Messages.

In this study we developed an infrastructure to perform simulations of Agilla middleware on TOSSIM which is a simulation platform of WSN with nodes running TinyOS [14]. We developed a tool which can be used to write the agent software and load this software on the motes being simulated. These agents, running on the Agilla middleware which is installed on the motes, are allowed to perform different tasks and the performance of the system when running these different software is evaluated. The results obtained in the simulation environment are also compared with those observed in the experiments using actual motes.

The paper is organized as follows: section 2 describes Agilla middleware and explains why it has been chosen in this study. Section 3 explains the simulations and section 4 presents the results of the simulations. Conclusions are presented in section 5.

SIMUTools 2008, March 3-7, 2008, Marseille, France.

2. AGILLA MIDDLEWARE

Agilla is a middleware which supports mobile agents on WSN [12]. Mobile agents are dynamic, localized, intelligent programs that can move or clone themselves across the nodes to perform a specific task. They are used when it is advantageous to move software around so that the sensor nodes may perform different tasks without the need to reload new programs on them. After being injected into the network, the mobile agent performs autonomously and executes its instructions upon reaching a sensor node. Mobile agents can also interact with other agents and through their use, network flexibility is considerably increased.

Agilla is initially developed for Mica2 motes where each mote in the network is considered to be a node. Agilla middleware is then loaded on the nodes and mobile agents are injected to work on this middleware. There is a neighbor list and a tuple space on each node which are maintained by Agilla. Neighbor list consists of the addresses of one hop neighbors and the tuple space stores data to be shared by the local and remote agents (agents which reside on other nodes). Multiple agents may also reside on different nodes simultaneously and they can migrate among the nodes. Migrating agents can carry their codes and execution states, but they cannot carry tuple space of the node. Mica2 motes use TinyOS operating system which is specifically designed for sensor nodes [17].

Mobile agents running on Agilla have many advantages such as adaptation to environmental changes and wireless reprogramming which are the two important challenges for WSN. To illustrate this situation, assume that a WSN is primarily deployed for intrusion detection in a building. Civil defense authorities may want to reprogram the network to detect fire or gas leak in an emergency situation. Installing all these applications at once is not flexible, manageable or scalable. Mobile agent middleware address this problem. It provides dynamic reprogramming of WSN by allowing new agents to be injected and allows old agents to die. Hence, mobile agent middleware support adaptability and mobility.

Since multiple agents can exist on a node simultaneously, mobile agent middleware support coexistence of multiple applications on a node.

Shared memory model of the tuple/tag spaces enables one agent to insert a tuple which contains data and another to retrieve this data later. This feature allows coordination of agents to perform a common task independently. This model provides scalability of the middleware.

Mobile agents also use resources of sensor nodes efficiently as they only need resources of the visited nodes. We can also say that mobile agent middleware are power aware.

These are the main reasons which led us to adopt a middleware which supports mobile agents such as Agilla in our studies.

3. SIMULATION METHOD

In real world applications Agilla is loaded on the motes and then agents are injected on these motes to run on this middleware. Later, agents copy or move themselves to other motes; hence the whole network is covered.

In this study, we used TOSSIM to simulate a WSN with agents loaded on each mote. The motes have TinyOS running on them. We have also used TinyViz, a visualization tool for TOSSIM simulator [15].

We developed a Java application which is used to embed the agent -which we wanted to simulate- into Agilla middleware. Moreover, upon embedding the agent, this tool compiles Agilla middleware and starts TinyViz simulation.

Agilla has different components which facilitate the simulation process. One such component that we utilized is the AgentMgrM.nc file, which manages the context of the agent.

AgentMgrM.nc file has several important functions:

1. It migrates an agent from host node to destination node.
2. Whenever a new agent has arrived at a node, it allocates resources for this agent.
3. If agent dies or moves, it frees all resources (memory etc.) occupied by the agent.

As mentioned before, in real experiments, applications are compiled on the (real) wireless sensor network components, namely motes. In simulations, on the other hand, the software should be compiled together with the middleware at the time of initialization of the simulation process. The number of nodes and other parameters may also be specified at this stage when running the application.

