JAFSPOT: Java Agent-Based Framework for Sun SPOT Wireless Sensor Networks

Hakan Cam, Ozgur Koray Sahingoz
Computer Engineering Department
Turkish Air Force Academy
Istanbul, Turkey
{h.cam, sahingoz}@hho.edu.tr

Ahmet Coskun Sonmez
Computer Engineering Department
Yildiz Technical University
Istanbul, Turkey
acsonmez@ce.yildiz.edu.tr

***Abstract*—Due to increasing capabilities of micro-sensors, wireless sensor networks have emerged as one of the key growth areas in recent years. They are a collection of sensor nodes deployed over a target region for observing physical phenomena, such as temperature, light, accelerometer, etc. Mobile agent model is a distributed computing paradigm, which is capable of solving problems effectively in dynamic and open environments like wireless sensor networks. Few mobile agent systems have been developed for wireless sensor networks so far. In this paper, we describe JAFSPOT, a Java Agent-based Framework for Sun SPOT. It uses event-based programming in which the core components communicate through events. To the best of our knowledge, JAFSPOT is one of the very few mobile agent-based frameworks for wireless sensor networks that support migration of isolates. First, we describe the core components of the proposed system, then present a sample application about monitoring Sun SPOT sensor node with mobile agents and finally, give the results of an experiment to evaluate the performance of this system in terms of time and energy consumption.**

Keywords- Wireless Sensor Networks; Mobile Agent Systems.

I. INTRODUCTION

Depending on recent developments in processing, power, storage, micro-sensor and wireless communication technologies, Wireless Sensor Networks (WSNs) have become a broad area of interest in military, academic and industrial circles [1]. WSNs are composed of hundreds of sensor devices coming together and communicating over a wireless radio. These sensor devices are low cost, tiny devices with low power, constrained storage, limited processing and short-range wireless communication capabilities [2]. A sample WSN architecture is depicted in Figure 1.

The mobile agent (MA) paradigm is a distributed computing mechanism used for remedying the problems of dynamically changing environments, such as WSNs. It is a software process that can operate autonomously and can migrate to its code and state. It provides a way to dynamic reprogramming and facilitates a powerful and flexible mechanism for complex distributed problems of WSN systems [3][4].

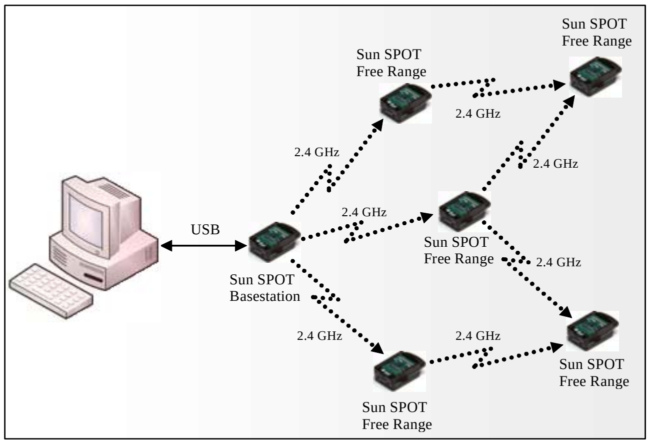

Figure 1. Wireless sensor network architecture

Since MAs can provide some reasonable, practical and inexpensive solutions for the WSNs limitations, integration of WSNs with MAs is emerging as an essential requirement. Some of the limitations of WSNs are: energy for long network lifetime, restricted bandwidth for wireless communication, hardware due to the small size of the sensor nodes, unstable network connections due to the mobility and lifetime of the sensor nodes, low-level re-programmability due to its distributed structure. Due to these constraints, deploying a new code into a distributed WSN and upgrading it is an extremely cumbersome issue. In addition to these constraints, while most WSNs have typically been developed in an application-specific manner, sensor devices can store and run multiple applications at the same time. Instead of storing and running all applications in a single sensor device, using of MAs seems to be a more practical solution [5]. MA paradigm is a way of smart programming and can be regarded as the further development of a distributed problem-solving of WSNs [6]. Performance of WSNs can be improved by using MAs via improving communication and coordination capabilities. A sample MA-based WSN structure is depicted in Figure 2.

