Towards A General Purpose Middleware Model for WSNs: A Literature Survey

Basem Y. Alkazemi
College of Computer and Information System
Umm Al-Qura University
Makkah, Kingdom of Saudi Arabia

Atif Naseer
Science and Technology Unit
Umm Al-Qura University
Makkah, Kingdom of Saudi Arabia
Email: anahmed [AT] uqu.edu.sa

Emad A. Felemban
College of Computer and Information System
Umm Al-Qura University
Makkah, Kingdom of Saudi Arabia

Abstract— **The applications of Wireless Sensor Networks (WSN) have recently become the core element in many industrial business models. The literature reports several aspects about the deployment and associated implementation protocols of WSN. However, conceptual taxonomization of the different elements of a WSN system seems lacking in the literature especially in terms of middleware models. As a result, this paper based on our survey work of middleware models available in the literature in order to serve as a base for the investigation of a general purpose middleware model that suits different business needs and network architectures. The survey is carried out by considering some key architectural characteristics defined in a prior work that have been identified to cause potential mismatches between business applications and hertogeneous WSNs. Our analysis is derived by the notion of middleware models and component models adopted by the software engineering community as we believe their defined practices can benefit the WSNs community to standardize their development activities.**

Keywords-component; Wireless Sensor Network, Middleware, Software Architecture

I. INTRODUCTION

In the past few years, wireless sensor networks (WSN) have played an important role in many industrial applications for monitoring different phenomena to fulfill certain business requirements. Conceptually, a wireless sensor network is a special type of adhoc network that utilizes sensors for data collection and transmission. Sensor nodes are small devices with very limited resources (i.e. storage and power) that can be deployed to perform any sort of data collection and aggregation. Every node in a network must comply with a set of pre-defined communication protocols and configurations to manipulate their lifecycle states and operation mode as per their corresponding business application requirements. A node, in a software engineering terminology is analogous to a component [17]. Every component model (e.g. COM [18], EJB [18]) defines a set of architectural characteristics that distinguishes it from other types of models. The selected component model must comply with the architectural characteristics required by a system in order to work as required. For instance a software system that expects to manipulate EJB components cannot deal with COM components. The same principles can apply to nodes in WSNs.

If a business application uses Java Messaging Service (JMS) [16] to interact with a network then all nodes within that network must implement the corresponding interfaces for JMS communication protocol and configuration settings. Although this is commonly recognized practice among the WSN community, the underlying network cannot be reused by any business applications those define different standards other than their deployer. There might be cases where heterogeneous nodes (i.e. nodes with different characteristics than what is available in a network) is used in a network. Although all nodes might follow similar communication protocols, there might be differences in terms of their states and configuration patterns. For example, if business applications invoke a method called "public void sleep ()" to switch the state of a node from data collection to sleep mode then a node with different method signature will not be managed. Considering that in mind, the notion of middleware can be utilized to tackle incompatibility between business application and the nodes of the underlying network.

In addition, nodes in WSNs usually have very limited power and storage capabilities. It may not be feasible to overload nodes with operations other than its basic functionality and expect a long lifetime of nodes. Thus, some operations, other than data collection and transmission, might be delegated to middleware components that have more resources and capabilities. In that sense, a middleware model must be general enough to deal with all sorts of interactions between business applications and WSNs. Therefore, this work aims at reviewing the current middleware models in the literature and examines their characteristics.

The rest of the paper is organized as follows: section 2 highlights the system definition for WSNs, section 3 describes some architectural characteristics of middleware; the related work is discussed in Section 4; we analyze some middleware in section 5; and finally, in section 6 we give concluding remarks and future work.

II. A SYSTEM MODEL FOR WSNS

A system model in the context of WSNs can be given as a composition of three main layers, namely:

- Business application layer
- Middleware layer
- Network layer

Figure 1 illustrates the overall architecture of a system model. The Business application layer in the system represents the main client who is interested in obtaining some data (e.g. warehouse monitoring) to initiate possible actions or feed into some sort of decision support systems. The middleware layer is a special type of components that separates business applications from the underlying networks. Finally, the network layer is the physical composite of nodes that can be deployed in a field for data collection.

