# CupCarbon: A Multi-Agent and Discrete Event Wireless Sensor Network Design and Simulation Tool

Kamal Mehdi
School of Computer Science & Informatics
University College of Dublin
Belfield, Dublin 4, Ireland
Kamal.Mehdi@ucdconnect.ie

Massinissa Lounis
LIMED Laboratory
University of Bejaia
Bejaia, Algeria
Massinissa.Lounis@univ-bejaia.dz

Ahcène Bounceur
Lab-STICC Laboratory
Universiy of Brest
Brest, France
Ahcene.Bounceur@univ-brest.fr

Tahar Kechadi
School of Computer Science & Informatics
University College of Dublin
Belfield, Dublin 4, Ireland
Tahar.Kechadi@ucd.ie

## ABSTRACT

This paper presents the first version of a Wireless Sensor Network simulator, called CupCarbon. It is a multi-agent and discrete event Wireless Sensor Network (WSN) simulator. Networks can be designed and prototyped in an ergonomic user-friendly interface using the OpenStreetMap (OSM) framework by deploying sensors directly on the map. It can be used to study the behaviour of a network and its costs. The main objectives of CupCarbon are both educational and scientific. It can help trainers to explain the basic concepts and how sensor networks work and it can help scientists to test their wireless topologies, protocols, etc. The current version can be used only to study the power diagram of each sensor and the overall network. The power diagrams can be calculated and displayed as a function of the simulated time. Prototyping networks is more realistic compared to existing simulators.

## Keywords

Wireless Sensor Network, simulator, multi-agent system, discrete event simulation, OpenStreetMap, mobility.

## 1. INTRODUCTION

Progress and development in the field of wireless communication and electronics in recent years have given rise to a new technology known as Wireless Sensor Networks (WSN). These are composed of a large number of sensors that can be deployed randomly and densely. A sensor is a small electronic device that can collect data from its environment and send it to a base station. The type of data collected varies depending on the application and the type of the sensors. Sensor networks have many applications in different fields such as health, environment, agriculture, geology, military, etc. Most applications of WSN are challenging for designers and this is due to the limited capacity of the nodes in terms of autonomy (battery), computing power and inaccessible areas. This makes the design of the algorithms and programs for WSN too constrained. Therefore, performance evaluation tools become very essential in the process of designing a wireless sensor network, and simulation is one of the most used tools for this evaluation.

There exists a large number of WSN simulator tools. Some of them are described in [12]. In this work we propose to classify these tools into two main categories: simulators and emulators and to introduce only some simulators that are of four different types. The first type represents simulators that are based on *NS2* [7], which is a simulator developed in general for traditional networks and adapted for WSN. It uses discrete event simulation. In this family we can find *NRL Sensorsim* [13], based on modules, and *RTNS* [11] for the real-time distributed systems. The second family is based on the *OMNET++* simulator [14]. It is a discrete event simulator. We can find *Castalia* [3], for the development of protocols and distributed algorithms, *MIXIM* [8], an inter-level platform, *NesCT* [15] which allows to run TinyOS applications, *Pawis* [6] with its modular design allows to simulate deferent types of nodes, and *Sensim* [9], which allows to develop new protocols and test their scalability. The third family of simulators is based on *Ptolemy II* [1], which is a framework for modeling, simulating and designing of parallel embedded real-time systems. We can mention two simulators of this family, *Viptos* [5] integrating graphical development framework to simulate WSN based on TinyOS and *VisualSense* [2] which can be used as an educational tool to understand the basics of WSNs. Unfortunately these two simulators are not available for download. The fourth family is specially developed for WSNs, such as *WSNet* [4] used to evaluate high-level designs, *Atarraya* [16] for teaching and research topology control algorithms and

Simutools 2014, March 17-19, Lisbon, Portugal

DOI 10.4108/icst.simutools.2014.254811

*J-Sim* [10] a framework for modeling and simulation.

