Wang MM, Cao JN, Li J *et al.* Middleware for wireless sensor networks: A survey. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 23(3): 305–326 May 2008

Middleware for Wireless Sensor Networks: A Survey

Miao-Miao Wang[1,2] (王苗苗), Jian-Nong Cao[2] (曹建农), Jing Li[1] (李　京), and Sajal K. Dasi[3]

[1] *Department of Computer Science, The University of Science and Technology of China, Hefei 230027, China*

[2] *Internets and Mobile Computing Lab, Department of Computing, The Hong Kong Polytechnic University, Kowloon Hong Kong, China*

[3] *Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, U.S.A.*

E-mail: {csmmwang, csjcao}@comp.polyu.edu.hk; lij@otcaix.iscas.ac.cn; das@uta.edu

Received January 15, 2008, revised March 4, 2008.

Abstract Wireless Sensor Networks (WSNs) have found more and more applications in a variety of pervasive computing environments. However, how to support the development, maintenance, deployment and execution of applications over WSNs remains to be a nontrivial and challenging task, mainly because of the gap between the high level requirements from pervasive computing applications and the underlying operation of WSNs. Middleware for WSN can help bridge the gap and remove impediments. In recent years, research has been carried out on WSN middleware from different aspects and for different purposes. In this paper, we provide a comprehensive review of the existing work on WSN middleware, seeking for a better understanding of the current issues and future directions in this field. We propose a reference framework to analyze the functionalities of WSN middleware in terms of the system abstractions and the services provided. We review the approaches and techniques for implementing the services. On the basis of the analysis and by using a feature tree, we provide taxonomy of the features of WSN middleware and their relationships, and use the taxonomy to classify and evaluate existing work. We also discuss open problems in this important area of research.

Keywords middleware, wireless sensor network, pervasive computing

1 Introduction

In recent years, a new wave of networks labeled Wireless Sensor Networks (WSNs) has attracted a lot of attentions from researchers in both academic and industrial communities. WSNs can be used for forming the underlying sensing and network infrastructure in pervasive computing environments. A WSN consists of a collection of sensor nodes and a sink node connected through wireless channels, and can be used for building distributed systems for data collection and processing, covering the functions of on-field signal sensing and processing, in-network data aggregation, and self-organized wireless communication. WSNs have found many applications in different areas, including environmental surveillance, intelligent building, health monitoring, intelligent transportations, etc.[1]

This survey paper is concerned with middleware for WSNs. Middleware refers to software and tools that can help hide the complexity and heterogeneity of the underlying hardware and network platforms, ease the management of system resources, and increase the predictability of application executions[2]. WSN middleware is a kind of middleware providing the desired services for sensing-based pervasive computing applications that make use of a wireless sensor network and the related embedded operating system or firmware of the sensor nodes.

The motivation behind the research on WSN middleware derives from the gap between the high-level requirements from pervasive computing applications and the complexity of the operations in the underlying WSNs. The application requirements include high flexibility, re-usability, and reliability. The complexity of the operations with a WSN is characterized by constrained resources, dynamic network topology, and low level embedded OS APIs. WSN middleware provides a potential solution to bridging the gap and removing the impediments. In the early time of the research on WSN, people did not pay much attention to middleware because the simplicity of the early applications did not show much demand for the support from the middleware. Along with the rapid evolution in this area, the gap becomes increasingly obvious and hinders the pop-

Survey

Supported by Hong Kong Polytechnic University under the ICRG Grant No. G-YE57, Hong Kong RGC under the Grant of A Research Center Ubiquitous Computing, and the National Hi-Tech Development 863 Program of China under Grant No. 2006AA01Z231.

ularity of WSN-based applications.

WSN middleware helps the programmer develop applications in several ways. First, it provides appropriate system abstractions, so that the application programmer can focus on the application logic without caring too much about the lower level implementation details. Second, it provides reusable code services, such as code update, and data services, such as data filtering, so that the application programmer can deploy and execute the application without being troubled with complex and tedious functions. Third, it helps the programmer in network infrastructure management and adaptation by providing efficient resource services, e.g., power management. It also supports system integration, monitoring, as well as system security.

Although middleware is a well established research area in distributed computing systems, WSN poses new challenges to middleware research. The traditional middleware techniques cannot be applied directly to WSNs. First, most distributed system middleware techniques[3,4] aim at providing transparency abstractions by hiding the context information, but WSN-based applications should usually be context-aware. Second, although many mobile computing middleware[5,6] supports context awareness, their major concern is how to continuously satisfy the interests of individual mobile nodes in the presence of mobility. In contrast, WSN-based systems are data centric, reflecting the whole application's interests. Thus, the locations and mobility of the sensor nodes should be handled by WSN middleware in a different way. For example, a node moving away from a phenomenon may choose to hand off the monitoring responsibility to a nearby node. Also, WSNs mostly use attribute-based addressing[1,7] rather than relying on network-wide unique node addresses. Third, data aggregation in intermediate nodes of the forwarding path is desirable in a WSN, but no such kind of support is provided in traditional distributed system middleware because of the end-to-end paradigm used. Finally, WSN requires the middleware to be light weight[7−9] for implementation in sensor nodes with limited processing and energy resources. WSNs also have new requirements on hardware (e.g., various sensors and computing nodes), operating systems and routing protocols[10], as well as the applications[11].

In recent years, a lot of work has been done on WSN middleware, focusing on different aspects and for different purposes. Although several survey papers can be found in literature[7−9,12−18], each of them addresses only a subset of the issues, e.g., the system architecture[7−9,16], the programming paradigm[13,14,17], and the run time supporting[12,18]. There is no reference model for classifying and analyzing the middleware functionalities, and no detailed discussion on the implementation techniques.

This paper presents a systematic study of recent research on WSN middleware to help identify the key services, challenging issues, and important techniques. Compared with the existing surveys, this paper makes the following distinct contributions. First, it proposes a reference model for analyzing the functionalities and key services of WSN-middleware. Second, it provides a detailed review of the existing work on the most important aspects in developing WSN middleware, covering the major approaches to and corresponding techniques of implementing the services. Third, the paper proposes a feature tree-based taxonomy[19] that organizes WSN-middleware features and their relationships into a framework to help understand and classify the existing work. The paper also discusses the open problems and identifies the directions in future research.

The remainder of this paper is organized as follows. In Section 2, we describe a reference framework to analyze the functionalities of and identify the key services to be provided by WSN middleware. In Section 3, we survey the approaches and the corresponding techniques for implementing the key services. In Sections 4 and 5, we look into the WSN runtime support and QoS support, respectively. In Section 6, we propose a feature tree-based taxonomy for classifying the features of WSN-middleware, and highlight the representative existing middleware projects with the evaluation of their advantages and disadvantages. In Section 7, we discuss the challenges, open problems, and future directions of WSN middleware research. Finally, we conclude this paper in Section 8.

2 Reference Model of WSN Middleware

2.1 Model Overview

As shown in Fig.1, a complete WSN-middleware solution should include four major components: programming abstractions, system services, runtime support, and QoS mechanisms. Programming abstractions define the interface of the middleware to the application programmer. System services provide implementations to achieve the abstractions. Runtime support serves as an extension of the embedded operating system to support the middleware services. QoS mechanisms define the QoS constrains of the system.

By analyzing the requirements of WSN-based applications and the characteristics of WSNs, we propose a reference framework, shown in Fig.2, to describe the or-

ganization and relationships of the above components. It should be mentioned that it is not necessary for a specific WSN-middleware to include all the components. Also, functions of several components may be combined together and implemented as one component.

In the deployment, the functions of WSN-middleware can be distributed to the sensor nodes, the sink nodes, and high level application terminals, as shown in Fig.3. The distributed middleware components located in different nodes of the network communicate with each other to achieve some common goals.

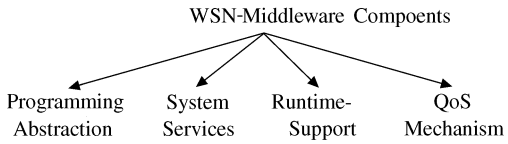


Fig.1. Major components of WSN-middleware.

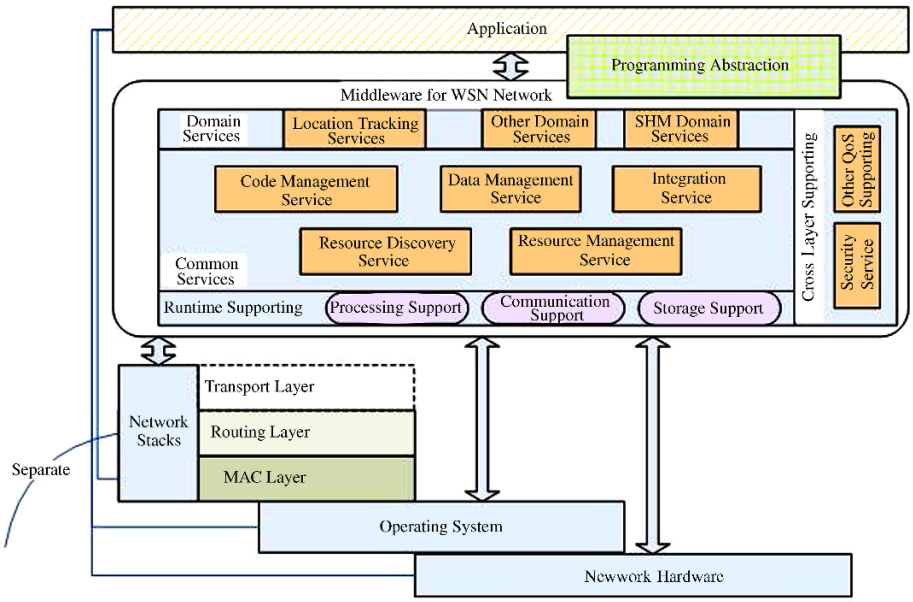


Fig.2. Reference model of WSN middleware.

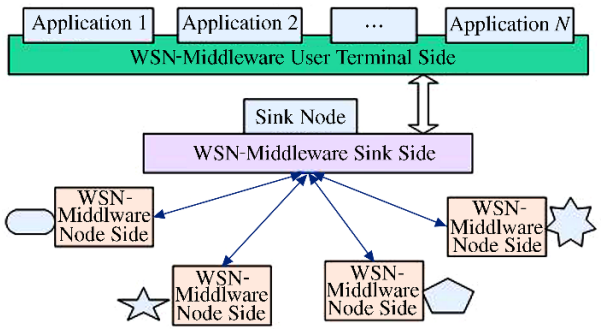


Fig.3. System architecture of WSN-middleware.

2.2 Programming Abstractions

Programming abstractions are the foundation of WSN-middleware. They provide the high-level programming interfaces to the application programmer, which separate the development of WSN-based applications from the operations in the underlying WSN infrastructures. They also provide the basis of developing the desirable middleware services. Three aspects are involved in developing the programming abstractions: *abstraction level, programming paradigm, and interface type.*

Abstraction Level refers to how the application programmer views the system. *Node level abstraction* abstracts the WSN as a distributed system consisting of a collection of sensor nodes, and provides the programmer the support for programming the individual sensor nodes for their actions and cooperation[20−22]. *System level abstraction* abstracts the WSN as a single virtual system and allows the programmer to express a single centralized program (global behavior) into subprograms that can execute on local nodes (nodal behavior), leaving only a small set of programming primitives for the programmer while making transparent the low-level concerns such as the distributed code generation, remote data access and management, and inter-node program flow coordination[23,24]. Generally speaking, node level abstraction facilitates the development of applications with more flexibility and energy saving, and less communication and interpretation overheads. On

the other hand, system level abstraction is easier to use because nodal behaviors can be generated automatically so the programmer can concentrate on the network-level actions, without worrying about how the sensor nodes collaborate with each other to perform the assigned tasks.