The following steps are taken when the proposed approach is used in the simulation of Agilla agents on TOSSIM:

1. Java tool is used to write the agent software using Agilla commands or open an Agilla agent file which has already been written.
2. Java program converts this code into nesC.
3. This agent is embedded into the Agilla code at the appropriate position.
4. Java program compiles Agilla middleware and loads it onto the motes.
5. TinyViz simulation is started and the agents which are already part of the program code are activated.

3.1 Advantages of the proposed approach

A similar agent simulation method (original method) is described in the Agilla website and it uses the same files as we used in our approach [16]. However, there are major differences between two methods:

1. User must write the agent code using NesC language in the original method.
2. User must manually open AgentMgrM.nc file and paste the NesC code into this file at the appropriate location. User may corrupt AgentMgrM.nc file because of these manual operations.
3. User must set some parameters manually before and after the simulation.
4. User must manually compile Agilla middleware and start TOSSIM.

5. Different agents can be simulated using our approach, while this is not possible with the original method.
6. Our method uses visual simulation with TinyViz, while this is not possible in the original approach.
7. Our method can run series of simulations with autorun property of TinyViz, while the original method cannot.
8. Simulation results can be saved to specified files for analysis with our method.

3.2 Simulations with TinyViz

TinyViz is a powerful tool with an autorun feature which permits setting of the parameters automatically, running multiple simulations, logging data to files, taking screenshots and some other operations.

An autorun file, shown in Figure 1, is used to specify the properties of TinyViz simulation, such as number of simulations, names of log files, number of simulation seconds elapsed etc.

```
# Set the layout (grid, random, grid+random)
layout grid
# This plugin takes debug messages
plugin DebugMsgPlugin
# Total number of simulated seconds to run
numsec 40
# Name of the executable file
executable build/pc/main.exe
# Number of motes
nummotes 25
# File to log all debug messages to
logfile 5hop_sim1
```

Figure 1. Autorun file used for simulations

The Java application developed executes the simulations with TinyViz via this autorun file. Each simulation is logged to the specified file. TinyViz runs each simulation for 40 simulation seconds. Once the simulation ends, TinyViz automatically terminates.

The relation between real time and simulation time is based on the clock cycle of the Mica2 motes. Because they use a clock frequency of 4 MHz, to convert the time produced by TinyViz into seconds one has to divide these numbers by 4,000,000 [13]. So we used a factor of 4000 to convert the time to milliseconds.

We had over 2000 log files after simulations. Manually analyzing these files was not feasible; hence they were analyzed using AWK scripting language.

4. SIMULATIONS

4.1 Strong migration vs. Weak migration

The term "migration" means moving or cloning of an agent. Moving an agent is transferring the agent from one node to another, while cloning means copying the agent. Special instructions are used to move or clone an agent. Cloning instructions are *sclone* and *wclone* and the moving instructions are *smove* and *wmove*.

First letters of these terms (s and w) describe whether the migration is strong or weak. The difference is the transferred parts of the agent. In strong migration, agent code, program counter, heap, stack and reactions are transferred and agent resumes running where it stopped. On the other hand, only the code is transferred and the agent starts execution from the beginning in a weak migration.

There are some trade-offs between using strong and weak migration. Since strong migration transfers everything, programming is simplified. However, strong migration consumes more memory, bandwidth and requires more processing.

We used test agents to benchmark strong and weak migrations. We measured consumed time during *smove* (strong move), *wmove* (weak move), *sclone* (strong clone) and *wclone* (weak clone) operations.

4.1.1 Smove vs. Wmove

We used two different agents to benchmark *smove* and *wmove*. The difference between them is the heap operations; 10 values are recorded to the heap in the first agent while heap operations have not been used in the other.

```
1:    pushc 1      // push "1" to stack
2:    setvar 0     // record "1" to heap[0]
3:    pushc 1      // push "1" to stack
4:    setvar 1     // record "1" to heap[1]
.....
.....
19:   pushc 1      // push "1" to stack
20:   setvar 9     // record "1" to heap[9]
21:   randnbr      // get a random neighbor
22:   smove        // (or wmove) move to the random neighbor
23:   halt         // agent terminates itself
```

Figure 2. Code of the agent with heap operations.

