MASIM: A Tool for Simulating Mobile Agent Applications on Wireless Sensor Networks

Marcos Camada[1], **Carlos Montez**[1], **Flávio Assis**[2]

[1]Pós-Graduação em Eng. Automação e Sistemas – Univ. Federal de Santa Catarina (UFSC)
88040-900 – Florianópolis – SC – Brazil

[2]Programa de Pós-Graduação em Mecatrônica – Univ. Federal da Bahia
40170-110 – Salvador – BA – Brazil

{mcamada, montez}@das.ufsc.br, fassis@ufba.br

Abstract. *A mobile agent is an autonomous software entity that is able to migrate between nodes of a distributed system, carrying its code and execution state. Recently, mobile agents have been proposed to be used in wireless sensor networks, as an approach to reduce energy consumption of wireless sensor nodes. The advantages provided by mobile agents, however, depend on specific application aspects and on the particular way how they are used to accomplish some task. Due to the lack of suitable tools for the simulation of mobile agent-based applications over wireless sensor networks, this paper introduces a tool for this purpose, called MASIM. This tool extends the toolboxes of MATLAB/TrueTime. This paper describes the general features of MASIM and illustrates its use to a specific scenario, where a protocol based on mobile agents and one based on message passing are compared.*

1. Introduction

A Wireless Sensor Network (WSN) is a type of wireless network whose nodes are computing devices equipped with sensors, such as temperature, light or humidity sensors [Yu et al. 2004]. These networks might be composed of tens or even thousands of nodes, that can be deployed randomly in a environment, present mobility and be self-organizable [Malik and Shakshuki 2007]. Sensor nodes are typically small autonomous devices with strongly limited processing power and memory capacity, with a short-range transceiver and with non-replaceable batteries [Ye et al. 2001]. Therefore, they should be designed so as to spend as less energy as possible, so that network lifetime can be increased.

Mobile agents have been proposed to be used in WSN as an approach to conserve energy of nodes (see, for example, [Chen et al. 2006]). A mobile agent is an autonomous software entity that has as its main feature the ability to migrate between nodes, carrying its code and execution state. Mobile agents may provide additional advantages such as reduction of network load, complexity reduction in the design of interfaces provided by nodes, and capacity to autonomously adapt to changes in the system, thus contributing to systems robustness and fault tolerance [Lange and Oshima 1999]. However, the advantages provided by mobile agents depend on specific application aspects and on the particular way how they are used to accomplish some task [Jansen et al. 1999].

Simulation is one important tool to help evaluating the potential advantages of using mobile agents in specific application scenarios. There are currently many simulation tools for WSN, but, to the best of our knowledge, none of them provides specific abstractions and functionality to simulate mobile agents over WSN.

This paper describes a tool to fulfil this need. We describe MASIM (Mobile Agent SIMulator in Wireless Sensor Network), a toolbox for simulating mobile agents on MATLAB [MathWorks 2009]. MASIM provides abstractions and functionality for modelling mobile agents, their environment and interactions between the many system components involved. The model used in MASIM is compatible with the mobile agents model defined by FIPA (Foundation for Intelligent Physical Agents) [FIPA 2004] and OMG (Object Management Group) [OMG 1997], two important standardized efforts in the context of mobile agent systems. MASIM uses an energy model that defines how each component spends energy which is based on the hardware specification of Mica2 [Crossbow 2009], one of the most important sensor node platforms. MASIM can also be easily extended to use energy models based on different platforms. We believe that extending MATLAB has the particular advantage of adding functionality to a system which is very frequently used for computer simulation.

This paper is structured as follows. In Section 2 we compare our approach to other existing wireless network simulation environments. In Section 3 we describe MASIM. In Section 4 we describe a particular use of MASIM to evaluate the advantages that mobile agents might bring for a specific scenario. Finally, in Section 5 we conclude the paper.

2. Related Work

MATLAB is a tool for the development and analysis of algorithms and data visualization with main focus on numeric computation [MathWorks 2009]. The MATLAB environment provides a set of libraries, called toolboxes, which can be developed in the native language of MATLAB or C++ MEX (based on ANSI C++). MATLAB can be used integrated with Simulink, which provides a graphical environment and a set of libraries for the design, simulation, testing and implementation of different types of systems, including communication, control and signal processing systems [Simulink 2009]. Simulink can be extended by libraries. A library of particular importance for this work is TrueTime [Ohlin et al. 2007]. TrueTime provides resources for the simulation of real-time control systems. It provides a set of functions and programming building blocks for specifying such systems: kernel block, network block, Wireless Network and a battery block [Ohlin et al. 2007]. MATLAB with Simulink and TrueTime thus provide resources for defining an energy model for a WSN and support IEEE 802.15.4 [IEEE 2006], which is becoming a *de facto* standard for the physical layer and medium-access control sublayer of sensor networks.

