Development of a Rule Based Wireless Sensor Network Middleware

Fei, X., & Yu, Z.

Proceedings of the 16th International Conference on Automation & Computing, University of Birmingham, Birmingham, UK, 11 September 2010

Development of a Rule Based Wireless Sensor Network Middleware

Xiang Fei*, Zixi Yu
Department of Computing and Digital Environment
Coventry University
Coventry, UK
*Contact email: x.fei@coventry.ac.uk

Abstract— **Recent years have witnessed significant interest in wireless sensor networking (WSN) due to its profound effect on the efficiency of many applications, plus the progress in sensor technology, wireless communications, and micro-processors. In order to support the development of these pervasive applications and the management of the underlying WSNs, middleware is needed to provide a uniform programming environment. REED (Rule Execution and Event Distribution) is a middleware solution that allows sensor networks to be programmed at run time via prescriptive rules. The contribution of this paper is two-folded: first, REED is extended to support not only data services, but also self-organization of WSNs; second, the prototype of REED is implemented and the test results show that REED is flexible enough to support WSN application development.**

Keywords- wireless sensor network, middleware, rules, self-organization

I. INTRODUCTION

Advances in sensor technology, wireless communications and micro-processors have made wireless sensor networking (WSN) a hot research topic in both academic and industrial communities. Generally speaking, a WSN consists of a collection of (heterogeneous) sensor nodes and a sink node connected through wireless links. As WSNs can be used in monitoring physical phenomena via data sensing, processing and distribution, they have found many applications in both military and civil areas, such as environmental surveillance, intelligent building, health monitoring, intelligent transportations, etc (Wang et al. 2008). One example is PROSEN (Networking of Distributed Sensors for Proactive Condition Monitoring of Wind Turbines) research project that aimed at developing a WSN system for proactively monitoring the wind farm conditions (PROSEN 2010).

The features of WSN systems, such as the distribution and heterogeneity of sensor nodes, the constrained resources (processing power, memory, and energy) of each sensor node, the error-prone wireless links and the dynamic network topology, make the WSN application development a challenging task (Wang et al. 2008, Römer, Kasten, and Mattern 2002, Yick, Mukherjee, and Ghosal 2008). To ease the wireless sensor data collection, processing and delivery, WSN middleware is introduced that provides an application programming interface (API) to shield the application developers from the complexities arising from the WSN. Some examples of WSN middleware solutions are: query-based solutions that take the whole WSN as a database (Madden, Franklin, and Hellerstein 2005), Turple-space based middleware that enable the sensed data on one node to be shared by the rest of the nodes (Murphy, and Picco 2005), mobile agent based paradigm that tracks mobile target (Fok, Roman, and Lu 2005), and system level abstraction that abstracts the whole WSN as a single virtual system (Welsh, and Mainland 2004). In order to support the (re)programmability of WSNs, Fei, and Magill (2008) proposed a middleware solution that supports rule execution and event distribution (REED). REED supports both the distribution of rules and the events that trigger them. REED employs a rule-based paradigm to allow sensor networks to be programmed at run time. This provides a flexible environment where applications and users can program the sensor nodes to allow their behaviors to be changed at run time. Further more, the behaviors of the WSN are described using descriptive rules and thus no knowledge of code programming is needed for the developers. In the paper authored by Fei, and Magill (2008), the architecture of REED was proposed; the rule management that enables run-time rule update was described, the pub-sub data service was constructed using rules, and the prototype implementation structure was provided. The contribution of this paper is two-folded: first, the REED is extended to be able to respond to not only data events, but also system events in order to enable WSN self-organization; second, the prototype of REED is implemented and tested.

The reminder of this paper is organized as follows: in Section 2, a rule based clustering algorithm is designed in order to demonstrate that the REED does not only support application related data service, such as pub-sub service as described in Fei, and Magill (2008), but also support dynamic system self-organization; in Section 3, the REED prototype implementation is describe in detail and test results are given; in Section 4 the related work is discussed; followed by the conclusion in Section 5.