Figure 2 shows the agent used to simulate *smove* instruction with heap operations. This agent saves 10 values to heap (from heap[0] to heap[9]) and finally moves to a random neighbor and terminates itself. We simulated this agent on a virtual wireless sensor network which had 25 nodes. We repeated the simulations 100 times for each instruction (*smove* and *wmove*) and we computed the average consumed time for *smove* and *wmove* operations. The results are displayed in Figure 3.

Average migration time of an agent from one node to another is 399 milliseconds with *smove* instruction; while *wmove* takes 218 milliseconds, 55% of the value recorded for *smove*. This result is expected, because *smove* instruction transfers agent's heap together with the agent's code, while *wmove* instruction transfers only the agent's code. As each variable in the heap is 40 bits and we used 10 variables, a total of 400 bits or 50 bytes are transferred. This extra-load increases the migration time of *smove* agent.

Figure 4 shows the second agent which has its heap operations removed. The agent performs *smove* (similar agent was used for

wmove) and the amount of the time taken for *smove* (and *wmove*) was measured with the same amount of data in both cases.

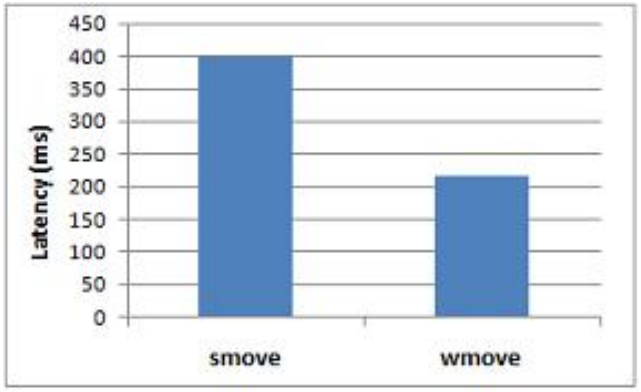

Figure 3. Latency of *smove* and *wmove* instructions with heap operations

```
1:   randnbr   // get a random neighbor
2:   smove     // (or wmove) move to a random neighbor
3:   halt      // agent terminates itself
```

Figure 4. Code of the agent without heap operations.

We also repeated the simulations for 100 times for each instruction (*smove* or *wmove*) for this agent. We then computed the average latency for these operations which are shown in Figure 5.

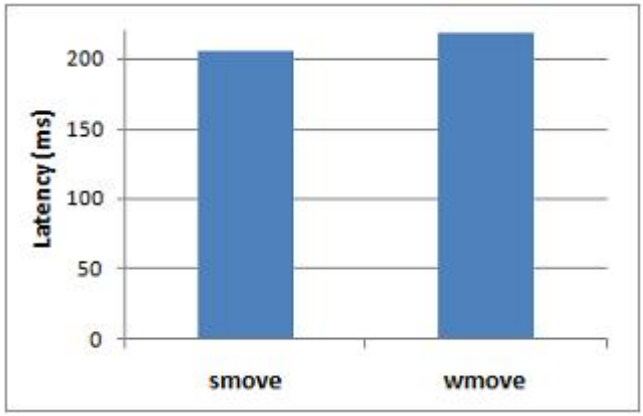

Figure 5. Latency of *smove* and *wmove* instructions without heap operations

As can be seen from Figure 5, there is only 6% difference between latencies of each move instruction (205 milliseconds for *smove* and 218 milliseconds for *wmove*). As the amount of data is the same we experienced similar latency in both cases.

4.1.2 Sclone vs. Wclone

Similar agents are used to benchmark *sclone* and *wclone* instructions. Code of these agents is similar to the above; only the line with *smove/wmove* was replaced by *sclone/wclone*. We repeated the simulations for 100 times on a simulated wireless sensor network which has 25 nodes. Figure 6 displays the results of the simulations for *sclone* and *wclone* with and without heap operations.

