Journal of Software Engineering and Applications, **2013, 6, 42-48**
doi:10.4236/jsea.2013.67B008 Published Online July 2013 (http://www.scirp.org/journal/jsea)

Introducing Intelligent Agents Potential into a competent Integral Multi-Agent Sensor Network Simulation Architecture Design

A. Filippou[1], D. A. Karras[2]

[1]University of Bolton, UK; [2]Sterea Hellas Institute of Technology, Greece, Automation Dept., Chalkis, Hellas (Greece), P.C..34400
Email: alexfilippoy@yahoo.gr, dakarras@ieee.org, dimitrios.karras@gmail.com.

Received June, 2013

ABSTRACT

During this research we spot several key issues concerning WSN design process and how to introduce intelligence in the motes. Due to the nature of these networks, debugging after deployment is unrealistic, thus an efficient testing method is required. WSN simulators perform the task, but still code implementing mote sensing and RF behaviour consists of layered and/or interacting protocols that for the sake of designing accuracy are tested working as a whole, running on specific hardware. Simulators that provide cross layer simulation and hardware emulation options may be regarded as the last milestone of the WSN design process. Especially mechanisms for introducing intelligence into the WSN decision making process but in the simulation level is an important aspect not tackled so far in the literature at all. The herein proposed multi-agent simulation architecture aims at designing a novel WSN simulation system independent of specific hardware platforms but taking into account all hardware entities and events for testing and analysing the behaviour of a realistic WSN system. Moreover, the design herein outlined involves the basic mechanisms, with regards to memory and data management, towards Prolog interpreter implementation in the simulation level.

Keywords: Wireless Sensor Networks (WSN); Simulation; MCU Emulation; CUDA; OpenCL; GPGPU; Intelligent Agents; Prolog Interpreter

1. Introduction

A WSN is a distributed system. It consists of a usually large number of autonomous devices that form a network. The diversity of missions and environments deployed in, introduces issues and parameters of paramount importance during design process. Success of this process is considered delivering **specific code running on specific hardware**, both meeting mission and production cost requirements.

In general, a mote is a device that consists of a medium access hardware interface, a processing module and a sensor array. In case of WSN, the medium is the RF channel, and the hardware interface is a RF transceiver. In case of submerged SNs the medium is water (acoustic signals) and the hardware interface is a microphone and a loudspeaker. The trivial case scenario is a Wireless Sensor Network, running on batteries, with limited computational ability and memory, operating in a harsh and hostile environment. By using simulation tools we gain pre-deployment knowledge estimating the network's behavior. In most cases, the design and implementation of application and protocol stack code, running on specific hardware setup, are viewed through energy consumption, security and production cost prisms.

The first task of a WSN after deployment is to configure itself. Every mote uses its transceiver to establish connections with its neighbors, in order to construct a topology. The mote acquires its location which is unknown at the beginning, through collaboration with other motes, starting from a few motes with known locations. The Localization protocol responsible for the above task uses physical quantities such as RSS AoA, ToA, to calculate the mote's location. After identifying its neighbors, and being identified, the mote is part of the network, able to produce sensor data, propagate data to sink, collaborate with neighboring motes to perform a computational or sensing task, create cluster, and so on. The total activity of a mote extends to multiple levels – or layers – each having its own procedure and parameters to calculate QoS. The overall performance is derived from the combination and cross layer code collaboration [1], and as stated in [2] does not necessarily means optimal performance in every layer.

