Mobile Agents Model and Performance Analysis of a Wireless Sensor Network Target Tracking Application

Edison Pignaton de Freitas[1,2], Bernhard Bösch[1], Rodrigo S. Allgayer[3], Leonardo Steinfeld[4], Flávio Rech Wagner[2], Luigi Carro[2], Carlos Eduardo Pereira[2,3], and Tony Larsson[1]

[1] School of Information Science, Computer and Electrical Engineering, Halmstad University, Halmstad, Sweden
[2] Institute of Informatics, Federal University of Rio Grande do Sul, Brazil
[3] Electrical Engineering Department, Universidade Federal do Rio Grande do Sul, Brazil
[4] Electrical Engineering Institute, Universidad de la República, Uruguay
{edison.pignaton,tony.larsson}@hh.se, berbos09@student.hh.se, leo@fing.edu.uy, {flavio,carro}@inf.ufrgs.br, {allgayer,cpereira}@ece.ufrgs.br

Abstract. Advances on wireless communication and sensor systems enabled the growing usage of Wireless Sensor Networks. This kind of network is being used to support a number of new emerging applications, thus the importance in studying the efficiency of new approaches to program them. This paper proposes a performance study of an application using high-level mobile agent model for Wireless Sensor Networks. The analysis is based on a mobile object tracking system, a classical WSN application. It is assumed that the sensor nodes are static, while the developed software is implemented as mobile agents by using the AFME framework. The presented project follows a Model-Driven Development (MDD) methodology using UML (Unified Modeling Language) models. Metrics related to dynamic features of the implemented solution are extracted from the deployed application, allowing a design space exploration in terms of metrics such as performance, memory and energy consumption.

Keywords: Wireless Sensor Networks, Multi-agents, Overhead, Energy Consumption.

1 Introduction

Wireless sensor nodes are embedded systems used to implement a large number of applications in different areas including those where wired solutions are not suitable. Their potential usage is increased when they form a network of cooperating nodes, i.e. a Wireless Sensor Network (WSN). A WSN is an ultimate technology for applications resembling environmental and area monitoring to acquire different types of measurements. WSNs are used for monitoring wildlife, water resources, security perimeter or collect data for disaster relief and prevention systems [1].

In spite of its enormous potential uses, the real deployment of WSN-based systems presents big challenges, particularly in relation to energy resource usage. Usually,

S. Balandin et al. (Eds.): NEW2AN/ruSMART 2011, LNCS 6869, pp. 274–286, 2011.
© Springer-Verlag Berlin Heidelberg 2011

sensor nodes have their energy supply provided by batteries, which represent limited resources. Even in the best cases in which the sensor nodes are able to harvest energy from the surrounding environment, like piezoelectric sensor nodes, the available energy is still a concern. Moreover, their processing capabilities are also limited, which makes it hard to run very complex applications in a single node. Based on that and on the inherent distributed nature of sensor network applications, distributed programming is highly recommended and used for WSN [2].

Another important aspect of WSN applications is related to their operation environment. Usually WSNs operate in harsh and very dynamic scenarios, in which constant changes imply in disturbances of the previous network topology or sensing conditions, by appearance of obstacles or occurrence of communication interferences, for instance. Moreover, the events of interest that have to be monitored per se might be highly dynamic, which requires flexible capabilities in terms of behaviors and functionalities. This dynamicity demands additional features from the WSN, such as reprogrammability, adaptability and autonomy in order to change the sensor nodes' behavior according to the current needs [3]. However, besides the original complexity came from the distributed nature of WSNs applications, these additional desirable features increase even more the overall system programming complexity.

As a consequence of this growing complexity of WSN applications, the deployment of such applications becomes even more challenging. Thus, the development of new programming models is an essential step towards solutions that address such complexity but at the same time present an acceptable level of energy efficiency. In spite of the existence of models used for distributed systems for ordinary computing platforms that could solve the problems related to the distributed nature of WSN applications, they do not address the needs related to the constrained energy and processing resources presented by WSN nodes. Recognizing such situation, research is being developed towards adaptation of such models to the WSN reality.