WSNs can benefit from MAs in several ways: First, MAs use the bandwidth more efficiently by transferring its code to the interested target area. Therefore, there is no need to circulate the raw data over the network. Second, MAs provide effective and dynamic re-

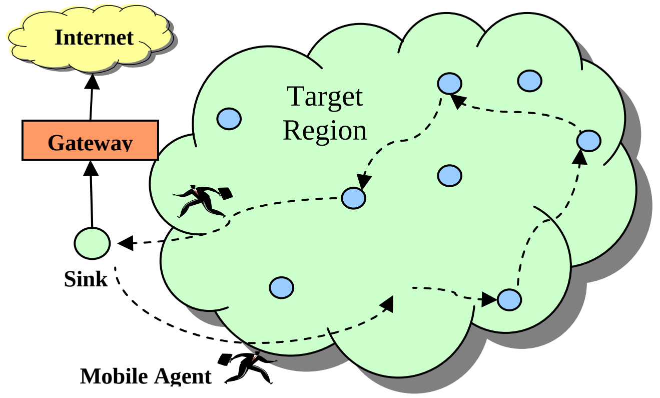

Figure 2. Mobile agent-based wireless sensor network

programming capability for cooperative data processing in WSNs [7].

MASPOT [8], MAPS [9][10] and MOBILE-C [11][12][13] are some of the studies regarding the integration of MAs in WSNs, but very few ones have been developed for Sun SPOT sensor devices [14]. They are compatible with Java 2 Micro Edition [15], and are supported by the Squawk Java Virtual Machine [16]. These studies are described in detail in related work section.

In this paper, it is aimed to give some insight on the issue of MA programming paradigm in WSNs and a **J**ava **A**gent-based **F**ramework for **S**un **SPOT-** JAFSPOT is proposed. This is a new agent-based framework programmable in Java for WSNs. It is based on Sun SPOT (Small Programmable Object Technology) sensor device technology that is an experimental platform for application programmers to develop WSN applications using Sun SPOT technologies. Because of their powerful structure, Sun SPOT sensor devices are widely used in the industry.

Agent-oriented programming of WSN applications are achieved in this framework using Java programming language. At the same time, event-based programming example is also used in the proposed framework. Therefore, all operations can be performed based on these events and the core components of the framework communicate through these events.

The remainder of the paper is organized as follows. MA systems recently developed for WSNs are investigated in Section 2. The proposed system architecture of the JAFSPOT framework and its main components are described in Section 3. In Section 4, a simple example is provided for exemplifying the MA-based application programming with JAFSPOT framework. Section 5 describes the testing and evaluation of proposed system. Finally, conclusion and the future work are mentioned.

II. RELATED WORK

Agent-based programming of WSNs is a very challenging issue because of the constrained resources of sensor devices. In addition, programming of WSNs is usually implemented as in an application specific manner and dependent on the application running in the system.

Some of the most popular MA-based WSN middleware systems proposed and implemented so far are described below.

MASPOT [8] is a MA-based system developed for Sun SPOT sensor devices. The authors claim that it is the only Java-based MA system for WSNs that currently provides code migration. Basic MA life cycles of creation, initialization, cloning and migration services are implemented in this framework. Its communication service provides primitives for agent-agent communications using tuple spaces and agents-base station communications using message passing. In addition, it only uses around 1.5% of the available flash memory and spends around 0.02% of the battery energy of sensor devices for moving an agent. It also extends the range of Java-based WSNs applications that can be built using current technology.

