IEEE DISTRIBUTED SYSTEMS ONLINE 1541-4922 © 2006 Published by the IEEE Computer Society
Vol. 7, No. 3; March 2006

Middleware: Middleware Challenges and Approaches for Wireless Sensor Networks

Salem Hadim, *Stevens Institute of Technology*
Nader Mohamed, *Stevens Institute of Technology*

Using middleware to bridge the gap between applications and low-level constructs is a novel approach to resolving many wireless sensor network issues and enhancing application development. This survey discusses representative WSN middleware, presenting the state of the research.

The emerging field of tiny, networked sensors offers an unprecedented opportunity for a wide spectrum of various applications. These tiny sensor nodes are low cost, low power, and easily deployable. When combined, they offer numerous advantages over traditional networks, such as a large-scale flexible architecture, high-resolution sensed data, and application adaptive mechanisms.

However, due to their tight integration to the physical world and the unique characteristics we've mentioned, sensor networks pose considerable impediments and make application development nontrivial. A middleware layer is a novel approach to fully meeting the design and implementation challenges of *wireless sensor network* technologies. Consider WSN middleware as a software infrastructure that glues together the network hardware, operating systems, network stacks, and applications. A complete middleware solution should contain a runtime environment that supports and coordinates multiple applications, and standardized system services such as data aggregation, control and management policies adapting to target applications, and mechanisms to achieve adaptive and efficient system resources use to prolong the sensor network's life.

Some research has surveyed WSNs. Ian Akyildiz and his colleagues focused more on WSN characteristics and challenges.[1] Others investigated potential WSN applications[2-3] and presented different routing protocols.[4] However, none have investigated the current state of research on WSN middleware design and development or provided any previous classification. In this article, we explore relevant middleware projects for WSNs and provide an exhaustive comparative study that expands on our previous work.[5] We devised a classification based on the programming approach used and not on the style of communication, as in traditional middleware for distributed systems. This will give you clearer insight to the current and proven ways to tackle issues on WSN middleware design.

Challenges and design principles

Military applications such as target detection, battlefield surveillance, and counterterrorism originally motivated sensor network applications. However, their advantages over traditional networks resulted in many other potential applications, ranging from infrastructure security to industrial sensing—for example, environment and habitat monitoring,[6] healthcare applications, home automation, and traffic control.

The design and development of a successful middleware layer must address many challenges dictated by WSN characteristics on one hand and the applications on the other.

Managing limited power and resources

The advent in microelectronics technology made it possible to design miniaturized devices on the order of one cubic centimeter.[7] Limited in energy and individual resources (such as CPU and memory), these tiny devices could be deployed in hundreds or even thousands in harsh and hostile environments. In cases where physical contact for replacement or maintenance is impossible, wireless media is the only way for remote accessibility. Hence, middleware should provide mechanisms for efficient processor and memory use while enabling lower-power communication. A sensor node should accomplish its three basic operations—sensing, data processing, and communication—without exhausting resources.[7] In energy-aware middleware, for example, most of the device's components (including the radio) are likely turned off most of the time depending on the application.

Scalability, mobility, and dynamic network topology

Scalability is defined as follows: if an application grows, the network should be flexible enough to allow this growth anywhere and anytime without affecting network performance. Efficient middleware services must be capable of maintaining acceptable performance levels as the network grows. Network topology is subject to frequent changes owing to factors such as malfunctioning, device failure, moving obstacles, mobility, and interference. Middleware should support sensor networks' robust operation despite these dynamics by adapting to the changing network environment. Middleware also should support mechanisms for fault tolerance and sensor node self-configuration and self-maintenance.

Heterogeneity

Middleware should provide low-level programming models to meet the major challenge of bridging the gap between hardware technology's raw potential and the necessary broad activities such as reconfiguration, execution, and communication. It should establish system mechanisms that interface to the various types of hardware and networks, supported only by distributed, primitive operating-system abstractions.