II. RULE BASED PROGRAMMING

A. *General architecture and definition*

Figure 1 shows the system architecture for PROSEN, which consists of a Policy Server (PS), a Processing Node (PN) for each wind-turbine, and sensors to measure parameters such as temperature, wind-speed, wind-

Sponsored by EPSRC PROSEN project

direction, battery-level, 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 responses to poor weather conditions. The PS converts the goals to a set of rules. These rules describe the behaviour of individual PNs. Hence the WSN distributes and executes these rules within each PN. It is also possible to transfer these rules between PNs.

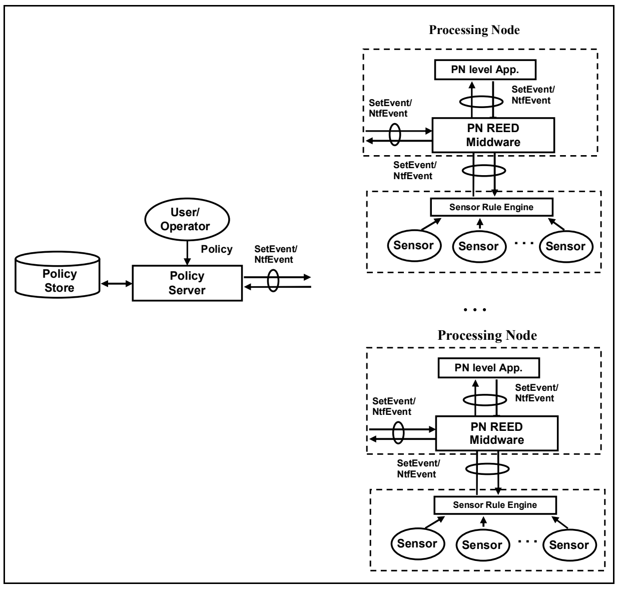


Figure 1. PROSEN system architecture

In addition to transferring the rules, the REED middleware also transfers events between the system components. It is these events that trigger the individual rules.

Conceptually, a *rule* takes the form of <*event*, *condition*, *action*> where:

- an event is received from any other component in the system. This is often an event carrying data values, but other events such as a timeout event, a sleep or wake-up event can also occur.
- 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.

To implement REED, a *rule-engine* has been designed and implemented. The functionality of the *rule-engine* includes:

- managing a rule-base that stores the rules for the middleware to allow the adding, removing, and overriding of rules
- verifying rule consistency, and
- executing the rules in response to received events.

Figure 2 shows the general architecture of the REED middleware. The middleware must record certain aspects of the state of the node and the events that have occurred. These are recorded in the *Fact-Base*. Here we borrow the terminology *Fact* from a separate rule-based WSN approach (Terfloth, Wittenburg, and Schiller 2006). The *Event-Manager* is responsible for receiving events, passing them to the *Rule-Engine*, where the engine executes any matching rules, and/or distributes any resulting events. The *Rule-Base* stores all the rules used by the engine.

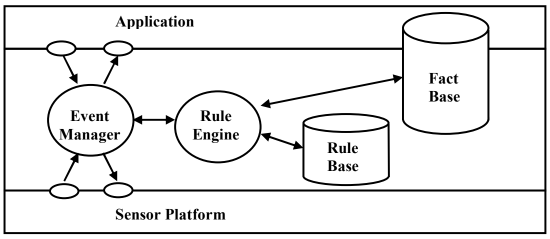


Figure 2. REED general architecture

In order to provide a clear description of the REED middleware, a formal notation is used. The notation is explained in Table 4 in Appendix and gives the core definitions.

B. Rule based pub-sub data service construction

Fei, and Magill (2008) described how to construct a typical WSN data service: publish-subscribe service:

- Data service subscription can be constructed by sending to sensor nodes a rule in which the subscribed data is expressed in event and condition of the rule; and the action of the rule is set as sending the data event back to the rule sender;
- Data publication is the result of executing the action of above rule in response of the data events that meet the condition of that rule.

An example of constructing rules for pub-sub over wind speed data event has been given in (Fei, and Magill 2008) and is described as follows:.