In the context of this study, we present a simulator called *CupCarbon* which is based on multi-agent and discrete event simulation. The current version belongs to the second family of simulators that is described above. It can be used to generate networks for *OMNET++*. It offers a simple and friendly graphical user interface for modelling the networks using the *OpenStreetMap* (OSM) framework. Each node is designed to be as close as possible to the real one. It is composed of four modules: mirco-controller the radio antenna, the capture unit and a battery. The current version of *CupCarbon* includes simulation of mobiles and it allows to represent the detailed energy diagram for each node versus the simulation time. This version can be also considered as a kernel which can be used to integrate different algorithms and modules making use of its advantages. It can simulate mobile tracking scenarios.

This paper is organised as follows. Sensor networks are introduced in the Section 2. Section 3 presents the *CupCarbon* design and simulation tool. Some simulation results are presented in Section 4. Some concluding remarks are given in Section 5.

## 2. WIRELESS SENSOR NETWORKS

Sensors are electronic components that operate in networks with autonomy. The term sensor is usually used to refer to a sensor node (having a dimensions of a few centimeters). It is composed of four components which are: a microcontroller (programmable integrated circuit), a radio antenna (for wireless communication), a battery and a set of sensors that we will call capture unit in order to avoid confusion. A capture unit (having a dimension of a few millimeters) captures or intercept environmental information (motion, temperature, humidity, gas, etc.). Henceforth, we use the term sensor to refer to a sensor node and sensor unit to refer to a sensor.

In order to be able to communicate (read or receive information), a sensor must be programmed. Figure 1 shows an example of a program for *Arduino* cards with an *ATmega* microcontroller. This program will send each second the byte 134 and will read the value of a sensor unit (motion sensor) which is connected to the input pin 12 of the card and then will send this read value wirelessly.

```
int p = 12;
void setup() {
   pinMode(p, INPUT);
   Serial.begin(9600);
}
void loop() {
   function();
   delay(1000);
}
void function() {
   Serial.write(134);
   int x = digitalRead(p);
   Serial.write(x);
}
```

**Figure 1: Example of *Arduino* program.**

A Wireless Sensor Network is an ad-hoc network and is composed of a number of sensor nodes that are able to collect, send and receive autonomously environmental data via wireless communications. Figure2 shows an example of a WSN. The position of these nodes is not necessarily predetermined. They can be dispersed randomly in a geographical area called "Wellfield" corresponding to the field of interest.

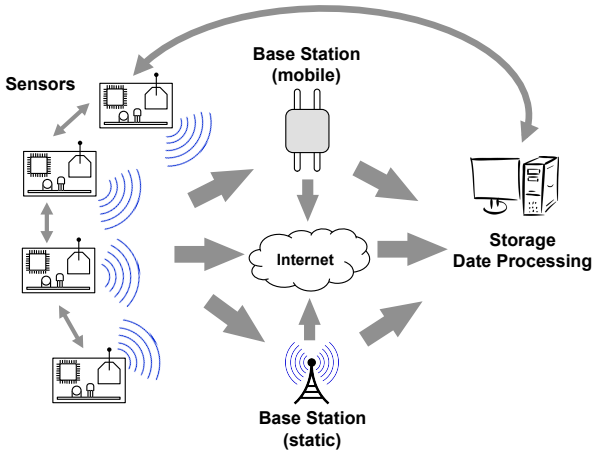


**Figure 2: Example of a Wireless Sensor Network.**

WSN is one of the most important technologies that have changed the world and facilitate many daily tasks. These WSNs offer the possibility of observing and controlling physical and biological phenomena in several areas; industrial, scientific (temperature, pressure, humidity, light, etc.), the environment (pollution, CO2, etc.), health (patient monitoring, epidemiological studies, etc..), transport (accident prevention, etc.), home automation, and so on.

After being captured, the data is routed based on a multihop routing to a node which is considered as a "collection point"; called Base Station. The Base Station, depending on its configuration, can store, process, or transmit the received data and it can be connected to the network via internet, satellite or any other system.

The user can send requests to the nodes of the network, specifying the type of data needed and collect environmental data from the base station. The data routing is a critical phase for the lifetime of the network, as the routing depends on the energy state of the sensors in the network, and on the data constraints.