MAPS (Mobile Agent Platform for Sun SPOT) [9][10] is a MA-based platform for Sun SPOT sensor devices. It is established on the agent paradigm and provides the programming of WSN applications using Java language. The architecture of MAPS is component-based and presents the core services to agents. Event-based, state-based and agent-based approaches are combined in this platform. Since Squawk Virtual Machine operations are relatively slow, the time of MA migration is quite high. The serialization of agents into a message is a very time consuming operation. The radio stream communication between sensor devices is quite slow.

MOBILE-C [11][12][13] is an agent platform for mobile C/C++ agents. This platform is compatible with the IEEE the Foundation for Intelligent Physical Agents (FIPA) [17]. It extends FIPA standards to support MAs. It integrates an embeddable C/C++ interpreter into the platform as a MA execution engine and defines an agent mobility protocol to direct agent migration process. For agent migration, it uses FIPA agent communication language (ACL) messages encoded in XML. This offers a good solution for inter-platform agent migration in FIPA compliant agent systems. In this framework, scriptable C/C++ is chosen as a MA language. It is written in C with a small footprint, and it uses an embeddable C/C++ interpreter named Ch [18][19][20] to support the execution of MA C/C++ source code.

The system we described in this paper, JAFSPOT, differs from other systems in several ways. It is, to the best of our knowledge, one of the very few MA-based frameworks using Java programming language for WSNs that supports weak migration with isolate mechanism. In this mechanism, inner state and the private data of the MA residing on a sensor device can be saved in a format which the destination MA can handle. The agent code must be present on both sensor devices, so that any agent method can be created and initialized with the transferred agent state on the destination sensor device.

This isolate migration mechanism facilitates the mobility and extends the range of possible applications of Sun SPOT sensor devices. Furthermore, JAFSPOT provides event-based communication between MAs. Event-based approach is a useful abstraction in the context of WSNs. Particularly; occurrence of a physical event and the resulting reaction according to this event provides a way to optimize the consumption of invaluable resources of resource-constrained WSNs.

III. System Architecture of Jafspot Framework

The block diagram of proposed system is depicted in Figure 3, and explained in detail below.

A. *Hardware*

Processor board, sensor board and battery are the three main components of Sun SPOT sensor devices. Interested readers may refer to [14] for detailed hardware components.

B. *SQUAWK Java Virtual Machine*

Squawk Java Virtual Machine is just over the hardware component. Here, fully capable J2ME CLDC 1.1 Java VM is supported by operating system. Hardware-independent and simultaneously working applications can be possible owing to the virtual machine.

Squawk JVM is realized for tiny devices with constrained capabilities using Java language and provides OS level mechanisms. It includes a mechanism for serializing the object graphs. All the pointers in a serialized object relocate in canonical addresses. This serialized form can be transformed into a new live object graph again.

Squawk JVM architecture is depicted in Figure 4. The most important characteristic feature of this architecture is a small and more remarkable compact byte code instruction set. Standard J2ME class files cover 35-45% less space compared with byte codes.

Isolate mechanism is one of the most important elements in the Squawk JVM architecture. Any application is represented as an object in this mechanism. More than one application can work in a single Squawk JVM. In this method, any application can conceptually run as fully isolated from other applications.