There are currently many different simulation tools for wireless sensor networks. Some of them, closely related to our system, are: TOSSIM [Levis et al. 2003], Atemu [Polley et al. 2004], SENSE [Chen et al. 2004], ns-2 [Fall and Varadhan 2008] and J-Sim [Sobeih et al. 2005].

TOSSIM is a discrete event simulator for wireless sensor nodes which are based on the TinyOS operating system. TOSSIM simulates the behavior of hardware components, such as the ADC, clock and EEPROM memory, the flash boot sequence, and components of the wireless communication protocol stack. The main feature of this simulator is the fact that the code used for simulation can be installed with no modifications on real devices.

Atemu (Atmel Emulator) is a sensor network simulator which is based on the standard Mica2 [Crossbow 2009] architecture and the AVR (Advanced Virtual RISC) microprocessor (although it can simulate heterogeneous WSN). The system provides components to be used in the specification of the simulation scenario, such as processor, clock and radio device. It allows the simulation of low-level operations on sensor nodes.

SENSE is a discrete event simulator for sensor networks which is based on the IEEE 802.11 standard with DCF (Distributed Coordination Function) as its physical / MAC sublayer. This simulator implements a battery model, which enables a control over energy spent based on the used electrical current.

NS-2 (Network Simulator 2) is a discrete event simulator that has a broad support for simulation of wired and wireless networks. Its support for modelling energy is, however, very limited.

J-Sim is a simulation tool for WSN developed in Java. It provides components for modelling typical WSN elements, such as battery, processor and radio models, and a phenomena generator. It is based on the IEEE 802.11 standard at the physical layer and MAC sublayer.

Among the systems cited above, MATLAB was the system which provided most adequately basic building blocks for constructing MASIM, specially due to its support for IEEE 802.15.4 and bulding blocks for specifying energy models.

3. MASiM

In this section we describe the main features and some implementation aspects of MASiM. MASiM was built using MATLAB with Simulink and TrueTime. We first provide a description of the system from a conceptual point-of-view, presenting the adopted WSN model (Section 3.1) and agent model (Section 3.2). We then describe the main system classes (Section 3.3), the system task model (Section 3.4).

3.1. WSN Model

In MASiM, a WSN is composed of two types of nodes: sensor node and base station. A WSN might have more than one base station. Base stations and sensor nodes have different hardware and software characteristis, but all sensor nodes are homogeneous. All nodes, including the base station, are capable of moving in the environment. Thus communication links between nodes are dynamic.

The main function of a sensor node is to monitor some physical characteristic of an environment and transmit the sensed data to the base station through wireless links. Each node maintains information about its energy level, its position in the environment (localization) and about time. The clocks of nodes are all synchronized. Each node is uniquely identified in the network.

Sensor nodes are organized in a mesh network (mesh) as defined by Zigbee[1]. Not all sensor nodes might reach the base station in a single communication hop. Thus, communication between sensor nodes and the base station will be primarily multihopping.

[1]http://www.zigbee.org

The range of the base station radio device is assumed to be larger than the range of sensor node radios. Thus, the resulting communication network is asymmetric, in the sense that a base station might reach a sensor node with a single hop, but this node might not reach the base station directly. The nodes are aware of their active neighbouring nodes and each node periodically broadcasts its location to its neighbours. Additional data, such as the energy level of the node, might be piggybacked in these periodical messages.

The network uses IEEE 802.15.4 without beacon [IEEE 2006] as the physical layer and MAC (Medium Access Control) sublayer. Since there is no beacon, there is no formation of superframe, and (unslotted) CSMA/CA (Carrier Sense Multiple Access / Collision Avoidance) is used as the medium access control mechanism.