## 3. THE CUPCARBON SIMULATOR

*CupCarbon* is a multi-agent and discrete event wireless sensor network simulator which is based on geolocation. It allows to model and simulate sensor networks on a digital geographic interface of *OpenStreetMap*. For this purpose, *CupCarbon* provides a set of easy-to-use and configurable objects. Figure 3 shows the graphical interface of this simulator. The use of multi-agent systems allows better optimisation of the simulation time by taking advantage of the parallelism of agents and events. *CupCarbon* is composed of three main components: a multi-agent simulation environment, mobile simulation, and finally the WSN simulator (*WiSen*). These three components will now be described.

### 3.1 Multi-agent Simulation Environment

*CupCarbon* provides an environment for a multi-agent simulation, which allows to run simulations and monitor various

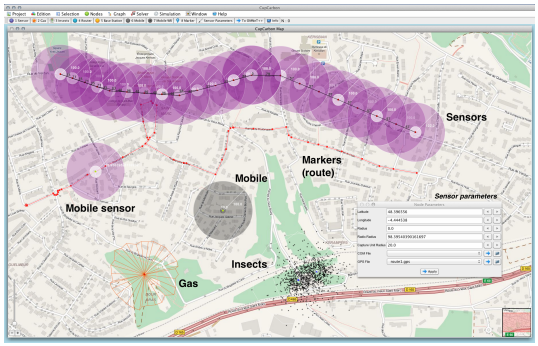


**Figure 3: The Interface of *CupCarbon*.**

```
00:00:00  36.7872914668200l5  4.965476989746094  0  25
00:00:01  36.768042206102585  4.956550598144531  0  25
00:00:02  36.74906320434928  4.95208740234375  1  25
00:00:03  36.73833386539537  5.000495910644531  0  25
```

**Figure 4: Example of a GPS file.**

```
HEAD PROTOCOL PROTOCOL-ID
send 154 70 S2
send 154 100 S3 S4
send 154 90 S1 S8 S9
wait 200
send 154 90 S2 S3 S6
send 154 30
wait 500
send 154 50 S1 S8 S9
break
```

**Figure 5: Example of a communication script.**

events and changes over time. The use of a multi-agent system allows a reproduction of an environment similar to the real world system, wherein each object in the system operates autonomously while communicating with other objects in the same environment. The integration of *OpenStreetMap* in this environment provides an interface and a database of digitised data related to geolocation such as roads, the positions of buildings, etc. Among the main agents in the system, one can find sensors for sending and receiving packets. There is also gas, insects and mobiles. They do not necessarily communicate between them but they interact with the sensors. In the next versions, the agents will be able to interact with the environment like buildings and meterological events (e.g. wind, temperature, etc.).

## 3.2 Mobility

*CupCarbon* offers the possibility to create paths, free or related to real roads, and assign them agents to make them mobile. Each path is defined by a set of referencing points on the OSM map, and each point is associated to a date corresponding to the exact arrival date of the object to it. There are two ways to generate trajectories in this environment:

1. `Manual generation`: it is possible to generate trajectories by manually selecting the points $(x, y)$ on the map and by specifying the arrival dates at each point.
2. `Automatic generation`: it is also possible to automatically generate trajectories on the OSM map by specifying the main points of the trajectory.

Paths must be saved. For memory optimisation reasons, only trajectories associated with objects included in the simulation are loaded into memory. Only one point is loaded in each move. The coordinates $(x, y)$ of given objects will be updated at the specific date for each point in the path during the simulation. One trajectory can be associated to multiple objects of different types. It is represented by a GPS file where each line describes a date, GPS coordinates, a boolean parameter, and the radius of the point to draw on the map. The boolean parameter is used to determine whether to draw the mobile objects connected at this point or just simulate without drawing. It differentiate a simulation for visualisation purposes and the simulation for the calculating the energy diagram. Figure 4 shows an example of a GPS file. Note that an agent who is not linked to a GPS file is considered as a static agent and a GPS file can be assigned to multiple agents.