Dynamic network organization

Unlike traditional networks, sensor networks must deal with resources that are dynamic, such as energy, bandwidth, and processing power.[3] Sensor networks also must support long-running applications, so routing protocols must be efficiently designed to enable the network to run as long as possible.[4] Because knowledge of the network is essential for it to operate properly, the middleware should provide ad hoc network resource discovery. A sensor node needs to know its location in the network and in the whole network topology. In some cases, self-location by GPS is impossible, unfeasible, or expensive. Important system parameter issues, such as network size and density per square mile, affect the trade-offs among latency, reliability, and energy.

Real-world integration

Most sensor network applications are real-time phenomena, where time and space are extremely important. Hence, middleware should provide real-time services to adapt to the changes and provide consistent data.

Application knowledge

Application knowledge's design principles dictate another important and unique property of WSN middleware.[8] Middleware must include mechanisms for injecting application knowledge of WSN's infrastructure. This lets developers map application communication requirements to network parameters, which enable them to fine-tune network monitoring. Much existing middleware is coupled to specific applications. However, middleware is intended support a wide range of applications. So, developers must explore the trade-offs between the degree of application specificity and middleware generality.

Data aggregation

Most sensor network applications involve nodes that contain redundant data and are located in a specific local region. These traits open the possibility for in-network aggregation of data from different sources, eliminating redundancy and minimizing the number of transmissions to the sink. This aggregation saves considerable energy and resources, given that communications cost is much higher than computation costs. This paradigm shifts the focus from the traditional address-centric approaches for networking to a more data-centric approach.[9]

Quality of service

Quality of service is an overused term with multiple meanings and perspectives from different research and technical communities.[10] In WSNs, we can view QoS from two perspectives: *application-specific* and *network*. The former refers to QoS parameters specific to the application, such as sensor node measurement, deployment, and coverage and number of active sensor nodes. The latter refers to how the supporting communication network can meet application needs while efficiently using network resources such as bandwidth and power consumption.

Traditional QoS mechanisms used in wired networks aren't adequate for WSNs because of constraints such as resource limitations and dynamic topology. Therefore, middleware should provide new mechanisms to maintain QoS over an extended period and even adjust itself when the required QoS and the state of the application changes. Middleware should be designed based on trade-offs among performance metrics such as network capacity or throughput, data delivery delay, and energy consumption.

Security

WSNs are being widely deployed in domains that involve sensitive information—for example, healthcare and rescue. The untethered and large deployment of WSNs in harsh environments increases their exposure to malicious intrusions and attacks such as denial of service.[11] In addition, the wireless medium facilitates eavesdropping and adversarial packet injection to compromise the network's functioning. All these factors make security extremely important. Furthermore, sensor nodes have limited power and processing resources, so standard security mechanisms, which are heavy in weight and resource consumption, are unsuitable. These challenges increase the need to develop comprehensive and secure solutions that achieve wider protection, while maintaining desirable network performance. Middleware efforts should concentrate on developing and integrating security in the initial phases of software design, hence achieving different security requirements such as confidentiality, authentication, integrity, freshness, and availability.

Classification of middleware approaches for WSN

For this survey, we selected research projects and design principles on the basis of how innovative they are in supplying new concepts and solutions for WSN requirements. Furthermore, we propose a classification based on the programming models they used.

Programming sensor networks raises issues that generally fall into two broad classes: *programming support* and *programming abstractions* (see figure 1). The first is concerned with providing systems, services, and runtime mechanisms such as reliable code distribution, safe code execution, and application-specific services. The second refers to the way we view a sensor network and provides concepts and abstractions of sensor nodes and sensor data. Although all subcategories of these two classes differ in their objectives and the way they deal with programming a WSN as a whole, the research is still maturing, so these subcategories might have some desired features in common.