A promise model to be used in WSN is based on mobile software agents [4]. By the inherent distributed nature of the computation performed by mobile agents, they fit to the distributed processing needs of WSN applications. An example of such usage is presented in [5], in which firefighters use WSN with mobile agents to monitor the progression of fire. However, a drawback of approaches as [5] is the ad hoc nature of the solution and the low level programming paradigm used to implement it, which makes it hard to evolve or adapt such systems. Thus, there is a need for higher level programming languages and methodologies that allow the adoption of flexible mobile agents' solutions. This alternative is becoming possible with the appearance of virtual machines for embedded systems with scarce resources, such as Squawk [6], allowing the use of programming languages with higher levels of abstraction such as Java. These advances combined with the usage of methods with higher abstraction methods, like Model-Driven Development (MDD) [7], allows the systems designers consider different alternatives for implementation and deployment in early stages of the software lifecycle. Moreover, these methods allow the traceability of changes across the design and implementation, which provides a smoother transition between different software versions, when changes are required. However, again the concern about resource usage has to be considered, as higher level programming alternatives for agents may impose high overhead depending on the

software design and how the implementation is carried on. Thus, a study about these issues is required to indicate how the adoption of this type technology with high level abstraction can be improved in WSN domain.

This paper aims to analyze the performance of a WSN system programmed with mobile software agents using a MDD methodology. To carry out this analysis, the classical mobile object tracking application for WSN was chosen [8]. The goal is to evaluate the overhead imposed by an agent support platform in relation to the application. Metrics are extracted from the implemented application to allow the analysis of its performance and the influence of the supporting platform. The outcome of these metrics provides valuable information for design space exploration, making the system designers aware about their decisions and choices during system design and implementation. The implementation was performed using AFME [9], which provides the support for the agents. It is a framework to develop agent-based systems using Java programming language. The implemented application was deployed on a network of SunSPOT sensor nodes [10].

The rest of the paper is organized as follows: Section 2 describes the methodology used to perform this study. In Section 3, the mobile object application is presented and the agent system is detailed. Section 4 presents the application models. Implementation details of the performed simulations along with acquired results are presented in Section 5. Finally, Section 6 presents the conclusion and future work.

2 Study Methodology

The road map for the proposed study is composed by three major steps: 1) Application modeling; 2) Application implementation; and 3) Metrics assessment and evaluation.

In the first step, the mobile object tracking application was abstractly modeled using UML use case diagrams to study the system requirements and to identify its functionalities. Then, based on the information achieved by this first analysis, the application was modeled according to the features provided by AFME framework [9]. The resulting model is composed by class and sequence diagrams.

AFME is a low scale agent framework, which was developed to enable the creation of planned agents for mobile devices and resource constrained devices. It is designed to handle the Constrained Limited Device Configuration (CLDC)/Mobile Information Device Profile (MIDP) subset of the Java Micro Edition (J2ME) specification. AFME is based on Agent Factory, a large framework for the deployment of multi-agent systems. The framework is complaint with FIPA specification enabling its interoperability with other FIPA-compliant environments. AFME uses a rule-based concept similar to expert systems [11] to represent the agents' behaviors and maintain a reduced set of meta-information about itself and its surrounding environment as the agents' belief. Rules' operations over the belief set determine the agents' commitments, which finally provide the actions that the agents should perform.

Based on the developed model, the second step could take place and the application was developed in Java programming language using AFME framework for execution on SunSPOTs.

The third step was achieved by acquiring the dynamic metrics of developed application, by instrumented code to retrieve performance measurements during the system execution on the SunSPOT platform. Finally, the acquired data is reported and discussed.

3 Mobile Object Tracking Application

The mobile object tracking application used in this study is based on the experiment described in [8], which uses the paradigm of software agents for location and tracking. However, unlike this referred study where agents were static, in the approach here presented the agents migrate among the sensor nodes enabling location and tracking with the cooperation among the nodes. This approach can reduce the amount of data exchanged among nodes, thus lowering the energy consumption.