A PS subscribes to a PN REED middleware by sending a rule to a particular PN as follows:

Rule = < wind_speed; [(*wind_speed.value* >> 60*;* send (PS, *WindSpeed*))]>

Later on, the REED receives an event from its wind speed sensor as follows:

WindSpeed = <wind_speed; *Value* = 67; *Time* = 23:14:12; *Date* = 01-02-08>

This event will trigger the execution of the rule above, and as a consequence, this event will be notified to the PS.

Here we consider another example: PNs are organized as a clustering structure, as shown in Figure 3. Clustering (Wokoma et al. 2005) is a useful technique adopted in sensor networks for aggregating and transmitting sensed data to a central base station. This approach is energy efficient due to the fact that the processing nodes are prevented from consume a lot of energy to transmit their data over large distances (data transmission and receiving consume most of the energy on sensor nodes). The clusters reduce the data dependency on individual nodes by encouraging collaboration between sensor nodes and by distributing the work load amongst the members of the clusters as fairly as possible. To subscribe periodic average wind speed from cluster heads (the period is denoted as *T_sub*), each PN should receive the following rules as listed in Table 1:

TABLE 1. RULES FOR CLUSTER BASED AVERAGE WIND SPEED SUBSCRIPTION

< wind_speed; [(*self.Identity* = = "head"; save (*WindSpeed, self.windSpeedArray*))]> //here *self* is a **StateID**, and *Identity* is its **PropertyName**.
< wind_speed; [(*self.Identity* = = "member"; send(*self.head*, *WindSpeed*))]>
<T_sub_timeout; [(*self.Identity* = = "head"; average(*self.windSpeedArray, Mean*), send (PS, *Mean*))]> //here average (X, Y) means calculating the average over array X, and save the result to Y.

The reason why each PN should keep the same subscription rules is that the cluster structure may be self-updated periodically (the period is denoted as T_update) for load balancing and robustness.

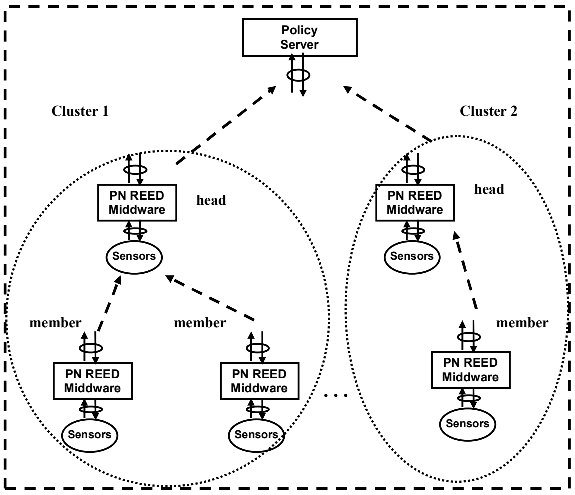


Figure 3. clustering structure of PNs

C. *Rule based clustering construction*

REED can not only ease the construction of WSN data services, but also support self-organisation of the sensor nodes, such as clustering mentioned above.

TABLE 2 gives the rules for one clustering algorithm that includes cluster head election, cluster head determination and cluster member determination. Each cluster member chooses its cluster head based on the SNR (Signal Noise Ratio) of its reachable cluster heads.

It can be seen from TABLE 2 that:

- The clustering is fully distributed and self-organized;
- Rules on each processing node are quite simple, and the emergent clustering comes from the interaction among these processing nodes.