3.2. Mobile Agent Model

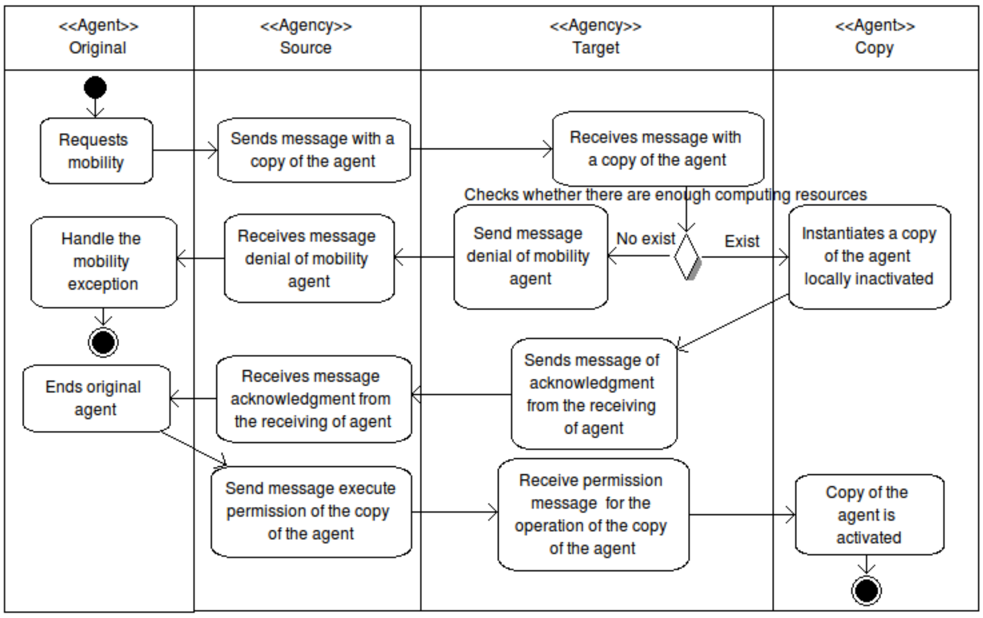
The mobile agent model adopted in MASiM is based on the model defined by FIPA [FIPA 2004]. A mobile agent can be in one of the following states: **Started**, **Active**, **Suspended**, and **Waiting**. The meaning of these states and the possible state transitions are described below.

On each node, mobile agents execute on logical places called here *agencies*. From a functional point-of-view, an agency represents the needed functionality that must be present on a node so that agents can execute there. We will, however, use the terms agency and node here interchangeably to denote the place where agents execute.

A mobile agent is instantiated at a node in the **Start** state. In this state, the agent receives its mission and its unique identifier. The user of MASiM is responsible for ensuring the uniqueness of this identifier. When the agent starts execution, it enters the **Active** state.

While at the **Active** state, an agent performs the operations defined in its mission. Two special operations are the movement and cloning operations. The implementation of these operations involves the execution of a specific protocol for mobility and cloning. Before an agent can move from an agency to another or before an agent can be cloned at a specific node, a negotiation protocol is carried out between the agency where the agent currently is, which will be called here the *original agency*, and the *target agency*, i.e., the agency to where the agent wants to move or the agency where the new instance of the agent will be created. The agent stays in the **Suspended** state from the beginning of the movement or cloning process until its end.

The mobility protocol is illustrated in the activity diagram in Figure 1. This protocol starts when an agent calls a movement operation, defined at the interface of the original agency. One of the parameters of this call is the identifier of the target agency. At this point, the agent enters the **Suspended** state. The original agency sends a message that contains a copy of the agent (code and state) for the target agency. The target agency will check whether there are sufficient resources for instantiating the agent locally. If the resources are sufficient, the agency creates the new instance of the agent. This instance is, however, created in the **Suspended** state. The target agency then sends a message to the original agency confirming that the movement operation was accepted. When the original agency receives this message, it destroys the local instance of the agent and, after that, sends a message to the target agency, allowing it to start the execution of its instance of the agent. When the target agency receives this message, it resumes the execution of the agent. The agent enters again the **Active** state. If the target agency does not have enough resources to execute the agent, it sends a message to the original agency denying receiving the agent.