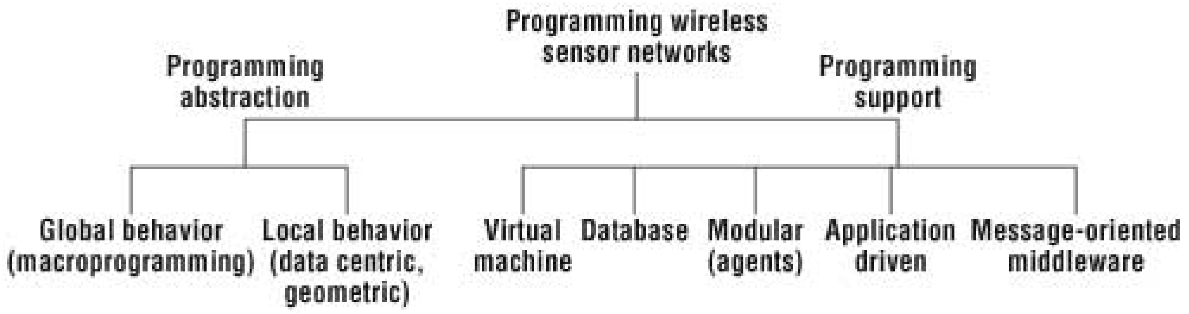

Programming wireless sensor networks
Programming abstraction
Programming support
Global behavior (macroprogramming)
Local behavior (data centric, geometric)
Virtual machine
Database
Modular (agents)
Application driven
Message-oriented middleware

Figure 1. Taxonomy of programming models for wireless sensor networks.

Programming support

In the programming support class, we identify five main subclasses: virtual machine based, modular programming based, database based, application driven, and message-oriented middleware.

Virtual machine. This system is flexible and contains virtual machines (VMs), interpreters, and mobile agents. It lets developers write applications in separate, small modules. The system injects and distributes the modules through the network using tailored algorithms, such as overall energy consumption and resource use are minimized. The VM then interprets the modules. This approach, however, suffers from the overhead that the instructions introduce.

Mate WSN middleware is among those that use a VM approach as an abstraction layer to implement its operation and tackle the challenges we've mentioned.[12,13] The project focuses on the need for new programming paradigms to overcome constraints such as limited bandwidth and network activities' large energy draw. Mate proposes a spectrum of reprogrammability, from adjusting simple parameters to uploading complete program updates using a VM. Sending a single bit of data can consume the same energy as executing thousands of instructions to produce that bit of data. The VM can use and support a content-specific routing and reprogramming model.

Mate is a byte code interpreter that runs on TinyOS.[14,15] TinyOS is the de facto operating system for sensor networks that run on *motes* (small devices with a small CPU and limited storage resources). Written using the NesC language,[16] TinyOS adopts a component-based model to build sensor network applications in an event-driven operating environment. TinyOS binds components statistically and then compiles them to a binary image that's programmed into a sensor node's flash memory. In TinyOS, communication between nodes is based on the active messages paradigm. According to this paradigm, each message contains the ID of a handle to be invoked on the target node and a data payload to pass as an argument.

Mate uses codes broken into capsules of 24 byte-long instructions. This benefits large programs, which are made up of multiple capsules and, thus, easily injected into the network. Mate's key components are the VM, the network, the logger, the hardware, and the boot/scheduler. Using a synchronous model that begins execution in response to an event such as a packet transmission or a time out, Mate avoids message buffering and large storage. The synchronous model makes application-level programming simpler and far less prone to bugs than dealing with asynchronous event notifications. Another Mate functionality is infection or network updates achieved by adding a version number to the capsule. So, comparison takes place at the neighbors, followed by installation of the new version. This process cascades with hop-to-hop communication.

Magnet is also in the middleware category that uses a VM.[17] MagnetOS is a power-aware, adaptive operating system specially designed for sensor and ad hoc networks. It constitutes a layer defined as *Single System Image*, which provides a higher abstraction for the heterogeneity of ad hoc networks' distributed nature. The abstraction lets the whole network appear as a single, unified Java VM. Following the JVM pattern, the system comprises dynamic and static components. The static component is responsible for rewriting regular java applications in the form of objects or modules, which explains MagnetOS's object-oriented nature. Then the component injects them into the network with special instructions to keep the semantics. At this point, a dynamic runtime component on each node monitors the object's creation, invocation, and migration, providing different services for applications.

For performance purposes, MagnetOS runtime provides flexibility for programmers to explicitly adjust object placement and migration. This lets them reduce network communication by moving objects closer to data sources, for instance. MagnetOS provides a robust, power-aware algorithm using object migration of the same application to nodes that are topologically closer together. This mechanism reduces application energy consumption and increases longevity.