Business applications can interact with WSNs either directly or indirectly through the middleware layer. The middleware layer hides the complexity of the underlying network from business application; and also resolves potential mismatches in the communication protocols between business applications and the WSNs. Thus, users can request data without much knowledge about the technical communication details of the targeted WSN. Middleware can also manage the configuration of the network topology to satisfy certain business requirements.

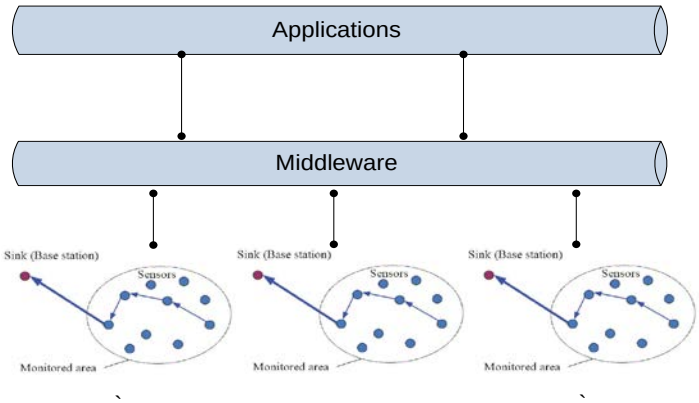

Figure 1. Wireless Sensor Network System

III. MIDDLEWARE CHARACTERISTICS

The concept of middleware is used differently among the WSN community. TinyOS [14] considers middleware as an internal layer within a WSN node that provides an abstraction to different physical components. It requires a number of APIs and drivers to control and manage the hardware elements.

Other works refer to middleware as a layer between the business application and the physical network. We adopted the latter view in this study and investigated the different work reported in the literature that fit into that context.

Consider a case where business application may not able to retrieve a required data from certain WSN due to incompatibilities between the business application and the underlying network of nodes in terms of communication protocols. Such a mismatch might negatively impact the operation of the overall system. Therefore, middleware can be utilized to facilitate the communication without any potential mismatches. We distinguish between two main types of characteristics of a system, functional and architectural. The functional characteristics describe the behavior of nodes in a network. For example, sensing through the transducer and transmitting data to other nodes or sink.

The architectural characteristics describe how to get at the functional characteristics. For example, how a node can start the sensing operation. In other words, what interfaces a business application needs to invoke so a node can start sensing. This classification is important in order to separate between the concerns that are relevant to a node and those that should be addressed by middleware. In this work, we are concerned only with the architectural characteristics as we believe they represent the core business of a middleware component. The survey conducted in this work aims at investigating the applicability of the reported middleware models in the literature to work as a general purpose component that effectively bridges the gap between business applications and WSNs in terms of interoperability and flexibility. So, business applications can interact with various types of networks regardless of being heterogeneous in nature. We have identified, in a prior work [13], a number of characteristics that are significant to be fulfilled by middleware from the perspective of business requirements. We utilize these characteristics to conduct a thorough survey in this work. The architectural characteristics of a middleware model should address the following:

A. Interaction Pattern

This characteristic defines how a middleware component can interact with business application and also the underlying network of nodes. From a business application perspective, some of the commonly used communication protocols are JMS and RPC. However, these protocols do not necessarily apply to all third party networks. Hence, instead of modifying a business application in order to comply with the protocol of a given network, middleware component can resolve such incompatibilities between business applications and networks. In addition, it checks user privileges to identify the type of information that must be delivered to business applications as per a predefined policy. For instance, one user might only get data about room temperature while another gets the complete set of data about the usages of that room. From the network perspective, the middleware model should define a standard messaging protocol and node configurations that different node types must comply in order to interact seamlessly with the middleware. So, middleware can query different networks to obtain data and synthesize them as appropriate for business applications.

