Applications Development on a Rule-Based WSN Middleware

Jiaxin Xie, Zixi Yu, Xiang Fei, Partheepan Kandaswamy
Department of Computing,
Coventry University
Coventry, UK
{xiej, yuzi, x.fei, kandaswp}@coventry.ac.uk

***Abstract*— Wireless Sensor Network (WSN) middleware eases the WSN application development by providing an application programming interface (API). Rule-based WSN middleware enables the applications and users to program the behavior of the sensor nodes. REED (Rule Execution and Event Distribution) is such a middleware solution that allows sensor networks to be programmed at run time. In this paper, we propose a method of developing WSN applications that uses finite state machine (FSM) as a bridge between application logics and the rules running on the REED, and demonstrate that for applications, if their behaviors can be described using finite-state machine (FSM), they can be directly described using the rules and thus implemented on the REED; further, we argue that rule-based middleware is useful for implementing bio-inspired mechanisms, such as self-organization, on WSN systems. Two WSN applications are implemented, as examples, on the REED: one is the de-synchronization of sensor nodes, and the other is the clustering-based self-organization. This paper is not aimed to study a specific application or control mechanism on WSNs, but to, via two prototype implementations, show that rule-based middleware such as the REED is useful and flexible enough to support the development of WSN applications, especially for bio-inspired mechanisms.**

Keywords-wireless sensor network; rules; finite state machine

I. INTRODUCTION

The advance in Wireless Sensor Networks (WSN) technology has led to a variety of WSN applications. One example is PROSEN [1] (PROactive SENsing) research project that aimed at developing a WSN system for proactive wind farm condition monitoring. The features of WSN systems, such as the distribution and heterogeneity of sensor nodes, the constrained processing power, memory, and energy for each sensor node; and the error-prone wireless links over which sensor nodes communicate, make the WSN application development a challenging task [2]-[4]. To ease the wireless sensor data collection, delivery and query, WSN middleware is introduced that provides an application programming interface (API) to shield the application (developer) from the complexities arising from the WSN. Rule-based WSN middleware enables the applications and users to programme the behaviour of the sensor nodes. Conceptually, a *rule* takes the form of <*event*, *condition*, *action*> where:

- an *event* is received from any other component in the system. This is can be an event carrying data values, or other events such as a timeout event, a sleep or wake-up event, and so on.
- a *condition* is a Boolean expression that will be evaluated when the *event* occurs.
- an *action* is executed if the above *condition* is true when the *event* is received. The action may manipulate or store data. It may also generate another *event* to other components in the system, such as an *event* to trigger other *rules*.

Fig. 1 shows the general architecture of the rule-based middleware [7]. The *Rule-Base* stores all the rules derived from the application while the *Fact-Base* stores the states of the node and the events that have occurred. The *Event-Manager* is responsible for receiving *events* and passing them to the *Rule-Engine*. The *Rule-Engine,* based on the current event and the states stored in the *Fact-Base*, executes matching rules stored in the *Rule-Base*. The results of the execution could be the update of the *Fact-Base*, or sending an event to other components in the system via the *Event Manager*.

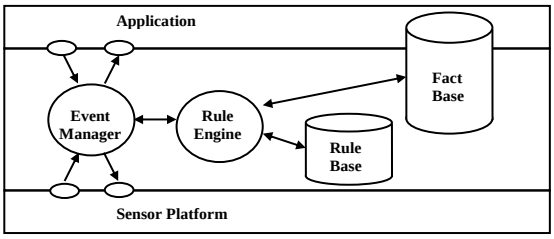

Figure 1. Architecture of rule-based middleware

A Rule Execution and Event Distribution (REED) middleware, originally for the PROSEN project, has been designed and implemented. REED is based on the general architecture described in Fig. 1; but further, enables programmability at run time, i.e., the system behaviour can be programmed by applications at run time, so as to be adaptive to the changing application goals and changing environment [5][6].