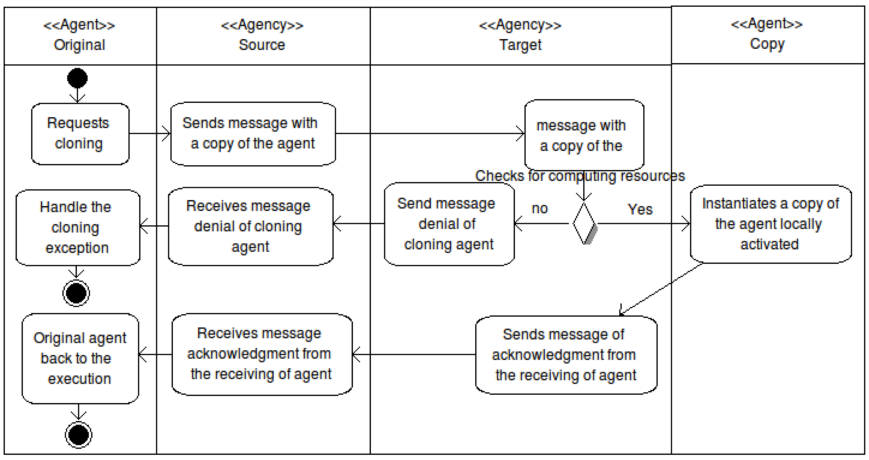
<<Agent>> Original
<<Agency>> Source
<<Agency>> Target
<<Agent>> Copy
Requests mobility
Sends message with a copy of the agent
Receives message with a copy of the agent
Checks whether there are enough computing resources
Handle the mobility exception
Receives message denial of mobility agent
Send message denial of mobility agent
No exist
Exist
Instantiates a copy of the agent locally inactivated
Sends message of acknowledgment from the receiving of agent
Ends original agent
Receives message acknowledgment from the receiving of agent
Send message execute permission of the copy of the agent
Receive permission message for the operation of the copy of the agent
Copy of the agent is activated

Figure 1. Activity Diagram of the agent mobility protocol.

The cloning protocol is illustrated in the activity diagram depicted in Figure 2. It begins in a way similar to the mobility protocol. The cloning operation has as one of its parameters the identifier of the remote agency where the clone agent shall be created (the target agency). After the agent calls the cloning operation, the agency where it is (original agency) sends a message to the target agency. The agent enters the **Suspended** state. This message contains the code and state of the agent to be cloned. The target agency checks if there are enough resources to create the local copy of the agent. If yes, it sends a message to the original agency confirming the creation of the clone agent, but, differently from the mobility protocol, the agent instance at the target agency immediately enters the **Active** state. When the original agency receives the confirmation message, the local instance of the agent enters the **Active** state too. As in the mobility protocol, if there are not enough resources for creating the agent at the target agency, the target agency sends a message to the original agency denying creating the clone.

During an agent movement or cloning operation, the agent execution and data state are transformed into a byte stream (a process called serialization), to be transfered to the target node. At the target node, this byte stream is used to reinstantiate the agent or create a new copy of it (deserialization) [Guenes et al. 2003]. This process is performed by the TrueTime toolbox, and how this is done is out of the scope of this work. However, the agent size can increase during its execution, due to data collected along its path. In MASiM, the user is responsible for determining how the agent size will grow at each migration.

An agent can send and receive messages. The process of receiving messages is blocking. When an agent is waiting for a message, it enters the **Waiting** state. It only



<<Agent>> Original
<<Agency>> Source
<<Agency>> Target
<<Agent>> Copy
Requests cloning
Sends message with a copy of the agent
message with a copy of the
Checks for computing resources
Handle the cloning exception
Receives message denial of cloning agent
Send message denial of cloning agent
no
Yes
Instantiates a copy of the agent locally activated
Original agent back to the execution
Receives message acknowledgment from the receiving of agent
Sends message of acknowledgment from the receiving of agent

Figure 2. Activity Diagram of the protocol for cloning the agent.

enters the **Active** state again when a message arrives. If a message is sent to a specific agent at an agency and this agency is not at this agency, the message is discarded.

3.3. Main MASiM Classes

The main classes in MASiM are: **NodeMessage**, **AgencyMessage**, **MessageInformationAgency**, **AgentMessage**, **InformationAgency**, **Phenomenon**, **Agent**, **Mission**, **State**, **Nodes** and **Agency**.

Nodes, agencies and agents communicate with each other by exchanging messages. In MASiM messages are exchanged asynchronously. If an entity sends a message to another non-existent entity, the message is discarded. MASiM defines a message class hierarchy. Three types of messages are defined: **NodeMessage**, **AgencyMessage** and **AgentMessage**.