Programming paradigm refers to the model of programming the applications. It is often dependent on the applications. WSN applications can be classified in two dimensions: application data collection feature and application dynamic feature. Data collections can be continuous, event-driven, or query-based. Application can be totally static and has some mobility characteristic, such as mobile target or mobile sink. Correspondingly, for different applications, WSN middleware may use different programming paradigms, such as database, mobile agent, and Publish/Subscribe (Pub/Sub). For example, the data base paradigm is often used for query-based data collection, while the Pub/Sub paradigm can be a good choice for event-driven applications. Mobile agent paradigm may be a choice for tracking mobile target applications.

Interface type refers to the style of the programming interface. As a matter of fact, programming abstraction is embodied as the programming interface. Descriptive interfaces provide SQL-like languages for data query[25,26], rule-based declarative languages for command execution[27], or XML-based specification files for context configuration[28]. On the contrary, imperative interfaces provide imperative programming languages for writing the code to interact with the WSN network[29]. Descriptive interfaces usually require the interpretation of the queries and thus consume more resources, while imperative interfaces require the programmer to specify the logic of execution, and are more flexible but more difficult to use.

The consideration of adopting a particular abstraction level and selecting an appropriate programming paradigm and applicable interface depends on the specific application requirements and the underling WSN infrastructure. Middleware providing similar paradigms may share the implementation techniques. For example, the database-based paradigm is usually implemented with a descriptive interface, while the event-driven paradigm can be implemented either with an imperative interface by providing the handlers to be recalled or with a descriptive interface by providing an event description scheme. We will discuss them in more details in the later sections.

2.3 System Services

System services embody the functionalities and form the core of WSN-middleware. They are exposed to the application programmer through the abstraction interface, and provide the support for application deployment, execution, as well as sensor and network management. We classify the system services into two broad categories: *common services* and *domain services*.

Common services are the basic services shared by all WSN applications. They help manage the application information and the WSN infrastructure. The functionalities provided by the common services include:

- *Code Management*: responsible for code migrating and code updating in a deployed network;
- *Data Management*: responsible for data acquisition, data storage, data synchronization, data analysis, and data mining;
- *Resource Discovery*: responsible for discovering newly joined sensor nodes and detecting nodes, which are becoming inaccessible either as a result of mobility or loss of battery power;
- *Resource Management*: responsible for managing the node resources (e.g., energy, memory, A/D device, communication module) and network resource (e.g., topology, routing, system time);
- *Integration*: responsible for integrating WSN and its applications into other networks, such as the Internet and Grid, for broader use.

We will explain the implementation details of the above services in the next section.

Domain services facilitate the development of applications in a specific domain. They can make use of the common services and add application-oriented functions of providing domain specific services. For example, EnviroTrack[30] is a WSN middleware that supports environmental Target tracking. Impala[22] is a middleware for the ZetbraBet project, a wildlife monitoring project. It has two layers: the upper layer contains the application specific protocols and functions, and the lower layer contains the common services such as code management. WSN-SHM middleware[31,32] is designed for developing structural health monitoring applications which have the requirements of high frequency sampling and high resource consumption.

2.4 Runtime Support

Runtime support provides the underlying execution environment of applications and can be seen as an extension of the embedded operating system which provides functions of scheduling of tasks, inter-process communication (IPC), memory control, and power control in terms of voltage scaling and component activation and inactivation. The need for runtime support in

WSN middleware comes from the facts that the hardware and firmware of the sensor nodes may not always provide enough support for the implementation of the middleware services described above.

The functionalities of the runtime support in WSN middleware include local processing support, communication support, and storage support. More specifically, the support is provided for multi-thread processing, smart task scheduling, and synchronization of memory access.

Runtime support of WSN-middleware is always embodied as a virtual machine over a specific embedded operating system. An example is Mate[20] built on top of TinyOS[33].

2.5 QoS Mechanism

Quality of Service (QoS) mechanism is an advanced feature of WSN-middleware. Providing QoS support in WSN is still an open issue for research[34]. QoS features are always crossing layers and crossing components, and are embodied in various functional services. For example, the data management service is required to be reliable and of high accuracy.

Typical parameters for expressing QoS of WSN network infrastructure include message delay, jitter, and loss, network bandwidth, throughput, and latency. Typical parameters for expressing QoS of WSN applications include data accuracy, aggregation delay, coverage, and system lifetime. Middleware acts as a broker between the applications and the network infrastructure. QoS support may translate and control the QoS metrics between the application level and the network level[35]. If the QoS requirements from an application are not feasible to fulfill in the network, the middleware may negotiate a new QoS guarantee with both the application and the network. QoS support may also provide the implementation framework for simplifying the QoS-aware WSN application development using QoS assurance algorithms.

In the rest of the paper, we will focus on the approaches and techniques for implementing the above identified middleware components.

3 Middleware System Services

3.1 Code Management

In the context of this paper, a WSN application consists of pieces of code that executes on the sensor nodes. Code management provides services for code deployment, i.e., allocation and migration of code to sensor nodes. Code allocation determines a set of sensor nodes, on which the execution will be activated. Code migration transfers the code on a sensor node to another node[36]. It not only helps conveniently re-task the network for network reprogramming (code updating), but also enables the data computation elements of an application to be re-located. Code can migrate to the nodes close to the area where relatively large amounts of data are collected, enabling potentially high energy saving, or migrate with the mobile phenomena. For example, the code of an application for fire alarm can be migrated from node to node along the path of fire spread.

Generally speaking, implementation of code allocation involves with checking conditions using comparisons. In SINA[37] code allocation is implemented in a sensor execution environment (SEE), which compares SQTL script parameters with the attributes of sensor nodes and executes the script only if there is a match. In Cougar[26], code allocation is implemented by a query optimizer that determines the energy-efficient query routes[36]. Code allocation services implemented by a query optimizer have good expressivity but bring network load, while the SEE approach has limited expressivity but good scalability. Another promising approach, as used in MiLAN[38], is to apply application-level QoS to the control of the code allocation in configuration adaptation. The approach enables the adaptation of the application operations on the basis of the current application requirements, which can be adjusted depending on the output of the application itself. In this way the code allocation is adaptive to the changing conditions. However, the technique used in MiLAN requires a centralized control.

Code migration can be implemented at not only the middleware layer but also in the underlying embedded operating systems, as in BerthaOS[39] and MagnetOS[40]. However, because WSN OS does not support code interpretation, code migration implemented at the OS level is error prone and subject to malicious attacks[36].

At the middleware level, most techniques for task migration rely on the use of mobile code, moving the code to the data origins to process the data locally[36]. Current implementations include code migration through mobile code (e.g., TCL script in Sensorware[41], SQTL scripts in SINA[37]) and mobile Java object (e.g., TinyLime[42]). An example of mobile code is mobile agent, which is an execution thread encapsulating the code as well as the state and data. Mobile agent makes migration decisions autonomously. The key of this approach is to make the application as modular as possible to facilitate their injection and dis-

tribution through the network. However, the nature of mobile agent code does not allow hardware heterogeneity. So, this approach is implemented on top of a VM for platform independency.

There is a trade-off between the complexity of the interpreter running on the nodes and the complexity of mobile code. Code migration services implemented by mobile code with TCL and SQTL have the advantages of small size and high dynamicity, but suffer from the complexity in specification and high communication cost. Implementations based on mobile agent and mobile Java objects have good scalability but high resource consumption. Code migration is very resource dissipative and should be used only when necessary.

To get more insights of the code management services, we take Agilla[29] as an example of the implementation techniques. Agilla is a mobile agent based WSN middleware. The idea behind Agilla is to initially deploy a network without any application installed. Agents that implement the application behavior can later be injected, effectively reprogramming the network. Agilla marks the first time that multiple mobile agents and tuple spaces are used in a unified framework for WSNs.

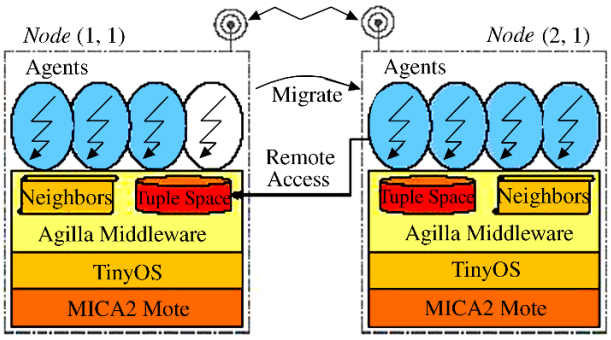


Fig.4. Agilla system model.

The Agilla system model is shown in Fig.4. Each sensor node supports multiple agents, and maintains a tuple space and a neighbor list. The tuple space is local and is shared by the agents residing on the node. Special instructions are given to allow the agents to remotely access another node's tuple space. The neighbor list contains the addresses of all the one-hop nodes. Agents can migrate carrying their code and state, but not their own tuple spaces.

Fig.5 shows the middleware architecture of Agilla. The tuple space manger implements the tuple space operations (e.g., out, inp and rdp) and reactions, and manages the contents of the local tuple space and reaction registry. The agent manager maintains each agent's context. It is responsible for allocating memory to an agent when it arrives and de-allocating it when the agent leaves or dies. The context manager determines the node's location as well as that of its neighbors. Instruction manager and Agilla engine provide runtime support. Instruction Manager is responsible for dynamic memory allocation, retrieving the next instruction to execute, and packing up the agent's code into the minimal number of messages. The Agilla engine controls the concurrent execution of all the agents on a sensor node.

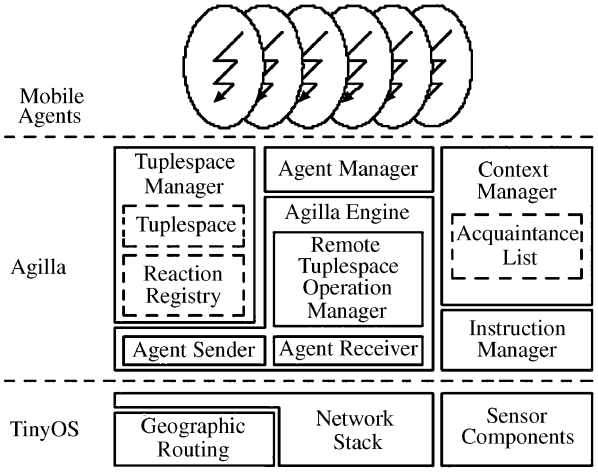


Fig.5. Agilla middleware architecture.

Fig.6 shows the agent architecture. An agent consists of a stack, a heap, and various registers. The heap is a random-access storage area that allows an agent to store variables. The registers contain the agent's ID, program counter (PC), and the condition code. The agent ID is unique to each agent and is maintained across migration operations. A cloned agent is assigned a new ID. The PC contains the address of the next instruction, and is used by the code manager for fetching the next instruction. When a reaction fires, the reaction manager changes the PC to point it to the first instruction of the reaction's code. To allow an agent to resume execution from where it was when the reaction fired, the original PC is stored on the stack. The condition code records the execution status.

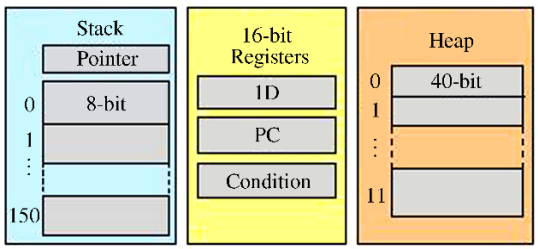


Fig.6. Agilla mobile agent architecture.