In the next part of this paper, we highlight topology,

Copyright © 2013 SciRes.

simulation and hardware design issues that back up our choices in design of our optimal simulator. We spot the parameters taken in consideration in order to calculate the simulation metrics in each case. Our goal is to design a sensor simulator able to perform cross layer code, communication medium and environmental simulation, while keeping an inside view in every aspect of this process, giving every detail needed in order to extract conclusions about network behavior, in mote, local (an area containing a number of motes) or global (entire network) level. Special attention is paid in the design of the basic mechanisms, related to memory and data management, of Prolog interpreter implementation in the mote simulation level, aiming at introducing intelligent decision making in the mote level and not in the base station only. Such a distributed intelligent decision making process would be of major importance in real world applications

2. The Proposed Improved Simulator Architecture

Although the major issues involved in the design of WSN simulators have been spotted in some detail in the literature [3-34], while considering a protocol or the entire protocol collection running on mote's MCU none of the so far proposed simulators employs generic hardware properties management. Hardware specific simulators like Tossim and Atemu, for instance, lack flexibility concerning hardware for they are bounded with a specific platform. Avrora emulates every mote in its own thread, and thus performs and scales according to thread ability of the computer it runs on. None of the above takes advantage of GPGPU abilities (CUDA, OpenCL)

We herein propose an architecture, able of scaling, providing fine grain detail of the simulation, and configuring all the parameters of RF channel, Environment and Hardware. Our goal is to provide a multi agent simulation tool, to serve among others as a WSN Simulator. Moreover, our goal is to introduce distributed intelligence in the decision making, by involving Prolog Interpreter capabilities in the mote level through implementation of its basic mechanisms with regards to data and memory management. Basic mechanisms in the Prolog implementation concerning runtime level (which is the most important in the simulation level), are the Activation Frames, the Binding Records, the Stack Module, the Backtracking and Unification mechanisms. The mechanism of primary importance in such an implementation is the stack module management (or heap in early implementations) [35,36].

Therefore, we herein introduce the design of a stack module management into the overall WSN simulation architecture towards achieving distributed intelligence in the mote level.

2.1. Overall Architecture

Our proposal consists of four basic elements: (a) Agents (b) The Controller (c) Interfaces (d) Services.

***Agents*:** There are two types of agents. (a) Built in agents. They are commonly used components such as plotters, visualizers, or emulators. They are at the disposal of the user and not necessarily part of every simulation. (b) User defined agents. User writes code to define the agent behaviour. These are implemented by an interpreter, whose functionality is described later.

***The Controller*:** The controller activates or deactivates agents according to simulation clock and/or event handling instructions written in user defined code. In case of a simulation clock, the controller uses a basic time step which is the greatest common divisor GCD of all time steps of the clocks of time dependant agents. In case of time independent agents, their code is activated at every pace of the controller.

***Interfaces*:** The interfaces of agents are implemented with a byte array. Two agents communicate through a set of bytes eg from index a to index b in this array, using b-a bytes. These two integers (a and b), describe the interface. All interchanged data (numerical quantities, text, fluctuation of a quantity for a period of time eg a waveform from t = k to t = k + 10, signals or combinations) is described in bytes. Every interface may be used by two or more agents putting interchanged data in wide scope. An event handling mechanism notifies every agent using the interface (eg form a to b) that contents are changed. An agent may use as many interfaces as user requires. By using interfaces the user may project data from inside the agent to defined scope. Below is an example of a mote built up using four agents and the interface array.

The parts of the interface array that are used by this mote are coloured with grey. In the above example the PIC emulator uses 3 inter-faces, of different width, as shown in **figure 1**. The pace of the emulator is one machine cycle,

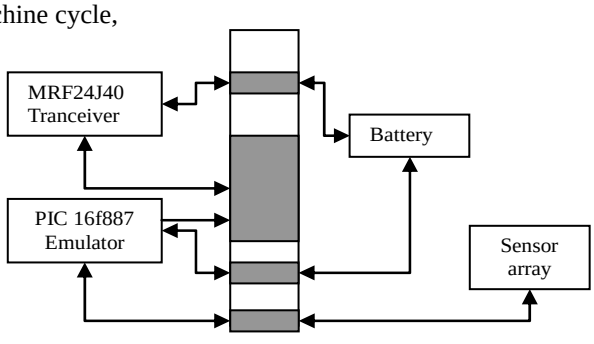

Figure 1. A Mote example.