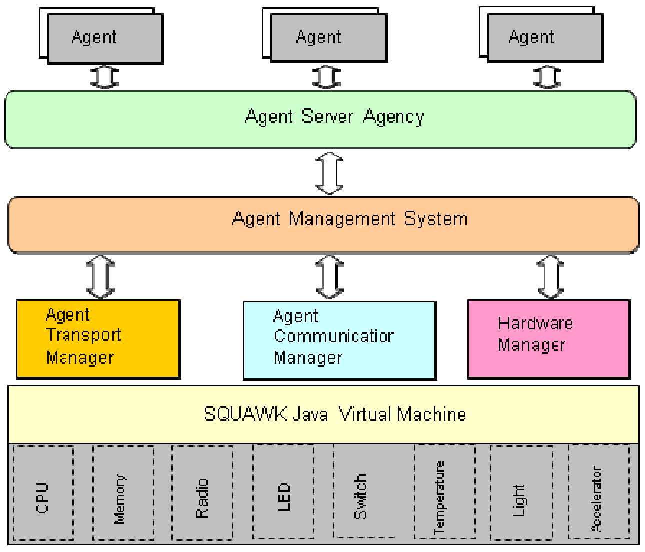

Figure 3. Block diagram of proposed system architecture

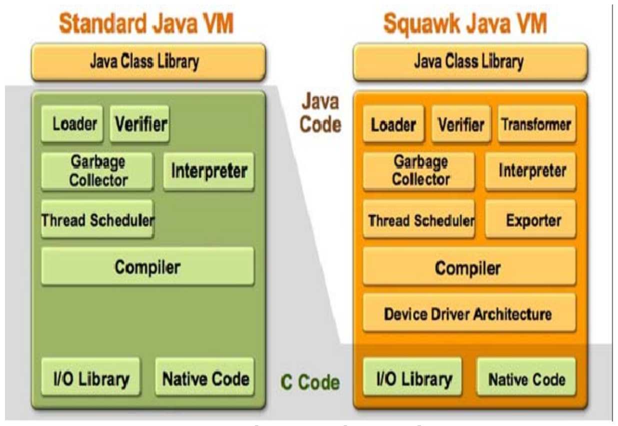

Figure 4. Squawk JVM architectural structure

It allows isolate migration which means that an isolate working in any instance of Squawk JVM to cease its operation, serialize into a file, send over network connection and work again in another instance of Squawk JVM.

1) *Isolate Mechanism*

While one application has multiple threads in standard Java ME applications, in practice only one application can work in standard Java VM at the same time. Nevertheless, Squawk JVM allows working of multiple applications on any Sun SPOT sensor device using a special class of Isolate. Thus, the operation of any application can be isolated from other applications. It prevents blocking the operation of one application from others. Each MIDlet-based application works in a separate isolate mechanism. All isolates can reach the resources of Sun SPOT hardware.

Isolate class provides a way to the instance of an isolate to work isolated from the instances of other isolates. Isolate mechanism is similar to the processes. Objects of any isolate are logically separated from objects of other isolates. Similarly, static variables of any isolate are logically separated from static variables of other isolates.

Isolate mechanism can be suspended in hibernation which stops working of both isolate and the threads of this isolate and can be serialized. The saved form of an isolate includes all the accessible objects, static variables and working contexts of all threads of this isolate. This saved form can be transferred to any file or other sensor devices over the sensor network. This saved isolate can be reopened, de-serialized and reactivated on the opposite side of the channel. An isolate can be in NEW, ALIVE, HIBERNATED and EXITED states.

C. *Agent Server Agency*

Agent Server Agency platform is located on the SQUAWK Java Virtual Machine component. This platform should be established on all Sun SPOT sensor devices and all components of the system work on this platform. This platform must be activated in the developed application in order to initialize the other components. New MAs that will work on the Sun SPOT sensor devices are to be added to the system with using this component. The 64-bit IEEE extended MAC address of the system running on the Sun SPOT sensor device is kept in this component.

D. *Agent Management System*

Agent Management System is the second component after the Agent Server Agency platform. This component operates as the brain of the system. It manages the events that happen throughout the system, connects and keeps the elements and provides the possibility of working together of the system components. This component makes connections with other components in order to perform the operations such as sending message, reading sensor values and synchronize timers performed by agents. This component is responsible for the required activities of agent creation, initialization, communication, migration, timing and termination. It provides the other components a way to work in a harmony.