B. Data exchange model

Usually, two types of data exchange model are defined by the WSN community; these are commonly known in the context of data streams as pull model and push model [15]. The pull model operates by receiving requests from the clients of business application and then responds by delivering the required data, whereas the push model sends the data to

registered set of clients as per the occurrence of an event. There are two subsidiary types of this model; interval based dispatching and threshold based dispatching. The former sends data to clients according to scheduled intervals (e.g. every 10 minutes). The latter sends data only once an interesting event has occurred that triggers a threshold value (e.g. If temperature exceeds 50c then send data). These two models, especially the push model, can be embedded in a middleware model in order to save storage space and power consumption of nodes. So, clients can register themselves in a middleware and define their threshold values in order to receive interesting data about an incident.

C. *Software as a Service*

Scalability is a desirable characteristic in any system to accommodate potential new business expansion. Therefore, the notion of services is established by the software engineering community to increase the flexibility and scalability of a system. So, services can be orchestrated and combined together at runtime instead of being hardcoded together at compile time. This capability can exhibit a considerable level of flexibility to fulfill continually changing business requirements. Thus, adopting the notion of software as a service (SaaS) [19] in a middleware model becomes an essence due to the wide variety of business client types and requirements. If a new type of parameters (e.g. QoS) needs to be considered by a client, then a new service can be deployed into a middleware component for client usage.

D. *Proxy pattern*

The notion of proxy pattern has been introduced in the software engineering community to resolve the problem of coupling between clients and servers. This pattern can be adopted by middleware model as it provides a similar sort of functionality of separating business applications from the underlying networks of nodes. A key advantage of this pattern can be obtained by means of deploying new modification to a proxy without the need to interrupt the node's operation. Moreover, middleware can interpret requests coming from business applications and send them in a specific format to a network and vice versa.

E. *Dynamic binding of Services*

Business applications should be able to manage and control the configuration of their corresponding networks. For example, if a business application requires fast response time from a network according to a predefined policy during peak hours only then a QoS service must be bound to the composite of services in order to satisfy this requirement. Moreover, if new types of services are introduced for business application usages then the binding mechanism of services only needs to be modified without affecting neither the application layer nor the network layer. However, dynamic binding of services requires high flexibility of service composition within a middleware. Thus, a middleware model should define a pool of services and also mechanisms of binding them together to comply with a named business application requirements without major impact on their clients.

F. *Parallel Processing*

One considerable problem might be encountered by business applications in terms of sharing nodes with multiple clients. For example, if the owner of a network has requested a node 'X' to switch its state into sleep mode in order to preserve the battery life. However, another application, which uses same communication protocols and frequency, might request data to be collected by the same node 'X' and requests its state to be switched into data collecting mood. This kind of overlap in state management might negatively impact the application of the network owner as they might find the node 'X' dead afterwards when they need to bring it to working mode again due to battery ran out. Thus, a middleware model can be utilized to control such ownership and coordinate interaction as per predefined policies given by network owners. A middleware should implement a means of queuing mechanism to buffer all coming requests and then executes them according to a pre-defined policy if available.

These characteristics represent the baseline for conducting our survey work of middleware models. We are going to investigate whether the available models satisfy, either fully or partially, these characteristics in order to evaluate their usefulness to serve as a general purpose middleware model.

IV. MIDDLEWARE SURVEY

Hermann et al. presents a middleware for wireless sensor networks called Senceive [1]. It also separates the network layer from the application layer. This model provides a GUI for network administrator to manage and configure the topology of the WSN. Senceive satisfies different requirements like Sensor Data Quality is increased by calculating noise level, zero crossing rate or signal energy, 2ndly it enables users to query each single sensor separately and collecting data using a well-defined interface. 3rdly a separate interface is also available for administrators to configure the whole network as well as individual nodes. And at last it gathers Meta information about the network infrastructure. Senceive is designed as a basic three tiered architecture by Crossbows Mote Works. The mote tier (denoted as WSN) provides a communication. The server tier provides a central control instance of the network and the client tier finally has one administrator for the network configuration and multiple applications for parallel data gathering. The implementation of this network is also based on TinyOS 2.x. Senceive is a good addition in WSN middleware. Sensitive is commonly used in motion modeling in the context of home security. It has a query interface to interact with business application; this interface processes queries from multiple applications, and provides the applications with expressive and adequate SQL-like query language. Senceive supports Java Remote Method Invocation.