Provided the rule-based middleware such as the REED, for application developers, their main focus should be on constructing the rules that describe the logics of the application tasks, and this leads to the question of how to effectively and efficiently express the application specific

 ISBN: 978-1-61208-207-3

domain behaviour using rules. In this paper, we propose a method of developing WSN applications that uses finite state machine (FSM) as a bridge between application logics and the rules running on the REED. Further, we argue that rule-based middleware such as the REED is especially useful for implementing bio-inspired mechanisms for WSNs as bio-inspired mechanisms imply that only simple rules are needed to be running on each node in order to achieve the emergent behaviour (such as self-organization) of the whole WSN system.

The rest of the paper is organized as follows: Related work is discussed in Section II. Then, the REED for the PROSEN project is briefly described in Section III. In Section IV, a general method of constructing WSN rules is proposed, followed by explaining the advantages of rule-based middleware for bio-inspired WSN mechanisms. The implementation of two bio-inspired mechanisms on the REED, i.e. sensor nodes clustering and de-synchronization, are described in Section V and tested and evaluated in Section VI. Section VII concludes the paper.

II. RELATED WORK

Fei and Magil [5] and Fei and Yu [6] have listed the related work on the rule-based middleware for WSNs, including a general architecture of the rule-based middleware proposed by Terfloth, Wittenburg, and J.Schiller [7]. Fei and Magil [5] also developed a rule-based middleware, called REED, for the PROSEN project. In addition, a survey across a broad array of WSNs and middleware including rule-based middleware has been provided by Mottola and Picco [2]. A systematic study on the same topic has been given in Terfloth [10]. As mentioned in Section I, for application developers, directly constructing rules for the WSN systems is still non-trivial. Using FSM, a primitive and useful graphic model for describing system behaviours, as a bridge between application logics and the rules running on the WSN systems will make it more straightforward for the application developers to construct rules running on the rule-based WSN middleware such as the REED. To the knowledge of the authors, this paper is the first to build up the relationship between FSMs and the rules running on the WSN systems.

Some bio-inspired mechanisms on WSNs have been proposed. The biologically-inspired clustering algorithm proposed by Wokoma, Shum, Sacks and Marshall [11] was inspired from quorum sensing, a biological process that is carried out within communities of bacterial cells. Based on how fireflies and neurons spontaneously synchronize, Geoffrey, Geetika, Ankit, Matt and Radhika [12] developed a fully distributed time synchronization mechanism among TinyOS-based motes; likewise, Julius, Ian, Ankit and Radhika [13] developed a fully distributed time de-synchronization mechanism. Boonma and Suzuki [14] developed the BiSNET, a biologically-inspired sensor networking architecture, to address issues such as scalability, energy efficiency, self-healing, etc. However, they were not implemented on rule-based middleware. We argue and demonstrate in this paper that rule-based middleware, plus the FSM-based rules construction method, will ease the applications development on WSN systems.

Jacobsen, Zhang and Toftegaard [15] provided an overview of bio-inspired principles and methods applicable to sensor network design. This overview also mentions that by using simple rules for the behaviour and the interaction among individuals a global optimum can be achieved on a large, system-level scale. However, no real case of rule-based bio-inspired mechanism is provided. In this paper, two bio-inspired mechanisms will be implemented on a rule-based middleware to demonstrate the effectiveness of the rule-based bio-inspired mechanisms development on WSNs.

III. REED FOR PROSEN

A. PROSEN Architecture

Fig. 2 illustrates the system architecture for PROSEN where REED lies in. The system consists of a Policy Server (PS), a PN (Processing Node) for each wind-turbine, and sensors to measure parameters such as temperature, wind-speed, wind-direction, and gearbox temperature. The PS interacts with users and operators to obtain the goals for the system. Such goals might describe a desirable power output or response to poor weather conditions. The PS converts the goals to a set of policies. These policies in turn are converted to low-level rules. These rules are then distributed to the REEDS on the PNs.

For the Rule-Engine, in addition to executing the *rules* in response to received *events* as described in Section I, in order to support on-line programmability, its functionality also includes:

- Managing a rule database to allow the adding, removing, and overriding of rules
- Verifying rule consistency
- Merging and filtering rules.