TABLE 2: RULES FOR CLUSTERING

1. cluster head election
< power_on; [(true; init, set(*self.identy*, "pending"), back_off (T_backoff))]> // here T_backoff is a random number and is used for cluster head election.
< T_update_timeout; [(true; set(self.identy, "pending"), back_off (T_backoff))]> //here T_update_timeout is for cluster self-updating
2.cluster header determination
<T_backoff_timeout; [(*self.identy* == "pending"; set (*self.identy*, "head"), broadcast (*Head_Beacon*))]> //here Head_Beacon is an event telling neighbouring nodes it is a clustering head.
3.cluster member determination
< head_beacon; [(*self.identy* == "pending", set (*self.identy*, "membet"), set (*self.head*, *Head_Beacon.PN_id*), set(*self.SNR*, *Head_Beacon.SNR*))]> //here SNR means signal noise ratio.
< head_beacon; [((*self.identy* == "member") && (*self.SNR* < *Head_Beacon.SNR*); set (*self.head*, *Head_Beacon.PN_id*), set(*self.SNR*, *Head_Beacon.SNR*))]> // if a cluster member received more than one head beacon, it chooses the one with best SNR as its cluster head.

III. PROTOTYPE IMPLEMENTATION

A. *Prototype implementation architecture*

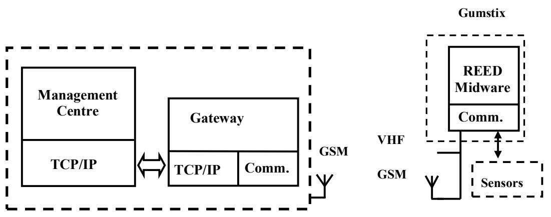


Figure 4. Prototype implementation architecture

Figure 4 shows the prototype implementation architecture, where the Management Center and the Gateway are running on PCs and the REED is running on a Gumstix^{TM} (Gumstix 2010) GS400K-XM. Gumstix is a miniature full function Linux motherboard based on low power Intel XScale® technology.

B. *Management Centre*

In PROSEN, the Policy Server is implemented by another partner in the project. The messages used for the interaction between the Policy Server and the Gateway have been designed, and the Management Centre uses the same messages for prototype implementation.

The functionality of the Management Centre is to provide a graphic user interface (GUI) via which:

- Operators can set rules and send them to the PNs;
- Sensor data collected can be displayed.

The Management Centre is implemented using Java as Java is a general purpose objected oriented programming language, and especially provides a variety of components (Swing components) for GUI development.

C. *Processing Node*

1) *REED*

The software structure for REED middleware is illustrated in Figure 5. REED sends and receives external messages via the interfaces provided by the UCP (Unified Communication Platform). The Event Constructor constructs events with the received messages. It classifies them either as SetEvents (rules), or as NtfEvents (e.g. data events), and then puts them onto their corresponding queues. These two queues may have different priorities. When any event is to be distributed, the Msg Constructor will transform it to the corresponding message format before delivering it to the UCP.

REED is implemented using Java. This is because, first, Java code can be running on any devices that support JVM (Java Virtual Machine) and thus REED is portable to other platforms with JVM; second, GS400K-XM has 16MB flash memory and can accommodate JamVM (JamVM 2010) which is a compact JVM.

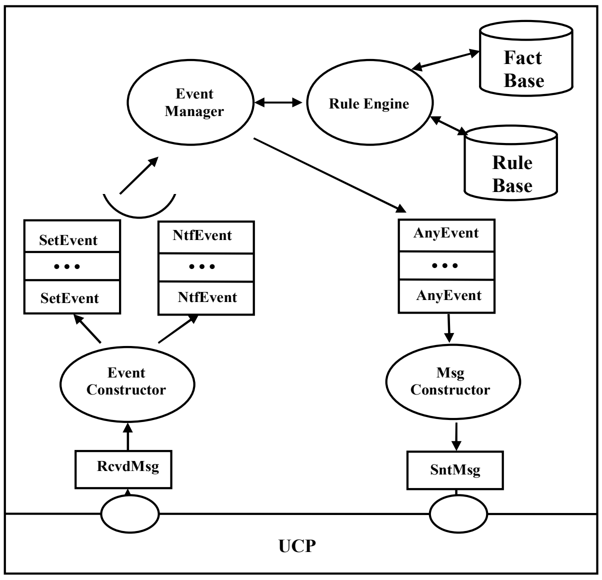


Figure 5. REED Software Structure

2) *UCP*

UCP aims to provide REED a unified interface to exchange messages so that REED doesn't need to care about the various communication links (VHF, GSM, Zigbee, etc) and detailed communication management. As compared to Java code, c code consumes less memory and takes less execution time to provide the same functionality (Horr´e et al. 2000), c is used to develop the UCP on the resource constrained PNs.