It provides the interaction with the user to collect information from sensor networks, but it still focuses on the same type of network configuration and assessment.

Jeon et al. [2] proposed a sensor node middleware to support web-based application over a Wireless Sensor Network (WSN). This middleware provides a web based interface to sensor nodes. The system proposed in this paper consists of three main parts: a gateway per sensor node, a data server that includes a Web Application Server (WAS), and application services. This system has different interfaces like the WSN interface gateway connect the client and the sensor node, here the gateway is an interface between WAS and WSN. This system supports various application services. This proposed middleware supports only some of the functional and architectural characteristics of the middleware like to manage nodes and dynamic binding of services. This middleware is currently used in the context of analyzing the effectiveness of mobile nodes in a GSM network and also the intercommunication between different nodes in an ad-hoc manner. The gateway transmits WSN sensor data to WAS through HTTP. It uses Java messaging Service (JMS) to transfer the data to the application. The gateway can transmit XML Documents and send hexadecimal data. It also sends HTTP request to nodes.

Heinzelman et al. [3] proposed application driven approach middleware called MiLAN. The application driven approach allows the programmers to tune the network according to the application need. Milan [3] uses a specialized graph to receive requirements for application. Through these graphs, Milan gets the application variables and the required QoS. On the basis of these graphs MiLAN can determine which sets of sensors satisfy all of the application QoS requirements. MiLAN supports scalability with its application driven approach and QoS issues, but it doesn't address mobility and lacks support for OS and hardware heterogeneity. Also MiLAN lacks the architectural characteristics of the middleware. Environmental Surveillance Home/Office Security, Medical Monitoring is the main business cases of Milan. The application uses an API to represent its requirement with regards to different sensors available. MiLAN uses a service discovery protocol (such as SDP or SLP) to find new nodes and learn when nodes are no longer accessible.

Based on Java language Li et al. [4] develops application event oriented WSNs middleware software. This system provides a standard interface and protocol for middleware. Also it provides compatibility, generality and operability. This middleware composed of following modules i.e. management module, event processing management module, event of information space, QoS processing module, and middleware interface. The main business cases are environment monitoring, health applications and military applications. This middleware uses web services, HTTP and JMS to communicate with business applications. Soap (Simple Object Access Protocol) technology is mainly applied in the interoperability between much heterogeneous program and the platform. This middleware software is a good addition as it provides an interface for communication but it lacks some standard architecture. Implementation of this software is also not available to formalize the results.

Padmanabh et al.[5] presents a middleware named MOJO that converts the actual sensor network node into a virtual network as a JAVA objects so that user can easily work with network and can easily deploy any application without knowing the complexity of Wireless sensor network. MOJO only works for similar type of network node and can only convert the same nodes into JAVA objects. These JAVA objects are connected to the outside world via a multiplicity of application programmable interfaces (APIs). The main business case for this middleware is a Conference room management system. This middleware exposed the APIs through RMI and web service (AXIS by Apache).

Aoki et al. [6] proposes Spinning sensors middleware that works for robotic sensor nodes. The middleware proposed in this paper provides the functionality of device coordination, data processing and management of spatiotemporal model of robotics sensor nodes. Spinning sensor middleware supports three kinds of applications i.e. environment monitoring, sensor controlled robot and context-aware service. Environment monitoring, radio control robot, and context-aware services are the main business case used by this middleware. The application uses Spinning sensor API to send their request. All the communication among nodes is conducted by using event driven architecture. This middleware only supports the robotics sensors data and is not providing a framework to use this middleware in any other wireless sensor network.