With regards to code allocation, Agilla uses a reaction approach. Reactions are added to the tuple spaces, allowing an agent to tell Agilla that it is interested in tuples which match a particular template. The tuple space manager remembers the reactions registered by each agent by storing them within the reaction registry. Whenever a tuple is inserted, the registry is checked to see whether the new tuple matches a reaction's template. If so, the tuple space manager notifies the agent manager, which updates the agent's program counter to execute the reaction's code.

Code migration is implemented by moving or cloning an agent from one node to another. The tuple space manager packages up all the reactions registered by an agent so they can be transferred along with the agent. When an agent moves, it carries its state and code, and resumes executing on the new node. When it clones, it copies its state and code to another node and resumes executing on both the old and new nodes. The multi-hop migration is handled by the middleware and is transparent to the user.

3.2 Data Management

As mentioned before, WSN applications are data centric. Here, data refers mainly to the sensed data. Sometimes it also refers to the network infrastructure information interested by the applications. Data management in WSN middleware provides services to applications for *data acquisition*, *data processing, and data storage.* The approaches to implementing the data management services depend much on the application data model.

3.2.1 Data Acquisition

Data acquisition is an essential service for WSN applications, responsible for delivering the relevant and accurate data required by the application.

For the event-based data model, data acquisition support is focused on the event definition, event register/cancel, event detection and event delivery. The application specifies the interest in certain state changes of the data. Upon detecting such an event, the middleware will help send event notification to interested applications. TinyDB[25], DSware[43], Mires[44], and Impala[22] all support event-based data acquisition. DSware also supports compound event detection.

A typical approach to implementing event-based data acquisition is the Pub/Sub paradigm, which has two advantages in supporting event-based data acquisition. First, it supports asynchronous communication. Second, it facilitates message exchanging between the sensor nodes and the sink node. The basic entities of Pub/Sub system are event subscriber and event publisher (some times event broker also). From the middleware's point of view, the event subscriber is the sink node and the event publishers are the sensor nodes.

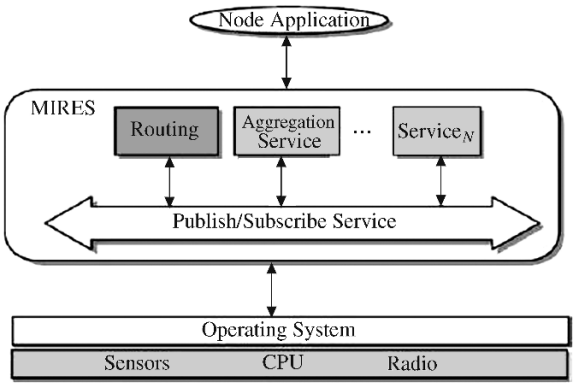


Fig.7. Mire's architecture.

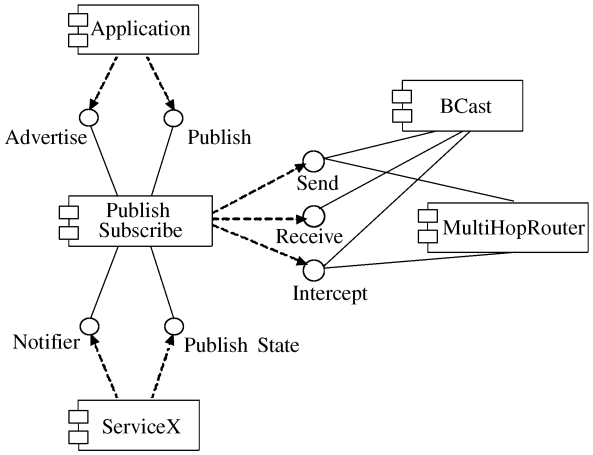


Fig.8. Mire's Pub/Sub component.

As an example of the Pub/Sub approach, let us have a look of Mires. Figs. 7 and 8 show Mires' architecture and its Pub/Sub component structure, respectively. Mire includes a core component, namely the Pub/Sub service, and some additional services. The communication between the sensor nodes consists of three phases. Initially, the sensor nodes in the network advertise their available topics (e.g., temperature and humidity) collected from the local sensors. Next, the advertised messages are routed to the sink node using a multi-hop routing algorithm. A user application connected to the sink node is able to select (i.e., subscribe) the desired advertised topics to be monitored. Finally, the subscribed messages are broadcasted down to the

network nodes. After receiving the subscribed topics, the sensor nodes are able to publish their collected data to the network. The Pub/Sub service maintains the topic list and the subscribing applications so as to marshal the right topic to the related application. In Mires, only the messages referring to the subscribed topics are sent, hence reducing the number of transmissions and energy consumption.

For query-based data model, data acquisition support is focused on the query processing model and methods. Middleware for query-based data model usually uses a declarative interface, with global level abstraction and database programming model. Example systems are TinyDB[25], Cougar[26], and SensorWare[41]. They leverage the techniques used in the traditional database system to implement data acquisition services, e.g., applying distributed query[45,46] or CACQ (continuously adaptive continuous queries over streams)[47,48].

TinyDB is a good example to illustrate the query-based approach. TinyDB is a query-processing system that extracts information from the data collected by the WSN using the underlying operating system TinyOS. TinyDB maintains a virtual database table, called SENSORS, whose columns contain information such as sensor type, sensor node identifier, and remaining battery power. The programmer can view the values of the SENSORS, and add new rows to it. Consider the following example, a user wants to be reported when the average temperature is above 80°F in any room on the third floor of a building monitored by sensors. The user inputs the following database query along with the rate at which the sensors are to collect the data:

```
SELECT AVG (temp) FROM sensors
(select rows from Sensors)
WHERE floor = 3
(at the 3rd floor)
GROUP BY room
(rows are grouped by room number)
AVG (temp) > 80F
(only groups with average temperature > 80F)
SAMPLE PERIOD 20 seconds
(perform every 20 seconds—rate of collection)
```

TinyDB uses a controlled-flooding approach to disseminate the queries throughout the network. The system maintains a routing tree (spanning tree) rooted at the end point (usually the user's physical location). Then, in a decentralized approach, every sensor node has its own query processor that processes and aggregates the sensor data and maintains the routing information. In every period, the parent node closer to the root agrees with its children in a time interval for listening to data from them.

3.2.2 Data Processing

Generally speaking, there are three different approaches to supporting data processing in WSNs. In centralized processing, all the data are collected and then sent to a central node for processing. In node level, distributed processing raw data collected in the sensor nodes are pre-processed to obtain partial results, which are then collected by the sink node for further processing to get the final result. In network level, distributed processing final results are obtained through both node-level distributed processing and information exchange between the sensor nodes, and between the sensor nodes and the sink node. In the extreme case, where every sensor node is involved with data processing and routing, and is aware of the final decision, it becomes completely distributed processing.

Given that the communication cost is much higher than the computation cost at a sensor node, WSN middleware should support in-network distributed data processing service, mostly through *data fusion/aggregation*. Although in-network data processing services are also supported at a lower level by some firmware in terms of signal conditioning, and data fusion and data aggregation can also be supported at the MAC and routing layers, middleware support has the following distinctive features. 1) It is more independent of the underlying network protocols, so different strategies can be applied according to different data accuracy requirements from different applications or different network conditions. 2) It facilitates high level data analysis such as feature-based fusion and decision-based fusion.

For event-based data model, data aggregation/fusion can be implemented in separate services. An example is the aggregation service in Mires[44]. In Mires, data aggregation is implemented in separate modules for functions such as AVG and SUM. The aggregation is executed by an "Aggregate Use" module that carries out an activity of de-multiplexing, passing requests for the correct aggregation module in accordance to its identifier. This way, the flexibility to add new aggregation functions is guaranteed, just requiring the creation of a module for the new function and adding the association between the function and an identifier to a configuration file.

In addition, for event-based data model, detecting the event boundary and determining the event area and its center should also be considered in WSN middleware.

For query-based data model, data aggregation/fusion services can be implemented by using the

pipelining techniques[46], as used in TinyDB and SensorWare.

Another data processing service is *data calibration* for ensuring the synchronization between the sensor nodes. Some applications, e.g., seismographic or building health monitoring, require precise time synchronization among the readings on different sensor nodes. How to achieve time synchronization is an important function of the middleware. More details on implementation techniques of time synchronization in WSN, e.g., hardware implementation, software implementation can be found in [55].

3.2.3 Data Storage

There are three approaches to implementing data storage support in WSNs[49,50]. External storage stores the data in the base station out of the WSN. Local storage stores the data where it is generated, reducing communication but increasing the inquiry cost. Data centric storage provides a tradeoff between the previous two approaches. Data-centric storage is the most popular approach implemented in existing WSN middleware.

Let us look at Data Service Middleware (DSWare)[43] as an example to show the data storage service implementation in WSN-middleware. As shown in Fig.9, DSWare is a specialized layer that implements various data services and, in doing so, provides a database like abstraction to WSN applications. Fig.10 shows the DSWare framework. The event detection component is responsible for providing the data acquisition service. The group management component provides the support for group-based decision and is responsible for data aggregation. The scheduling component schedules the services to all DSWare components with two scheduling options: energy-aware scheduling and real-time scheduling. Here, we focus on the data storage and caching components.

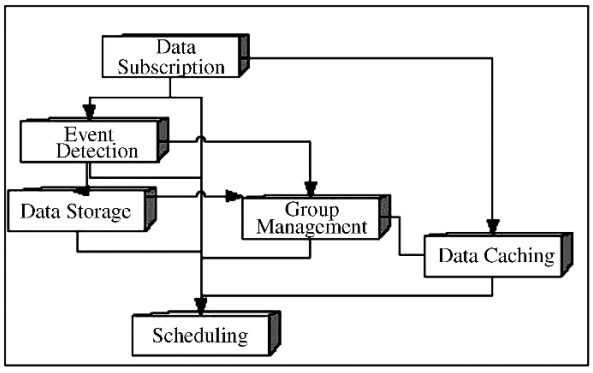


Fig.9. DSWare framework.

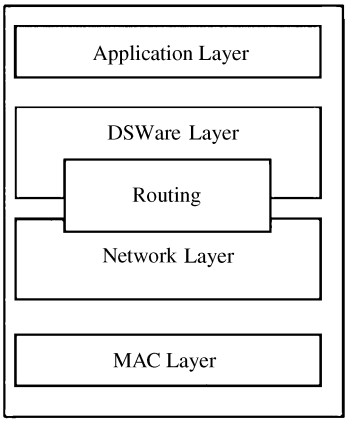


Fig.10. DSWare system model.

The Data Storage component in DSWare stores data according to the semantics associated with the data. It has a data look-up operation and provides fault tolerance should there be node failures. It also has operations for storing correlated data in geographically adjacent regions. This has two advantages: enabling data aggregation and making it possible for the system to perform in-network processing.