(usually for MCUs execution of one instruction). In the lab we may measure the exact energy consumption of

Copyright © 2013 SciRes.

one cycle. This emulator uses as input the compiled program to be uploaded to the real device. The battery agent subtracts from a starting quantity energy consumption of MCU and radio, sending a shut- down signal to MCU emulator in case of energy depletion deactivating the virtual mote. Sensor array agent sends analog data to MCU emulator. The interface is as wide as needed to describe the possible value range of data. Due to ADC conversion latency, the ADC may be modelled separately using a fifth agent.

***Services*:** There are two types of services: (a) built in library. They are functions and procedures widely used in fields of engineering or science. (b) User defined: user writes code to describe the functionality of their procedure or function. Services are sets of procedures and/or functions in scope of any agent. They are activated when called by agents and their data is valid even when they are inactive. An implementation example is the RF medium. When a mote transmits, sends to RF medium service its coordinates, time, and transmission power along with data. RF medium service stores these values in a table. When a mote listens, the service calculates the signal at his location, according to all active signals above a given SNR. In case of an CSMA-CD MAC layer, both services (transmitting, listening) are active at the same time.

2.2. The Basic Interpreter

The interpreter consists of three main parts: (a) A 4XN integer array (b) The actual code that executes user programs (c) A byte array where data is stored.

1) 4*XN Integer Array* The simple program is:

1. If (A>0) 2. B ← B + 1 3. else 4. C ← C + 5 5. Endif 6. D ← B + C

Ends up translated in tokens and stored in the 4XN integer array. There are 4 basic types of token supported: (a) decision (b) operator (c) assignment (d) variable or value. At the first column contains the token code. Columns 2 and 3 contain pointers pointing at the next token to execute in the token (4XN) array. Column 4 contains pointer pointing to data memory. In detail, as shown in **figures 2** and **3**:

***Decision*:** if A > 0 is true then the next token to execute is the assignment token in line 2, else executes assignment token in line 4. Assuming that decision token code is number 1, and x and y are the indexes that tokens in lines 2 and 4 are stored respectively, then the line in which the decision token is stored in the 4XN would be like:

1	x	Y	

There is no pointer to data memory for there is no value involved. The expression A > 0 is stored directly below the decision token row. When the interpreter reaches a decision token, evaluates the expression below and according to its truth selects next token (x or y). With the decision token we may also implement for, while, do until statements using the pointers x,y accordingly.

***Operator*:** Operator > compares variable A with value 0. Assuming that operator > token code is 2, and variable-value token code is 3 then the expression is stored:

The algorithm that evaluates expressions is

```
Evaluate (x)
{y=Token(x,2); z=Token(x,3)
IF z=0 and y=0 THEN return mem (Token (x,4));
ELSEIF z=0 THEN return op (Token(x,1), evaluate(y))
ELSEIF y=0 THEN Return op(Token(x,1), evaluate(z))
ELSE Return op(Token(x,1),evaluate(y), evaluate(z))
ENDIF},
```

where x, y, z are indexes in the 4XN array of tokens, and m, n are indexes in the data memory (array of type byte).

***Assignment*:** The assignment token evaluates the expression on its right, stores the result in memory, and proceeds to next token execution. From assignment token in line 2, next to be executed is the assignment token in line 6. The same for assignment token in line 4. The token code for the assignment is 4 and for the + is 5. The memory pointer in column 4 in an assignment row, points the location to store the results. Completing the arrays:

***Data Types*:** The interpreter engine uses its own set of basic data types: Char, string, int, real, boolean, byte, binary and their combinations (structs). Binary is used for binary numbers bigger than 255. From the user's point of view, data memory and interfaces are transparent. From inside the agent, interface is seen and used as a simple variable or a struct. The interpreter's engine is responsible for any transformation needed, relieving user from the task. The basic advantage of this interpreter implementation is that is simple and can be used also:

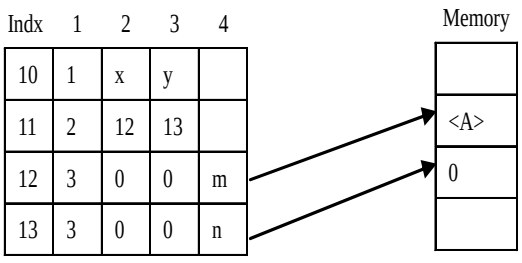

Figure 2. 4XN token array and memory array.