To enable the interaction between REED and UCP, two techniques are explored. The first one is JNI (Java Native Interface) (Java Native Interface 2010). JNI is a programming framework that allows Java code and code written in other languages, such as c and c++, to interact with each other. The other technique is using named pipes. A named pipe (Named pipe 2010), also know as a FIFO, is an extension to the traditional pipe concept on Unix and UNIX-like systems, and is one of the methods for inter-process communication.

Compared to pipes-based inter-process communication solution which can only be used in UNIX and UNIX-like environment (pipes in Microsoft Windows have substantially different semantics), JNI can be used in both UNIX and Microsoft Windows environment. However, pipes-based technique is more flexible and straightforward to implement. Both techniques have been implemented. The one that are used in our system test, called UCM (Unified Communications Manager), is the combination of JNI and pipe-based solution (our Gumstixs provide Linux environment): using JNI to negotiate system unique named pipes, and then using these pipes to exchange messages. The UCM is developed by other partners in the PROSEN project.

D. *Gateway*

The introduction of the Gateway is due to the following reasons:

- The PS has been developed by one of the partners in the PROSEN project. The communication interface provided by the PS is TCP/IP while the interfaces to WSN may be VHF (174 MHz) or GSM. So the gateway is needed to inter-connect the WSN and TCP/IP;
- The WSN may be deployed in an area that is not wirelessly reachable to its PS. So the gateway can be used as a relay;
- Most importantly, to minimise the energy consumption caused by communicating, messages sent and received by each PN should be as concise as possible by using special coding schemes, while the messages to and from the PS are readable plain texts. So the Gateway should take the role of an interpreter. Table 3 gives two examples that need interpreting.

TABLE 3. EXAMPLES OF MESSAGES ON MANAGEMENT CANTER AND THEIR CORRESPONDING MESSAGES ON PNS

Management Centre		PN
device_out(set_rule, wind_speed, 2, ,[max_wind_speed, 60.0, alert_ps])	⇒	9;0; ; [1_0; 60.0; 0]
device_in(above_threshold, wind_speed, 2, 15:30:00, [65])	⇐	0;2;15:30:00; 0;1;65

The Gateway is implemented using Java.

E. *System test*

As shown in Figure 6, Management Center and Gateway are running on PCs and these two PCs are connected to the Internet. REED and UCM are running on a Gumstix. The Gumstix and the Gateway are connected via two VHF wireless modems. Sensor readings are simulated via a random number generator.

Figure 7 shows the snapshot on the Management Centre. Via the Management Centre GUI, operators can subscribe the wind speed sensor readings that are higher than the upper threshold (60 kph in this case), and view the detailed results (sensor reading, sensor ID and timestamp) published by the PNs.

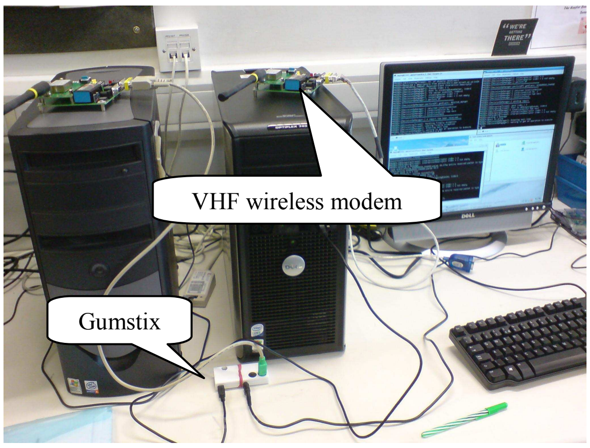
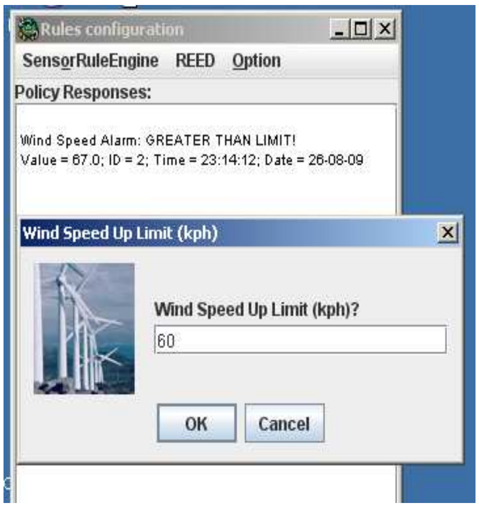