To facilitate data look-up, DSWare maps data to physical storage using two levels of hash functions. At the first level, the hash function maps a key, which is a unique identifier assigned to each data type, to a logical storage node in the overlay network. As a result of this operation, the storage nodes form a hierarchy at this level. The second level involves the mapping of single logical node to multiple physical nodes such that a base station performing a query operation has the data fetched from one of the physical locations. There is a big risk in mapping a given data type to a single node as this data could be lost as a result of node failure. Furthermore, mapping data to a single node in the sensor network causes bursts of traffic to the node which may lead to collision and higher rate power consumption. DSWare uses replication to store data in multiple physical sensor nodes that can be mapped onto a single logical node. Load balancing is achieved since queries can be directed to any one of the physical nodes and the lifetime of individual nodes is prolonged since power consumption is substantially reduced. With replication of data amongst multiple nodes there come consistency issues. DSWare adopts "weak consistency" to avoid peak time traffic since only the newest data amongst nodes is bound to lack consistency. This new data is propagated to other nodes and the size of inconsistent data is bounded so that replication occurs when the workload in individual nodes is low.

Data caching in DSWare provides multiple copies of data that are most requested. DSWare spreads the cached data over the network to achieve high availability and faster query execution. A feedback control scheme is used for dynamically deciding whether or not copies should reside in frequently queried nodes. The scheme uses various inputs, including proportion of periodic queries and average response time, from data source to guide the nodes in making decisions about whether or not a copy should be kept. This component also monitors the usage of the copies to decide whether to increase or reduce the number of copies, or move them to a new location.

In conclusion, 1) data management is an important topic in WSNs. One of the distinguished features that middleware offers in data management is the appropriate abstraction of data structure and operation. Without this abstraction, the developer has to manage the heterogeneous data and low level operation in the application. Various exiting data management algorithms can be implemented as reusable and alternative middleware services with certain of parameters. The middleware system can even automatically adjust the service parameters according to its current status. Application specific data management algorithms can be written based on those common data services. This also facilitates the development process. 2) Most existing WSN middleware provides some kinds of data management services. However, high level in-network analysis services related to the WSN application domain, e.g., data mining, are not implemented yet, to which more attention should be paid.

3.3 Resource and Information Discovery

Resources in a WSN usually refer to the sensor node hardware resource, e.g., energy, memory, A/D device, and communication module. The resource discovery service returns the data type that a discovered node can provide, the modes in which it can operate, and the transmission power level or residual energy level of a sensor node. On the other hand, the information discovery service returns the information about the network topology, the network protocols, and the neighbors and the locations of the discovered nodes. The service can also be used for discovering new nodes and finding out when nodes become inaccessible as a result of either mobility or loss of battery power. However, many of the above mentioned service features are not being available in existing WSN middleware yet.

Compared to the resource discovery in traditional networks[51,52] which is involved with identifying and locating (relocating) the services and resources in the system, resource and information discovery services in WSN are more difficult to implement due to the lack of unique node ID and the lack of generic service specification, and because the services need to be provided in a power-aware way. Some existing WSN middleware systems adopt service discovery protocols from traditional computer network solutions, e.g., SLP[53] and Bluetooth SDP[54]. MiLAN is an example. Other systems, e.g., TinyLime, use tuple space to implement the resource discovery service. However, these implementations need unique ID of the resource, but many WSNs are content-based without unique ID for sensor nodes.

Although many localization algorithms have been developed for different kinds of systems, for example, Ultrasound, RF, and ultra-wideband, RSSI techniques are used for accurate localization via carefully placed beacons. Few existing WSN middleware has integrated location discovery service. To our opinion, this is mainly because the implementation of this kind of service depends very much on hardware and the underlying environment. For large scale use of WSN for pervasive computing, standard and adaptive location discovery services should be provided.

3.4 Resource Management

Resource and information discovery services described in the previous section have two main functions: 1) providing the underlying network information to applications that are required to be reflective or adaptive (e.g., context-aware); 2) providing the underlying network information for the purpose of supporting adaptive resource management services. Resource management in WSN middleware is mainly for providing common and reusable services of supporting the applications that have the requirements for self-organization. Resource management services are usually used for resource configuration at setup time and resource adaptation at runtime, and they are essential to ensure the QoS of WSN which we will discuss in details later on.

Resource management at the OS layer is platform-dependent, so changes at this level might affect different resource requirements of the applications running in a sensor node. On the other hand, application-level resource management imposes an extra burden on the application, and adaptation mechanisms developed at this level cannot be reused. In contrast, resource management at the middleware layer has more flexibility. Most existing WSN middleware provide services including cluster service[56,57], schedule service, and data routing service. These services are supported by finer granular services such as power level management, transmission level management, etc. These fine granular services

should be supported and constrained by the underlying OS, the firmware, and the hardware. Otherwise, it is impossible for the middleware to provide the corresponding services.

The cluster service refers to the cluster member maintenance for layered WSN. For cluster service, many middleware systems, including EnviroTrack[30], MiLAN[38], DSWare[43], AutoSec[58], and SINA[37] addressed the implementation issues according to different objectives. For examples, EnviroTrack[30] provided the cluster member re-allocation service that redefines the clusters after deployment; the MiLAN[38] and AutoSec[58] provided automatically cluster organization service according to the QoS information obtained from network infrastructure and WSN application. SINA[37] and DSWare[43] also provided automatically cluster organization service, but the objectives were to achieve appropriate clusters so as to facilitate the data aggregation process. Except for the above examples, the work reported in [57] provides a function for generic cluster management of sensor nodes. The function arises either in terms of non-functional requirements (e.g., security, reliability) or according to dynamic system conditions. (e.g., power level, connectivity).

The schedule service refers to the node wakeup/sleep scheduling. It is used for reducing the energy consumption by allowing the sensor nodes to be put to sleep and to be waken up according to specific policy. For example, when not being allocated tasks to a sensor node can sleep in order to save energy. Implementation of this service may make use of the services, such as sleep scheduling protocols in the MAC layer and CPU voltage scaling[1] in the physical layer.

The data routing service can be implemented in several different ways. Some middleware such as Mate[20] do not provide any specific routing management service, but provide architecture which allows the implementation of arbitrary routing protocols. For systems that provide routing management services, three main approaches can be identified. The first approach is implementing a new higher level routing protocol at the middleware level. An example is MagnetOS[40] that implements a multi-hop routing protocol in a middleware component. The second is maintaining an overlay, and supporting routing mechanism, as well as routing reconfiguration on top of this overlay. For example, Mires[44] makes use of a Pub-Sub mechanism to support the routing management. Owing to the loosely coupled interactions between the nodes in the Pub-Sub paradigm, it is very flexible to provide new kind of data routing implementation. The third approach is implementing a mechanism that allows for switching between different routing protocols, as what is done in Impala[22], or providing a mechanism that allows for the adaptation of different routing protocols, as what is done in MilAN[38]. As an example, Fig.11 shows the data routing management of MilAN. The routing management of MiLAN is intended to sit on top of multiple physical networks. It acts as a layer that allows network-specific plug-ins to convert MiLAN commands to protocol-specific ones that are passed through the usual network protocol stack. Therefore, MiLAN can continuously adapt to the specific features of whichever network is being used in the communication.

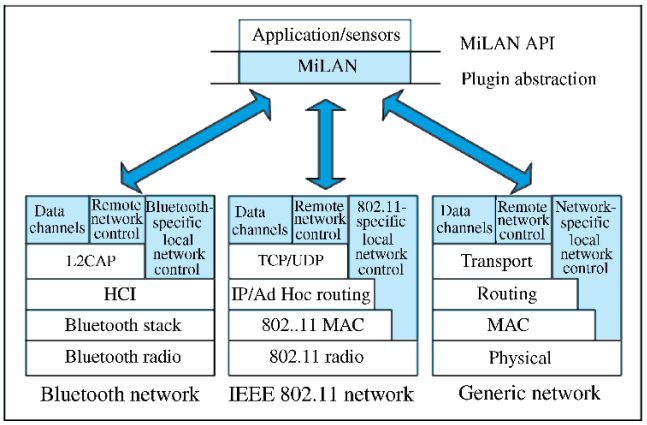


Fig.11. MiLAN network protocol management.

In conclusion, most existing WSN middleware adopts localized resource management. Policy-based management has been shown to be a good approach to supporting the design of self-adaptive resource management. Currently, resource management services in existing WSN middleware are tightly coupled with applications and generic resource management services need to be developed.

3.5 WSN Integration

For broader applications, WSN needs to be integrated into other exiting network infrastructures, such as the Internet, Gird, and database-based systems. Because a WSN is a "close" network, it is not easy to implement the integration service at the lower layers (e.g., OS or MAC layer), and thus middleware should provide this service[59−63].

For WSN middleware, integration is related to both task coordination as well as data sharing, and can be implemented at the application level or data level. Application level integration is more related to task coordination, where the applications are running in both

the WSNs and the leverage system. Data level integration, on the other hand, is more related to data sharing, where only the data provided by the WSNs are used in the leverage system.

Proxy server is a common mechanism for integration implementation. The client can access non-standardized services in a wireless sensor network by inquiring a proxy server which translates the standardized protocol to the proprietary protocol, and vice versa. For integrating WSNs into existing legacy middleware, authors of [65] described an "edge server middleware" which performs application-specific processing at the boundary between the WSN and the legacy middleware. For integrating WSNs into a Gird middleware infrastructure, several early systems have been reported[60−63] on sensor-grid computing.

The service-oriented approach[28,64] to implementing WSN integration is based on standard open architecture technologies such as Web services[66]. It provides a common information and communication format to facilitate the integration. In this approach, the sink node is modeled as web service provider that exposes the services provided by the network using a standard service interface. The WSDL language and SOAP protocol[66] are used for describing the services and formatting messages used by the underlying communication protocol. This approach does not combine the proposal with any other particular underlying data dissemination protocols. Thus, the WSN can be used as a system to supply data for different applications and users.

On the basis of the SOA approach, some researchers proposed the concept of "Sensor-Web"[67−69]. The Sensor Web aims at making various types of web-resident sensors, instruments, image devices, and repositories of sensor data, discoverable, accessible, and controllable via the World Wide Web. A lot of efforts have been made in order to overcome the obstacles associated with the connecting and sharing of these heterogeneous sensor resources. The Sensor Web Enablement (SWE) standard has been defined by the OpenGIS Consortium (OGC), which is composed of a set of specifications, including SensorML, Observation & Measurement, Sensor Collection Service, Sensor Planning Service and Web Notification Service. OGC has also proposed a reusable, scalable, extensible, and interoperable service-oriented sensor Web architecture, which conforms to the SWE standard, integrates Sensor Web with Grid Computing, and provides middleware support for Sensor Webs.

In comparison, the proxy server approach is more application dependent and less scalable because adding or removing a proxy server will impact the network structure. The service-oriented approach is more flexible and scalable, but needs more energy consumption. Most of the integration services are now still in very preliminary stage. WSN middleware for this kind of services are resting on the architecture concept level.

4 WSN Runtime Support

As mentioned in Section 2, because the underlying WSN platform, mostly the embedded OS does not always provide enough support for implementing the middleware services, WSN middleware needs to develop runtime support. Runtime support extends the functions provided by the embedded OS for processing, communication, and storage management in order to provide a well-defined execution environment for the execution of application and system programs. The basic functions of runtime support include inter-process communication (IPC), memory control, and power control in terms of voltage scaling and component activation and inactivation. These functions are used by higher level middleware services such as multi-thread processing, smart task scheduling, synchronization of memory accessing and the spread signal spectrum management. Runtime support in WSN middleware is always embodied as some kinds of virtual machines over the underlying platform. It can be implemented as a platform specific kernel on top of the embedded OS, but with platform independent primitives for the generic WSN middleware services.