E. *Agent Transport Manager*

Agent Transport Manager Component enables naming of agents, specifying the neighboring sensor nodes and neighboring agents dynamically and migration of agents from one sensor node to the other. In doing so, it provides a mechanism for serializing the agents into a message and migration of these agents to the neighboring sensor nodes. It also receives the messages containing the serialized agent coming from neighboring sensor nodes and opens and activates these agents with a reverse operation. This component is used for keeping the address and lists of the agents while migrating from a Sun SPOT sensor device to others within the communication range. It is also used for establishing a RadiostreamConnection between sensor devices before migrating of the agents to other sensor devices. Finally, it is used for receiving the agents sent from the other sensor devices.

F. *Agent Communication Manager*

Agent Communication Manager is the other component located on the Agent Server Agency platform. This component provides message-based asynchronous communication capability between agents working on the Sun SPOT devices. Thus, agents working on the different sensor devices can communicate with each other. Similarly, this component provides a way about communication of all the other components located on the Sun SPOT device. RadiogramConnection link can be established through the communication port between sender device and receiver device using this component. After the connection had been established, the Datagram was created for sending and receiving operations. In addition, all the events publish through this communication port to the neighboring sensor devices within the communication range.

G. *Hardware Manager*

The Hardware Manager component is located at the bottom of the Agent Server Agency platform. This component allows reading of data values from hardware resources like temperature sensor, light sensor, acceleration sensor, battery, switch and LED from Sun SPOT devices. Therefore, it can be possible to operate on the reading both sensor values and input/output values of hardware resources of Sun SPOT devices. It provides a way to perceive the physical temperature, light and three-dimensional acceleration as analog, to convert these analog values to numerical values and to evaluate these values as required.

IV. Mobile Agent Application

In this part of the paper, a MA application designed, developed and implemented using Java programming language. The purpose of this application is to demonstrate the rationale behind MAs working over the proposed system architecture on Sun SPOT devices. This MA-based application works on two real physical Sun SPOT devices. While the first one serves as the sender device, the second one serves as the receiver. In this application there are two MAs working on the sender device and one MA working on the receiver device. One of the sender side MAs is named MobileAgentSender and the other for MobileAgentMediator. MobileAgentMediator agent is migrated from sender device to the receiver device over wireless communication channel. MobileAgentReceiver agent works on the receiver device.

There are two different components of the proposed system, namely, the primary components and the secondary components. The primary ones are Agent Server Agency, Agent Management System, Agent Transport Manager, Agent Communication Manager and Hardware Manager. The secondary ones are Light Sensor, Temperature Sensor, Accelerometer Sensor, LED and Switch. All these components on the two Sun SPOT devices should be activated and worked in order to MAs to work. After all the components had been activated on the two Sun SPOT devices, the MAs added to the system.

A. *MobileAgentSender*

There are six conditions for MobileAgentSender agent working on sender Sun SPOT device in this application. These conditions are Begin State, Wait_Message State, Event_Creation State, Capture_Value State, Transmit_Value State and End State. These conditions are illustrated in Figure 5.

```
BEGIN:
    AGENT_START:
        Create LED_ON/LED_OFF events used for producing a binary count up
        to 256 on the 8 tri-color LEDs on Sun SPOT sensor device in red color
WAIT_MESSAGE:
    MESSAGE:
        Create LED_FLASH event used for flashing all the 8 tri-color LEDs ten
        times in blue color
        Create SWITCH_ON event
EVENT_CREATION:
        Create TEMPERATURE, LIGHT, ACCELERATION, BATTERY event
CAPTURE_VALUE:
    TEMPERATURE, LIGHT, ACCELERATION and BATTERY:
        Add light, temperature, acceleration and battery values to related variable
    SWITCH_ON:
        Create MobileAgentMediator agent
        Create LED_FLASH event used for flashing the first LED of 8 tri-color
        LEDs in red color
TRANSMIT_VALUE:
        Create related event for MobileAgentMediator agent to be sent
        Piggy-back the data to the MobileAgentMediator agent as a payload
END:
        Stop agent operation
```