The **NodeMessage** class models messages exchanged between nodes. The main attributes of this class are: *sender*, *target* and *identifier*. The *sender* refers to the node that sends the message. The *target* attribute represents the node to which the message is sent. The *identifier* attribute refers to the unique identifier of the message. The contents of a **NodeMessage** is the data that is sent to the destination node. This content is represented by the **AgencyMessage**.

An agency message, represented by the **AgencyMessage** class, like **NodeMessage**, has as attributes the agency that sent the message (*sender*), the agency to which the message is being sent (*target*) and the unique identifier of the message (*identifier*).

MessageInformationAgency is an abstract class that can be extended by the classes **Agent**, **InformationAgency** and **AgentMessage**. The **MessageInformationAgency** class contains the phenomenon of the sender agency. Class **Phenomenon** represents the phenomenon that a given node monitors. This class has an attribute that identifies the type of phenomenon (*type*).

The **Agent** class models the agents. This class has as attributes a unique (agent) *identifier*, an attribute that contains the set of agencies that the agent has already visited (*trace*) and, if the agent is a clone of another agent, it contains also an attribute that identifies the agency from where the agent was created (*parentAgency*). This reference becomes obsolete if the original agent moves and does not send a message to its cloned agents informing its new location. At the time an agent is instantiated, the agent receives its identifier and its mission. The **Agent** class contains an additional attribute which represents this mission (*mission*). This attribute can not be changed.

In addition to its attributes, the **Agent** class has also the following methods: *move*, to move the agent to a target agency; *clone*, to create a clone of the agent; *sendMessage*, to send a message to one or a set of agents - if a particular agent is not specified, the message is sent to all agents in a particular agency; *getMessage*, to receive a message; *enabled*, to verify if the agent is in the **Suspended** state; and *executeMission*, to initiate agent execution.

The mission of the agent is represented by the **Mission** class. This class has a set of states. These states determine the set of operations that the agent must execute. These operations are defined by the user. The **Mission** class contains an identifier of the current execution state (*currentStateIdentifier*) and the identifier of the next execution state (*nextStateIdentifier*) of the agent. These attributes are used to determine what state the agent must enter after moving or being cloned.

The **MessageAgent** class models messages that an agent sends to another agent at a certain agency. This class has the following attributes: *sender*, which identifies the agent that has sent the message; *receiver*, which identifies the agent to which the message is being sent; *targetAgency*, which identifies the agency where the receiver agent must be to receive the message; and *messageContents*, which conveys the actual information to be transmitted. When the target agency receives a message, it stores it in an appropriate memory buffer, where the agent can read it. If the receiver agent is not at the target agency when the message arrives, the agency discards the message. No confirmation of receipt is sent to the sender agent.

The **AgencyInformation** class represents the data that **AgencyMessage** might contain during the execution of the mobility or the cloning protocol.

The **Node** class represents a network node. Each node is uniquely identified by the attribute *identifier*. The battery energy level of the node is represented by the attribute *battery*. The value of the local clock of the node is represented by the attribute *clock*. The attribute *memory* determines the amount of free memory available on the node. A node in a WSN can be of one of two types: base station or sensor node. The base station is modelled with the **Node** class. A sensor node is modelled with the **SensorNode** class. This class extends the features of the **Node** class and contains an additional attribute, called *Sensor*, which is the current value of the sensor. The type of phenomenon monitored by the node is represented by the attribute *phenomenon*.

An agency is modelled by the **Agency** class. This class is uniquely identified by the attribute *identifier*. The value of this attribute will match the same value of the identifier of the node where the agency is. The methods it provides are used by agents to access information about the resources of the node. To get information on the energy level of the node, the method *getEnergyLevel* is invoked. The method *getFreeMemory* returns how much free memory is available on the node. The method *getValueSensor* returns the current value of the sensor. The type of phenomenon monitored by the node is represented by the method *getPhenomenon*.

The role of the controller class **Simulation** is performed by **MATLAB** with **Simulink** and the **TrueTime** toolbox. The **Simulator** is responsible for executing the simulation, controlling time and the simulation variables, like the level of the nodes batteries, nodes mobility and the execution of the methods of the network components.

In all classes described above there are methods for reading and writing all the attributes of the class.

