Extending Middleware frameworks for Wireless Sensor Networks

Syed Rehan Afzal, Christophe Huygens and Wouter Joosen
IBBT DistriNet Research Group, Department of Computer Science,
Katholieke Universiteit Leuven
Leuven, Belgium
{Rehan.Afzal, Christophe.Huygens, Wouter.Joosen}@cs.kuleuven.be

Abstract— **We define sensor middleware as the binding code mainly running between the sensor OS and applications providing programming abstractions to bridge the gap between application developers and low-level hardware. Hence it serves the purpose of simplified integration of components developed by multiple technology vendors. Middleware for distributed systems is a relatively mature research area with a considerable amount of work done on the aforementioned topics. In this paper we explain why standard distributed systems middleware solutions are not suitable to address Wireless Sensor Network (WSN) problems. Illustrated by a health monitoring use case, we propose an enhanced middleware framework that better addresses the needs of WSN applications.**

Keywords- *Sensor Networks, Middleware, Security*

I. INTRODUCTION

Over the past decade, Wireless Sensor Network (WSN) applications have evolved from being a niche technology to a wide range of complex industrial and business applications. Beginning as battlefield surveillance devices, sensing capabilities today span every aspect of life, including: acoustics, automotive, chemical, electric, environment, airflow, pressure, optical and thermal sensing. Sensor devices have grown smaller, more power-efficient and more capable, becoming less expensive at the same time. Current WSN technologies comprise a large array of technological alternatives in terms of hardware vendors, operating systems and communication technologies; hence, reflecting a wide range of potential future applications. Nevertheless, there remains a huge gap between the ongoing state-of-the-art research and its exploitation in real-world application scenarios. We suggest a strongly application driven design for wireless sensor networks which is devised keeping the state-of-the-art in sensor network technology in mind.

The future, in our opinion, lies in heterogeneous wireless devices working together to serve common or conflicting end user goals by providing a seamless interface to remote sensor services, wherein end-users and application developers are unaware of the heterogeneity underneath. Heterogeneous devices cannot work together by default because of several reasons, including: different hardware implementations, native operating system concepts, programming models and programming abstractions. Consequently it's necessary to have a middleware that overcomes and protects developers and end-users from these hardware and software intricacies.

A considerable amount of work has been done in the area of WSN middleware such as database, mobile agents or event driven middleware solutions [9]. Hadim et. al. [22] classified existing solutions consisting of Virtual machines, database, modular programming, application driven or message-oriented approaches. These approaches simplify tasking sensor networks at different levels. However, what's missing is a holistic middleware solution that addresses the existing gaps by providing support for required programming abstractions and positioning them right from the hardware platforms up to end user applications. Middleware for state-of-the-art distributed systems is a relatively mature area [15], which when implemented properly, can help to:

- Shield software developers from low-level, tedious, and error-prone platform-specific details
- Offers patterns in reusable frameworks, rather than rebuilding software monolithically for each use
- Provides a consistent set of higher-level network-oriented abstractions that better match application requirements
- Provides a wide array of non-functional services, such as logging and security that have been proven necessary to operate effectively in a networked environment

Though WSNs application development also requires the aforementioned support, contemporary distributed systems frameworks may not be appropriate for WSNs primarily because of three key reasons *(1) constrained nature of WSNs in processing capability and battery life (2) instead of being static and fixed as in traditional distributed systems, WSN nodes are dynamic and mobile in nature (3) Unlike traditional distributed systems, in WSNs most, if not all, of the aspects related to mobility, routing and security are managed by the nodes themselves; since messages are handled by nodes instead of routers.* In this paper we make two primary contributions:

9781-4244-3941-6/09/$25.00 ©2009 IEEE

- Identify middleware design goals and requirements for applications built upon heterogeneous WSNs
- Present our WSN middleware architecture aimed to serve as a framework for future WSN middleware development.

The rest of the paper is structured as follows: Section II outlines some of the notable middleware approaches proposed, section III describes our problem domain and discusses the example of our ongoing project on wireless sensor networks based remote health monitoring. After discussing the desired role of the middleware in section IV, we then present our middleware architecture in section V. Finally we conclude in section VI.