For detailed information on the REED and its language description, refer to [5].

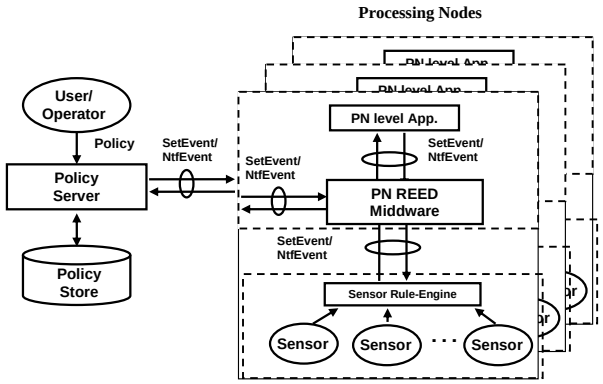

Figure 2. PROSEN system architecture

B. General Prototype Structure

The general prototype system structure, as shown in Fig. 3, consists of one PC functioning as a PS, one PC as a Gateway [6] and PNs that are the combination of PCs (as PN emulators) and a GumstixTM [8]. GS400K-XM is a miniature full function Linux motherboard based on low

power Intel XScale® technology. It has 16MB flash memory which can accommodate JamVM [9], which is a compact JVM (Java Virtual Machine). REED is developed using Java and running on PNs (PCs and the Gumstix). The PS and the gateway are connected via the Internet, and the Gateway and the PN are connected via a 174MHz wireless link. The sensor reading is simulated via random number generator.

Figure 3. General prototype system structure

IV. APPLICATIONS DEVELOPMENT ON THE REED

A. A General Method of Constructing WSN Rules

Rule-based middleware, such as the REED, to some extent ease the development of WSN applications However, as each rule is encoded by a structure of textual data format; application developers may still face difficulties in constructing the rules that describe the behaviour of the application tasks. For them, graphical models, such as FSMs and UML statechart diagrams, are the convenient ways of expressing the behavior of the application. We argue that the FMS as shown in Fig. 4 has the direct relationship with the rules supported by the REED, and the mapping from the FMS to the rules are as follows:

- The Event part in the FSM can be mapped to the event of a rule;
- The State part in the FSM can be mapped to the condition of a rule;
- The Action part in the FSM can be mapped to the action of a rule plus an extra action that is to update the current state;
- The number of the $\frac{\text{Event}}{\text{Action}}$ in the FSM is equal to the number of the *rules* for an application.

So, as long as the application behaviour can be described using finite-state machine (FSM), the construction of rules from FSM is straightforward. Two cases in Section V provide examples of how to construct rules via FSMs.

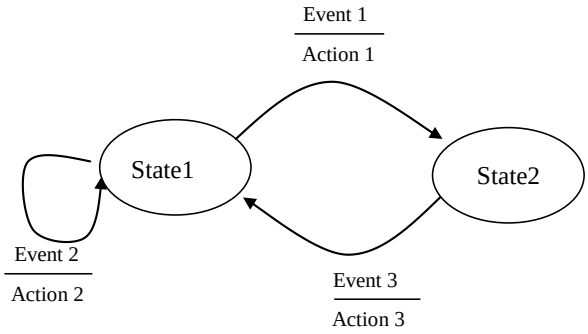

Figure 4. Finite state machine

B. Support for Bio-inspired WSN Mechanisms

Biological systems achieve complex goals reliably via the collaboration of a large number of cheap, unreliable components. Such collaboration is based on each component executing simple tasks in respond to the stimuli. For WSN, sensor nodes share the similar features with the components in biological systems: distributed, resource limited, unreliable, and etc.

Bio-inspired mechanisms, such as self-organization, have been applied to WSN. If the simple tasks executed on each node in response to the stimuli can be expresses as a small set of *rules*, as mentioned above, rule-based middleware such as the REED will be advantageous in facilitating the implementation of the bio-inspired mechanisms on WSNs. Both two cases in Section V are bio-inspired.