Copyright © 2013 SciRes.

Index	1	2	3	4
10	1	14	18	0
11	2	12	13	0
12	3	0	0	m
13	3	0	0	n
14	4	22		o
15	5	16	17	0
16	3	0	0	o
17	3	0	0	p
18	4	22		q
19	5	20	21	0
20	3	0	0	q
21	3	0	0	r
22	4			s
23	5	24	25	0
24	3	0	0	o
25	3	0	0	q

indx	
m	<A>
n	0
o	<B>
p	1
q	<C>
r	5
s	<D>

Figure 3. The representation of the program in the two arrays.

***MCUs*:** The interpreter is uploaded in the MCU EEPROM. Application and protocol coding now becomes data. Sink transmits code along with data altering WSN mission at runtime.

In the 4XN array we may store 1 or more programs. Each program starts at a specific index of this array. The interpreter using a priority mechanism may execute these programs concurrently, by executing a number of tokens of each, in every pace. This feature makes the interpreter able to serve as a base of a WSN OS.

***GPGPU*:** The function evaluate(x) is the part of the interpreter that calculates expressions. The other part is a simple switch (x) statement where x is the token code. Using a non recursive version of evaluate(x) we eliminate external dependencies and calls. One would expect difficulty in handling the sum of data involved due to variety of data types. In our case all data and code are contained in two arrays. Code and data can fit in a CUDA thread. In other words, one agent in every thread. The amount of data is constant (arrays). Both are copied to GPU memory using:

```
#define X 100; #define Y 4; #define Z 1000;
Int Tkn [X] [Y]; Byte mem [Z]; cudaMemcpy
(dTkn,Tkn, X*Y*sizeof(int), cudaMem-
cpyHostToDevice); cudaMemcpy(devicemem,
mem,z,cudaMemcpyHostToDevice);
```

2.3. The Interpreter Equipped with Complex Data Types Capabilities towards Gaining the Potential of Building Intelligent Mote Agents through Provision of Prolog Mechanisms

Implementation of stack module management regarding basic Prolog mechanisms realization, with respect to data types and memory management implies a methodology for Function Calls application. In the next paragraphs we provide the principles of the relevant design.

A Specific token is used for all user defined functions that return a value. An array is added to implement the stack mechanism in order to provide the recursive ability. Every function has the input struct, (the set of input parameters) and the output value. Every instance of the function created in the recursive chain, uses the struct and output value locations in memory only once, at the beginning and at the end respectively. Thus there are no instances created for the struct and the output value. When function starts, it copies the input struct contents to its local variables. An instance of local variables is created for every instance of the function. This is implemented using a 3XL array of pointers as shown below in **Figures 4** and **5**, where, also, its association with memory and the 4 X N token array is defined.

The T1 is the token for the user defined function. The Pointer P1 points at the first line of function code, P2 points at memory workspace where the input struct is stored. Pointer P3 points at the location where the output value is to be stored.

Pointers P4 and P5 point at the Stack Pointer Array (SPA). At SPA (P4,3) = P6 is the pointer that points at the initial location of the variable V1 in memory workspace. At every function call, the interpreter copies values from the input struct to local variables (V1,V2). Pointers P4, P5, P6, P7 are initialized during parsing. <var1.1> and <var2.1> are the first instances of V1 and V2 respectively. Every recursive function call pushes a new instance of local variables to stack. In case of V1, the second call pushes to the next available location in stack, in this case S1. The SPA (P4,1) = S1 points at this location. In SPA (S1,3) = P8 points at the location in memory of the second instance of V1 (<var1.2>). When the second instance of the function completes execution, then the stack of its local variables backtracks to SPA(S1,2)=P4 and SPA(S2,2) = P5 for V1 and V2 respectively, and performs SPA(P4,1)← 0 and SPA(P5,1)←0.