II. RELATED WORK

During the past few years, researchers have devoted much effort in designing and developing suitable algorithms and programming paradigms for middleware for WSNs. In this section we outline some of the notable middleware approaches each representing a different technique e.g. Sensorpedia [23] (Web 2.0 based), TinyDB [24] (Database oriented), Mate [25] (Virtual Machine based), Agilla [26] (Mobile Agent), TinyLime [27] (tuple space) and TinyCubus [28] (cross-layered) middleware.

Sensorpedia [23] - an ongoing project at the Oak Ridge National Laboratory - connects sensors composed of various standards through a Wikipedia-like platform with three important distinctions: links to near-real-time, streaming data, support of interactive mashups and restriction of authorship to approved personnel. Sensorpedia is based on the same underlying social networking and collaboration principles used by popular web sites such as Wikipedia, Squidoo, Google Maps, and Facebook. It provides near-real-time collaboration among communities with requirements to share sensor information. Sensorpedia possesses an open API and claims to have flexible access controls mechanisms. Its beta version is expected to be released soon.

TinyDB [24] is a query processing middleware system based on TinyOS [1]. TinyDB provides power-efficient in-network query processing system for collecting data from individual sensor nodes which reduces number of messages that must be sent. This results in reduced energy consumption. It has two different types of messages for query processing – Query Messages and Query Result Messages. It also has Command Messages for sending command to sensor nodes. While TinyDB provides solid abstraction support and has good aggregation model, it does not provide much functionality as part of middleware service. Consequently most of the services have to be provided in the applications running on top of it.

Mate [25] is a virtual machine for sensor networks implemented on top of TinyOS [1]. It hides the asynchrony and race conditions of underlying TinyOS. Mate has a stack-based architecture with three execution contexts – clock, send, and receive. Mate breaks down the program into small self-replicating capsules consisting of 24 instructions. These capsules are self-forwarding or self-propagating. Although Mate has a small, concise, resilient, and simple programming model, its energy consumption is high for long running programs. Mate's virtual machine architecture increases security somewhat and it takes care of malicious capsules.

Agilla [26] is a Mobile Agent based middleware. Its stack-based architecture reduces code size. Agilla allows agents to move from one node to another using two instructions – clone and move. Up to four agents can run on a single sensor node. Since one node can run multiple agents at the same time, multiple applications can be supported on the network simultaneously. To save energy, Agilla can move its agent to bring computation closer to data rather than transmitting data over unreliable wireless network.

TinyLime [27] is implemented on top of TinyOS exploiting Crossbow's Mote platform. It is an extension of Lime [28]. TinyLime follows an abstraction model based on shared tuple space. This tuple space contains sensed data. It supports data aggregation to find more information from collected data. TinyLime consists of three main components i.e. Lime integration component, mote interface and mote level subsystem. TinyLime however does not have any built in security support.

TinyCubus [29] is a cross-layer framework implemented on top of TinyOS. Flexibility and adaptation are two major issues behind the design philosophy of TinyCubus. To achieve this, TinyCubus architecture is divided into three parts namely tiny cross-layer framework, tiny configuration engine and tiny data management framework. Although TinyCubus's flexible architecture allows it to be used in different environments, overhead due to cross-layer design may be prohibitive in some environments.

III. PROBLEM STATEMENT AND BACKGROUND

Over the past decade, the domain of wireless sensor networks has been expanding over the following dimensions:

- *Hardware* i.e. RAM, flash memory, radio range, sensing capabilities, clock, battery life etc.
- *Programming paradigms* with varied levels of abstraction such as native code, byte code or interpreted code including nesC, C, TCL, Java, Maté, Agilla [6] etc
- *Networking support* is offered by a wide range of routing protocols [7] and security enhancements including: key distribution, authentication and encryption/ decryption mechanisms
- *Communication Medium Support* comes with several choices with different tradeoffs as well. We start with *IEEE 802.15.4* which provides Physical and MAC support for WLAN by means of low cost but low speed and less reliable communication between devices. It forms the basis of other WLAN communication protocols such as *Zigbee* [20] which

provides a complete networking solution by developing the upper layers at the cost of added complexity or *6LoWPAN* [21] which unlike Zigbee implements *IPv6* and uses header compression mechanisms to allow IPv6 packets to travel over IEEE 802.15.4. In comparison to 802.15.4, *Bluetooth* does not provide mesh-networking support and limits the number of nodes that can actively maintain a connection, while *Wifi*, on the other hand is too expensive and power-hungry to serve as a WSN communication medium.