Figure 5. States/actions of MobileAgentSender agent

B. *MobileAgentReceiver*

There are four conditions for MobileAgentReceiver agent working on the receiver side Sun SPOT sensor device in this application, including Begin State, Switch_On State, Circulated_Data State and End State. These conditions are illustrated in Figure 6.

```
BEGIN:
    AGENT_START:
        Create LED_ON/LED_OFF events used for producing a binary count up to
        256 on the 8 tri-color LEDs on Sun SPOT sensor device in red color
        Create SWITCH_ON event
SWITCH_ON:
        Discover the other mobile agents
        Create and send message event
CIRCULATED_DATA:
    MESSAGE:
        Fragment obtained data into meaningful parts using StringTokenizer
        Print out light, temperature, acceleration and battery values on the screen
END:
        Stop agent operation
```

Figure 6. States/actions of MobileAgentReceiver agent

C. *MobileAgentMediator*

There are three conditions for MobileAgentMediator agent working on the sender side device and will migrate to the receiver side in this application. These conditions are Begin State, Migration State and End State. These conditions are illustrated in Figure 7.

```
BEGIN:
    AGENT_START:
        Obtain the value of data from related variable
        Discover the other neighboring mobile agents
        Declare its migration request to these identified sensor devices
MIGRATION:
        Migrate to these identified sensor device
END:
        Stop agent operation
```

Figure 7. States/actions of MobileAgentMediator agent

V. TEST AND EVALUATION

This section describes the testing and evaluation of proposed system explained in Section IV. The purpose of this test scenario is to demonstrate the rationale behind MAs working over the proposed system architecture on Sun SPOT Java Development Kit with Sun SPOT SDK v5.0 (Red). This kit includes a base station and two Sun SPOT sensor devices equipped with sensor boards and rechargeable batteries. This application is developed using the Apache Ant Server, Java Development Kit 1.6.0_31 and NetBeans IDE 7.1.1 Integrated Development Environment.

Since the amount of energy spent by sender and receiver side agents is a critical issue for WSNs we evaluated this parameter. Here, we tested five issues namely, battery current drawn while the MA migration, available capacity, MA creation time, MA migration time and MA termination time. We repeated and obtained 15 different values for this test.

There are three key points for testing the migration cost of MA. The first point is where the MA residing on the sender side sensor node asks for the migration. The second point is where the sender side MA received an ACK used for acceptance of migration from receiver side sensor node. The third point is where the sender side MA successfully migrates to the destination side node. At these points, we captured the maximum battery current drawn from sender and receiver side nodes. These experiment values are depicted in Figure 8.

Here, we compare our results with the values of Table 1 in related research paper on MASPOT [8]. As it can be seen from these results, the mean spent energy of JAFSPOT was 0.71559986 milliampere for the sender side and 0.453732553 milliampere for the receiver side. Here this gives an overall mean of 0.584666207 milliampere. Similarly, mean spent energy of MASPOT [8] was 0.1577 milliampere for the sender side and 0.1257 milliampere for the receiver side, giving an overall mean of 0.1417 milliampere.

We have also studied the available capacity of the sender and receiver side sensor nodes. Each Sun SPOT sensor node is equipped with a 3.7 V rechargeable 770 milliAmperehour Lithium-Ion battery. The obtained values are depicted in Figure 9. Here, the mean percentage of battery use is on average 0.099388869% for the sender side and 0.06301841% for the receiver side battery capacities. Considering these average values obtained with these tests, sender side sensor node spent around 57.71% more energy than the receiver side sensor node.