An algorithm example is the n factorial (n!):

```
Function Factorial (n)
  if (n==0)        {
     Factorial ← 1
                   }
  Else             {
     Factorial ← n * Factorial (n-1)
                   }
End Function
```

Copyright © 2013 SciRes.

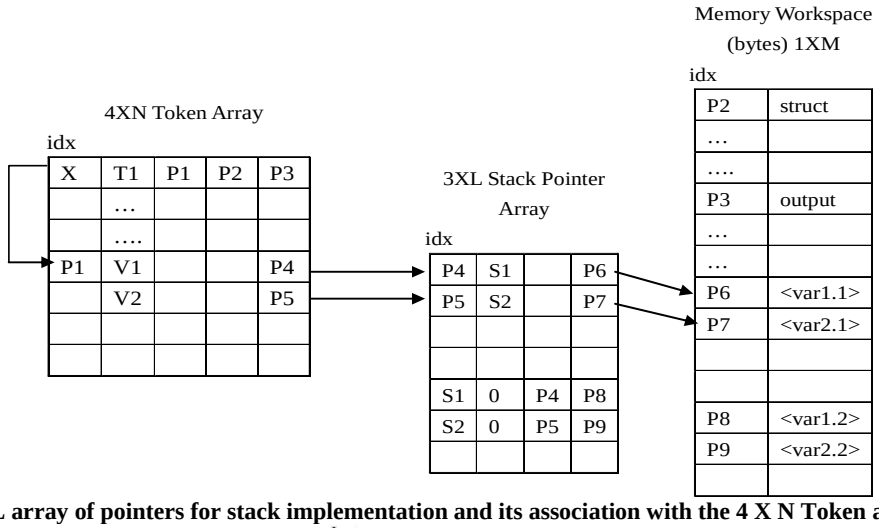

Figure 4. The 3XL array of pointers for stack implementation and its association with the 4 X N Token array and Memory