Hence, it can be seen that sensor application architects and programmers have a large pool of technologies to choose from in order to build modern sensor applications. Nevertheless, applications composed of such heterogeneous platforms, programming languages and networking protocols cannot coexist together unless there are modules that bridge this assortment of underlying heterogeneous technologies.

A. Use case - Remote Patient Monitoring & Tracking

Our Networking Task force in Distrinet Labs, K. U. Leuven is conducting an ongoing project DEUS (Deployment & Ease of Use of wireless Services) [3]. Our vision is to make wireless sensor networks easy to use by providing seamless interfaces and transparent service access. Figure 1 explains the proposed architecture for patient monitoring where a patient is wearing multiple sensor nodes to read particular health data of interest such as pulse rate or body temperature.

As shown in the figure, the health monitoring data is collected by Tmote sky motes [11] and after node level filtering and aggregation is transmitted via more powerful sensor nodes, which in this case are SunSPOTs [8]. Finally, after further filtering, aggregation and redundancy checks, the data reaches the even more powerful wireless nodes, which in this case are iMote 2.0 [10].

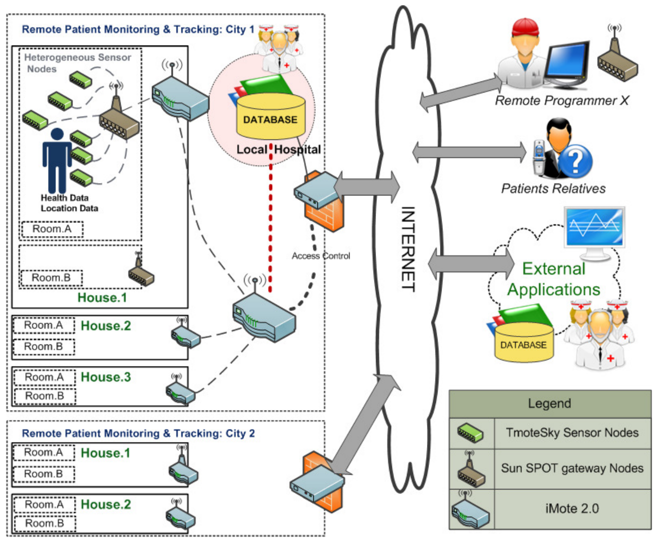


Figure 1. DEUS architecture

Mote	RAM	External Memory	Operating System
Tmote Sky	10KB	48KB Flash	TinyOS & Contiki[2]
SunSPOT	512KB	4MB Flash	Squawk VM [11, 12]
iMote 2.0	32MB	32 MB Flash	TinyOS

Figure 2. Employed sensor hardware

The abovementioned hardware comes from different vendors. Figure 2 demonstrates the processing capabilities and supported Operating Systems for each node. Since each hardware platform possesses distinct traits and tradeoffs; it makes sense to using a combination of these devices where each device type should be deployed where it is most suited. This result in the use of different routing protocols for each subset of motes based on characteristics such as their processing power, battery life and transmission range. Hence we choose the following routing architecture:

- *TmoteSky:* Employs Dynamic Source Routing Protocol (DSR) [4], which is a very efficient source routing based protocol.
- *SunSPOT:* Make use of AODV [13] protocol supported by default by the SunSPOT motes.
- *Imote 2.0:* On the most powerful motes, we employ Robust Secure Routing Protocol (RSRP) [14] based on DSR (Dynamic Source Routing). However unlike DSR, RSRP provides security, confidentiality and integrity attributes and deploys symmetric cryptography for encryption and hashing.