## 3.3 Communication script

*CupCarbon* allows to configure each agent of the simulated system using script files to program their communications. Such a file describes how an agent will communicate with its neighbours and its environment. It contains all the main actions to be performed by a sensor during the simulation. Each line of the file represents an event that is characterised by its type, date, value, signal power, and the list of agents involved in it. An event is a command that can be of five different types, as described below:

- `COM_SEND`: for the packet (message) sending,
- `COM_DELAY`, `COM_WAIT`: to force a waiting,
- `COM_BREAK`: to mark the end of a communication,
- `COM_RECEIVE`: to accept the reception of a packet (message),
- `COM_UNKNOWN`: for unknown commands.

This file can, in some cases, have a header part which allows to configure the communication protocol and describe the type of algorithm to use in the simulation, subject to the fact that the algorithm is already integrated into the simulator. The header commands are given in the following:

- `COM_HEADER`: indicates the inclusion of a header,
- `COM_PROTOCOL`: this is for the communication protocol to be used in the simulations.

Note that the headers are optional in the communication script files and in the case of the absence of the header, a standard protocol will be used. Also, in the absence of the `break` command the execution of the file will be done repeatedly until the end of the simulation. Figure 5 shows an example of a communication script file.

## 3.4 The WSN simulator: WiSeN

*WiSeN* is the name of the module that represents the kernel of *CupCarbon* for simulating events related to sensors (sending, receiving, waiting, etc.). It supports and manages the evolution of the state of each object in the system (energy, position, etc.). Its implementation in a multi-agent environment allows for each agent to be executed independently and in parallel. Thus, it is possible to include mobility aspects and the detection of the target. Each agent generates events based on a existing saved script. The simulator organises events generated by agents (sensors, mobiles, etc.) according to their creation dates and executes them in the same order, and it updates their status (energy, position, etc.). Two types of files are automatically generated during the simulation:

1. `log file`: a file in which all the events executed by the simulator are stored for more detailed analysis and debugging purposes. Each line in the file corresponds to an event. One log file is generated per simulation.

2. `rst file`: a result file generated for each sensor. It contains the energy evolution of a sensor during the simulation.

## 3.5 CupCarbon Architecture

*CupCarbon* is developed in *Java*. Its architecture consists of two layers: The first layer concerns the modules used to build the simulation. The second layer concerns the simulation itself. Figure 6 shows the different modules of *CupCarbon*. Four main modules can be distinguished:

- Agents Module: it includes devices and events necessary to prototype wireless sensors networks and to prepare and configure the simulator.
- *OpenStreetMap* Module: it allows designing wireless sensors on an OSM map.
- *WiSen* Simulator Module: it allows to simulate wireless sensor networks. This module is connected to the simulator agents.
- Solver Module: it integrates a set of optimisation algorithms such as routing, coverage, etc. The current version integrates an algorithm for solving the set covering problem used to determine the minimum number of sensors that detect the maximum of targets. This part will not be presented because it is not a part of the objectives of this paper.

In order to simulate wireless sensor networks, two main types of actors are needed: the agents and the scheduler. Once created and initialised, the simulator then starts the agents and the scheduler and synchronises all critical sections between these agents.

Each sensor agent executes its associated communication script after its initialization step and begins to move using its associated gps file. The script allows the generation of events (send, wait, etc.) which are recovered by the scheduler to execute them in the order of their creation dates. The