The network is composed by distributed nodes, which have ability to perform sensing, processing and communication with their neighbor nodes. It is assumed that the sensor nodes have sensor devices which are able to measure the distance to the target. It is possible to use various types of sensors (sound, light and radiofrequency) for this purpose, which may be active or passive to measure the distance to the target, depending on the characteristics of the target object. This work considers that the target object emits radiofrequency signals, called beacons, which are detected by receivers located at each sensor node in the network. Depending of the signal strength received by the nodes, the software calculates the distance between the sensor and the target object based on an omnidirectional model of electromagnetic waves propagation. This model provides that the signal power decays quadratically with the distance between transmitter and receiver.

There are three types of nodes in the network: sensor node, coordinator node and target object. Each node has a distinct function in the application. The target object is characterized by the emission of beacons on the network. The sensor node is responsible for performing the sensing of signals sent by the target object. These nodes are in a greater number in the network. The coordination node is responsible to manage the entry and exit of agents in the network and stores a database with the trajectory of the target object. Figure 1 illustrates this scenario.

The algorithms for location and tracking of the target object in the network are performed by software agents which can be of two types: Resident Agent (RA) and Collaborative Agent (CA). The RA agents can be found fixed in a sensor node and can be a RA_Coordinator (RAC) when the node is a coordinator node or RA_SensorNode (RAS) when it is in a sensor node. They communicate with CA agents when these agents are on the same node. The CA agents have the ability to move among the nodes of the network, being responsible for performing tracking and calculating the position of target objects. They can be a CA_Master (CAM) or a CA_Slave (CAS). As CAM they have the task of coordinating a cluster formed by the sensor nodes that are closer to the target to calculate the location of the target object. This is done by requesting data from others agents and checking when an agent should migrate to another node. As CAS they have the function of cooperating with the agent CAM by sending data.

The injection of agents in the network is accomplished by the coordinator node. When a sensor node detects a beacon from a target object for the first time, it informs the coordinator node (Figure 1 (a)). The coordinator node waits to receive notification of at least three nodes to inject the CAM in network. The sensor node which receives the CAM will be chosen by the shortest distance between the sensor node and the target object (Figure 1 (b)). When the CAM is initialized in the sensor node, it notifies the resident agent about its presence in the sensor node and that it is able to cooperate. Then the resident agent begins to send the distance information to the collaborative agent. This node makes two clones of the agent generating the CASs, and sends these clones to neighboring nodes (Figure 1 (c)). The CASs are initialized in the sensor nodes that received them and inform the respective resident agent, which initiate to send the measured distances. Then CASs send the distance to the CAM.

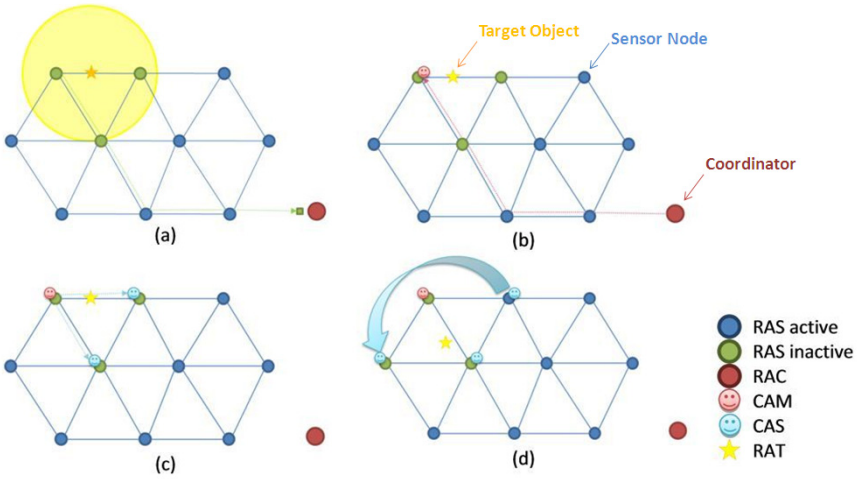


Fig. 1. Agents movement in the sensor network for locations and tracking of target object