3.4. Programming Tasks

MASIM was programmed using C ++ MEX programming language and was based on the resources provided by the TrueTime toolbox. The use of TrueTime requires that the programming follow a model based on tasks, which can be periodic or aperiodic. A task can communicate with other tasks through shared memory regions called **Mailboxes**. To follow this programming model, MASIM tasks were organized into two groups: Node Tasks and Agency Tasks. Node Tasks are those related to operations present in all nodes, and Agency Tasks are those existing only in nodes that run the agency. Thus, the Node Task are:

- **Node's Received Message Handler:** aperiodic task triggered by an interrupt when a message arrives. This task has the role of receiving and forwarding the message to the task responsible for it;
- **Neighborhood Message Handler:** aperiodic task triggered by Node's Received Message Handler with a role to process messages with information about neighboring nodes. This neighborhood information are stored in memory for the use of agents;
- **Simple Message Handler:** task triggered by Node's Received Message Handler with a role of dealing with messages defined by a user. The way that this message should be handled is specified by the user;
- **Sent Message Handler:** aperiodic task responsible for sending messages to another node. This task is triggered after a particular task to put a information to be sent in a Mailbox. This task gets the information and creates a message to be sent;
- **Notification to the Neighborhood Node:** periodic task whose role is to send a message to neighboring nodes with information about the current node. The information that are sent are: the energy level, the node identifier and coordinates. This task puts these information in the Mailbox and triggers an event that activates the task Node's Received Message Handler. This last task will be responsible for sending messages on the node to all neighboring nodes.

The tasks that have a role to perform operations related to the agency are as follows:

- **Agency's Received Message Handler:** aperiodic task triggered by the Node's Received Message Handler. This task has the role of receiving and processing messages in its Mailbox. If the message destination is the agency, this task will process this message. This occurs when a message is related to mobility or cloning protocol. However, if the message contains an agent, this task forwards it to Creator Agent task, which is responsible for instantiating the agent on the local node.

The message destination may a local agent. Thus, this message is forwarded to the Controller Execution Agent task, which is responsible for delivering the message to the agent, if one exists. Otherwise, this message is discarded;

- **Creator Agent:** aperiodic task triggered by the task Agency's Received Message Handler. This job role is to instantiate a particular agent that has been received during mobility or cloning for the current node;
- **Controller Execution Agent:** periodic task whose role is to run the agents following a run queue. This task executes and controls the cycle of the mission of each agent.

4. Assessment of The MASIM Tool through Simulation Scenarios

4.1. Manufacturing environment scenarios

Aiming to build scenarios with mobile agents in a distributed system, a manufacturing environment was adopted, similar to that proposed by [Krishnamurthy and Zeid 2004]. In such environments, it is necessary to monitor the operation of various equipments, collect data and forward them to a centralized operation room where decisions are taken based on individual or collective information obtained by this monitoring.

A set of sensor nodes distributed in some factory environments can monitor equipments, and mobile agents could be used both to collect data in a intelligent and selective way and to convey information about the plant to destination (a base station). It was considered scenarios where a node can communicate directly with the base station only if it is in a certain distance of the base station (they are neighbors). Moreover, there is the possibility that there are unconnected nodes, that is, nodes without neighbors in the network. Finally, the data collected can be related to information of alarms from equipment malfunctioning, and there may be a need for a maximum time (deadline) for which this information reaches the operations room, and the data has a freshness constraints.

Therefore, through the use of Simulink/Matlab files (MDL files) in MASIM, some manufacturing scenarios were modeled based on two different approaches: (i) a simple diffusion-based approach; and (ii) mobile agent-based approach.

4.1.1. First Approach: A Simple Diffusion Protocol (without mobile agents)

In this approach, when a node receives a message with a monitored event from another node, it attaches the received data to its message and forwards it to neighbors. After that, when the first message arrives to the base station, it is considered that the mission is complete.

4.1.2. Second Approach: Agents that Moves Beyond The Range of Base Station

The protocol based on agents has the goal to achieve a trade-off between energy consumption and network coverage (Figure 3). In **State 0**, there is an agent in the base station that clones itself to all neighboring nodes. In **State 1**, clones choose to move to their neighboring nodes of higher energy level until they find a particular node where the target phenomenon was observed. When this happens, in **State 2**, each agent tries to send a message to the agent in the base station. However, it is possible there may be a clone agent far from the base station. In this case, this agent moves back toward base station, until it reaches a node neighboring the base station, before it send the message:

To simplify the understanding of this algorithm, states **State 3, 4, 5 and 6** only carry out the operation to complete the mission of the agent. The **States 4 and 5** handle exceptions that may occur in the cloning and mobility, respectively. When the operations of receiving target phenomenon and find phenomena are carried out successfully, the final **States 3 and 6** are invoked.