FOK et.al. [7] presents a mobile agent middleware for Self-Adaptive Wireless Sensor Networks called Agilla. It's an agent based middleware provides a programming model consists of different agents that shares a wireless sensor network. Agents are able to enter and exit a network and are working according to environmental conditions. These mobile agents will allow the application to adapt the changing on the requirements. This model is not meant for data collection applications The architecture of agile is composed of an engine which is a virtual machine kernel that controls the middleware, underlying network and agents layers. Agilla provides the scalability, security and the capability for applications to self-heal. Agilla provides a model that consists of agents but is unable to provide a framework for middleware. Agilla middleware only supports the model proposed by Agilla and is only agent based system.

TinyDB [8] is a query based processing middleware build on top of TinOS [9] operating system. In a sensor network TinyDB runs on each of the nodes. Every device in a network contains a sensor table to produce and store their readings. TinyDB provides power-efficient in network query processing system for collecting data from sensor nodes. Also TinyDB

supports multiple queries running simultaneously and is able to prioritized Data Delivery. TinyDB provides a strong query based mechanism for wireless sensor network but it does not provide much functionality as part of a middleware service. Marrh et.al [10] presents a framework for middleware called TinyCubus for TinyOS based sensor networks. TinyCubus consists of three parts: the Tiny Data Management Framework, the Tiny Cross-Layer Framework, the Tiny Configuration Engine and Tiny Data Management. TinyCubus flexible architecture allow it to be used in different types of application but its not provide a framework that fixes all the issues of middleware and supports different types of hardware architecture nodes. Murphy et al. [11] bridges the gap between sensor network and applications through a middleware called TinyLime. TinyLime is an extension of Lime middleware [12]. This paper contributes on two issues, proposing a new operational setting for sensor network applications and a middleware that support this development. TineLime is also implemented on top of TinyOS platform. In the previous schemes data are collected centrally by a central node and application should communicate to that node to collect any information but in the proposed architecture data is collected by mobile monitors interconnected through a MANET, which can access only those sensors that are directly available to them. TinyLime has three main components i.e. Lime integration component, mote interface and mote level subsystem.

V. Discussion

In the conducted survey, we have explored many middleware architectures available for wireless sensor networks with their separate merits and demerits. We try to analyze these middleware according to different business cases they are using also how they interact with different application and underlying networks. Table 1 shows the analytical study of each middleware according to the architectural characteristics we identified earlier. We observed that all the surveyed models employ sophisticated interaction patterns and data exchanging model. It is obvious that these two characteristics represent the key attributes in all middleware models even though they vary in their implementation details. However the surveyed middleware varies with respect to the other architectural characteristics we identified due to being designed to serve specific application domain only regardless of potential variety of business requirements.

In terms of flexibility, it has been observed that a considerable number of models employ software as a service (SaaS) as part of their model building block. However, they only expose services externally to applications but cannot be composed internally within the middleware itself to satisfy different application domains. Another characteristic of importance when dealing with distributed system is the parallel processing capability. Surprisingly, we observed that some of the surveyed models can only process one request at a time and cannot support multi-threading. We believe that such a deficiency contradicts with the nature of distributed environment where services need to handle multiple types of requests independently. Dynamic binding characteristic is not addressed by many middleware models as most of them are built on hardcoded components that cannot be manipulated or re-structured at runtime.

The proxy design pattern is implemented in many middleware models with sophisticated capability to separate between interfacing and implementation components. Other models that lack this characteristic (e.g. MOJO) assume that middleware is a tightly coupled set of components without separating their interfaces from the underlying computational components. Their main concern is to satisfy the functional requirements only regardless of architectural considerations. Based on the analysis we conducted in this paper, we strongly believe that a general purpose middleware model is required that can effectively bridge the gap between application domains and the underlying network of nodes. So, different application domains can interact seamlessly with different networks regardless of any potential incompatibility in their architectural characteristics. We believe a service oriented architecture (SOA) based model will overcome many of the limitations observed in the current models, and this is going to be part of our planned future work.