Aiming to limit the complexity of application, some characteristics were identified and restrictions applied for the implementation:

- Only one target object is presented within the network at a time;
- The sensor nodes and the target object are on a flat surface (2D). Thus, it is possible to determine the position of the target based on the distance measured by at least three sensor nodes. For the case of a 3D surface, not addressed in this work, it would be required the distance values from at least four sensor nodes;
- The relation of the distances between nodes and the communication range assures direct communication between at least three nodes, i.e. an error free communication is assumed;
- The routing protocol allows sending messages to the coordinator, and its current position is known by the network nodes.

3.1 Calculation of the Target Object Position

The calculation of the target object position is accomplished by triangulation among three neighboring sensor nodes in the network. The network has a regular shape where the nodes are arranged in a triangular fashion equidistant from one another. The formula for determining the position of the target object (x_0, y_0) is represented by (1) where it is necessary to know the position of three sensor nodes (x_i, y_i) and the distances of these nodes to the target object (r_i) for i = 1, 2, 3.

The CAM agent is responsible for execute the triangulation algorithm. It has the knowledge of the position of two neighboring nodes and obtains the distance measured by the CAS agents in relation to the target object.

$$p_0 = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = A^{-1} \cdot b \tag{1}$$

where:

$$A = 2 \cdot \begin{bmatrix} x_3 - x_1 & y_3 - y_1 \\ x_3 - x_2 & y_3 - y_2 \end{bmatrix} \tag{2}$$

and

$$b = \begin{bmatrix} (r_1^2 - r_3^2) \\ (r_2^2 - r_3^2) \end{bmatrix} - \begin{bmatrix} (x_1^2 - x_3^2) + (y_1^2 - y_3^2) \\ (x_2^2 - x_3^2) + (y_2^2 - y_3^2) \end{bmatrix} \tag{3}$$

The position calculation can degrade the performance of sensor nodes which have a limited processing power and energy sources like traditional sensor network nodes. Thus, optimizations can be performed in the algorithm for calculating the relative position of the target object given by (1)-(3). The matrix A, given by (2), will remain constant until one of the collaborating agents have to perform a migration. Then, the result of the inverse operation of matrix A can be stored until a migration occurs, which will reduce the number of operations performed by the algorithm. The same can be applied to a part of the vector b, represented by (3).

3.2 Calculation of Agent Movement (Migration)

The collaborating agents have the characteristic of mobility between the sensor nodes to follow up the target object, minimizing the amount of messages exchanged between network nodes which is one of the goals of distributed processing and collaborative information processing performed by the multi-agents. This feature requires that agents have knowledge about the network topology and to which node they have to migrate in order to be closer to the target object. In Figure 1 (d) depicts the migration of agents between network nodes.

The CAM determines if a CAS agent, or itself, needs to migrate by comparing the distance between the current node that hosts the agent and the target object to a predetermined threshold value. Then, due to the equilateral triangular nodes distribution in the network, the CAM agent can calculates the position of the node

which the agent has to migrate, by using (4). In this case the agent located at (x_2, y_2) has to migrate to the node located at (x_4, y_4).

$$\begin{aligned} x_4 &= x_1 + x_3 - x_2 \\ y_4 &= y_1 + y_3 - y_2 \end{aligned} \tag{4}$$