In our application scenario the body sensor nodes which in this case are Tmote Skys read the health data like blood pressure, glucose level, and pulse rate. As soon as an anomaly occurs, an alert is passed to the local hospital using the SunSPOTS and iMotes as routing infrastructure. The body sensor nodes also periodically forward data to the local administrator so that it can be accessed by other interested parties over the internet e.g. patient's caregivers.

B. Challenges

These capability centric design choices result in several interoperability and compatibility challenges that need to be addressed by middleware functions.

1) Multiple Routing Protocols & Networking support: Networks running on different communication protocols cannot communicate with each other directly e.g. nodes running 6LowPAN or TSMP over 802.15.4 cannot communicate with the nodes running ZigBee [19]. Every routing protocol has its own packet format and security schemes (key distribution, authentication algorithms, hashing mechanisms) etc. Hence messages originating from one routing domain cannot directly be interpreted by nodes belonging to the other.

2) Interoperability: In case of remote code deployment, interoperability problems occur if (a) the source of the code and deployment destination use different programming paradigms/ platforms (b) deployment destination nodes is a set of heterogeneous nodes running on different hardware and programming models.

3) *Difficulty of Integration:* Such an ad-hoc composition of sensor components may result in a lack of usability as there exists no distinction between novice users and application developers. Hence all external users, applications and application developers would require binding code in order to integrate with the WSN infrastructure.

4) *Resource Constraints:* WSNs comprise of sensors with very limited resources in terms of battery life, energy and bandwidth etc. Hence fulfilling functional and non-functional application requirements is a challenging task.

5) *Connectivity:* Dynamic and mobile transmission nature of WSN hinders the objective of continued connectivity. As the network topology changes, the neighboring nodes lose track of each other's coordinates.

IV. Framework Requirements

Based on the domain use case and challenges we have highlighted, we now state the requirements for the desired middleware framework.

1) *Transparency:* One of the major desired characteristics is to provide transparent access to local, remote, and mobile resources. Transparency when applied to middleware generally refers to the fact that infrastructural details are hidden from the application and end users. Hence, the end user can acquire the needed data without knowledge of the underlying platform heterogeneity e.g. the patient X's caregiver can request the glucose level of patient X, without requiring knowledge of infrastructure details.

2) *Platform independence:* During the application life cycle, it's quite natural to update the policies within the network based on past experiences. However this would be very cumbersome if (1) the nodes need to be reprogrammed onsite (2) one particular policy update triggers the need to reprogram all different nodes in their respective programming languages. Hence it is needed to have sensor network middleware that supports remote code deployment with platform independence.

3) *Seamless integration:* It is also desired that the data from the sensor nodes could be made useful for the interested stakeholders all across the internet, for example in the form of Sensor Services where the sensor services interface could be easily integrated with other external applications without needing some service oriented plumbing code.

4) *Enhanced In-network support:* In order to make the most efficient use of scarce sensor resources, in-network services must make intelligent use of available resources e.g. through data aggregation and filtering.

5) *Mobility Support:* Finally, specific mechanisms are required to support node mobility and handle network dynamism. An ideal middleware solution should provide methods e.g. periodic route discoveries and routing table updates so that the nodes view of the network topology gets refreshed continuously.

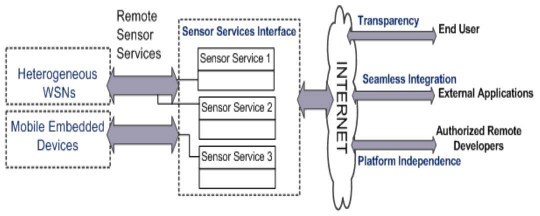


Figure 3. Service oriented architecture

V. Proposed Architecture

Considering the previously mentioned challenges as common to most of the future Wireless Sensor Network applications, we now propose our middleware architecture. The constrained and dynamic nature of WSN and other networked embedded devices sets them apart from state-of-the-art distributed systems. Furthermore concerns like mobility, routing and security are handled by nodes in contrast to routers as in the case of standard distributed systems. Our proposed framework builds on the fundamental concepts of Distributed Object Computing (DOC) middleware [15] focusing specifically on wireless sensor networks and mobile embedded devices. The resulting middleware framework is designed to address the distinctive characteristics of WSN. The middleware functionality is structured in four layers of components as shown in Figure 4.