VI. Conclusion ad Future Work

Middleware plays an important role in WSN. This paper discusses the architectural characteristics of middleware and reviews some of the middleware already available in the literature. The paper also analyzes these middlewares according to their architectural characteristics to find out the similarities and differences among their architectures. All of these middlewares are compared and presented in a tabular format; Table 1 provides a summary of these middlewares.
The next step is to propose a general purpose service-oriented architecture (SOA) based model for middleware. This proposed framework is able to provide services to sensor networks and dynamically handles and build all types of applications. This architecture will provides standard layers of services to different types of nodes from different vendors, hence to communicate with different types of nodes in a single network is achievable.

Acknowledgements

This work is funded by grant number 11-INF1674-10 from the Long-Term National Plan for Science, Technology and Innovation (LT-NPSTI), the King Abdul-Aziz City for Science and Technology (KACST), Kingdom of Saudi Arabia. We thank the Science and Technology Unit at Umm A-Qura University for their continued logistics support.

References

[1] C. Hermann and J.Dargie.Senceive: A Middleware for a Wireless Sensor In Advanced Information Networking and Applications, pages 612 - 619, Gino-wan, Okinawa, Japan, 2008, IEEE.

[2] Y.J. Jeon, S.H. Park, J.S. Park. Sensor Node Middleware to Support Web-Based Applications over Wireless Sensor Networks, The 2nd IEEE Workshop on Wireless and Internet Services (WISe 2009) Zrich, Switzerland; 20-23 October 2009.

[3] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo, Middleware to Support Sensor Network Applications, IEEE Network, 2004.

[4] Qingfeng Li, Chen Li, Jifang Shi. Research and Application on Application-oriented Event-Based Middleware of WSNs, In 2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing, pages 498- 501, Wuhan, Hubei 24-25 April 2010.

[5] Kumar Padmanabh, LakshyaMalhotra, AdiMallikarjun Reddy V, Amrit Kumar, Sunil Kumar V and Sanjoy Paul, MOJO: A Middleware that Converts Sensor Nodesinto Java Objects, In 2010 Proceedings of 19th International Conference on Computer Communications and Networks, pages 1-6, Zurich, Switzerland; 2-5 Aug. 2010.Schmitt, J.B; Roedig, U., "Worst case dimensioning of wireless sensor networks under uncertain topologies", In IEEE Proceedings of the 1st workshop on Resource Allocation in Wireless Networks, (2005)

[6] Soko Aoki, Jin Nakazawa and Hideyuki Tokuda, Spinning Sensors: A Middleware for Robotic Sensor Nodes with Spatiotemporal Models, In The 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pages 89 98, 25-27 Aug. 2008, Kaohsiung.

[7] Fok, C.-L., Roman, G.-C., and Lu, C. 2009. "Agilla: A mobile agent middleware for self-adaptive wireless sensor networks". ACM Trans Autonom. Adapt. Syst. 4, 3, Article 16 (July 2009), 26 pages. DOI = 10.1145/1552297.1552299 http://doi.acm.org/10.1145/1552297.1552299.

[8] 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):122173, 2005.

[9] 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.

[10] P. J. Marrn, 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.

[11] 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 6172. IEEE Computer Society, March 2005.

[12] 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.

[13] Alkazemi, B.Y, Felemban, E.A, Abid, A.Z., Al-Zahrani, F.A. "Middleware model for Wireless Sensor Networks", International Conference on Multimedia Computing and Systems (ICMCS), 10-12 May, 2012, Tangier-Morocco.

[14] HILL, J., SZEWCZYK, R., WOO, A., HOLLAR, S., CULLER, D., AND PISTER, K. 2000. System architecture directions for networked sensors. SIGPLAN Not. 35, 11, 93104.

[15] M., Duller, R., Tamosevicius, G., Alonso, and D., Kossmann, XTream: Personal Data Streams. In Proceedings of the ACM International Conference on Management of Data (SIGMOD), ACM, 2007.

[16] http://www.afceurope.com/documents/JMS.pdf [Access: 11th June 2013].

[17] Dogru, A.H.; Tanik, M.M., "A process model for component-oriented software engineering," Software, IEEE , vol.20, no.2, pp.34,41, Mar/Apr 2003 doi: 10.1109/MS.2003.1184164.