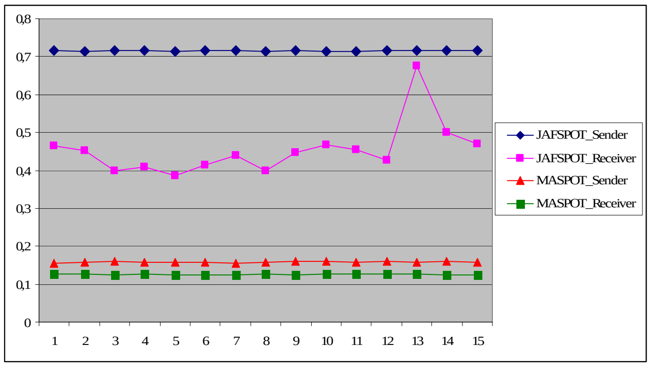

Figure 8. Agent Migration Cost in milliampere

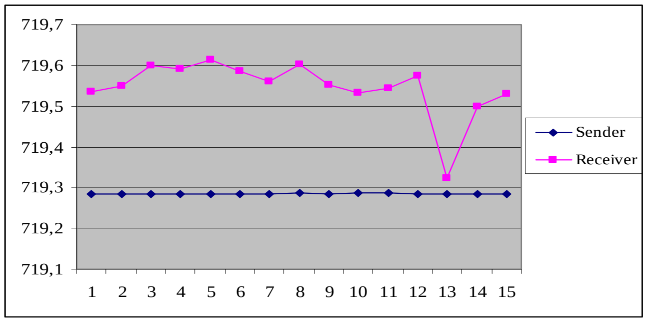

Figure 9. Available Capacity

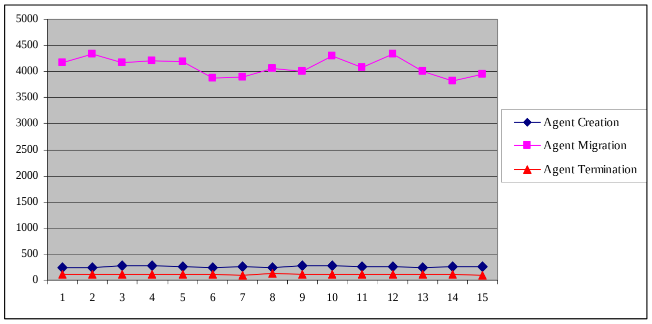

Figure 10. Elapsed time in milliseconds

We have also tested the MA creation time, migration time and termination time for the evaluation of cost of these parameters. The results are shown in Figure 10. It can be seen that, the mean agent creation time was 259.2666667 milliseconds. Similarly, the mean agent migration time and agent termination time were 4093.4 milliseconds and 108.9333333 milliseconds, respectively. Considering these values, agent migration time is much greater than the agent creation time and agent termination time. Because the operations of Squawk JVM are relatively slow, the serialization of agents into a message is a very time consuming process, and the radio stream communication between sensor devices is quite slow.

VI. Conclusion

The design, development, deployment and implementation of a Java Agent-based Framework for Sun SPOT- JAFSPOT, has been presented in this article. To the best of our knowledge, JAFSPOT is one of the very few Java-based, MA-based and event-based systems for Sun SPOT sensor devices of WSN that supports isolate migration. With this framework, it is possible to specify real world scenarios using static and/or MA-based applications. It also facilitates the programming of event-based and agent-based applications. Event-based approach is a particularly useful abstraction in the context of WSNs. Particularly; this approach provides a way to optimize the consumption of invaluable resources of resource-constrained WSNs. It exemplifies how to program different dynamic behaviors of the MAs during their lifetime. Providing weak migration of isolate mechanism is an important facility when considering usage of MAs to support WSN reprogramming. The possibility of executing MAs on Sun SPOT sensor devices extends considerably the varieties of applications for this platform.

Our ongoing efforts have been devoted to extending this framework for a real world application and adding a security aspect.

References

[1] I. F. Akyildiz, Y. Sankarasubramaniam, and E. Cayirci, “A survey on sensor networks”, IEEE Communications Magazine, vol. 40 (8), pp. 104-112, 2002.