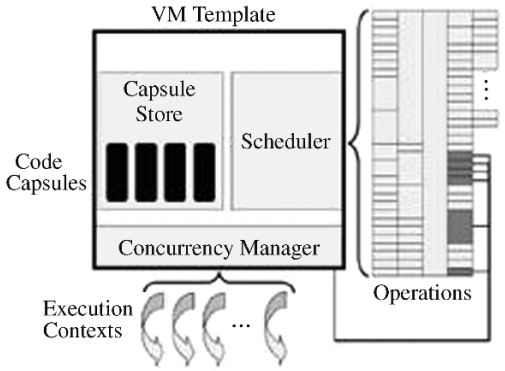


Fig.12. Architecture of Mate.

Let us look at an example, Mate[20], which is a middleware built on top of TinyOS. Fig.12 illustrates its architecture. Mate takes the role of a traditional OS kernel. Instead of system calls, it provides a set of primitives for programming. As shown in the figure, Mate has a byte code interpreter that runs on TinyOS which adopts a component-based model to build sensor network applications in an event-driven operating

environment. The core of the Mate architecture is a simple FIFO scheduler. The scheduler maintains a queue of run-able contexts, and interleaves their execution. The scheduler executes a context by fetching its next byte code from the capsule store, and dispatches it to the corresponding operation component. The Mate concurrency model is based on statically named resources, such as shared variables. Operations specify the shared resources that they use. The analysis that determines a handler's complete resource usage is language-specific. Mate forwards the programs using the Trickle algorithm which uses broadcast-based suppressions to quickly propagate new data but minimizes the overheads when the nodes share data. Just as with explicit forwarding, once a user installs a single copy of a program in the network, Mate installs it on every mote. Mate proposes a spectrum of reprogrammability, from simply adjusting parameter to uploading complete program updates. The support can be used for developing more complex code management services, which will be discussed later on.

Another example is Magnet[40], a runtime based on MagnetOS, which is a power-aware, adaptive operating system specifically designed for sensor and ad hoc networks. With its support for a single system image, MagnetOS overcomes the heterogeneity of distributed, ad hoc sensor networks by exporting the illusion of the Java VM on a top of distributed sensor networks. Magnet has both static and dynamic components. The static components are responsible for rewriting applications in the form of object modules. The runtime components (dynamic components) on each node monitor the object's creation, invocation, and migration.

Task cooperation is an important component of runtime support, which includes the means for communication between distributed tasks. Task cooperation is useful when multiple tasks reside on the nodes in the WSN and need to interact with each other. In existing systems, e.g., TinyTime[42] and Agilla[29], implementation of task coordination mainly uses the tuple space approach. The concept of tuple space was proposed originally in Linda. Tuples are collections of passive data values. A tuple space is a pool of shared information, where tuples are inserted, removed, or read. Data are global and persistent in the tuple space and remain until being explicitly removed. In WSN middleware, tuple spaces are used for inter-agent communication and context discovery. In the tuple space, a task does not need to know its peer task and tasks do not need to exist simultaneously, because they do not need to communicate directly.

Let us look at the Agilla example again for task coordination support. The tuple space ensures that the agents run autonomously by allowing them to communicate in a decoupled fashion. For example, suppose there are a fire detection agent and a habitat monitoring agent residing on the same node when fire is detected. The fire detection agent inserts a fire tuple into the local tuple space to indicate the presence of fire and activates a tracking agent before dying. The habitat monitoring agent reacts to this tuple, and voluntarily kills itself to free additional resources. The fire detection agent does not need to know who receives the fire tuple. The sending and the reception operations can occur respectively at different time, and reception can occur even when the sender is no longer present. This spatial and temporal decoupling ensures each agent operates autonomously. In Agilla, agents also need to coordinate with the agents residing on remote nodes. Agilla allows agents to coordinate across nodes by introducing special remote tuple space operations. They are synonymous with local operations, except that they take an additional location parameter that specifies on which node to perform the operation. Note that Agilla does not support tuple spaces that span across nodes. Each node maintains a distinct and separate tuple space. The dedicated remote tuple space instructions rely on unicast communication with the specific node hosting the tuple space.

5 WSN QoS Support

QoS support is important for applications with requirements on performance, both functional and non-functional. These requirements include fault-tolerance[70,71], reliability[72,73,77], security[74−76], and real-time[77−80]. It is also very important for context-aware applications. Under different contexts, applications can adopt different QoS policies. As mentioned in Section 2, QoS support in WSN middleware is still an open issue for research. First of all, the QoS metrics are not well defined in the context of WSN applications. For expressing QoS of WSN network performance, parameters such as packet delay, jitter and loss, throughput, and latency are defined. For expressing QoS of WSN application performance, new parameters are defined, including data accuracy, aggregation delay, aggregation degree, coverage, and precision[34]. The literature [34] surveyed both of above parameters, and proposed that in WSN middleware level, *collective QoS metrics* should be considered. Because WSNs are always densely deployed, single sensor accuracy or time delay between two respective sensors may not be meaningful. Collective behavior and effect of a set of logically related sensors become very important. However, nei-

ther did [34] nor other existing work provides a clear definition of *collective QoS* metrics and the way of how to implement them in WSN middleware.

Implementation of QoS support in WSN middleware depends on the middleware services we have described before, mainly the resource discovery service and the resource management service. Provision of QoS in WSN middleware, on the other hand, also affects other services, such as data acquisition in the data management service.

WSN middleware may provide a QoS support implementation framework with the fundamental QoS assurance algorithms to simplify the development process. For example, in [70] a framework is proposed with fault-tolerant algorithms. But a typical implementation approach of QoS support in WSN middleware is to translate and control the QoS between the application and the networks. That is, if the QoS requirements from an application are not feasible to be satisfied in the network the middleware may negotiate a new quality of service with both the application and the network.

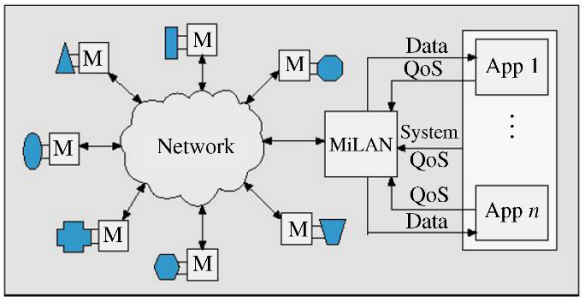


Fig.13. QoS support in MiLAN.

Take MiLAN, for example, let us look at how existing systems implement QoS support. As shown in Fig.13, MiLAN assumes that application performance can be described by the QoS expressed in different variables of interest to the application. The QoS variables depend on the fact that sensors provide data to the application. Each sensor has a certain QoS in characterizing each of the application's variables. In order to determine how to best serve the application, MiLAN needs to know the variables of interest to the application, the required QoS for each variable, and the level of QoS that the data from each sensor or set of sensors can provide for each variable. Note that all of these may change according to the application's current state.

During the initialization of the application, the QoS information is conveyed from the application to MilAN via State-based Variable Requirements Graph and Sensor QoS Graph. The State-based Variable Requirements Graph specifies to MiLAN the application's minimum acceptable QoS for each variable based on the current state of the application. For a given application, the QoS for each variable can be satisfied using the data from one or more sensors. Given the information from the two kinds of graphs as well as the current application state, MiLAN can determine which sets of sensors satisfy all of the application QoS requirements for each variable. These sets of sensors define the application feasible set *FA* where each element in *FA* is a set of sensors that provides QoS greater than or equal to the application-specified minimum acceptable QoS for each specified variable. MiLAN also determines which sets of sensor nodes can be supported by the network using Sensor QoS Graph. The subsets of the nodes that can be supported by the network define a network feasible set *FN*. MiLAN combines these two constraints to obtain an overall set of feasible set $F = FA \cap FN$.

QoS support in WSN-middleware is still in a very preliminary stage. Although work has been reported, there is a lack of implementation and simulation results.

6 Taxonomy of WSN Middleware Features

Current works on developing middleware for WSNs have focused on different aspects and for different purposes. In this section, we highlight the characteristics of WSN middleware and propose taxonomy of the desirable features. We first classify the features of WSN middleware by using feature trees, and then on the basis of the classification provide an overview of the exiting works. The use of a feature tree facilitates structuring the middleware features and describing the relationships between them.

The feature tree is derived by analyzing the system model and the implementation approaches of WSN middleware functions discussed in the previous sections. In Fig.14, the solid dots represent the necessary features, while the hollow dots represent optional or alternative features. Features can be decomposed into sub features. The relationship between sub features can be either the inclusive relationship, denoted by solid branch cross, or the alternative relationship, denoted by the hollow branch cross.

As seen in the root tree (see Fig.14(a)), the features are from both the WSN middleware and the underlying network. The features from the middleware include the services provided by and system architecture of the target middleware. The features from the WSN consist of two parts. The first part contains the features describing the characteristics of the applications that the target middleware can support. The second part contains the features that describe the underlying WSN platform support of the target middleware.

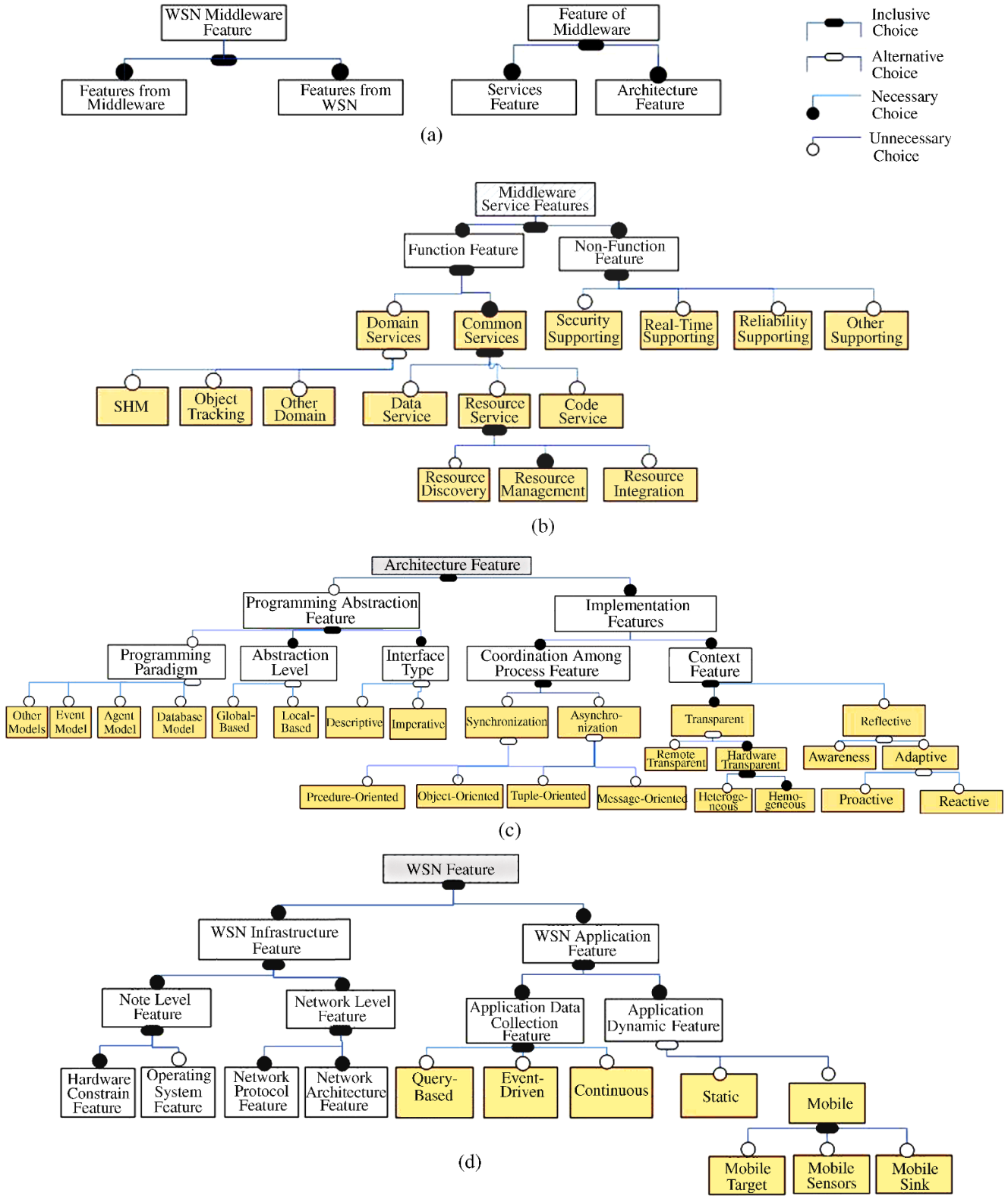


Fig.14. Feature trees of WSN middleware. (a) Root feature tree of WSN middleware. (b) Feature tree of middleware services. (c) Feature tree of middleware architecture. (d) Feature tree of WSN.