[18] Emmerich, W.; Kaveh, N., "Component technologies: Java beans, COM, CORBA, RMI, EJB and the CORBA component model," Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on , vol., no., pp.691,692, 25-25 May 2002

[19] http://en.wikipedia.org/wiki/Software_as_a_service [Access: 24 August 2013].

TABLE I. ARCHITECTURAL CHARACTERISTICS OF VARIOUS DISCUSSED MIDDLEWARE IN LITERATURE REVIEW

Author(s)	**Middleware**	**Summary**	**Identified Problem**	**Architectural Characteristics**					
				Interaction Pattern	Data exchanging model	Software as a Service	Proxy pattern	Dynamic binding of Services	Parallel Processing
Hermann et al. [1]	Senceive	This middleware support multiple applications through lightweight query language and query engine.	• Focuses on same type of network configuration and assessment. • Dynamically configure a network to carry some complex processing tasks within the network is also lacking in this proposed middleware.	✓	✓	✓	✓	✗	✓
Jeon et al. [2]	Sensor Node Middleware to Support Web-Based Applications over Wireless Sensor Networks	This middleware provides a web based interface to sensor nodes. It consists of three parts: a gateway per sensor node, a data server that includes a Web Application Server (WAS), and application services	• It supports only some of the functional and architectural characteristics of the middleware like manage nodes and dynamic binding of services.	✓	✓	✓	✓	✓	✗
Heinzelman et al. [3]	MiLAN	MiLAN is an Application driven approach. it allows programmers to tune the network according to the application need.	• MiLAN supports scalability with its application driven approach and QoS issues, but it doesn't address mobility and lacks support for OS and hardware heterogeneity. • MiLAN lack the architectural characteristics of the middleware.	✓	✓	✓	✓	✓	✗

Li et al. [4]	Application-oriented Event-Based Middleware of WSNs	This system provides a standard interface and protocol for middleware. Also it provides compatibility, generality and operability.	• Lacks some standard architecture and also not able to dynamically configure a new node. • Implementation of this software is also not available to formalize the results.	✓	✓	✗	✗	✗	✓
Padmanabh et al. [5]	MOJO	MOJO converts the actual sensor networks node into a virtual network as a JAVA objects so that user can easily work with network and can easily deploy any application without knowing the complexity of Wireless sensor network.	• MOJO only works for similar type of networks node and can only convert the same nodes into JAVA objects	✓	✓	✗	✗	✗	✓
Aoki et al. [6]	Spinning Sensors	Spinning Sensors provides the functionality of device coordination, data processing and management of spatiotemporal model of robotics sensor nodes.	• Only supports the robotics sensors data. • Not providing a framework to use this middleware in any other wireless sensor network.	✓	✓	✓	✗	✗	✓
FOK et.al. [7]	Agilla	It's provides a programming model consists of different agents that shares a wireless sensor network.	• It consists of agents but is unable to provide a framework for middleware.	✓	✓	✗	✗	✓	✓
Madden et al. [8]	TinyDB	TinyDB is a query based processing middleware build on tip of TinOS operating system. Every device in a network contains a	• Provides a strong query based mechanism for wireless sensor network but it does not provide much functionality as part of middleware service.	✓	✓	✗	✓	✓	✗

		sensor table to produce and store their readings.							
Marrh et al. [10]	TinyCubus	TinyCubus consists of three parts: the Tiny Data Management Framework, the Tiny Cross-Layer Framework, the Tiny Configuration Engine and Tiny Data Management.	• It not provides a framework that fixes all the issues of middleware and supports different types of hardware architecture nodes.	✓	✓	✓	✗	✓	✗
Murphy et al. [11]	TinyLime	In TinyLime data is collected by mobile monitors interconnected through a MANET, which can access only those sensors that are directly available to them.	• It only designs for a specific type of network. TinyLime is based on previously developed Lime model.	✓	✓	✗	✓	✗	✗