[2] J. Yick, B. Mukherjee, and D. Ghosal, “Wireless sensor network survey”, Computer Networks, vol. 52 (12), pp. 2292-2330, 2008.

[3] F. Aiello, G. Fortino, R. Gravina, and A. Guerrieri, “A Java-based platform for programming wireless sensor networks”, The Computer Journal, vol. 54 (3), pp. 439-454, 2010.

[4] S. González-Valenzuela, M. Chen, and V. C. M. Leung, “Applications of mobile agents in wireless networks and mobile computing”, Advances in Computers, Marv Zelkowitz (Ed.), vol. 82, pp. 113-163, 2011.

[5] M. Chen, T. Kwon, and Y. Choi, “Mobile agent-based Directed Diffusion in wireless sensor networks”, EURASIP Journal on Applied Signal Processing, (1), 2007.

[6] E. Shakshuki, H. Ghenniwa and M. Kamel, “Agent-base system architecture for dynamic and open environments”, Journal of Information Technology and Decision Making, vol. 2 (1), pp. 105-133, 2003.

[7] P. Wang, “A brief survey on cooperation in multi-agent system”, International Conference on Computer Design and Applications (ICCDA), vol.2, pp. 39-43, 25-27 June 2010.

[8] R. Lopes, F. Assis, and C. Montez, “MASPOT: A mobile agent system for Sun SPOT”, Tenth International Symposium on Autonomous Decentralized Systems, Tokyo, 2011.

[9] F. Aiello, G. Fortino, R. Gravina, and A. Guerrieri, “MAPS: A mobile agent platform for Java Sun SPOTs”, Proc. 3rd Int. Workshop on Agent Technology for Sensor Networks (ATSN), Budapest, Hungary, 12 May 2009.

[10] F. Aiello, G. Fortino, R. Gravina, and A. Guerrieri, “A Java-based agent platform for programming wireless sensor networks”, The Computer Journal, vol. 54 (3), pp. 439-454, 2010.

[11] B. Chen, H. H. Cheng, and J. Palen, “Mobile-C: a mobile agent platform for mobile C/C++ agents”, Software: Practice. and Experience, vol. 36, pp. 1711-1733, 2006.

[12] B. Chen and H. H. Cheng, “A runtime support environment for mobile agents”, Proceedings of the 2005 ASME/IEEE International Conference on Mechatronic and Embedded Systems and Applications (MESA05), Long Beach, CA, September 2005. American Society of Mechanical Engineers: New York, pp. 37-46, 2005.

[13] B. Chen, “Runtime support for code mobility in distributed systems”, PhD Thesis, Department of Mechanical and Aeronautical Engineering, University of California, 2005.

[14] Sun™ Small programmable object technology (Sun SPOT). (2012), http://www.sunspotworld.com/.

[15] Sun Microsystems. Java 2 Platform, Micro Edition (J2ME)-Connected Limited Device Configuration-Specification-version 1.1, Mar. 2003.

[16] D. Simon and C. Cifuentes, “The Squawk Java Virtual Machine: Java on the Bare Metal”, Proc. 20th Object-Oriented Programming, Systems, Languages and Applications (OOPSLA 2005), San Diego, CA, USA, October 16-20, pp. 150-151.ACM, NewYork, NY, USA, 2005.

[17] IEEE Foundation for Intelligent Physical Agents (FIPA). Agent Communication Language Specifications. (2012), http://www.fipa.org/repository/aclspecs.html.

[18] H. H. Cheng, “Scientific computing in the Ch programming language”, Scientific Programming, vol.2 (3), pp. 49-75,1993.

[19] H. H. Cheng, “Ch: A C/C++ interpreter for script computing”, C/C++ User’s Journal, vol. 24 (1), pp. 6-12, 2006.

[20] Softintegration, Inc. “Ch: An Embeddable C/C++ Interpreter”, (2012), http://www.softintegration.com.