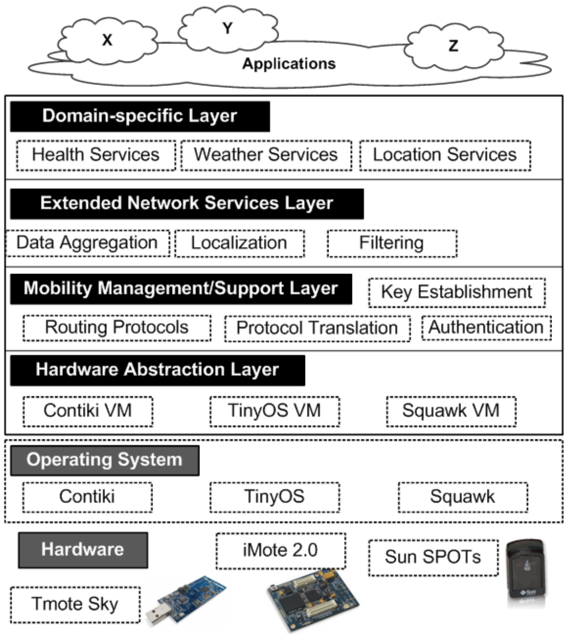


Figure 4. WSN middleware framework

A. Hardware Abstraction Layer

The Hardware Abstraction Layer is positioned directly over the sensor node operating system. The primary task of this layer is to provide transparency & platform independence by hiding machine level complexities, individual OS peculiarities etc. and provide a simple function call based interface to the upper layers. Squawk Java Virtual Machine for Sun SPOT sensor nodes [11] is an existing example of Hardware Abstraction Layer. Squawk runs on the bare ARM-9 hardware and provides low-level operating system support, as well as application isolation (isolates).

Furthermore, we envision that the hardware abstraction layer will provide abstractions to support *remote policy updates and code deployment.* For instance in Figure 1, the *programmer X* is asked to update a particular policy on the sensors currently present in *Room 1* of *house 1*. Typically the programmer is required to update policies individually for each particular set of sensor hardware corresponding to a specific programming language. However in ideal case, the programmer would write a policy update e.g. change the patients periodic temperature reading interval from reading/10mins to reading/15mins. This update may trigger parametric changes on different sensor hardware running different programming platforms; hence a code update for each distinguished platform. Ideally, the programmer defines a policy change and each VM translates it to suit its underlying hardware.

Matthys et. al. [18] presented a policy based management framework for sensor networks. There is also a good amount of work in progress in the sensor network programming community to define a formal programming model [16, 5] that provides higher-level programming abstractions for WSNs [17]. The pre-requisites for such a technique to be effective are (1) target virtual machines consenting upon a consistent standard for the input file (2) run-time support on each sensor node in their native languages. Where possible, such an approach should make the most of the distinct features each programming model offers.

B. Mobility Management/Support Layer

The mobility management layer is strictly oriented to address the dynamic and mobile nature of Wireless Sensor Networks. As opposed to infrastructure-based wired distributed systems where network routing and transmission packet management activities are taken care by dedicated intermediate routers, in wireless sensor networks and wireless ad hoc networks each node acts as a router. Hence the mobility management layer provides support for routing and communication protocols, transmission packet management and network security components etc. It provides transparency by providing continuous connectivity to the mobile participating nodes without hard-coding dependencies on the aforementioned attributes and thus shielding heterogeneity from the layers above.

The role of the mobility management support layer is twofold:

1) Basic Network Support: the mobility management layer shields applications from heterogeneous key distribution, authentication, routing & communication protocols. On one hand this provides unified communication over distinct communication standards e.g. Zigbee, 6LoWPAN or other IEEE 802.15.4-based protocols. Furthermore, it provides network support for packets being transmitted between network domains that use different routing protocols. Figure 5 zooms further into the idea of heterogeneity from Figure 1&2 where we chose different routing protocols for each class of sensor nodes.