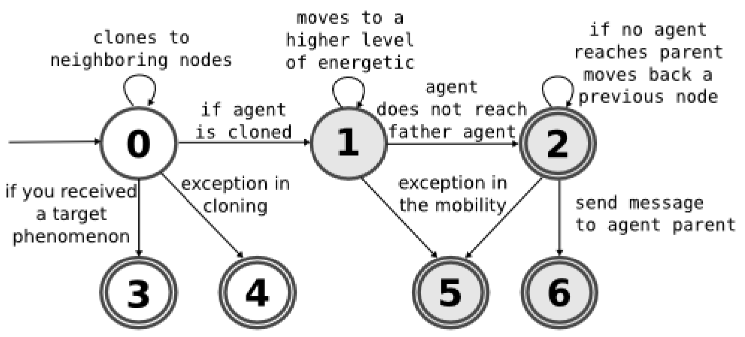

clones to neighboring nodes
moves to a higher level of energetic
if no agent reaches parent moves back a previous node
if agent is cloned
agent does not reach father agent
0
1
2
if you received a target phenomenon
exception in cloning
exception in the mobility
send message to agent parent
3
4
5
6

Figure 3. Specification of the agent's mission.

4.2. Comparison of agent-based and message-based approaches

In order to exercise the simulator, both approaches based on agents and based on diffusion messages were implemented with two densities of nodes: 50 and 100 nodes. For each density, 20 simulations were performed, each one with the duration of $100s$. In each simulation, the nodes are redistributed randomly in area.

The nodes used in the simulations are based on the hardware configuration of the Mica2. Thus, the energy source of the sensor nodes are two batteries with $3V$ and $27000J$ of energy. These nodes were randomly scattered over a fully plan area of 300m^2, and after the deployment, the nodes are considered fixed (non-mobile nodes). In each simulation scenario, it is also considered that only 10% of nodes monitors the target phenomenon. The radio signal from the antenna has an operating range of $99m$ and there may be disconnected nodes in the network. In each node there is energy consumption due to data processing and communication (sending and receiving messages). In this work, is not considered the energy consumption due to sensor or actuator tasks. Thus, the receipt and sending of one message and processing performed by the processor spends $16.62 * 10^{-4}J$, $9.6 * 10^{-4}J$ and $4.8 * 10^{-4}J$, respectively [Polastre et al. 2004].

Simulations show that the proposed approach based on agents is the better choice in respect to energy consumption (Table 1). There was a considerable increase in energy consumption by the nodes in the approach based on diffusion, because, as the number of nodes increase, the amount of messages received for each node also increase. However, the diffusion approach is more effective in coverage, because, at the end of the simulation, all nodes have been achieved by the protocol in all scenarios. The coverage parameter means the amount of network nodes that were recognized by the base station through a

Table 1. Energy consumption and coverage.

	Energy consumption			*Coverage*	
	50 *nodes*	100 *nodes*		50 *nodes*	100 *nodes*
Agent-based	0,238J	0,239J		14%	12%
Diffusion-Based	0,418J	0,601J		100%	100%

specific protocol. The average coverage for the agent-based approach were 14% and 12% for densities of 50 and 100 nodes, respectively.

In respect of time constraints, the diffusion-based protocol may be more effective to attend tight deadlines, because messages arrive at base station in a minimum time ($2s$). This value was much smaller than that obtained by agent-based approach (average of $15s$).

5. Conclusions

This work aimed to develop a tool for simulating agents in Wireless Sensor Networks called MASIM. With this tool it is possible to specify simulation scenarios using static or mobile agents. The flexibility in MASIN was achieved through the provision of functions and block diagrams, which facilitate the programming of scenarios and agents mission, determining different behaviors of the agent during its lifetime.

The programming interface developed allows the use of functions and data structures facilitating users to program the simulation models: the network topology may be defined using the block diagrams available; the programming of nodes and defining the mission of agents are made through the C++ MEX; and mission of agents can be modeled as a state machine. To exercise the simulator, showing its flexibility, was built simulations of scenarios based on mobile agents and dissemination of messages in a WSN. Results from these simulations shown possible advantages and disadvantages of using the approach of mobile agents concerning energy saving and network coverage metrics.