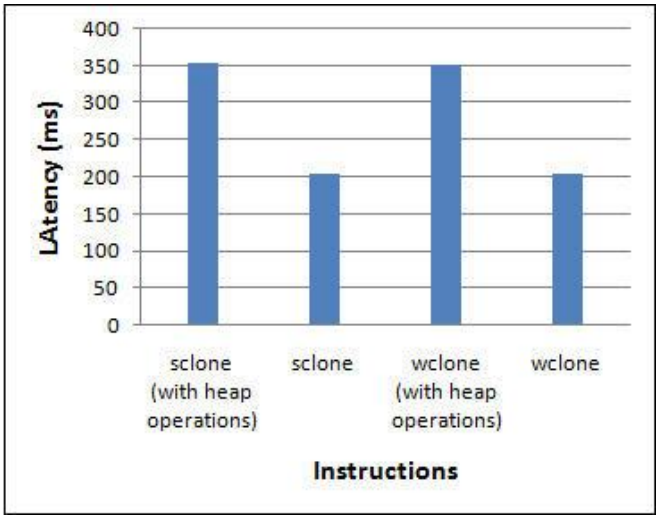

Figure 6. Latency of *sclone* and *wclone* instructions

4.2 Comparison of Simulations with Experiments Performed Using Actual WSN

Latency and reliability of individual Agilla instructions are evaluated by Agilla developers in a technical report [12]. They used a 25-node network which was formed in a 5x5 grid topology. All messages except those from neighbors are filtered in the grid. Grid coordinates (x,y) are used to identify each node.

We compared the results of simulations and the results of actual experiments to evaluate the usability of mobile agent simulations. Same number of nodes, same network topology and same agents are used for consistency.

4.2.1 Smove

In this simulation, the *smove* agent, shown in Figure 7, moves from node (1,1) to a remote node. Remote node could be 1 to 5 hops away. This experiment is repeated 100 times for 1 to 5 hops. Average latency of successful executions and number of failures are computed. Then, simulation results are compared with actual experiments (Figure 8 and Figure 9). According to the figures, differences between values of the simulation environment and real experiments differ between 0% and 3%. This outcome indicates that the simulation environment produces very similar results as the real life experiments for migration instructions.

```
1:  pushloc 5 1     // push location (5,1)
2:  smove           // strong move to node at (5,1)
3:  pushloc 0 0     // push location (0,0)
4:  smove           // strong move to node at (0,0)
5:  halt            // agent terminates itself
```

Figure 7. Code of *smove* agent

4.2.2 Rout

Rout (remote out) agent, shown in Figure 10, inserts a tuple in a remote node's tuple space. Simulations are performed to test this operation and tests are repeated 100 times again for 1 to 5 hops. Comparisons of simulation results with real world experiments are shown in Figure 11 and Figure 12. These figures show that difference between of the results of the simulated system and the real system is under 1%.

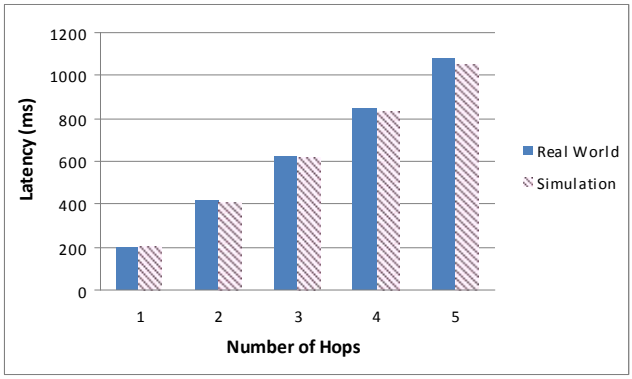

Figure 8. Comparison of simulations and real experiments for the latency of *smove* instruction

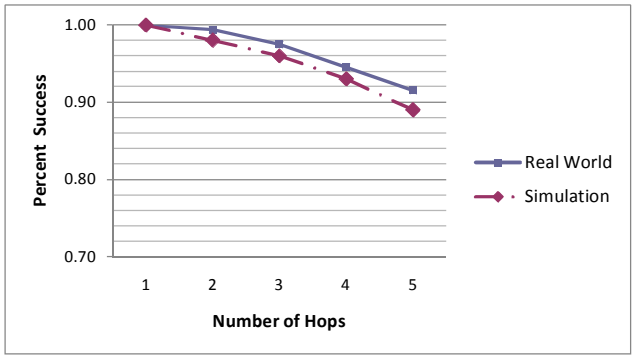

Figure 9. Comparison of simulations and real experiments for the reliability of *smove* instruction

```
1:  pushc 1         // push value 1 to stack
2:  pushc 1         // tuple <value:1> on stack
3:  pushloc 5 1     // push location (5,1)
4:  rout            // remote out to node at (5,1)
5:  halt            // agent terminates itself
```