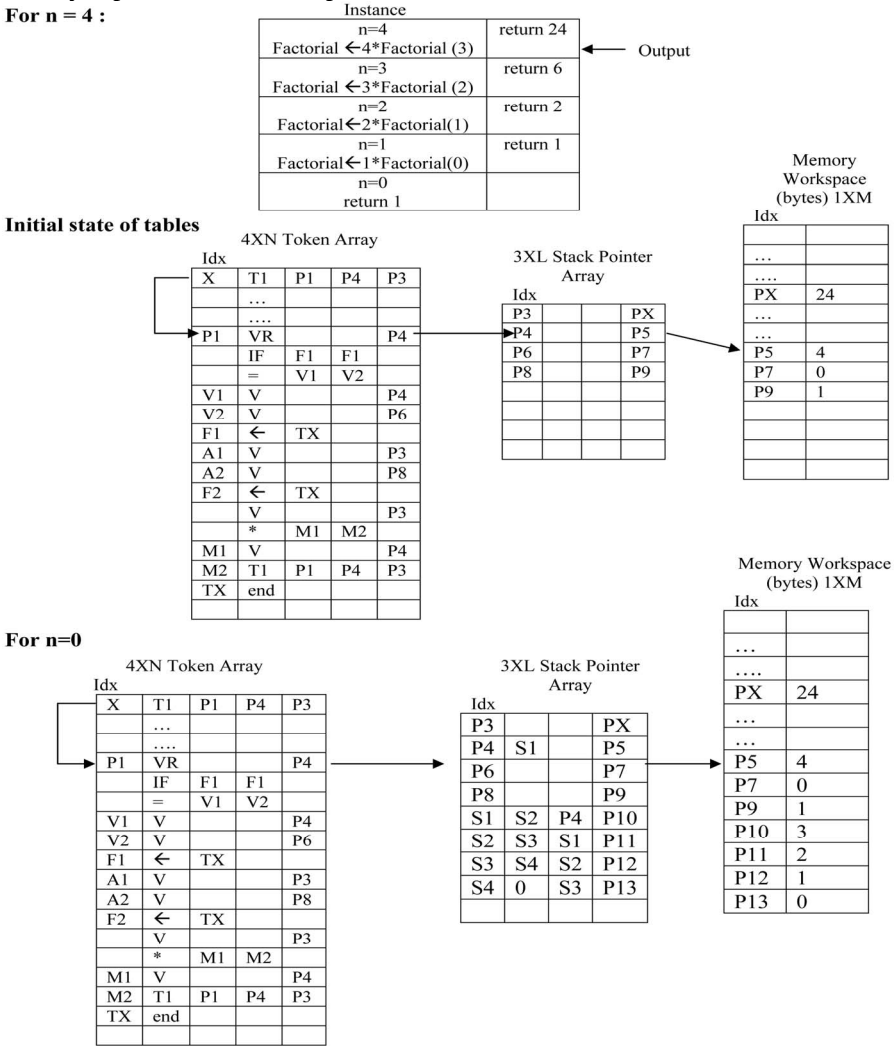

Figure 5. The 3XL array of pointers for stack implementation of the factorial example and its association with the 4 X N Token array and Memory

Copyright © 2013 SciRes.

The only variable used is n. At first call the interpreter copies the input struct (a single value of 4) at memory workspace MW(SPA(p4,3)). A double connected list is created to store the instances of n with entry point the pointer P4. The pointer P4 points next to S1 etc and the list expands till P4 → S1 → S2 → S3 → S4 When n reaches 0, the expansion stops and the function returns the value 1. The list backtracks form S4 to SPA(S4,2) = S3, and so on. For every local variable (in this case only n) a double connected list is created. Variables may backtrack independently, and may be of any data type.

This mechanism is used by function tokens, but may be used also for dynamic data types or structs such as lists, stacks etc. thus it may serve as base for implementation of **prolog** clauses, and basic rules.

3. Other Simulator Approaches and Prospects for the Proposed Design

The proposed architecture is similar with the architecture of SENSE [32]. SENSE is built on top of COST. A simulation is dealt as a composition of components. Each component is implemented in C++, and communicates with other components via inports and outports.

Inports and outports are used to reduce interdependencies between components and allow code reuse. The universal interface mechanism in our architecture provides the ability of interchanging any type of data and of any size and organization (structs). An interface may serve as a private channel used by two agents, or as a group channel, only by the use of two indexes that specify the channel width and location on the byte array.

Avrora, AvroraZ, tossim and Atemu, are AVR emulators, with AvroraZ [33] giving the ability of emulating the cc2420 chip. All are limited to AVR MCU's implementations, and they do no support network-communication level simulation [34]. SensorMaker [34] is written in C, provides network information in fine grain, (packet collisions, packet path, cluster layout) along with mote information mainly energy level. However, modelling – coding gap still remains for the designer to fill. Collecting ideas and implementations, we conclude the set of desired features of a simulating tool:

(1) Easy to use [34] (2) code reuse [32] (3) Hardware emulation [14,33] (4) Visualization and interpretation of data [34] (5) Network toolbox [34] (6) precise timing [14,33].