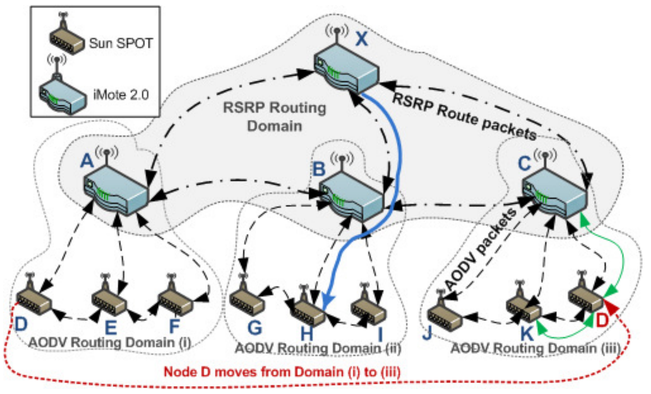
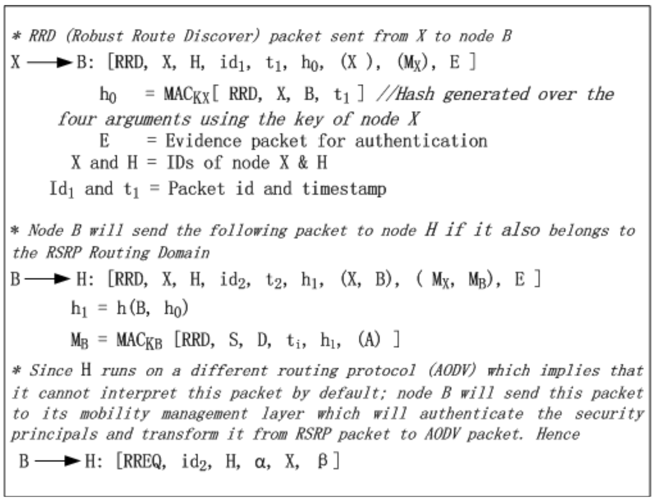


Figure 5. Sensor Network topology demonstrating of heterogeneous hardware, routing protocols, key distribution and data aggregation

Each of those routing protocols happen to have different packet formats, key structures and authentication schemes and it is desirable to keep these unwanted details hidden from application developers & end-users accessing the network from outside. Hence at the heart of the mobility management layer are protocol translators that allow nodes to interoperate across networks regardless of the communication protocols and interconnects. Figure 6 provides an insight on how these modules conceal these details.



** RRD (Robust Route Discover) packet sent from X to node B*

$X \longrightarrow B$: [RRD, X, H, id_1, t_1, h_0, (X), (M_X), E]

h_0 = MAC_{KX}[RRD, X, B, t_1] *//Hash generated over the four arguments using the key of node X*

E = Evidence packet for authentication

X and H = IDs of node X & H

Id_1 and t_1 = Packet id and timestamp

** Node B will send the following packet to node H if it also belongs to the RSRP Routing Domain*

$B \longrightarrow H$: [RRD, X, H, id_2, t_2, h_1, (X, B), (M_X, M_B), E]

h_1 = h(B, h_0)

M_B = MAC_{KB} [RRD, S, D, t_i, h_1, (A)]

** Since* H *runs on a different routing protocol (AODV) which implies that it cannot interpret this packet by default; node B will send this packet to its mobility management layer which will authenticate the security principals and transform it from RSRP packet to AODV packet. Hence*

$B \longrightarrow H$: [RREQ, id_2, H, α, X, β]

Figure 6. Route packet transformation at Mobility Management Middleware for packet destined from X to B2

2) Mobility Management: This layer also assists node mobility e.g. by reinitiating location and route discovery or acting as a key establishment agent etc. Efficient key handshake in dynamic and mobile networks is a challenging concern that needs to be handled by this layer. Going back to Figure 5, we can note that all nodes in each routing sub-domain have a secret key shared with each other. Now if for instance node D moves from domain (i) to (iii), it needs to initiate handshake with the nodes at the new routing domain as shown by the green arrows. For this reason the mobility management layer also implements functionality for new keys to be set up.