4 Application Model and Implementation

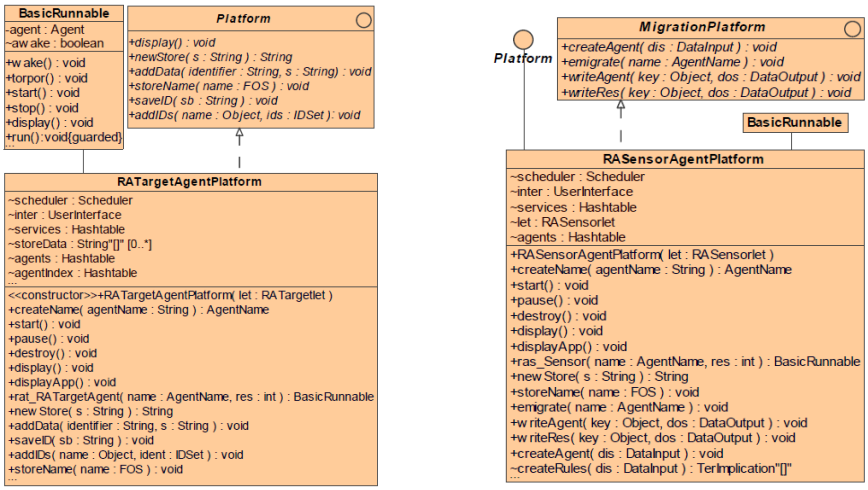
The software was developed based on models created for the application. This software development approach based on the creation and specification of models for the describing application is called Model Driven Development (MDD) [7]. This approach uses specifications (called models) developed in high-level domain languages for software specification, in which UML [12] is a widely used standard and the one chosen for this work.

Some issues needed special attention. First, the application considered in this work, as any collaborative one, involves different parts that run independently in each node of the network, so each of them need to be modeled. Another important issue is the interaction between the network nodes through communication messages may be modeled in two different ways: modeling the communication through a "network" object, where all network messages are sent to and by this object; or modeling the communication channel as an association between the objects that need to interchange messages, so they directly call the appropriate method of each other. This second option translates better the distributed nature of WSN applications, such as the tracking application studied in this work, besides it is simpler to be implemented than the first one, due to the fact of avoiding an intermediary element in the system. For these reasons the second alternative was the selected one.

The Class Diagram is one of the main structural diagram in the UML, in which the classes, interfaces and their relationships are represented. The developed model was divided in three main class diagrams, one for each type of node, namely sensor node, coordinator node and target object. Additionally other class diagrams were created for the CAs. The model includes the classes from the application itself plus the essential ones from the AFME framework. Notice that the framework has many other classes, which are not presented in this paper because they were not used in this project or they are not essential for the overall understanding of the software structure. Parts of the developed class diagrams are presented in the following, which were selected in order to show the relationship between the most important classes from the framework and those from the application. Figure 2 shows the class diagrams for the classes that provided the support to run the agents in the three different kind of nodes, i.e. target object (Figure 2a), sensor node and coordinator node (Figure 2b).

The class diagram of Figure 2a show the class `RATargetAgentPlatform` which implements the interface `Platform` from the AFME API, which provides the basic functionalities to the agents that are hosted in a node. This class contains an instance of the `BasicRunnable` class, which provides the basic functionalities to execute an agent, updating its beliefs and proceeding the agent's control process. The instance of this class that represents an agent is the `RATarget` in the case of target

node. Figure 2b presents a similar diagram for the sensor and coordinator nodes. In the case of these two nodes, besides the implementation of the `Platform` interface, the interface `MigrationPlatform` is also implemented by the class `RASensorAgentPlatform`, which will instantiate an object `RASensorNode` for the sensor nodes and an object `RACoordinator` for the coordinator node from the class `BasicRunnable`. The interface `MigrationPlatform` provides the functionalities that are need for both types of nodes, sensor and coordinator, to send and receive the mobile agents in the application, i.e. the CA_Master and CA_Slave.