The middleware feature sub-tree (see Fig.14(a)) itself has two sub-trees: middleware service features (see Fig.14(b)) and architecture features (see Fig.14(c)). Middleware service features are composed of functional features, which are related to the middleware services, and none-functional features, which are related to the QoS support. The architecture features are composed of the programming abstraction features and the implementation features. We have discussed the programming abstraction features in Subsection 2.2.

Implementation-related features have two main perspectives: the feature of coordination among the processes and the context features. In the branch representing the process coordination, the procedure-oriented method is not very applicable to the WSN Middleware due to its poor scalability[81]. Object-oriented method is scalable, but it is based on the object context whose weight is too heavy for the WSN context. Message-oriented and tuple space oriented methods can scale easily and support the event-driven paradigm[82]. Thus, they are often used in the WSN middleware.

In the branch that represents the context features, hardware transparent features are those that hide the hardware implementation details from users. The features supporting heterogeneous hardware transparency hide the differences of various hardware platforms. These kinds of features are rarely supported because of the diversity of the WSN hardware and the tight application dependent characteristics. Remote transparency means hiding the location and other network attribute

Table 1. Overview of Existing Projects and Proposals

(a)

Features	Programming Abstraction Features			Middleware Service Features	
Proposal	Interface Type	Abstraction Level	Programming Paradigm	Functional	Un-Functional
Sensor Ware	Declarative	Local	Database	CodeManagement	None
MagnetOS	Imperative (java language)	Global	Virtual machine	CodeManagement ResourceManagement	None
Mate	Imperative (motle, byte code)	Local	Virtual machine	CodeManagement ResourceManagement	Security
MiLAN	None	None	None	ResourceManagement ResourceDiscovery	None
SINA	Declarative (cells data format SQTL)	Global	Database	DataManagement	None
TinyDB	Declarative (ACQP language)	Global	Database Graphic UI	DataManagement ResourceManagement	None
Cougar	Declarative (XML data format, SQL-like language)	Global	Graphic UI	DataManagement	None
Impala	None	None	Mobile agent	CodeManagement ResourceManagement StorageSupporting	Security
Agilla	Imperative (compile-like)	Local	Mobile agent	CodeManagement	None
FACTS	Declarative (facts, rules, functions, XML like)	Global	Rule based	CodeManagemet DataManagemet	None
AutoSec	None	None	None	ServiceDiscovery ResourceManagement	None
DSWare	Declarative	Global	None	DataManagement ResourceManagement StorageSupporting	Real-time Reliability
A Service Approach	Declarative (XML,SOAP)	Global	None	DataManagement Integration Resource discovery	None
Mire	Imperative	Local	Graphic UI	DataManagement	None
Enviro Track	Imperative object-based (context object and) tracking objects	Global	None	DataManagement ResourceDiscovery ResourceManagement	None

Table 1. Overview of Existing Projects and Proposals

(b)

Features	Internal Implementation Features			Supporting Application Features		WSN Features	
Proposal	Process	Context		Application	Application	Network	Node
	Communication	Transparent	Reflective	Dynamic	Dada Collection	Level	Level
Sensor Ware	Asynchronous Tuple-Oriented	Homo	None	Mobile Phenomena	All	None	Linux IPAQ Based
MagnetOS	Asynchronous	Homo/ Hetero	None	All	All	Ad-Hoc On-Demand Routing	Build on PC
Mate	Synchronous Asynchronous	Homo	None	All	All	Content-Based Routing	TinyOS-Based;
MiLAN	Synchronous Asynchronous	Hetero	Proactive	All	Event-Driven	Switch Between Different Protocols	None
SINA	ASynchronous	Remote Homo	None	Static	Enquire	Cluster-Based	None
TinyDB	Asynchronous	Homo	None	Static/Mobile Sensor	Enquire/Event Driven	Spanning-Tree-Based Routing	TinyOS Mote
Cougar	Asynchronous	Remote Homo	None	Static	Enquire	Directed Diffusion Routing Cluster-Based	None
Impala (designed for ZebraNet)	Asynchronous	Homo	Adaptive	Mobile Sensor	Event-Driven	Switch Between Different Protocols	iPAQ/PC Linux
Agilla	Asynchronous (tuple based)	Homo	None	All	Any	Multi-Protocol Routing	Tiny OS, MICA2,
FACTS	Asynchronous Tuple-Oriented	Homo	None	All	Event-Driven	None	None
AutoSec	Asynchronous Broker-Based	Homo	Proactive	All	No	None	Tiny OS
DSWare	Asynchronous Pub-Sub-Based	Homo	None	All	Event-Based	Cluster-Based	None
A Service Approach	Asynchronous (Pub-Sub)	Homo Remote	Reflective Proactive	All	Event-Driven	Directed Diffusion	None
Mire	Asynchronous (Pub-Sub)	Homo	None	All	Event-Driven	Cluster-Based	TinyOS
Enviro Track	Asynchronous Synchronous	Homo	None	Mobile-Phenomena	Continues/ Event-Driven	Group-Based Content-Based Routing	TinyOS

details from the user. Remote transparency does not fit WSN systems which need the information to support the applications and manage the resources.

For context features, in order to achieve resource management adaptation, developers must introduce open resource configuration and reconfiguration. This requires the context to be reflective[83]. Computational reflection refers to the ability to reason about (being aware of) and alter (being adaptive to) its own behavior. Both "proactive" and "reactive" middleware systems belong to the category of adaptive middleware. Compared to "reactive" middleware systems, which react only by themselves when changes occur within the network, "proactive" middleware systems enable applications (or middleware) to actively participate in the process of configuring the network where the middleware operates.

The branch representing the WSN network features (see Fig.14(d)) includes the features of WSN applications such as the application dynamic features and the application data collection features (see the right branch of Fig.14(d)) as explained in [84]. Application data collection feature defines the data acquiring approach. By observer initiated approach, the data acquiring process is initiated by the data query command. By event-driven approach, data will be sent to the sink node whenever a defined event occurs. By continuous approach, data is sent to the sink node during every de-

Table 2. Summary of Existing WSN Middleware Production

Proposal	Test Environment	Simulation and Testing Tools	Published Evaluation Results
Sensor Ware	Prototype	SensorSim	Framework Size, Execution Delays, Energy Consumption
MagnetOS	JVM	Custom Simulator	Internal Algorithm Comparison in Simulator
Mate	Prototype	TOSSIM	Byte Code Overhead, Installation Costs, Code Infection Performance
MiLAN	None	None	None
SINA	Simulation	GloMosim	SINA Networking Overhead, Application Performance
TinyDB	Simulation Prototype	Custom Environment	Query Routing Performance in Simulation, Sample Accuracy and Sampling Frequency in Prototypes
Cougar	None	None	None
Impala	Simulation	ZnetSim	System Implementation and Overhead, Impala Event Processing Time, Software Transmission Volume
Agilla	Prototype	Mica 2	Reliability of Remote Tuple Space and Agent Migration Operations, Overhead of Agilla's Instructions, Case Study
FACTS	None	None	None
AutoSec	Simulation	Custom Environment	Information Collection Overhead, Overall Performance Efficiency
DSWare	Simulation	GloMoSim	Performance in Reduction of Communication, Impact of Node Density, Performance in Reaction Time
Mire	None	None	Case Study
Enviro Track	Prototype	Mica 2	Communication Performance Data, Effect of Sensory Radius on Maximum Trackable Speed
Service Oriented Middleware	None	None	WSDL Documents for Describing, the WSN Services and the SOAP Messages Format and Content

fined period. The application dynamic features define the mobility characteristic of WSN application. Mobility can be divided into three classes, namely, mobile target, mobile sensors and mobile sink. Location discovery service of WSN middleware is especially important for mobile sensor model. WSN infrastructure features (see the left branch of Fig.14(d)) also have important impact on the WSN middleware. For example, if TinyOS is used, then synchronization process coordination can be easily implemented because of the inherent support from the TinyOS's active message mechanism.

An overview of the exiting work can be found in Table 1, which compares the existing WSN middleware systems and the proposals by applying the feature tree. We also provide Table 2, which summarizes the exiting WSN middleware by classifying them into Test-beds, simulation and testing tools, and evaluation results.

7 Challenges and Directions

Before concluding this paper, in this section, we discuss the challenges, research issues, and future directions in the area of WSN middleware.

We classify the challenges in designing and implementing WSN-middleware into three main types. The first type of the challenges comes from the conflict between the contexts of distributed computing and the embedded sensor devices. Distributed computing should support scalability, reliability, dependability, and heterogeneity, but this demands the careful the design under the context of resource limited devices and dynamic network topology.

The second type of challenges comes from the tradeoffs between the degree of application specificity and middleware generality. It is important to integrate application knowledge into the services provided by the middleware because it can significantly improve the resource and energy efficiency of the operations. However, since middleware is designed to support and optimize a broad range of applications, tradeoffs need to be explored between the degree of application-specific requirements and the generality of the middleware. A practical policy is to embed the unique features of an application into the application code or specification, which can be interpreted by the middleware at runtime.

The third type of challenges comes from the new context of QoS. Because of the limited resources, it is very likely that the performance requirements of all the running applications cannot be simultaneously satisfied. Therefore, it is necessary for the middleware to

smartly trade the QoS of various applications against each other.

Existing work uses different approaches to provide different WSN-middleware services. For example, the database-based approach is more appropriate for data management services, and mobile agent is more appropriate for code management services. In fact, the mobile agent approach can also be used in data management services especially in distributed in-network data processing services. So, how to choose or integrate different approaches, and how to make their advantages together deserves consideration.

With regards to the system architecture, most of the current middleware have not considered how to integrate the components into generic middleware architecture to help developers match different requirements. Because many features need to be considered in the design of WSN middleware, re-factoring WSN middleware with aspects[85] is helpful. Services should be reusable, and a generic framework with customizable component-based architecture is desirable. Component-based architecture has good support for dynamic configuration, and it can make use of the component interface definition to provide slandered service interface. Components need not always exist, and they can be initialized when needed, so as to save the system resource. The relationship between components and services should be carefully considered. Moreover, design patterns are widely used in traditional middleware, but until now, there is very little study on design patterns for WSN middleware[86]. Also, Domain specific design of WSN middleware functions deserves more attention.

Furthermore, evaluation metrics, as well as testing and simulation tools for WSN-middleware are needed.