Figure 6. prototype system for test



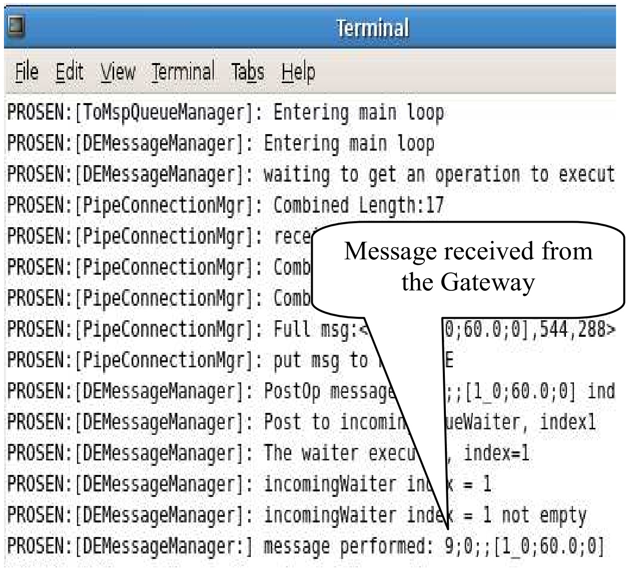
Policy Responses:

Wind Speed Alarm: GREATER THAN LIMIT!
Value = 67.0; ID = 2; Time = 23:14:12; Date = 26-08-09

Wind Speed Up Limit (kph)? 60

Figure 7. snapshot on the Management Center

Figure 8 shows the REED debug information on the Gumstix, from which it can be seen that the message received from the Gateway has been transformed to the concise format understandable by the REED.


Terminal
File Edit View Terminal Tabs Help
Message received from the Gateway

```
PROSEN:[ToMspQueueManager]: Entering main loop
PROSEN:[DEMessageManager]: Entering main loop
PROSEN:[DEMessageManager]: waiting to get an operation to execut
PROSEN:[PipeConnectionMgr]: Combined Length:17
PROSEN:[PipeConnectionMgr]: rece
PROSEN:[PipeConnectionMgr]: Comb
PROSEN:[PipeConnectionMgr]: Comb
PROSEN:[PipeConnectionMgr]: Full msg:<          0;60.0;0],544,288>
PROSEN:[PipeConnectionMgr]: put msg to          E
PROSEN:[DEMessageManager]: PostOp message      ;;[1_0;60.0;0] ind
PROSEN:[DEMessageManager]: Post to incomin     ueWaiter, index1
PROSEN:[DEMessageManager]: The waiter execu    , index=1
PROSEN:[DEMessageManager]: incomingWaiter in   x = 1
PROSEN:[DEMessageManager]: incomingWaiter index = 1 not empty
PROSEN:[DEMessageManager:] message performed: 9;0;;[1_0;60.0;0]
```

Figure 8. debug information on the Gumstix

In addition, the rules can be updated during runtime. E.g. the upper threshold of the wind speed (*condition* part of the *rule*) can be changed and then sent to PNs, and as a result, the sensor data notified by PNs will be changed accordingly.

The test results show that REED is flexible enough to support WSN application development (e.g. pub-sub data service). The ability of the REED to support WSN self-organization will be proved when the rule based clustering is implemented and tested.