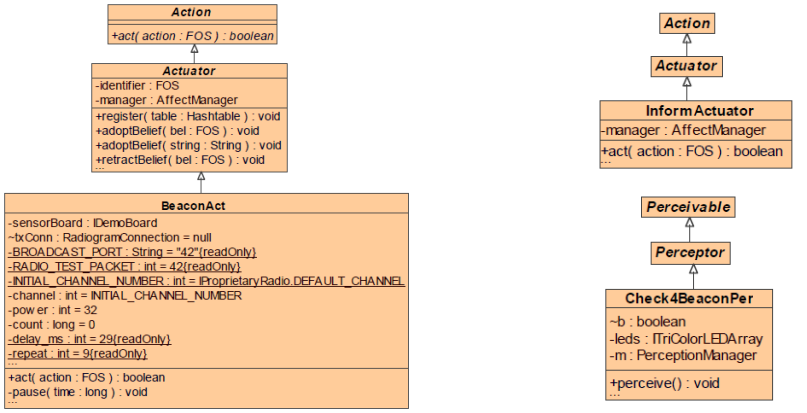
(a) (b)

Fig. 2. Class diagram for the basic classes supporting the agents in each node: (a) Target Object; (b) Sensor and Coordinator Nodes

The classes shown so far do only provide the support to run the agents in the nodes. In AFME, the real semantics of the agents are expressed classes representing the agents' perceptions, the Perceptors, and the classes that implement their actions, the Actuators. Figure 3a presents the classes that model the actuator for the RA_Target, while Figure 3b presents the actuator and perceptor of the RA_Sensor.

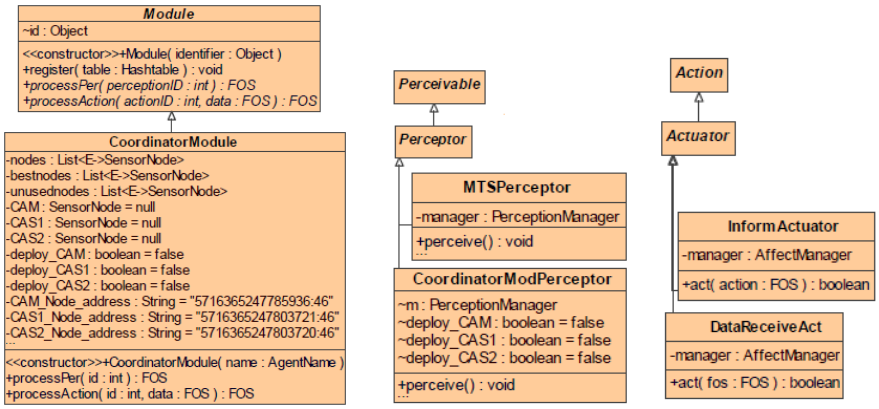
Notice that the RA_Target has no perceptor and just an actuator, the `BeaconAct` in Figure 3a, which is responsible for the action related to sending beacons which will be perceived by the RASensorNode, by means of the functionality implemented in the class `Check4BeaconPer` presented in Figure 3b. This perception will be stored in the agent's belief and informed to other agents resident in the node by means of the functionality in the implemented in the `InformActuator` class (Figure 3b).

Figure 4 present the class diagram with the perceptors and actuator for the RA_Coordinator.



(a) (b)

Fig. 3. Class diagram: (a) RA_Target Actuator; (b) RA_Sensor Perceptor and Actuator



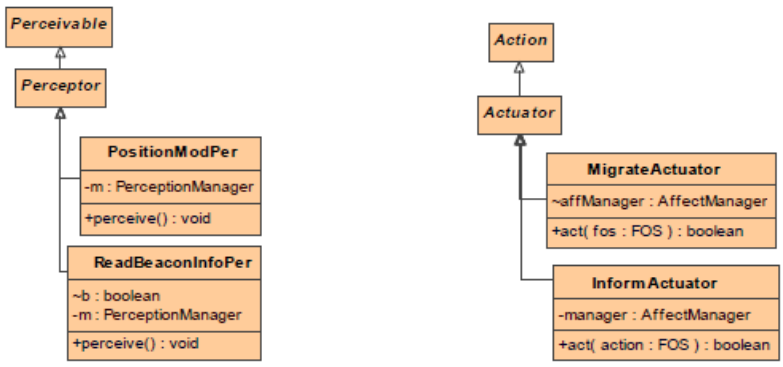
(a) (b) (c)

Fig. 4. Class diagram for RA_Coordinator: (a) Module; (b) Perceptors; (c) Actuators

The `CoordinatorModule` presented in Figure 4a extends the class `Module` from the AFME API, and it is mainly responsible for deploying the CA agents in the network. The result of this deployment is perceived by the `CoordinationModPerceptor` (Figure 4b), which keeps track of the CA agents after the deployment, updating the RACoordinator beliefs. This deployment of the CA agents is done after the RASensorNodes have sent the information about the first beacon to the RACoordinator, which handles this information by the actuator `DataReceiveAct` (Figure 4c).