Modular programming (mobile agents). The key to this approach is that applications are as modular as possible to facilitate their injection and distribution through the network using mobile code. Transmitting small modules consumes considerably less energy than a whole application. For example, *Impala*[18] provides mechanisms for network updates that are efficient enough to support dynamic applications. Its autonomic behavior increases its fault tolerance and network self-organization. However, the nature of its code instruction doesn't allow hardware heterogeneity, which makes it unsuitable for devices with limited resources.

The insight for Impala stems from the observation that sensor networks are long running and autonomous. Impala was specially designed as part of the wildlife-watch project ZebraNet. It proposes an asynchronous, event-based middleware layer that uses program modules (mobile agents) compiled into binary instructions then injected into the network. The approach ensures application adaptation and can automatically discern needed parameter settings or software uses. Programmers can plug in new protocols at anytime and switch between protocols at will.

The middleware itself is separated into two layers. The upper layer contains all of the ZebraNet applications and protocols. These applications use various strategies to achieve the common task of gathering environment information and routing it to a base station. The lower layer contains three middleware agents: event filter, adapter, and updater. The event filter controls different operations and initiates processing chains: timer, packet, send, done data, and device events. Armed with the Application Finite State Machine (AFSM), the adapter agent handles application adaptation on the basis of different scenarios, such as energy efficiency and other attributes the applications determine. The updater agent is in charge of achieving effective software updates with resource constraints by taking into account trade-offs such as high node mobility, constrained bandwidth, wide range of updates, propagation protocol, and code memory management.

Impala adopts a module-based system with a version number. Each application as a whole also has a version number. Before exchanging software updates, nodes exchange an index of application modules. They then request only the changed modules for transmission, which saves network bandwidth. The system compiles program modules into binary instructions before injecting them into the network. Impala won't link a module to the main program for installation until it receives the whole update.

Database. This approach views the whole network as a virtual database system. It provides an easy-to-use interface that lets the user issue queries to the sensor network to extract the data of interest. However, the approach provides only approximate results, and it lacks the support of real-time applications that need the detection of spatio-temporal relationships between events.

Cougar introduces a new dimension in middleware research by adopting a database approach in which sensor data are considered a virtual relational database.[19] Cougar (http://www.cs.cornell.edu/database/cougar) implements WSN management operations in the form of queries, using an SQL-like language. The system defines a sensor database system comprising sensor database and sensor queries. The sensor database, in its turn, contains stored data and sensor data. Cougar represents stored data as relations that include the set of sensors that participate in the sensor database and the sensors' or physical environment's characteristics. Signal-processing functions generate the sensor data, which is represented as time series to facilitate sensor query formulation. Cougar then uses *abstract data types* with virtual relations to model the signal processing functions. Because object-relational databases support ADT, Cougar represents these signal-processing functions as sequences. An ADT represents all sensors of the same type in the physical world. With algebra operators, the system formulates the sensor queries as SQL-like language with little modifications. To maintain a persistent view of long running queries, Cougar supports such queries using incremental results.

TinyDB is a query-processing system for extracting information from a network of sensor devices using TinyOS as an operating system.[20] As we mentioned, other solutions using TinyOS generally force the user to write embedded C code (NesC) to extract sensor data. However, TinyDB relieves the user from this complexity by providing an easy, SQL-like interface for extracting the data of interest from sensor nodes with limited power and hardware resources.

The queries use simple data manipulation to indicate the type of readings, such as light and temperature, as well as the subset nodes of interest. For that purpose, TinyDB maintains a virtual database table (called SENSORS), where columns contain information such as sensor type, sensor node identifier, and remaining battery power. We can view reading out the sensors at a node as adding a row to SENSORS. Consider the following example: A user wants to report when the average temperature is above 80° F in any room on the third floor of a building sensors that are monitoring. The user inputs the following database query along with the rate at which the sensors are to collect 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 send 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 sensor data and maintains all routing information. The parent node closer to the root agrees with its children on a time interval for listening for data from them. The whole process repeats for every period and query.