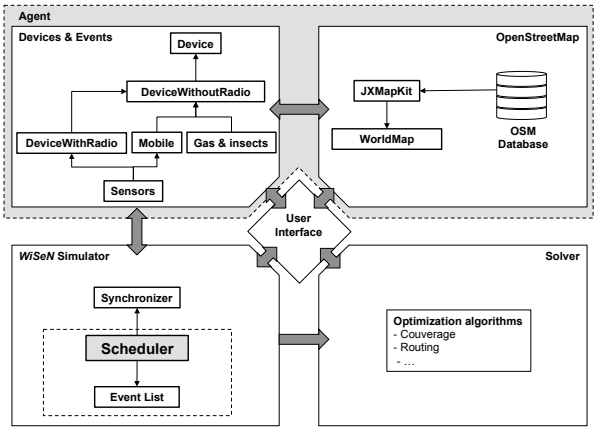


**Figure 6: The *CupCarbon* Architecture.**

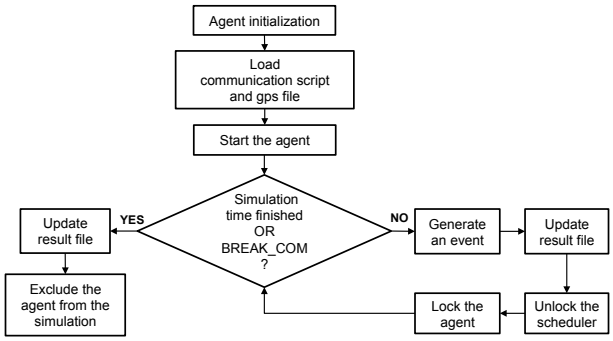


**Figure 7: The behaviour of a simulated agent.**

process describing the behaviour of an agent is described in Figure 7.

During a simulation, an agent starts to generate events described in the associated communication script and those related to mobility. These will then be communicated to the scheduler int order to sort them for a next processing phase. As long as the agent has not met the stopping conditions, it generates one event at a time whenever the scheduler unlocks it by calling the synchronisation semaphore. The agent unlocks the scheduler after generating the event. It then enters into idle state until the next call of the scheduler.

The scheduler is an independent module that runs in parallel with the agents. It organises events generated by agents in order to execute them. It is composed of a list of events, a management algorithm, and processing. Figure 8 shows the main role of the scheduler.

The scheduler is started just after its initialisation. Then, it will run in the background by collecting at each iteration the next events. If the agent of generating events has the necessary energy then the event will be executed otherwise the scheduler move to the next event of the event list. After its execution, it updates all the agents associated to the event, such as receiving a message from a sensor, and then add this action to the log file. The scheduler agent indicates the event to the generator agent so that it can generate and

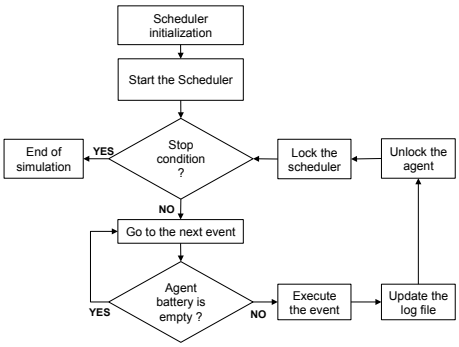


**Figure 8: The scheduler.**

add the next event in the scheduler list. Then, it switches to the idle state while the list is used by the agent (the list of events is a critical resource). The scheduler repeats this cycle until the stopping conditions are met. The overall simulation stops when one of the following three conditions is fulfilled:

- there is no events to execute in the scheduler list,
- the simulation time is exceeded,
- there is a forced stop or real time limitation of the simulation.

# 4. CASE STUDY

In this section we will show how to use *CupCarbon* to simulate the energy diagram of a sensor network which is designed on the *OpenStreetMap*.

## 4.1 Simulation Setup

Once the *CupCarbon* tool is started and a new project is created, it is possible to create a network of sensors directly on a geographical map which represents the main window of the software. It is possible to create routes to be assigned to mobiles. It is also possible to create mobiles, gas and a cloud of insects. For the sake of simplicity, we have chosen as an example a simple network consisting of eight sensors as shown in Figure 9.

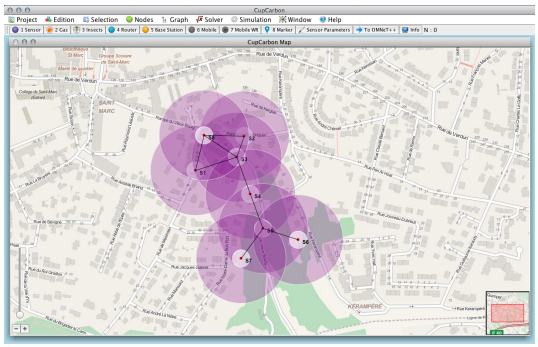

**Figure 9: A WSN designed with *CupCarbon*.**

Once the network is developed, the different communication scripts must be created and assigned to each sensor. One script can be assigned to multiple sensors. In our case, we create a single script that is assigned to all sensors. The script is shown in Figure 10. This script allows a sensor to send 10 bytes with the maximum signal power (100%).

```
send 100 100
send 110 100
send 120 100
send 130 100
send 140 100
send 150 100
send 160 100
send 170 100
send 180 100
send 190 100
delay 100
```

**Figure 10: Communication script of the sensors.**

As soon as all communication scripts are associated with the sensors, the simulation can begin. The next section describes the results for the network presented above.