Figure 5 presents perceptors and actuators of the CA_Master agent. The `ReadBeaconInfoPer` class is responsible for the update of the belief state upon a

received beacon from the target node, while the `PositionModPer` updates the target position information (Figure 5a). The `MigrateActuator` is responsible for the agent migration when the target leaves the range of sensing of the current node in which the agent is hosted, while the `InformActuator` is responsible to send messages to the CA_Slave in the neighbor sensor nodes.



(a) (b)

Fig. 5. Class diagram for CA_Master: (a) Perceptors; (b) Actuators

The class diagram for the CA_Slave has similar classes as presented for the CA_Master, but with differences in the semantic implemented in its actuators and perceptors, such as sending of information about the target position to the CA_Master and the migration upon receiving a message from the CA_Master.

5 Results: Metrics Analysis

The developed software was evaluated by means of extraction of dynamic metrics. These metrics represent information about the application execution, which were achieved during the system runtime by instrumented code. The measurements provide information about the cost in running the application in terms percentage of CPU time utilization, average energy consumption and memory usage.

The testbed was deployed in a network of six SunSPOT nodes: one represent the target object to be detected by other four sensor nodes and the last one was used as a reference node. The reference node was without any agent running on it during the whole time of the experiment, namely Configuration 1. The node representing the target object ran the RA_Target agent, Configuration 2. The other sensor nodes were initially running the RA_SensorNode agent to perform the target detection which was out of the sensors' range, Configuration 3. Then, the nodes were classified during the experiment according to its workload (processing and communication) and depending on the agents they were hosting. When the target entered to the network, three of them performed the detection, Configuration 4. Finally, they performed the target tracking,

two nodes running a CA_Slave agent, Configuration 5, and one a CA_Master, Configuration 6. The RA_Coordinator was running in a PC that represented a base station.

Table 1 presents the results for CPU utilization and current consumption, directly related to the energy drain. Configuration 2 does not consume much processing resources, but energy, which is explainable because the target node periodically sends beacons to the network and wireless communication module demands significant energy to send data. Configuration 3 has a very small CPU utilization if compared with the reference Configuration 1, which is explainable by the fact that it is just waiting for receiving beacons. Its energy consumption is not much high either. However, by the appearance of the target in the network, the sensor nodes in the target range will be in Configuration 4, which presents higher energy consumption, which is explainable by the first beacon message that they send to the coordinator node. The CPU utilization is increased, but not much significantly. With the injection of the mobile CA agents and consequently move to Configurations 5 and 6, it is noticed a significant increase in the CPU utilization, as well as in the energy consumption, which are explained by the execution of the calculations presented in Section 3, as well as the communication among the CA agents.

Table 1. Results for the CPU Utilization and Current Consumption

Sensor Node Configuration	CPU Utilization (%)	Current Consumption (mA)
1 - No agent - Reference node	1,52	59,7
2 - Target Sunspot	17,26	74,4
3 - No CA / No Target present	3,44	65,8
4 - No CA/ Target detection	15,73	71,7
5 - CA_Slave	28,32	73,9
6 - CA_Master	60,85	85,1

Table 2 presents the amount of memory used in each node according to its configuration, and the remaining available memory space. In terms of memory, the burden of the agent oriented approach is not as strong as it is for the energy and CPU usage. As expected, the sensor node running the CA_Master requires more memory, but the difference in relation to the other configurations is not too significant.