V. TWO CASE STUDIES

A. Rule-Based Sensor Nodes Clustering

Clustering sensor nodes is one of the self-organisation mechanisms applied to WSNs. There exist quite a few clustering mechanisms for WSNs [11]. The clustering algorithm implemented in this paper features first the clusters are formed dynamically and updated periodically; second, the process overhead imposed on the cluster heads is balanced among all the sensor nodes.

To describe the behaviour of the clustering algorithm, its FSM is drawn first as in Fig. 5. As 'UpdateCurrentState' is the default action, due to limitations of space, it is not listed in the *action* part of the *rules*. According to the Section IV, the rules can be directly derived from the FSM as shown in Table 1. Further, it is found that if the *action* set for an *event* is always the same across the whole states, the *condition* part of the rules for that event can be simply replaced by "TURE"; this is especially the case if for an events, it occurs only when the PN is in a specific state. This replacement not only makes the rule set concise, but also reduce the processing time for the PNs to check the *condition* part of the rules. So for R1, R2, and R3, their *condition* part is simply "TRUE".

 ISBN: 978-1-61208-207-3

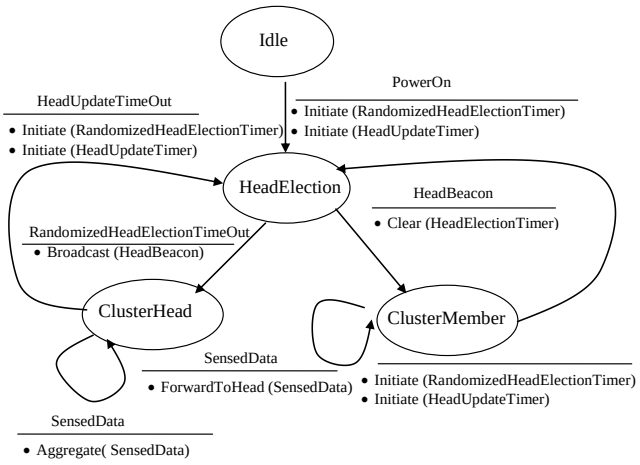

Figure 5. FSM for the clustering algorithm

B. *Rule-Based Sensor Nodes De-synchronization*

De-synchronization implies that sensor nodes perfectly interleave periodic events to occur in a round-robin schedule. It is useful in that it enables the sensor nodes to evenly distribute sampling burden in a group of nodes, schedule sleep cycles, or organize a collision-free TDMA schedule for transmitting wireless messages [13]. DESYNC, proposed by Julius, Ian, Ankit and Radhika [13], is a biologically-inspired self-maintaining algorithm for de-synchronization in a single-hop network. In this paper, the algorithm of the DESYNC will is implemented on the REED. Due to limitations of space, for detailed description of the DESYNC, please refer to [13].

Fig. 6 describes the behaviour of each sensor node in order to achieve de-synchronization, from which it can be seen that only four rules running on each node are enough to achieve the emergent de-synchronization of the sensor nodes in a fully distributed way. Table 2 lists the four rules derived from Fig. 6.

TABLE I. RULES FOR SENSOR NODES CLUSTERING

Rule 1 is triggered when the sensor node is powered on.
R-1 = power_on
[TRUE; Initiate (RandomizedLeaderElectionTimer), Initiate (LeaderUpdateTimer)]

Rule 2 is triggered for cluster head election.
R-2 = leader_election_timeout
[TRUE; Broadcast (LeaderBeacon)]

Rule 3 is triggered for cluster members.
R-3 = leader_beacon
[TRUE; Clear (LeaderElectionTimer)]

Rule 4 is triggered for cluster head to aggregate data.
R-4 = sensed_data
[Head; Aggregate (SensedData)]

Rule 5 is for cluster members to send data to its head.
R-5 = sensed_data
[Member; ForwardToHead (SensedData)]]