SINA (System Information Networking Architecture) models the network as massively distributed objects.[21] SINA is cluster-based middleware, and its kernel is based on a spreadsheet database for querying and monitoring. Each logical datasheet comprises cells, and each cell represents a sensor node attribute (in the form of a single value, such as power level and location, or multiple values, such as temperature changes history). Each cell is unique, and each sensor node maintains the whole datasheet. The sensor network as whole is a collection of datasheets. The spreadsheet approach is the abstraction that allows information management to meet application changes and needs. SINA, then, incorporates two robust mechanisms: hierarchical clustering allowing scalability and energy savings, and an attribute-based naming scheme based on an associative broadcast to manage the spreadsheets. The cells are initiated in a node by requests from other nodes. The nodes make requests in a SQL-like statement. The nodes maintain the cells through four possible approaches: on-demand content retrieval, content coaching, periodic content update, and triggered content update.

DsWare (Data Service Middleware) is a database-like abstraction approach tailored to sensor networks on the basis of event detection.[22] DsWare provides more flexibility by supporting group-based decision-making and reliable data-centric storage. It improves real-time execution performance and aggregated results' reliability and reduces network communication (overhead). DsWare provides applications with services supported by its architecture modules such as data storage, data caching, group management, event detection, data subscription, and scheduling. Like Cougar, DsWare uses an SQL-like language for registering and canceling events.

Application driven. This approach introduces a new dimension in middleware design by supplementing an architecture that reaches the network protocol stack. This will let programmers fine-tune the network on the basis of application requirements—that is, applications will dictate network operations management, providing a QoS advantage. However, the tight coupling with applications might result in specialized, not general purpose, middleware.

Milan's (Middleware Linking Applications and Networks) masterpiece is its focus on high-level concerns by providing an interface that's mainly characterized by applications that actively affect the entire network.[8] Milan lets sensor network applications specify their quality needs and adjusts the network characteristics to increase application lifetime while still meeting those needs. To accomplish that, it receives

- the individual applications' QoS requirements over time and a way to meet these QoS requirements using different sensor combinations,
- the different applications' relative importance to the overall system and the user, and
- the network's available sensors and resources, such as energy and channel bandwidth.

With specialized graphs incorporating state-based changes in application needs, Milan can determine the adequate combination (F_A) of sensors satisfying an application's QoS requirements. Then, with its architecture that extends into the network protocols stack and an abstraction layer that lets network-specific plug-ins convert commands to protocol-specific commands, Milan can configure and manage the network. Depending on the application feasible set (F_A), the network plug-ins determine which sets of sensor nodes (F_N) best meet the requirement, as well as other information such as each node's role. Finally, Milan combines the two constraints to get an overall set of feasible combinations: $F = F_A$ F_N.

Message-oriented middleware. Mainly a communication model in a distributed-sensor network, message-oriented middleware uses the publish-subscribe mechanism to facilitate message exchange between nodes and the sink nodes. The strength of this paradigm lies in that it supports asynchronous communication very naturally, allowing a loose coupling between the sender and the receiver. This approach is quite suitable in pervasive environments such as wireless sensor networks, where most applications are based on events.

Mires proposes an adaptation of a message-oriented middleware for traditional fixed distributed systems.[23] Mires provides an asynchronous communication model that's suitable for WSN applications, which are event driven in most cases, and has more advantages over the traditional request-reply model. Mires is built on TinyOS using NesC. It adopts a component-based programming model using active messages to implement its publish-subscribe-based communication infrastructure.

Mires' architecture includes a core component (a publish-subscribe service), a routing component, and some additional services, such as data aggregation. The publish-subscribe service coordinates the communication between middleware services. Communication occurs in three phases. First, the nodes in the network advertise their sensed data (topic) through the publish service. Next, Mires routes the advertised messages to the sink, using the multi-hop routing algorithm. Finally, the user application subscribes to topics of interest using only a suitable GUI. The publish-subscribe service also maintains the topics list and the subscribed application to marshal the right topic to the related application. Mires sends only messages referring to subscribed topics, hence reducing the number of transmissions and energy consumption. It includes a data aggregation service that lets the user specify how data will be aggregated and the association between topics of interest and aggregates.