Our proposal is a multi agent simulator equipped with the capability of distributes intelligence in the mote level through Prolog interpreter mechanisms realization regarding complex data types and memory management via stack management. Using agents and services the user may implement hardware emulation, channel – medium (RF sound) modelling, environmental phenomena modelling, event monitors and handlers, data visualisation interpretation and storage. All the components of the simulator are connected directly, or via the multi type interface structure. The simulation controller activates each agent (a) according to a time interval (time dependant agents), (b) at every pace (continually), (c) or according to an event (a change in an interface, a memory location, a variation of an environmental parameter etc). Agent or service behaviour is implemented in code which is executed by the interpreter or the simulator. The interpreter's back end is an array of tokens in which the user's code is translated. The interpreter's front end is a C like language, but in the future others will be supported (Delphi, Basic). Controller may control channel (RF or sound) simulation, Environmental (temperature variations over an area) simulation and device simulation/ emulation (model/real code)

REFERENCES

[1] W. Masri and Z. Mammeri, "On QoS Mapping in TDMA Based Wireless Sensor Networks," Wireless and Mobile Networking IFIP Joint Conference on Mobile Wireless Communications Networks MWCN 2008.

[2] B.-L. Wenning, *et al*., "Environmental Monitoring Aware Routing in Wireless Sensor Networks," Wireless and Mobile Networking IFIP Joint Conference on Mobile Wireless Communications Networks MWCN 2008.

[3] S. Dziembowski, A. Mei and A. Panconesi "On Active Attacks on Sensor Network Key Distribution Schemes," Algorithmic Aspects of Wireless Sensor Networks 5th International Workshop ALGOSENSORS 2009 Rhodes Greece July 10, 11, 2009.

[4] C. Karlof and D. Wagner, "*Secure routing in wireless sensor networks*: *attacks and countermeasures*," *Ad Hoc Networks*, Vol. 1, No. 2-3, 2003, pp. 293-315. doi:10.1016/S1570-8705(03)00008-8

[5] E. Serpedin and Q. Chaudhari, "Synchronization in Wireless Sensor Networks Parameter Estimation Peformance Benchmarks and Protocols," Cambridge University Press 2009. doi:10.1017/CBO9780511627194

[6] A. Giridhar and P. R. Kumar, "The Spatial Smoothing Method of Clock Synchronization in Wireless Networks," Theoretical Aspects of Distributed Computing in Sensor Networks Springer 2011.

[7] S. Slijepcevic and M. Potkonjak, "Power Efficient Organization of Wireless Sensor Networks," *Proceedings of IEEE International Conference on Communications*, Vol. 2, 2001, pp. 472-447.

[8] G. Q. Mao and B. Fidan "Localization Algorithms and Strategies for Wireless Sensor Networks," Premier Reference Source – information Science Reference 2009.

[9] www.opnet.com.

[10] T. He, R. Stoleru, A. Stankovic John, "Range Free Localization. Technical Report," , University of Virginia

Copyright © 2013 SciRes.

(2006)

[11] Timo Ojala et al "UBI-AMI: Real-Time Metering of Energy Consumption at Homes Using Multi-Hop IP-based Wireless Sensor Networks" Advances In Grid and Pervasive Computing GPC Oulu Finland 2011, pp. 274-284

[12] M.A Labrador, P.M. Wightman "Topology Control in Wireless Sensor Networks," Springer 2009.

[13] T. Watteyne, *et al*. "Centroid Virtual Coordinates – A novel Near – Shortest Path Routing Paradigm," Computer Networks 17 October, 2008.

[14] P. Levis, N. Lee, M. Welsh and D. Culler, "Tossim: Accurate and scalable simulation of entire tinyos applications," First International Conference on Embedded Networked Sensor Systems (SenSys 2003) (November 2003).

[15] Khan M, Abdelzaher T.,Gupta K.K "Towards Diagnostic Simulation in Sensor Networks," Distributed Computing in Sensor Systems 4th IEEE International Conference DCOSS 2008 Santorini Island Greece June 11-14, 2008.

[16] W. B. Heinzelman, A. P. Chandrakasan and H. Balakrishnan, "An Application-specific Protocol Architecture for Wireless Microsensor Networks," *IEEE Transactions on Wireless Communications*, Vol. 1, No. 4, 2002, pp. 660-670. doi:10.1109/TWC.2002.804190

[17] S. Lindsey and C. S. Raghavendra, "PEGASIS: Power Efficient Gathering in Sensor Information Systems," in: Proc. IEEE Aerospace Conference, Big Sky, Montana, March 2002. doi:10.1109/AERO.2002.1035242

[18] A. Manjeshwar and D. P. Agrawal, "TEEN: A Protocol For Enhanced Eficiency in Wireless Sensor Networks," in Proceedings of the 1International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing, San Francisco, CA April 2001.