C. *Extended Network Services Layer*

This layer augments the mobility management middleware by defining higher level in-network services such as data aggregation, localization and filtering to achieve efficiency, optimization and quality of data. A principal cause of energy use in a sensor networks is data transmission from multiple sensor nodes, many of which may report the same information. Hence, sensor nodes may implement data-centric forwarding techniques to reduce unnecessary data transmission. Reasons for data transmission removal include such factors as duplication, out-of-range, or errors in data. Consequently, the extended network layer provides (1) transparent use of in-network algorithms using a unified communication model provided by the Mobility Management/Support Layer and (2) easy to use information collection mechanisms to the applications running on top.

For instance, aggregation is an important technique which fulfills the function of efficient data collection. The idea is to combine and redirect the data coming from different sources, eliminating redundancy, minimizing the number of transmissions and thus saving energy. However the network in our Figure 5 may contain a subset of information critical nodes for which it is desirable to present the raw data directly to the next tier. Furthermore, it could be required to use different aggregation schemes at different parts of the networks e.g. AODV routing domain *(i)* & *(iii)* uses some clustering based efficient aggregation mechanism in contrast to some semantic based approach used by the routing domain *(ii)*. Hence this layer provides (1) the transparency of extended in-network mechanisms to the applications and end users and (2) flexible platform to support data management mechanisms to maximize efficiency of the constrained resources.

D. *Domain-specific Layer*

This layer is commissioned to address the requirements of particular domains such as health care, environmental monitoring, aerospace, surveillance etc. In the example of DEUS health monitoring system, it is desired to have an integrated set of domain specific middleware services for medical diagnosis tasks and applications such as cardiac systems, patient monitoring systems and life support systems etc. Hence this layer provides applications with high-level domain-centric abstractions and services resulting in easily accessible remote sensor services.

VI. Conclusion and Future Work

Middleware for sensor networks is an emerging and promising research area. Most of the current research on sensor middleware focuses on developing individual algorithms and components for data aggregation, localization, service discovery, synchronization, etc. These projects, however, often lack attention for integrating these algorithms and components into a generic middleware architecture, and for helping application developers to compose a system that matches their requirements. In this paper we attempted to pursue this rationale by presenting a middleware architecture better suited to integrate WSN components and technologies. Our framework is composed of four layers positioned between applications and platforms spanning all the sensor hardware, components and technologies. In this phase of our work, we have proposed and illustrated the generic framework based on scenarios but without any structural evaluations. Keeping the breadth of the domain in mind, we will proceed with an individual layer's perspective, with our first goal to further develop and evaluate mechanisms for mobility/management support layer.

Acknowledgment

Syed Rehan Afzal is a Ph.D. fellow at IBBT-Distrinet, Katholieke Universiteit Leuven, 3001 Leuven, Belgium. This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund K.U. Leuven.

References

[1] P. Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, and David Culler. Tinyos: An operating system for wireless sensor networks. In W. Weber, J. Rabaey, and E. Aarts, editors, Ambient Intelligence. Springer-verlag, 2004.

[2] A. Dunkels, B. Groenvall, and T. Voigt. "Contiki - a lightweight and flexible operating system for tiny networked sensors., in Proceedings of the First IEEE Workshop on Embedded Networked Sensors. Florida. USA, Nov. 2004.

[3] IBBT-DEUS (Deployment and Ease of Use of Sensor Services) URL: www.ibbt.be/en/project/deus-0 (accessed in August 2009).

[4] D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. In Computer Communications Review. Proceedings of SIGCOMM '96, Aug. 1996.

[5] Luís Lopes, Miguel S. Silva, and João Barros. Robust Programming for Sensor Networks. Technical Report DCC-2008-01, Department of Computer Science, Faculty of Sciences, University of Porto, 2008

[6] Ryo Sugihara and Rajesh K. Gupta. Programming models for sensor networks:A survey. ACM Transactions on Senor Networks, 4(2):1–29, 2008. 42, 43, 44,45.