IV. Related Work

Zhang, Li, and Pan (2005) proposed an ECA (Event, Condition and Action) rules based middleware model for WSN. However, no prototype implementation was provided. In the paper authored by Terfloth, Wittenburg, and Schiller (2006), a rule-based middleware architecture for WSN, called FACTS, was proposed, and Terfloth, Wittenburg, and Schiller (2006) described its programming primitives and implementation using the Haskell programming language. However, the rule set in FACTS is static while the *rule-base* in REED is dynamic as the rules for REED can be updated at run time. Furthermore, the REED prototype has been implemented, which demonstrates not only the functionality but also the usability of REED.

Keoh et al. (2007) and Keoh et al. (2006) proposed a policy based middleware architecture for managing body sensor networks, in which the policies take on the same form (<*event*, *condition*, *action*>) as used in REED. Indeed Keoh et al. (2006) conclude that the policy based middleware provides flexibility to reprogram the sensor with new adaptation strategies without requiring installation of new code. However, they did not demonstrate such reprogramming scenarios.

JESS is a rule-engine written entirely in Sun's Java language (JESS 2010). It is for general purpose and not dedicated for a WSN environment. As a consequence, the memory usage is not optimized (Terfloth, Wittenburg, and Schiller 2006) for running on sensor nodes. In addition, in JESS, all the facts are stored in its working memory before executing the rules while in REED, any received data event will be filtered by rules first and only those needing further processing will be saved to the *fact-base*. As a result, the overhead for memory consumption is expected to be lower than using JESS.

Dressler et al. (2009) proposed a rule-based sensor network (RSN) for sensor and actor networks. The rules take the form of < *if PREDICATE then { ACTION }* >. Each RSN node stores all received messages in a buffer. The rule interpreter is either started periodically or after the reception of a new message. Simulation results showed the feasibility of such an approach. However, no prototype implementation was provided

V. Conclusion

REED is a rule based middleware for WSNs. This paper extends the REED described in the paper authored by Fei, and Magill (2008) by supporting not only data services, such as pub-sub data service, but also self-organization of WSNs, such as clustering. Rules for clustering algorithm have been constructed. In addition, a prototype system has been implemented and the test results show that REED is flexible enough to support WSN application development.

The rule based clustering is being implemented with the hope to prove the ability of the REED to support WSN self-organization.

VI. ACKNOWLEDGEMENTS

The authors would like to thank EPSRC (Engineering and Physical Sciences Research Council) for funding the PROSEN project, and the support from PROSEN research colleagues.

REFERENCES

[1] Wang, M., Cao, J., Li J., and Dasi, S. K. (2008) ‘Middleware for Wireless Sensor Networks: A Survey’, Journal of Computer Science and Technology 23 (3), 305-326

[2] PROSEN [online] available from < http://www.cs.stir.ac.uk/~kjt/research/prosen/ > [8 July 2010]

[3] Fei, X., and Magill, E. (2008) ‘Rule Execution and Event Distribution Middleware for PROSEN-WSN’, Second International Conference on Sensor Technologies and Applications (SENSORCOMM), 580-585

[4] Murphy, A. L., and Picco, G. P. (2005) ‘TinyLIME: Bridging Mobile and Sensor Networks through Middleware’, PERCOM (Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications), 61 – 72

[5] Madden, S. R., Franklin, M. J., and Hellerstein, J. M. (2005) ‘TinyDB: An Acquisitioned Query Processing System for Sensor Networks’, ACM Trans. Database Systems 30(1) 122–173

[6] Welsh, M., and Mainland, G. (2004) ‘Programming sensor networks using abstract regions’, Proceedings of First Symposium on Networked Systems Design and Implementation (NSDI ‘04), 29–42

[7] Fok, C., Roman, G., and Lu, C. (2005) ‘Mobile Agent Middleware for Sensor Networks: An Application Case Study’, Proc. of the 4th Int’l Conf. Information Processing in Sensor Networks (IPSN 05), 382–387

[8] Zhang, C. , Li, M. , and Pan, Q. (2005) ‘An ECA Rules Based Middleware Architecture for Wireless Sensor Networks’, Proceedings of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT), 586 – 588