## 4.2 Simulation Results

Once the network is ready, the real simulation parameters must be specified: the simulation time and the simulation step. Thus, the simulation can begin. In our case, we chose to simulate a period of 100 hours ($= 360K$ seconds) with a simulation step of one hour. Indeed, this parameters are given by the user but the execution will be based on the discrete-event simulation.

Figure 11 shows the curves representing the energy diagrams obtained for each sensor. In this graph eight curves are drawn but we distinguish only four because the other curves are superimposed. In this example, these curves are directly related to the number of neighbours of each sensor. There are four types of sensors corresponding to the four families of curves. We distinguish the family of sensors with only one neighbour (sensors S6 and S7) in blue, the sensors with two neighbours (sensors S1, S2 and S4) in red, the sensors with three neighbours (sensors S5 and S8) in purple and only one sensor with four neighbours (S3 sensor) in green. As we can see on the graph, for the script of the Figure 10 without the last line (`delay`) and for a given capacity of the battery, the sensor S3 having four neighbours (in green curve) dies after 57 hours ($= 205,200$ seconds). Note that the selected script means that the sensors send and receive messages continuously while the battery is charged.

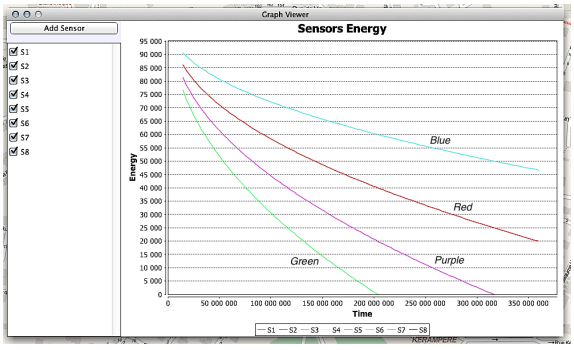


**Figure 11: Energy diagrams related to sensors (without the delay instruction).**

If we take into account the `delay` of 100 milli-seconds for each outgoing 10 bytes, as shown in the last line of the script of Figure 10 we can improve significantly the lifetime of the sensors. This new situation is illustrated by the curves in Figure 12. After 100 hours functioning, the battery of the sensor S3 reaches 70% of its total capacity. In other words, the sensor S3 consumes 30% of its battery each 100 hours functioning with an additional delay of 100 milli-seconds in its communication script and it dies after only 57 hours without this delay.

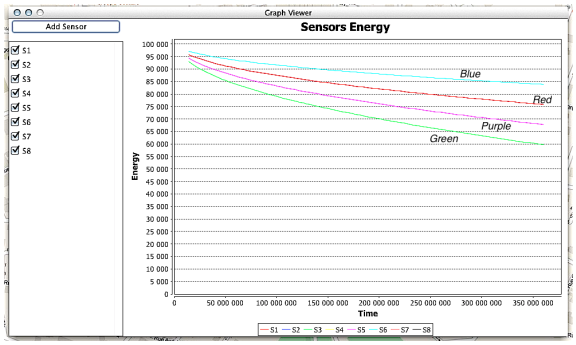


**Figure 12: Energy diagrams related to sensors (with the delay instruction).**

# 5. CONCLUSION

A simulator, *CupCarbon*, for the design and study of wireless sensor networks is presented. It allows to design networks using the *OpenStreetMap* framework. Networks can include sensors and other elements such as mobiles, gas, fires, etc. The presented tool can run two types of simulations. The Multi-agent simulation is used to parallelise the behaviour of each sensor to make it independent. The discrete event simulation is used to simulate the communications between agents and especially between sensors. The main objective of this simulator is educational. It will demonstrate the use of wireless sensors under almost the same conditions as in the real world. The simulator can also be used to calculate the energy diagram of each sensor. A case study showing how to obtain this type of diagrams is also presented. The results are consistent with the structure of the network. We have shown that a simple delay of only 100 milli-seconds in the communication script of a sensor can improve significantly its lifetime.