[19] H. Frey and I. Stojmenovic, "On delivery guarantees of face and combined greedy-face routing algorithms in ad hoc and sensor networks," in twelfth ACM Annual International Conference on Mobile Computing and Networking (MOBICOM) Los Angeles CA USA ACM September 23-29, 2006, pp. 390-401.

[20] T Watteyne, *et al*., "On using virtual coordinates for routing in the context of wireless sensor networks," in 18th Annual International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC) Athens Greece IEEE, September 3-7, 2007.

[21] C. Inatanagonwiwat, R. Govindan and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networksm," Mobicom 2000, Boston, MA, USA (2000).

[22] Jon S. Wilson ed. "Sensor Technology Handbook," Elsevier 2005.

[23] Patrick Kuckertz *et al*, "Sniper Fire Localization Using Wireless Sensor Networks and Genetic Algorithm Based Data Fusion," Military Communications Conference 2007 MILCOM 2007 IEEE (2007).

[24] Muhammad Imran *et al*. "Application-Centric Connectivity Restoration Algorithm for Wireless Sensor and Actor Networks," Advances in Grip and Pervasice Computing GPC 2011, pp. 243-253.

[25] H. M. Ammari "Challenges and Opportunities of Connected k Covered Wireless Sensor Networks - From Sensor Deployment to Data Gathering," Springer 2009.

[26] J. Fraden, "Handbook of Modern Sensors – Physics Designs and Applications," Springer Verlag, 2004.

[27] P. Balister, S. Kumar, Z. Zheng and P. Sinsha, "Trap Coverage : Allowing Coverage Holes of Bounded Diameter in Wireless Sensor Networks," Infocom/IEEE 2009.

[28] M. Cardei, M. T. Thai, Y. Li and W. Wu, "Energy-efficient Target Coverage in Wireless Sensor Networks," In: IEEE INFOCOM (2005).

[29] M. X. Cheng, L. Ruan and W. Wu, "Achieving minimum coverage breach under bandwidth constraints in wireless sensor networks," IEEE INFOCOM (2005).

[30] O. Saukh, R. Sauter and P. J. Marron, "Time-Bounded and Space-Bounded Sensing in Wireless Sensor Networks," Distributed Computing in Sensor Systems DCOSS 2008, LNCS 5067, pp. 357-371.

[31] C. K. Liang and Y. T. Chen, "The Target Coverage Problem in Directional Sensor Networks with Rotatable Angles," Distributed Computing in Sensor Systems DCOSS 2008, LNCS 5067, pp. 264-273.

[32] C. Gilbert, *et al*., "SENSE: a wireless sensor network simulator," *Advances in Pervasive Computing and Networking* (2005), pp. 249-267.

[33] de P. Alberola, Rodolfo and D. Pesch, "AvroraZ: extending Avrora with an IEEE 802.15. 4 compliant radio chip model," *Proceedings of the 3nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*. ACM, 2008.

[34] Y. Sangho, *et al*., "SensorMaker: A Wireless Sensor Network Simulator for Scalable and Fine-grained Instrumentation," *Computational Science and Its Applications–ICCSA 2008*, pp. 800-810.

[35] M. Bruynooghe, The Memory Management of PROLOG Implementations, in Clark, K. L. and S. Tarnlund (1982, eds.), Logic Programming, Academic Press, London.

[36] P. Civera, G. Piccinini and M. Zamboni, "Implementation Studies for a VLSI Prolog Coprocessor," in IEEE Micro, Vol.9/1 February 1989 pp.10-23. doi:10.1109/40.16791

Copyright © 2013 SciRes.