Finally, WSN middleware can be seen as a kind of lower layer pervasive computing middleware. Until now, all the exiting work on WSN middleware has focused on the sensor networks. In the future, more kinds of sensor nodes, RFID, and camera sensors will put new challenges to this research field. Also, with the maturity of new kinds of sensor networks, such as delay tolerant sensor networks, sensor and actuator networks, and mobile sensor networks, we will face new research issues in developing WSN-middleware, in order to deal with the more complex heterogeneity problems. So, we should consider more integration of WSN-middleware and higher level pervasive computing middleware. WSN middleware focuses on information gathering from the physical world and pervasive computing middleware focus on the use of the information, and the appropriate division of functions and cooperation between these two kinds of middleware are very important for supporting efficient pervasive computing applications.

8 Conclusion

In this paper, we have presented a review of the state of arts of middleware for wireless sensor networks. We described a reference model for WSN middleware architecture as a basis for our discussion. We discussed and evaluated the programming abstractions and middleware services of the model in detail. Using a feature tree-based taxonomy, we classified the features of WSN middleware and compared existing projects and proposals. We also discussed the challenges as well as future research directions in developing WSN middleware.

The main objective of this paper is to provide a comprehensive understanding of the current issues in this area for better future academic research and industrial practice of WSN-middleware. WSN is a new and rapidly evolving field, and advances and new capabilities are constantly introduced into the design of WSN middleware. This paper is presenting what essentially constitutes a "snap shot" of the state of the art around the time of its writing, as is unavoidable the case of any survey of a thriving research field. We do, however, believe that the core information and principles presented will remain relevant and useful to the reader.

References

[1] Akyildiz IF, Su W, Sankarasubramaniam Y, Cayirci E. A survey on sensor networks. *IEEE Communications Magazine*, 2002, 40(8): 102–114.

[2] Cecilia Mascolo, Stephen Hailes. Survey of middleware for networked embedded systems. Technical Report for Project: Reconfigurable Ubiquitous Networked Embedded Systems, University College London, 2005.

[3] Java Soft. Java Remote Invocation specification. Revision 1.5, JDK1.2 Edition, Oct. 1998, http://java.sun.com /j2se/1.4.2/docs/guide/rmi/.

[4] OMG. The common object request broker: Architecture and specification, Rev. 2.2, 1998, http://en.wikipedia.org/ wiki/CORBA.

[5] Licia Capra, Wolfgang Emmerich, Cecilia Mascolo. Middleware for mobile computing. Technical Report, Department of Computer Science, University College London, 2005.

[6] Murphy A L, Picco G P, Roman G C. Lime: A middleware for physical and logical mobility. In *Proc. the 21st Int. Conf. on Distributed Computing Systems*, Orland, USA, May 2001, pp.524–533.

[7] Kay Röomer, Oliver Kasten, Friedemann Mattern. Middleware challenges for wireless sensor networks. *Mobile Computing and Communications Review*, 2002, 6(1): 12–17.

[8] Matthew Wolenetz, Rajnish Kumar, Junsuk Shin. Umakishore Ramachandran. Middleware guidelines for future sensor networks. Technical Report 30332-0280, College of Com-

puting, Georgia Institute of Technology, Atlanta, Georgia, 2004.

[9] Bartolome Rubio, Manuel Diaz, Jose M. Troya. Programming approaches and challenges for wireless sensor networks. In *Proc. the 2nd International Conf. Systems and Networks Communications* (*ICSNC07*), Cap Esterel, French Riviera, France, August 25–31, 2007, p.36.

[10] Jiang Q, Manivannan D. Routing protocols for sensor networks. In *Proc. the 1st IEEE Consumer Comm. and Networking Conf.* (*CCNC 04*), Las Vegas, NV, USA, IEEE Press, 2004, pp.93–98.

[11] Chong C, Kumar S P. Sensor networks: Evolution, opportunities, and challenges. In *Proc. the IEEE*, August 2003, 91(8): 1247–1256.

[12] Sadjadi S M. A survey of adaptive middleware. Technical Report, Software Engineering and Network Systems Laboratory, Department of Computer Science and Engineering, Michigan State University, 2005.

[13] Wassim Masri, Zoubir Mammeri. Middleware for wireless sensor networks: A comparative analysis. In *Proc. the 2007 IFIP International Conference on Network and Parallel Computing*, Dalian, China, Sept. 18–21, 2007, pp.349–356.

[14] Karen Henricksen, Ricky Robinson. A survey of middleware for sensor networks: State-of-the-art and future directions. In *Proc. the Int. Workshop on Middleware for Sensor Networks Table of Contents*, Melbourne, Australia, 2006, pp.60–65.

[15] Salem Hadim, Nader Mohamed. Middleware for wireless sensor networks: A survey. In *Proc. the 1st Int. Conf. Comm. System Software and Middleware* (*Comsware06*), New Delhi, India, Jan. 8–12, 2006.

[16] Hill J, Szewczyk R, Woo A, Hollar S, Culler D, Pister K. System architecture directions for networked sensors. In *Proc. the 9th Int. Conf. Architectural Support for Programming Languages and Operating Systems* (*ASPLOS-IX*), New York, NY, USA, ACM Press, 2000, pp.93–104.

[17] Kay Röme. Programming paradigms and middleware for sensor networks. Technical Report, Institute for Pervasive Computing, ETH Zurich, 2002.

[18] Nuno Costa, António Pereira, Carlos Serõdio. Virtual machines applied to WSN's: The state-of-the-art and classification. In *Proc. the 2nd International Conference on Systems and Networks Communications* (*ICSNC 07*), Cap Esterel, French, Riviera, France, August 25–31, 2007.

[19] Kang K, Cohen S, Hess J, Nowak W, Peterson S. Feature-oriented domain analysis (FODA) feasibility study. Technical Report, CMU/SEI-90-TR-21, Pittsburgh, Software Engineering Institute, Carnegie Mellon University, PA, 1990.

[20] Levis P, Culler D. Mate: A tiny virtual machine for sensor networks. In *Proc. the 10th Int. Conf. Architectural Support for Programming Languages and Operating Systems* (*ASPLOS-X*), San Jose, USA, ACM Press, 2002, pp.85–95.

[21] Levis P, Gay D, Culler D. Bridging the Gap: Programming sensor networks with application specific virtual machines. In *Proc. the 6th Symp. Operating Systems Design and Implementation* (*OSDI 04*), San Francisco, USA, 2004, pp.273–288.

[22] Liu T, Martonosi M. Impala: A middleware system for managing autonomic, parallel sensor systems. In *Proc. PPoPP'03*, San Diego, California, USA, June 2003, pp.107–118.

[23] Welsh M, Mainland G. Programming sensor networks using abstract regions. In *Proc. the 1st Usenix/ACM Symp. Networked Systems Design and Implementation* (*NSDI 04*), San Francisco, CA, March, 2004, pp.29–42.

[24] Gummadi R *et al.* Macro-programming wireless sensor networks using kairos. In *Proc. the Int. Conf. Distributed Computing in Sensor Systems* (*DCOSS 05*), Marina del Rey, USA, *LNCS 3560*, Springer, 2005, pp.126–140.

[25] Madden S R, Franklin M J, Hellerstein J M. TinyDB: An acquisitioned query processing system for sensor networks. *ACM Trans. Database Systems*, 2005, 30(1): 122–173.

[26] Bonnet P, Gehrke J, Seshadri P. Towards sensor database systems. In *Proc. the 2nd Int. Conf. Mobile Data Management* (*MDM 01*), Hong Kong, China, 2001, pp.314–810.

[27] Kirsten Terfloth, Georg Wittenburg, Jochen Schiller. FACTS — A rule-based middleware architecture. In *Proc. the IEEE/ACM International Conference on Information Processing in Sensor Networks* (*IPSN*), Los Angeles, USA, 2006.

[28] Fla' Via, Paulo F Pires. A Service Approach for Architecting Application Independent Wireless Sensor Networks. Cluster Computing, Netherlands: Springer Science Business Media, Inc, 8, 2005_C, 2005, pp.211–221.

[29] Fok C, Roman G, Lu C. Mobile agent middleware for sensor networks: An application case study. In *Proc. the 4th Int. Conf. Information Processing in Sensor Networks* (*IPSN 05*), UCLA, Los Angeles, California, USA. Apr. 25–27, 2005, pp.382–387.

[30] Abdelzaher T, Blum B, Cao Q, Evans D, George J, George S, He T, Luo L, Son S, Stoleru R, Stankovic J, Wood A. EnviroTrack: Towards an environmental computing paradigm for distributed sensor networks. In *Proc. the 24th Int. Conf. Distributed Computing Systems* (*ICDCS 04*), Tokyo, Japan, March 23–26, 2004, pp.582–589.

[31] Krishna Chintalapudi, Jeongyeup Paek, Omprakash Gnawali, Tat S Fu, Karthik Dantu, John Caffrey, Ramesh Govindan, Erik Johnson, Sami Masri. Structural damage detection and localization using NETSHM. In *Proc. the 5th Int. Conference on Information Processing in Sensor Networks* (*IPSN 06*), Nashville, TN, USA, April 19–21, 2006, pp.475–482.

[32] Musiani D, Lin K, Simunic Rosing T. Active sensing platform for wireless structural health monitoring. In *Proc. the 5th International Conference on Information Processing in Sensor Networks* (*IPSN 07*), Cambridge, Massachusetts, April 25–27, 2007.

[33] TinyOS communities. TinyOS specification, http://www.tinyos.net.

[34] Chen D, Varshney P K. QoS support in wireless sensor networks: A survey. In *Proc. the Int. Conf. Wireless Networks* (*ICWN 04*), Las Vegas, Nevada, USA, June 21, 2004.

[35] Mohsen Sharifi, Majid Alkaee Taleghan, Amirhosein Taherkordi. A middleware layer mechanism for QoS support in wireless sensor networks. In *Proc. the Int. Conf. Networking, Int. Conf. Systems, and Int. Conference on Mobile Communications and Learning Technologies* (*ICNICONSMCL 06*), Mexico, 2006, p.118.

[36] Mauri Kuorilehto, Marko Hännikäinen, Timo D Hämäläinen. A survey of application distribution in wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking*, 2005, 38(5): 774–788.

[37] Srisathapornphat C, Jaikaeo C, Shen C. Sensor information networking architecture. In *Proc. the Int. Workshop Parallel*, IEEE CS Press, 2000, pp.23–30.

[38] Heinzelman W B *et al.* Middleware to support sensor network applications. *IEEE Network*, 2004, 18(1): 6–14.

[39] Lifton J, Seetharam D, Broxton M, Paradiso J. Pushpin. Computing system overview: A platform for distributed, embedded, ubiquitous sensor networks. In *Proc. the 1s Int. Conference on Pervasive Computing* (*Pervasive'02*), Zurich, Switzerland, August 2002, pp.139–151.

[40] Barr R *et al.* On the need for system-level support for ad hoc and sensor networks. *Operating Systems Review,* 2002, 36(2): 15.

[41] Athanassios Boulis, Chih-Chieh Han, Mani B Srivastava. Design and implementation of a framework for efficient and programmable sensor networks. In *Proc. the First International Conference on Mobile Systems, Applications, and Services* (*MobiSys 03*), SAN Francisco, CA, USA, May 5–8, 2003, pp.187–200.

[42] Carlo Curino, Matteo Giani, Marco Giorgetta, Alessandro Giusti. TinyLIME: Bridging mobile and sensor networks through middleware. In *Proc. the 3rd IEEE Int. Conf. Pervasive Computing and Communications*, Kauai Island, Hawaii, March 8–12, 2005.