Table 2. Results for the memory usage

Sensor Node Configuration / Memory	Conf 2 Target	Confs 3, 4 Sensor	Conf 5 Slave	Conf 6 Master
Free (Bytes)	337164	334852	320300	301164
Used (Bytes)	106359	108668	123221	142358

By analyzing the achieved results, it is possible to evaluate the imposed overhead due to the use of the adopted agent-oriented approach to implement the WSN application. This information can be used by the system developer to consider alternatives to reduce this overhead, for example, observing the high cost in the node running the CA_Master, the designer can consider redistribute some of the computation to the other nodes that run the CA_Slave. Another possible consideration is in relation to messages exchanged among the nodes, which will impact the design of the application.

6 Conclusion and Future Work

This paper presented the application of mobile agents to implement WSN applications. The classical target tracking application was chosen as case study, which was implemented, deployed in real sensor nodes, the SunSPOTs, and evaluated. The achieved results indicated a significant consumption of processing resources and moderate energy consumption. The memory utilization was not a specific concern, for this particular application in the SunSPOT platform, as a significant amount of memory was not used. These results can be used as basis for new designs in which high level decisions can have their impact considered in the system performance.

There are a number of tasks in the pipeline to proceed with this work. One of them is to perform the implementation of the same design using different agent frameworks, so that the specific burden due to the supporting framework can be evaluated and compared. Moreover, other metrics can be acquired, such as static ones to compare different implementations. Another future work is to breakdown the energy consumption for different workload cases, to better understand the impact of an agent-base framework in the communication channel utilization, usually pointed as the major energy consumer.

References

1. Arampatzis, T., Lygeros, J., Manesis, S.: A Survey of Applications of Wireless Sensors and Wireless Sensor Networks. In: Proceedings of the 13th Mediterranean Conference on Control and Automation, Limassol, Cyprus, pp. 719–724 (2005)
2. Zhao, F., Guibas, L.: Wireless Sensor Networks: An Information Processing Approach. Elsevier, Amsterdam (2004)
3. Allgayer, R.S., Götz, M., Pereira, C.E.: FemtoNode: reconfigurable and customizable architecture for wireless sensor networks. In: Rettberg, A., Zanella, M.C., Amann, M., Keckeisen, M., Rammig, F.J. (eds.) IESS 2009. IFIP Advances in Information and Communication Technology, vol. 310, pp. 302–309. Springer, Heidelberg (2009)
4. Lange, D.B., Oshima, M.: Seven Good Reasons for Mobile Agents. Communications of the ACM 42(3), 88–89 (1999)
5. Fok, C.-L., Roman, G.-C., Lu, C.: Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications. In: Proc. of the 24th ICDCS, pp. 653–662 (2006)

6. Simon, D., Cifuentes, C.: The squawk virtual machine: Java on the bare metal. In: Proceedings of Conference on Object-oriented Programming, Systems, Languages, and Applications, pp. 150–151. ACM Press, New York (2005)
7. France, R., Rumpe, B.: Model-driven Development of Complex Software: A Research Roadmap. In: Proceedings of Future of Software Engineering 2007, pp. 37–54. IEEE Computer Society, Washington, DC (2007)
8. Tseng, Y.C., Kuo, S.P., Lee, H.W., Huang, C.F.: Location tracking in a wireless sensor network by mobile agents and its data fusion strategies. In: Information Processing in Sensor Networks Book, p. 554. Springer, Berlin (2003)
9. Muldoon, C., O'Hare, G.M.P., Collier, R., O'Grady, M.J.: Agent Factory Micro Edition: A Framework for Ambient Apps. Comp. Science, pp. 727–734. Springer, Berlin (2006)
10. SunSPOTWorld, `http://www.sunspotworld.com` (accessed April 2011)
11. Giarratano, J.C., Riley, G.: Expert Systems: Principles and Programming. Brooks/Cole Publishing Co., Pacific Grove (1989)
12. Object Management Group, Unified Modeling Language, UML (2011), `http://www.uml.org` (acessed March 11, 2011)