Figure 10. Code of *rout* agent

4.2.3 Remote Operations

Agilla middleware enables remote coordination of agents by remote operations. Most important ones are *rout, rinp, rrdp* (remote probing rd) and migration instructions (*smove, wmove, sclone, wclone*).

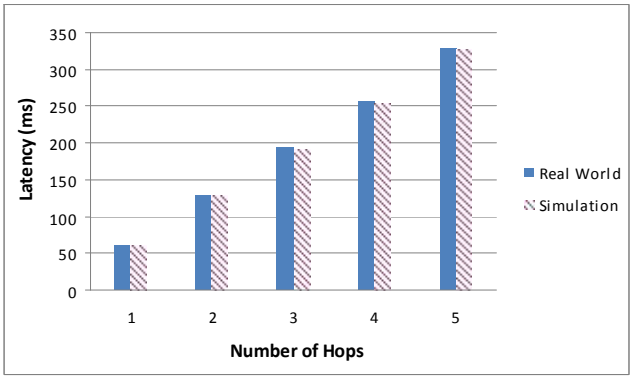

Figure 11. Comparison of simulations and real experiments for the latency of *rout* instruction

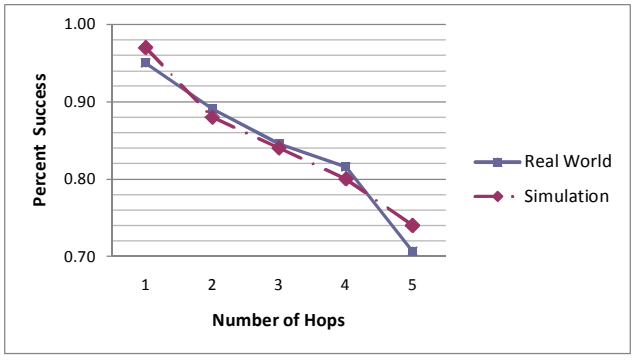

Figure 12. Comparison of simulations and real experiments for the reliability of *rout* instruction

Rout instruction and migration instructions are mentioned before. *Rinp* (remote probing in) and *rrdp* (remote probing rd) search remote node's tuple space for a matching template. If one is found, *rinp* removes this tuple from remote node's tuple space, while *rrdp* doesn't.

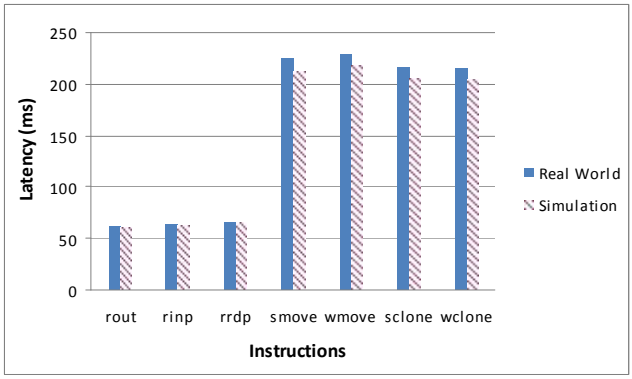

Figure 13. Comparison of remote operations

Comparisons of simulation results of remote operations with those obtained in real life experiments are shown in Figure 13. The figure shows that, again, the simulations produce similar results as experiments using real nodes for both remote tuple space operations (*rinp, rout and rrdp*) and migration operations (*smove, wmove, sclone, wclone*).

5. CONCLUSIONS

In this study we performed simulations of mobile agents running on Agilla using TOSSIM. We tested the different Agilla commands and measured the time it takes to execute them on the simulator. We used TinyViz to help us collect the simulation data and also to visually observe the simulation process. We compared the simulation results with those obtained in experiments where real sensor motes were used.