# 6. REFERENCES


[1] P. Baldwin, S. Kohli, E. A. Lee, X. Liu, and Y. Zhao. Modeling of sensor nets in ptolemy ii. *In the 3rd International Symposium on Information Processing in Sensor Networks (IPSN'04), Berkeley, USA*, pages 359–368, April 26-27, 2004.

[2] P. Baldwin, S. Kohli, E. A. Lee, X. Liu, and Y. Zhao. Visualsense: Visual modeling for wireless and sensor network systems. *Technical Memorandum UCB/ERL M04/08*, University of California, Berkeley, USA, April 23, 2004.

[3] A. Boulis. Castalia: a simulator for wireless sensor networks and body area networks. *Users's Manual, version 3.2*, March 2011.

[4] G. Chelius, A. Fraboulet, and E. Fleury. Worldsens: development and prototyping tools for application specific wireless sensors networks. *In the 6th International Conference on Information Processing in Sensor Networks*, pages 176–185, Cambridge, USA, April 25-27, 2007.

[5] E. Cheong, E. A. Lee, and Y. Zhao. Viptos: A graphical development and simulation environment for tinyos based wireless sensor networks. *Demo Abstract in the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys'05)*, page 302, San Diego, USA, November 2-4, 2005.

[6] J. Glaser, D. Weber, S. A. Madani, and S. Mahlknecht. Power aware simulator framework for wireless sensors network and nodes. *EURASIP Journal on Embedded Systems - C-Based Design of Heterogeneous Embedded Systems archive*, 2008(3), January 2008.

[7] T. Issariyakul and E. Hossain. Introduction to network simulator ns2. *Book, Springer Ed.*, 2011.

[8] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. K. Haneveld, T. Parker, O. Visser, S. L. Hermann, and S. Valentin. Simulating wireless and mobile networks in omnet++: The mixim vision. *In the 1st International Conference on Simulation Tools and Techniques for Communications (SIMUTools'08)*, Marseille, France, March 3-7, 2008.

[9] K.-W. Lee and W. K. D. Sensim: A simulation program for solid-state pressure sensors. *In the IEEE Transactions on Electron Devices*, 29(1):34–41, August 9, 2005.

[10] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. *In the 1st International Conference on Embedded Networked Sensor Systems*, pages 126–137, New York, USA, 2003.

[11] P. Pagano, M. Chitnis, and G. Lipari. Rtns: an ns-2 extension to simulate wireless real-time distributed systems for structured topologies. *In the 3rd International Conference on Wireless Internet (WICON'07)*, Austin, USA, October 22-24, 2007.

[12] A. Rastegarnia and V. Solouk. Performance evaluation of castalia wireless sensor network simulator. In *the 34th International Conference on Telecommunications and Signal Processing (TSP)*, pages 111–115, Budapest, Hungary, August 18-20, 2011.

[13] A. Sethi, J.P.Saini, and M. Bisht. Wireless adhoc network simulators: Analysis of characterstic features, scalability, effectiveness and limitations. *International Journal of Applied Information Systems (IJAIS)*, 5(9), July 2012.

[14] A. Varga. Omnet++. *User's Manual, version 4.3.*

[15] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. *In the 1st International Conference on Simulation Tools and Techniques for Communications (SIMUTools'08)*, Marseille, France, March 3-7, 2008.

[16] P. M. Wightman and M. A. Labrador. Atarraya: A simulation tool to teach and research topology control algorithms for wireless sensor networks. *In the 2nd International Conference on Simulation Tools and Techniques for Communications (SIMUTools'09)*, Rome, Italy, March 2-6, 2009.