[9] Römer, K. , Kasten, O. , and Mattern, F. (2002) ‘Middleware Challenges for Wireless Sensor Networks’, ACM SIGMOBILE Mobile Computing and Communications Review 6(4), 59 - 61

[10] Yick, J. , Mukherjee, B. , and Ghosal, D. (2008) ‘Wireless sensor network survey’, Computer Networks 52(12), 2292-2330

[11] Terfloth, K. , Wittenburg, G., and Schiller, J. (2006) ‘FACTS - A Rule-Based Middleware Architecture for Wireless Sensor Networks’, First IEEE International Conference on Communication System Software and Middleware (COMSWARE 2006), 1-8

[12] Terfloth, K. , Wittenburg, G. , and Schiller, J. (2006) ‘Rule-oriented Programming for Wireless Sensor Networks’, International Conference on Distributed Computing in Sensor Networks (DCOSS) / EAWMS Workshop

[13] Wokoma I., Shum L., Sacks L., Marshall I.W. (2005) ‘A Biologically-Inspired Clustering Algorithm Dependent on Spatial Data in Sensor Networks’, 2nd European Workshop on Wireless Sensor Networks, Turkey

[14] Gumstix [online] available from < http://gumstix.com > [8 July 2010]

[15] JamVM [online] available from < http://jamvm.sourceforge.net > [8 July 2010]

[16] Java Native Interface [online] available from < http://en.wikipedia.org/wiki/Java_Native_Interface > [8 July 2010]

[17] Named pipe [online] available from < http://en.wikipedia.org/wiki/Named_pipe > [8 July 2010]

[18] Keoh, S. L., Dulay, N., Lupu, E., Twidle, K., Schaeffer-Filho, A. E., Sloman, M., Heeps, S., Strowes, S., and Sventek, J. (2007). Self managed cell: A middleware for managing body sensor networks. In Proceedings of the 4th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (Mobiquitous)

[19] Keoh, S. L., Twidle, K., Pryce, N., Schaeffer-Filho, A. E. , Lupu, E., Dulay, N., Sloman, M., Heeps, S. , Strowes, S., Sventek, J., and Katsiri, E. (2006) ‘Policy-based Management for Body-Sensor Networks’, 4th International Workshop on Wearable and Implantable Body Sensor Networks (BSN), 26 – 28

[20] Dressler, F., Dietrich, I. , German, R. , and Kruger, B. (2009) ‘A Rule-based System for Programming Self-Organized Sensor and Actor Networks’, Computer Networks 53 (10) 1737-1750

[21] JESS [online] available from < http://www.jessrules.com/ > [9 July 2010]

[22] Horr´e, W. , Matthys, N., Michiels, S. Joosen, W., and Verbaeten, P (2000) An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl [online] available from < http://page.mi.fu-berlin.de/prechelt/Biblio//jccpprt_computer2000.pdf > [8 July 2010]

APPENDIX

TABLE 4. CORE LANGUAGE DEFINITION FOR REED

Property = <PropertyName “=” PropertyValue>

State = <StateID “;” Property | State “;” Property>

Event = < EventID “;” Property | Event “;” Property >

FactID = < StateID | EventID >

Fact = <State | Event>

ComparisonOperatior = < “>>” | “<<”| “>=”| “<=”| “= =”| “!=”>

Connector = < “&&”>

ExistOperator = < “∃” >

Condition = <”True” | EsistOperatpor “(” FactID “)”| FactID “.” PropertyName ComparasionOperator Threshold | FactID “.” PropertyName ComparasionOperator FactID “.” PropertyName >

ConditionSet = <Condition | Condition Connector Condition>

Action = < Set “(”FactID.PropertyName, PropertyValue“)” | Send “(”Destination “,” Event“)” | FunctionCall “(”Event “)” | …>

ActionetSet = < Action | Action “,” Action>

EventHandler = < “(” ConditionSet “;” ActionSet “;” Priority “)” | EventHandler ; “(” ConditionSet “;” ActionSet “;” Priority “)”>

Rule = <Event_ID “;”“[”EventHandler“]” >