As future work, the tool is going to allow the user to specify custom information about the neighborhood. Moreover, it is going to allow the definition of custom protocols for cloning and mobility of agent. It is expected too that the user is going to able to program the settings using the MATLAB language, in addition to C++ MEX.

References

Chen, G., Branch, J., Pflug, M. J., Zhu, L., and Szymanski, K. (2004). Sense: A sensor network simulator. http://www.cs.rpi.edu/ szymansk/papers/wpcn.04.pdf. Accessed on Jan. 2010.

Chen, M., Kwon, T., Yuan, Y., and Leung, V. C. (2006). Mobile agent based wireless sensor networks. *Journal of Computers*, 1.

Crossbow (2009). Mica2 - wireless measurement system. ”`http://www.xbow.com/products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf`”. Accessed on Jan. 2010.

Fall, K. and Varadhan, K. (2008). *The ns Manual*. UC Berkeley and LBL and USC/ISI and Xerox PARC.

FIPA (2004). Fipa agent management specification. ”`http://www.netlib.org/lapack`”. Accessed on Jan. 2010.

Guenes, M. H., Tiersem, M. E., Yildiz, M., and Kuru, S. (2003). Performance analysis of mobile agents using simulation. In *Proc. of the Advanced Engineering Design Conference (AED2003)*, Praga, Czech Republic.

IEEE (2006). Part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (wpans). Technical report, IEEE Computer Society.

Jansen, W., Mell, P., Karygiannis, T., and Marks, D. (1999). Applying mobile agents to intrusion detection and response. Technical report, National Institute of Standards and Technology Computer Security Division, Washington, D.C, EUA.

Krishnamurthy, S. and Zeid, I. (2004). Distributed and intelligent information access in manufacturing enterprises through mobile devices. In *Journal of Intelligent Manufacturing*, pages 175 – 186. Kluwer Academic.

Lange, D. B. and Oshima, M. (1999). Seven good reasons for mobile agents. *Commun. ACM*, 42(3):88–89.

Levis, P., Lee, N., Welsh, M., and Culler, D. (2003). Tossim: accurate and scalable simulation of entire tinyos applications. In *Proc. of the 1st Int. Conf. on Embedded Networked Sensor Systems*, pages 126 – 137, Los Angeles, California, EUA. ACM.

Malik, H. and Shakshuki, E. (2007). Data dissemination in wireless sensor networks using software agents. In *Annual Int. Symp. on High Performance Computing Systems and Applications*, page 28, Saskatoon, Saskatchewan, Canadï¿$\frac{1}{2}$. IEEE Computer Society.

MathWorks, T. (2009). *MATLAB Getting Started Guide*. The MathWorks, Inc.

Ohlin, M., Henriksson, D., and Cervin, A. (2007). *TrueTime 1.5 - Reference Manual*. Department of Automatic Control, Lund University.

OMG (1997). Mobile agent system interoperability facilities specification. "`http://www.omg.org`". Accessed on Jan. 2010.

Polastre, J., Hill, J., and Culler, D. (2004). Versatile low power media access for wireless sensor networks. In *Proc. of the 2nd Int. Conf. on Embedded Networked Sensor Systems*, pages 95 – 107, Baltimore, MD, EUA. ACM.

Polley, J., Blazakis, D., McGee, J., Rusk, D., and Baras, J. (2004). Atemu: a fine-grained sensor network simulator. In *First Annual IEEE Comm. Society Conf. on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004*, pages 145 – 152.

Simulink (2009). *Simulink 7 - Simulation and Model-Based Design*. The MathWorks. Accessed on January 2010.

Sobeih, A., Chen, W.-P., Hou, J. C., Kung, L.-C., Li, N., Lim, H., ying Tyan, H., and Zhang, H. (2005). J-sim: A simulation and emulation environment for wireless sensor networks. *IEEE Wireless Communications magazine*, 13:2006.

Ye, W., Heidemann, J., and Estrin, D. (2001). An energy-efficient mac protocol for wireless sensor networks. pages 1567–1576.

Yu, Y., Krishnamachari, B., and Prasanna, V. K. (2004). Issues in designing middleware for wireless sensor networks. *IEEE Network*, 18:15 – 21.