[43] Li S, Son S, Stankovic J. Event detection services using data service middleware in distributed sensor networks. In *Proc. the 2nd Int. Workshop Information Processing in Sensor Networks* (*IPSN 03*), Palo Alto, California, USA, April 22–23, 2003, pp.502–517.

[44] Souto E *et al.* A message-oriented middleware for sensor networks. In *Proc. the 2nd Int. Workshop Middleware for Pervasive and Ad-Hoc Computing* (*MPAC 04*), Toronto, Canada, Oct. 2004, ACM Press, pp.127–134.

[45] Madden S, Szewczyk R, Franlin M J, Culler D. Supporting aggregate queries over ad-hoc wireless sensor networks. In *Proc. the IEEE Workshop on Mobile Computing Systems and Applications*, 2002.

[46] Madden S *et al.* TAG: A tiny aggregation service for ad-hoc sensor networks. In *Proc. the 5th Symp. Operating Systems Design and Implementation* (*OSDI 02*), Boston MA, ACM Press, 2002, pp.131–146.

[47] Madden S, Frankin M J. Fjording the stream: An architecture for queries over streaming sensor data. In *Proc. the 18th International Conference on Data Engineering* (*ICDE 02*), San Jose, California, USA, IEEE Computer Press, 2002, pp.555–666.

[48] Madden S, Shah M, Hellerstein J M, Raman V. Continuously adaptive continuous queries over streams. In *Proc. the ACM SIGMOD Int. Conf. Management of Data*, Madison, USA, Jun, 2002, pp.49-60.

[49] Ratnasamy S, Estrin D *et al.* Data-centric storage in sensornets. In *Proc. 1st ACM Int. Workshop on Wireless Sensor Networks and Applications* (*WSNA 02*), Alanta, GA, September 2002, pp.78–87.

[50] Sameer Tilak, Nael B. Abu-Ghazaleh, Wendi Heinzelman. Collaborative storage management in sensor networks. *International Journal of Ad Hoc and Ubiquitous Computing*, 2005, 1(1/2): 47–58.

[51] Hector A Duran-Limon, Gordon S Blair, Geoff Coulson. Adaptive resource management in middleware: A survey. *Distributed Systems Online*, IEEE, July 2004, 5(7).

[52] Toni A Bishop, Ramesh K Karne. A survey of middleware. Technical Report, Computer & Information Science Dept., Towson University Towson University, 2000.

[53] Guttman E, Perkins C, Veizades J, Day M. Service location protocol, version 2, IETF, RFC 2608, June 1999. http://en.wikipedia.org/wiki/Service_Location_Protocol

[54] Bluetooth Special Interest Group. Bluetooth specification, version 1.1, Feb. 2001, http://en.wikipedia.org/wiki/Bluetooth.

[55] Bharath Sundararaman, Ugo Buy, Ajay D Kshemkalyani. Clock synchronization for wireless sensor networks: A survey. Technical Report, Department of Computer Science, University of Illinois at Chicago, 2005.

[56] Yao-Chung Chang, Z-S L, Jiann-Liang Chen. Cluster based self-organization management protocols for wireless sensor networks. *IEEE Trans. Consumer Electronics*, 2006, 52(1): 75–80.

[57] Mardoqueu Souza Vieira, N.S.R. A reconfigurable group management middleware service for wireless sensor networks. In *Proc. the 3rd International Workshop on Middleware for Pervasive and Ad-Hoc Computing*, Grenoble, France, 2005.

[58] Han Q, Venkatasubramanian N. Autosec: An integrated middleware framework for dynamic service brokering. *IEEE Distributed Systems Online*, 2001, 2(7).

[59] Kwang-il Hwang, J I NhoKyung Park, Doo-seop Eom. A design and implementation of wireless sensor gateway for efficient querying and managing through world wide web. *IEEE Trans. Consumer Electronics*, 2003, 49: 1090–1097.

[60] Vipul Hingne, Anupam Joshi, Elias Houstis, John Michopoulos. On the grid and sensor networks. In *Proc. the IEEE 4th International Workshop on Grid Computing* (*GRID 03*), Phoenix, Arizona, USA, Nov. 17, 2003, p.166.

[61] Mark Gaynor, Steven L Moulton, Matt Welsh, Ed LaCombe, Austin Rowan, John Wynne. Integrating wireless sensor networks with the grid. *Distributed Systems Online*, IEEE Computer Society, July–August, 2004.

[62] Jan Humble, Chris Greenhalgh, Alastair Hamsphire, Henk L Muller, Stefan Rennick Egglestone. A generic architecture for sensor data integration with the grid. *Lecture Notes in Computer Science* (*LNCS*) 3458, Herrero P, Pérez M S, Robles V (eds.), Berlin Heidelberg: Springer-Verlag, 2005, pp.99–107.

[63] Chen-Khong Tham, Rajkumar Buyya. SensorGrid: Integrating sensor networks and grid computing. Techical Report (61), National University of Singapore, 2003.

[64] Manish Kushwaha, Isaac Amundson, Xenofon Koutsoukos, Sandeep Neema, Janos Sztipanovits. OASiS: A programming framework for service-oriented sensor networks. Technical Report, Institute for Software Integrated Systems (ISIS), Vanderbilt University Nashville, TN 37235, USA, 2007.

[65] Sean Rooney, Daniel Bauer, Paolo Scotton. Edge Server Software Architecture for Sensor Applications, In Proc. of the Symposium on Applications and the Internet (SAINT 05) Trento, Italy, 31 Jan.–4 Feb. 2005.

[66] W3C (World Wide Web Consortium) Note. Web Services Description Language (WSDL) http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[67] Nickerson B G, Sun Z. A sensor web language for mesh architectures. In *Proc. the 3rd Annual Communication Networks and Services Research Conference*, Halifax, Canada, May 16–18, 2005.

[68] Vincent Tao, Steve Liang, Arie Croitoru, Zia Moin Haider, Chris Wang. GeoSWIFT: Open Geospatial Sensing Services for Sensor Web. GeoSensor Networks, Anthony Stefanidis and Silvia Nittel, CRC Press, Volume 1, Part 5, June 2005, pp.267–274.

[69] Reichardt M. Sensor web enablement: An OGC white paper. Open Geospatial Consortium (OCG), 2005 http://www.crisisgrid.org/html/ogc-swe.html.

[70] Ruiz L B, Siqueira I G, Oliverira L B. Fault management in event-driven wireless sensor networks. In *Proc. the 7th ACM/IEEE Int. Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, Italy, 2004.

[71] Xuanwen Luo, Ming Dong, Yinlun Huang. On distributed fault-tolerant detection in wireless sensor networks. *IEEE Trans. Computers*, Jan. 2006, 55(1): 58–70.

[72] Gregory Chockler, Murat Demirbas, Seth Gilbert, Calvin Newport. A middleware framework for robust applications in wireless ad hoc networks. Technical Report, MIT Computer Science and Artificial Intelligence Laboratory Cambridge, MA 02139, USA. 2006.

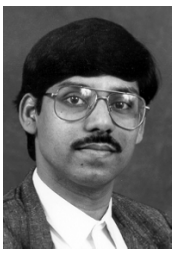
[73] Vana Kalogeraki. Middleware for reliable real-time sensor data management. *Lecture Notes in Computer Science*

(*LNCS*) 4125, Moro G *et al.* (eds.), Berlin/Heidelberg: Springer, 2007, pp.235–246.
[74] Perrig A, Szewczyk R, Tygar J, Wen V, Culler D. SPINS: Security protocols for sensor networks, wireless networks, *ACM Wireless Network*, 2002, 8: 521–534.
[75] Jeffery Undercoffer *et al.* On security for sensor networks. CADIP Research Symposium, Technical Presentations, Oct. 25–26, 2002, http://www.cs.umbc.edu/cadip.
[76] C. Karlof, N. Sastry, and D. Wagner. TinySec: Security for TinyOS. Presentation given at NEST group meeting, Nov. 21, 2002.
[77] Felemban E *et al.* Probabilistic QoS guarantee in reliability and timeliness domains in wireless sensor networks. In *Proc. the IEEE INFOCOM*, Miami, March 13∼17, 2005, Vol.4, pp.2646–2657.
[78] Stankovic J A *et al.* Real-time communication and coordination in embedded sensor networks. In *Proc. IEEE*, 2003, 91(7): 1002–1022.
[79] Luca Caviglione, Franco Davoli. Peer-to-peer middleware for bandwidth allocation in sensor networks. *IEEE Communication Letters*, February 2005, 9(3): 285–287.
[80] Younis M *et al.* On handling QoS traffic in wireless sensor networks. In *Proc. the 37th Hawaii Int. Conf. System Sciences*, 2004, 40(8): 102–116.
[81] Isabelle Rouvellou, Stanley M Sutton Jr, Stefan Tai. Multidimensional separation of concerns in middleware. Technical Report, IBM T. J. Watson Research Center, New York, USA, 2001.
[82] René Meier, Vinny Cahill. Taxonomy of distributed event-based programming systems. Technical Report, Department of Computer Science, Trinity College Dublin, Ireland, 2004.
[83] Delicato F C *et al.* Reflective middleware for wireless sensor networks. In *Proc. the 20th Annual ACM Symposium on Applied Computing* (*ACM SAC*), USA, 2005, pp.1155–1159.
[84] Sameer Tilak, Nael B Abu-Ghazaleh, Wendi Heinzelman. A taxonomy of wireless micro-sensor network models. Technical Report, Computer System Research Laboratory, Dept. of CS, Binghamton University Binghamton, NY, 2005.
[85] C. Zhang and H. A. Jacobsen. Aspectizing middelware systems. Technical Report CSRG-466, University of Toronto, 2003.
[86] Kenji Tei, Yoshiaki Fukazawa, Shinichi Honiden. Applying design patterns to wireless sensor network programming. Technical Report, Waseda University, Okubo, 2007.

Miao-Miao Wang is a Ph.D. candidate majoring in computer science at the University of Science and Technology (USTC). Now she is a research assistant in the Department of Computing at Hong Kong Polytechnic University. Her current research interest includes: wireless sensor network, software architecture, and pervasive computing.

Jian-Nong Cao received the Ph.D. degree in computer science from Washington State University, USA, 1990. He is a professor in the Department of Computing at Hong Kong Polytechnic University. He is the coordinator in Asia of Technical Committee on Distributed Computing (TPDC) of IEEE Computer Society. He is a vice chairman and member of the Technical Committee on Computer Architecture, an executive member of the Technical Committee on System Software, and a senior member of China Computer Federation. He is a senior member of the IEEE (including the IEEE Computer Society and the IEEE Communication Society), and a member of the ACM. His main research interests include: wireless sensor network, distributed computing, and middleware.

Jing Li received the Ph.D. degree in computer science from University of Science and Technology of China (USTC) in 1993. He is a guest professor in USTC. His main research interests include: software architecture, distributed object technology, network distributed computing, middleware and Web technology.



Sajal K. Das received his Ph.D. degree in computer science from University of Central Florida in 1988. He is the director of Center for Research in Wireless Mobility and Networking in Department of Computer Science and Engineering at University of Texas. He is a member of Association for Computing Machinery (ACM), ACM Special Interest Group in Mobile Computing (SIGMOBILE), Institute of Electrical and Electronics Engineers (IEEE), IEEE Computer Society, IEEE Technical Committee on Parallel Processing (TCPP), IEEE Technical Committee on Communications (TCCC), and IEEE Technical Committee on Supercomputing. His main research interests include: wireless multimedia, sensor networks/computer security, pervasive computing, parallel and distributed processing.