Rule 6 and 7 are triggered for cluster head updating.
R-6 = leader_update_timeout
[Head; SendToHost (AggregatedData), Initiate (RandomizedLeaderElectionTimer), Initiate (LeaderUpdateTimer)]
R-7 = leader_update_timeout
[Member; Initiate (RandomizedLeaderElectionTimer), Initiate (LeaderUpdateTimer)]

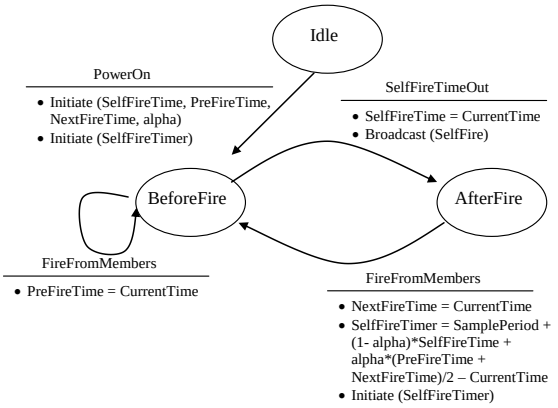

Figure 6. FSM for the de-synchronization algorithm

TABLE II. RULES FOR SENSOR NODES DE-SYNCHRONIZATION

Rule 1 is triggered when the sensor node is powered on.
R-1 = power_on
[TRUE; Initiate (SelfFireTime, PreFireTime, NextFireTime, alpha), Initiate (SelfFireTimer)]

Rile 2 is triggered when the sensor node receives firing signals from its neighbour before it fires
R-2 = fire_from_members
[BeforeFire; PreFireTime = CurrentTime]

Rile 3 is triggered when the sensor node fires
R-3 = self_fire_timeout
[TRUE; SelfFireTime = CurrentTime]

Rile 4 is triggered when the sensor node receives a firing signal after its neighbour before it fires
R-4 = fire_from_members
[AfterFire; NextFireTime = CurrentTime, SelfFireTimer = SamplePeriod + (1 - alpha) * SelfFireTime + alpha * (PreFireTime + NextFireTime)/2 – CurrentTime, Initiate (SelfFireTimer)]

 ISBN: 978-1-61208-207-3

VI. PROTOTYPE IMPLEMENTATION AND EVALUATION

The prototypes of the two cases have been implemented based on the general structure as shown in Fig. 3. To ease the implementation, the gateway is not included and the underlying communications among the PS and PNs are TCP/IP. However, the interfaces provided by the REED middleware are kept the same.

A. Evaluation of the Clustering Algorithm

To evaluate the rule-based clustering mechanism, a prototype containing three PNs in a single-hop network has been implemented. The cluster head update period is set as 12 seconds and the sampling period is 5 seconds. Two tests are carried out:

1. Formation of cluster heads: Fig. 7 shows the debugging information on the clustering algorithm, from which it can be seen that in this specific period, node A, with its ID being 101.0, becomes the cluster head/leader, and aggregated information (average of the sensed data) has been collected and sent by node A, the cluster head/leader.
2. Distribution of the cluster head: the system is tested for one hour and 20 minutes which equals to 400 cluster head update periods. Fig. 8 provides the information on the distribution of the cluster head across the PNs, where Node A was elected 135 times, Node B 138 times and Node C 137 times. It presents an overall uniform distribution which demonstrates that the process overhead imposed on the cluster heads are balanced among all the sensor nodes.
3. The test results show that the clustering mechanism, with the features being those mentioned in Section V, has been implemented on the REED.