[7] I.F. Akyildiz, T. Melodia and K.R. Chowdhury, A survey on wireless multimedia sensor networks, Computer Networks Elsevier 51 (2007), pp. 921–960.

[8] SunSPOT World Home. URL: www.sunspotworld.com/ (accessed in August 2009).

[9] K. Römer. Programming Paradigms and Middleware for Sensor Networks, GI/ITG Workshop on Sensor Networks, pp. 49-54, Karlsruhe, Germany, February 2004

[10] Adler R, Flanigan M, Huang J, Kling R, Kushalnagar N, Nachman L, Wan CY, Yarvis MD. Intel mote 2: an advanced platform for demanding sensor network applications. In SenSys. 2005; 298

[11] J. Polastre, R. Szewczyk, and D. Culler. Telos:Enabling ultra-low power wireless research. In Proc. IPSN/SPOTS'05, Los Angeles, CA, USA, April 2005.

[12] Doug Simon, Cristina Cifuentes, Dave Cleal, John Daniels and Derek White. Java™ on the bare metal of wireless sensor devices: the squawk Java virtual machine. Second International Conference on Virtual Execution Environments, VEE'06, Ottawa, Canada, June 14-16, 2006.

[13] S. Das, C.E. Perkins and E. M. Royer. Ad hoc On Demand Distance Vector(AODV) Routing. Mobile Ad-hoc Network (MANET) Working Group, IETF, January 2002.

[14] Syed Rehan Afzal, Subir Biswas, Jong-bin Koh, Taqi Raza, Gunhee Lee, and Dong-kyoo Kim. RSRP: A Robust Secure Routing Protocol for Mobile Ad hoc Networks. Wireless Communications and Networking Conference, WCNC 2008

[15] R.E. Schantz and D.C. Schmidt, "Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications," Encyclopedia of Software Eng., Wiley & Sons, New York, 2001

[16] L. Lopes, F. Martins, M. S. Silva, and J. Barros. A Formal Model for Programming Wireless Sensor Networks. In International Conference on Distributed Computing in Sensor Systems, DCOSS'07, Springer-Verlag, 2007.

[17] R. Newton and M. Welsh. Region Streams: Functional Macroprogramming for Sensor Networks. In DMSN Workshop, 2004.

[18] N. Matthys and W. Joosen, Towards policy-based management of sensor networks. In the third international workshop on Middleware for Sensor Networks (MidSens'08)

[19] 802.15.4 vs ZigBee. URL:www.sensor-networks.org/index.php?page=0823123150. (accessed in August 2009).

[20] ZigBee Alliance. Zigbee specication. Technical Report Document 053474r06, ZigBee Alliance, June 2005.

[21] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, IETF, Sept. 2007.

[22] S. Hadim and N. Mohamed, Middleware challenges and approaches for wireless sensor networks, IEEE Distributed Systems Online 7 (3) (2006), pp. 853–865.

[23] Sensorpedia. URL: http://www.sensorpedia.org/. (accessed in August 2009)

[24] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: An acquisitional query processing system for sensor networks. ACM Transactions on Database Systems, 30(1):122–173, 2005.

[25] P. Levis and D. Culler, "Mate: A Tiny Virtual Machine for Sensor Networks". In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X), October 2002.

[26] C.-L. Fok, G.-C. Roman, and C. Lu, "Agilla: A Mobile Agent Middleware for Sensor Networks, "International Conference on Information Processing in Sensor Networks (IPSN'05), Los Angeles, CA, April 2005.

[27] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. L. Murphy, and G. P. Picco. TinyLIME: Bridging mobile and sensor networks through middleware. In 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom), pages 61–72. IEEE Computer Society, March 2005.

[28] A. L. Murphy, G. P. Picco, and G. C. Roman. "Lime: A Middleware for Physical and Logical Mobility," 21st IEEE International Conference on Distributed Computing Systems (ICDCS '01), April, 2001, pp. 524-533.

[29] P. J. Marrón, D. Minder, A. Lachenmann, and K. Rothermel. "TinyCubus: An Adaptive Cross-Layer Framework for Sensor Networks," it - Information Technology, vol. 47(2), 2005, pp. 87-97.