Our results show that simulations produce quite similar outcomes as real life experiments and they can be used to evaluate the performance of Agilla agents. As it is much cheaper and more flexible than using actual motes, this approach could save a lot of time and expense.

These experiments will be carried out using different network topologies and different agents in the future. Actual network will be set up and experiments will be compared with simulations as future work.

6. REFERENCES

[1] Molla M.M., Ahamed S.I. 2006. A survey of middleware for sensor network and challenges. In Proceedings of IEEE International Conference on Parallel Processing Workshops (Milwaukee, Wisconsin, August 14 – 15, 2006)

[2] Yao, Y. and Gehrke, J. 2002. The cougar approach to in-network query processing in sensor networks. SIGMOD Rec. 31, 3 (Sep. 2002), 9-18.

[3] Yu, X., Niyogi K., Mehrotra, S. and Venkatasubramanian N. 2003. Adaptive Middleware for Distributed Sensor Environments. IEEE DS Online 4, 5 (May 2003).

[4] Madden, S. R., Franklin, M. J., Hellerstein, J. M., and Hong, W. 2005. TinyDB: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst. 30, 1 (Mar. 2005), 122-173.

[5] Shen, C. C., Srisathapornphat, C. and Jaikaeo, C. 2001. Sensor information networking architecture and applications. IEEE Pers. Commun. 8, 4 (Aug. 2001), 52-59.

[6] Levis, P. and Culler, D. 2002. Maté: a tiny virtual machine for sensor networks. In Proceedings of the 10th international Conference on Architectural Support for Programming Languages and Operating Systems (San Jose, California, October 05 - 09, 2002). ASPLOS-X. ACM Press, New York, NY, 85-95.

[7] Barr, R., Bicket, J. C., Dantas, D. S., Du, B., Kim, T. W., Zhou, B., and Sirer, E. G. 2002. On the need for system-level support for ad hoc and sensor networks. SIGOPS Oper. Syst. Rev. 36, 2 (Apr. 2002), 1-5.

[8] Murphy A. and Heinzelman, W. Milan: Middleware linking applications and networks. 2002. University of Rochester, Tech. Rep. TR-795, (Jan 2002).

[9] Han, Q. and Venkatasubramanian, N. 2001. AutoSeC: An Integrated Middleware Framework for Dynamic Service Brokering. IEEE DS Online 2, 7 (2001).

[10] Fok, C., Roman, G., and Lu, C. 2005. Mobile agent middleware for sensor networks: an application case study. In Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (Los Angeles, California, April 24 - 27, 2005). Information Processing In Sensor Networks. IEEE Press, Piscataway, NJ, 51.

[11] Kang, P., Borcea, C., Xu, G., Saxena, A., Kremer, U., and Iftode, L. 2004. Smart Messages: A Distributed Computing Platform for Networks of Embedded Systems. The Computer Journal, Special Issue on Mobile and Pervasive Computing, Oxford Journals 47, 4 (January 2004), 475-494.

[12] Fok, C. L., Roman, G. C., and Lu, C. 2006. Agilla: A Mobile Agent Middleware for Sensor Networks. Technical Report, WUCSE-2006-16, Wa. Univ. in St. Louis (March 2006).

[13] http://mail.millennium.berkeley.edu/pipermail/tinyos-beta-commits/2005-May/000451.html

[14] Levis, P., Lee, N., Welsh, M., and Culler, D. 2003. TOSSIM: accurate and scalable simulation of entire TinyOS applications. In Proceedings of the 1st international Conference on Embedded Networked Sensor Systems (Los Angeles, California, USA, November 05 - 07, 2003). SenSys
'03. ACM, New York, NY, 126-137.

[15] http://www.tinyos.net/tinyos-1.x/doc/tutorial/lesson5.html

[16] http://mobilab.wustl.edu/projects/agilla/docs/tutorials/9_debug.html

[17] Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, JL, Welsh, M., Brewer, E. and Culler, D. 2004. TinyOS: An Operating System for Sensor Networks. In Ambient Intelligence. Weber, W., Rabaey, J.M., and Aarts, E., Eds. Springer-Verlag New York, Inc. Secaucus, NJ. 115-148.