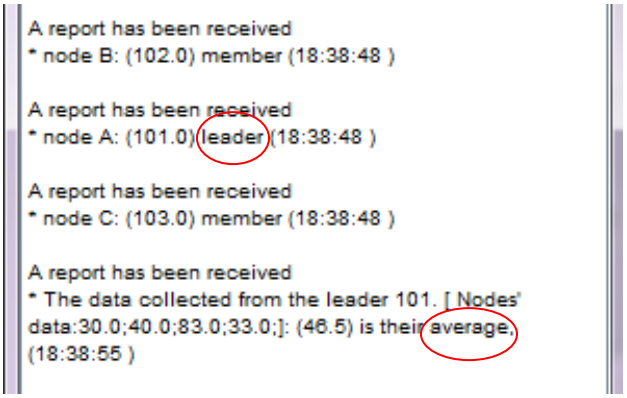

Figure 7. Debugging information on the clustering mechanism

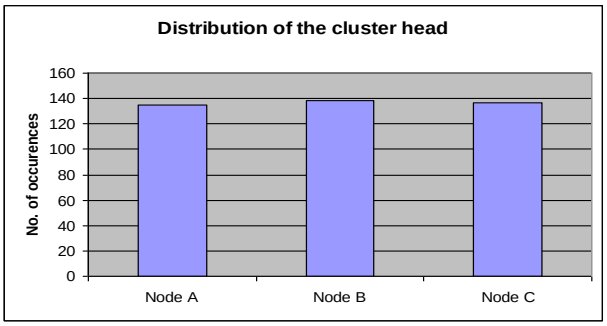

Figure 8. Distribution of the cluster head

B. Evaluation of the De-synchronization Algorithm

To evaluate the de-synchronization algorithm, a prototype containing three PNs in a single-hop network has been implemented. The sampling period for each PN is set as 12 seconds and alpha is set as 0.95. By dividing the sampling period by the number of the PNs, the desired time slot size, which in this case is 4 seconds, can be obtained. Fig. 9 shows the debugging information on the de-synchronization algorithm, from which the firing time difference between two time-wise adjacent nodes, i.e. the time slot size, can be derived. Time slot size is the core evaluation metric for the de-synchronization mechanism.

Fig. 10 illustrates the firing time differences between two PNs over time. It can be seen that at the very beginning, deviation from the desired slot size is non- negligible. This is because each PN initially starts their sampling tasks at random time. By running the de-synchronization algorithm, the firing time differences converge to the desired slot size quickly.

Figure 9. Debugging information on the de-synchronization

 ISBN: 978-1-61208-207-3

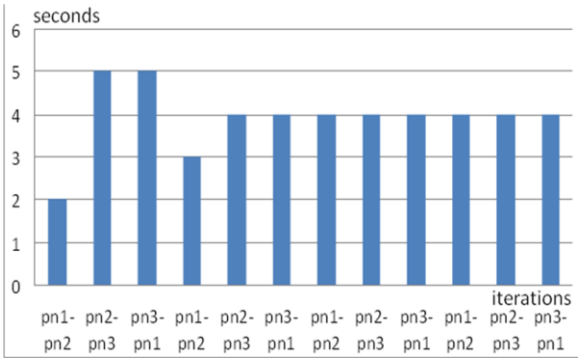

Figure 10. Firing time differences between two nodes

The system was running for 10 hours to test the dynamic behaviour of the de-synchronization algorithm. It was observed that the time slot size may oscillate when reaching the desired value, as shown in Fig. 11. This is mainly due to the choice of alpha, the degree of accuracy provided by the Java libraries, and etc. To get rid of the oscillations, for R-4, no adjustment on the next firing time will be applied if:

$$\left|SelfFireTime - (PreFireTime + NextFireTime)/2\right| \leq Threshold$$

In our case, the threshold is set as 55 milliseconds. After this modification, on noticeable oscillations were observed when the system is running for 10 hours.

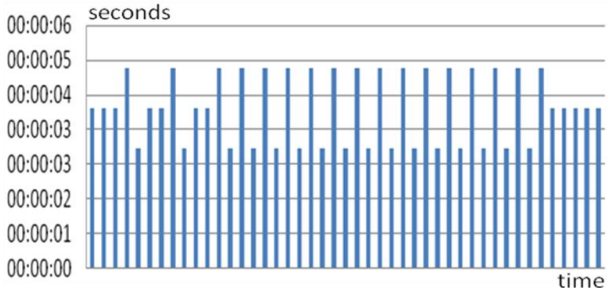

Figure 11. Oscillations of the time slot size

It should be noted that to realize the bio-inspired de-synchronization, only four rules are needed to be running on each node, which makes easier developing bio-inspired mechanisms on resource constraint sensor nodes.

VII. CONCLUSION AND FUTURE WORK

In this paper, the REED, a rule-based WSN middleware is briefly described. A general method to develop WSN applications is given which uses FSM as a bridge between application logics and the rules running on the REED. Especially we argue that rule-based middleware is especially useful for implementing bio-inspired mechanisms, such as self-organization, on WSNs. Prototypes of two WSN applications: sensor nodes clustering and de-synchronization, are implemented on the REED following the proposed developing method. The test results demonstrate the usability of the REED, the effectiveness of the proposed developing method, and especially the advantages of the rule-based realization of bio-inspired mechanisms. Further, we stress-out that the REED provides a framework that makes the application development more straightforward.

To further evaluate the effectiveness of the application development method proposed by this paper, in the future, we aim to extend our research by on the one hand, implementing more existent bio-inspired mechanisms on the REED; and on the other hand, developing real-world applications using the proposed FSM-based rules construction method.

REFERENCES

[1] PROSEN: http://www.cs.stir.ac.uk/~kjt/research/prosen/ [retrieved: August, 2012]

[2] L. Mottola and G. P. Picco, "Programming Wireless Sensor Networks: Fundamental Concepts and State of the Art", ACM Computing Surveys (CSUR) Volume 43 Issue 3, April 2011

[3] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey", Computer Networks,Vol. 52 , Issue 12, pp. 2292-2330, 2008

[4] M. Kuorilehto, M. Hannikainen, and T. D. Hamalainen, "A Survey of Application Distribution in Wireless Sensor Networks", Journal on Wireless Communications and Networking, Vol. 5, pp. 774–788, 2005

[5] X.Fei and E. Magil, "Rule Execution and Event Distribution Middleware for PROSEN-WSN", SENSORCOMM-2008, pp.580-585, 2008

[6] X. Fei and Z. Yu, "Development of a Rule-Based Wireless Sensor Network Middleware", Proceedings of the 16th International Conference on Automation & Computing, pp. 13-18, Sept. 2010

[7] K. Terfloth, G. Wittenburg, and J.Schiller, "FACTS - A Rule-Based Middleware Architecture for Wireless Sensor Networks", COMSWARE 2006, New Delhi, India, January 2006

[8] Gumstix: http://gumstix.com/ [retrieved: August, 2012]

[9] JamVM: http://jamvm.sourceforge.net/ [retrieved: August, 2012]

[10] K. Terfloth, "Doctoral Dissertation: A Rule-Based Programming Model for Wireless Sensor Networks", Freie Universitat, Berlin. June 2009.

[11] I. Wokoma, L. Shum, L. Sacks, and A. Marshall, "A Biologically-Inspired Clustering Algorithm Dependent on Spatial Data in Sensor Networks", Proceeedings of the Second European Workshop on Wireless Sensor Networks, pp. 386 - 390, 2005

[12] W. A. Geoffrey, T. Geetika, P. Ankit, W. Matt, and N. Radhika, "Firefly-Inspired Sensor Network Synchronicity with Realistic Radio Effects", Sensys'05, pp. 142–153, November, 2005

[13] D. Julius, R. Ian, P. Ankit, and N. Radhika, "DESYNC: Self-Organizing Desynchronization and TDMA on Wireless Sensor Networks", IPSN, pp. 11-20, April 2007.

[14] P. Boonma and J. Suzuki, "BiSNET: A biologically-inspired middleware architecture for self-managing wireless sensor networks", International Journal of Computer and Telecommunications Networking, Vol. 51 Issue 16, pp. 4599-4616, November, 2007

[15] R. H. Jacobsen, Q. Zhang, and T. S. Toftegaard, "Bioinspired Principles for Large-Scale Networked Sensor Systems: An Overview", Sensors, Vol. 11, pp. 4137-4151, 2011

 ISBN: 978-1-61208-207-3