Other approaches. Other middleware research efforts also provide programming support in the sensor network community. However, most of them fall into one of the approaches we've mentioned, and they use similar mechanisms. These efforts include AutoSec, Agilla, Garnet, and SOS.

AutoSec (Automatic Service Composition) is an application-driven middleware framework that focuses on providing support for dynamic service brokering.[24] In a distributed system, this focus allows better resource use. Distributed applications have QoS requirements that programmers can translate into the underlying system-level resources. AutoSec has an architecture that goes deep in the network. It performs resource management and organization in a sensor network by providing the required QoS for applications on per-sensor basis. It does this by choosing a combination of information-collection and resource-provisioning policies from a given set on the basis of user and system needs.

Agilla is Based on Mate and extends that approach by providing mechanisms for better injection of a mobile code into the sensor network to deploy user application.[25] Mobile agents can intelligently move or clone themselves into desired locations based on network changes. This method is more suitable than the flooding mechanisms Mate uses for the same purpose.

Garnet is a middleware framework that focuses on managing data streams as an abstraction within a sensor network.[26] Garnet offers a collection of system services such as receivers, filtering and dispatching services, resource manager, and orphanage.

SOS is considered an operating system specifically designed for motes sensor networks class.[27] However, we judged useful to include it here because it provides some middleware services. Moreover, its kernel includes a sensor API, which makes developing sensor network applications on top of it easy. It's actually a suitable operating system to develop more middleware services for WSN applications. SOS's motivation stems from the need for a general purpose operating system that allows dynamic configurability. This lets programmers dynamically change software modules in sensor nodes after they have deployed and initialized the network.

SOS offers incremental software updates, addition of new software modules, and removal of unnecessary or unused modules. These functions should be done with minimal energy and resources consumption, which is critical in sensor nodes. TinyOS adopts a single system image that's statistically linked at compile time, where there is no kernel mode and user mode. In contrast, SOS adopts a more general-purpose approach by having its architecture composed of a common kernel and dynamic application modules that can be loaded and unloaded at runtime.

Programming abstractions

The programming abstractions class has two main subclasses (see figure 1). The first focuses on the global behavior of a distributed sensor network as a whole—also called macroprogramming. The second deals with the local behavior of the sensor network nodes in a distributed computation.

Global behavior (macroprogramming). This subclass introduces a new and completely different view on how to program sensor networks. Macroprogramming involves programming the sensor network as a whole, rather than writing low-level software to drive individual nodes. The WSN global behavior is programmed at a high-level specification, enabling automatically generated nodal behaviors. This relieves application developers from dealing with low-level concerns at each network node.

Kairos focuses on providing the necessary notions and concepts to design, develop, and implement a macroprogramming model on WSN.[28] Kairos allows developers to express a single centralized program (global behavior) into subprograms that can execute on local nodes (nodal behavior). It includes compile time and runtime subsystems, 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 internode program flow coordination. Kairos extends to a programming language by providing three main abstractions:

- a node abstraction for manipulating nodes and lists of nodes,
- a list of one-hop neighbors for tracking the current list of the node's radio neighbors, and
- a remote data-access mechanism to read from variables at named nodes.

Kairos addresses node synchronization in either a loose synchronization or a tight synchronization. Programmers choose the method on the basis of trade-offs between efficiency and overhead.

Local behavior. In most sensor network applications, the focus is more on the nature of the sensed data, which generally involves a group of sensor nodes in a specified region. In region or Hood style, the interest is on a specific location in a sensor network—for example, applications are more likely to ask for a location where the temperature exceeds a certain value, rather than for an individual sensor reading. In data-centric sensor networks, nodes are addressed according to the data produced —for example, detect a target having a shape of "tank" in military applications.

Abstract regions[29] stem from the observation that most sensor network applications involve the cooperation of a group of nodes over which local computation, data aggregation, and communication occur. For example, in the military, tracking a mobile target involves aggregating all sensor readings from nodes near the object to generate accurate information on the characteristics of that target. Based on this observation, abstract regions are a suite of general-purpose communication primitives for WSN that provide addressing, in-network data aggregation,[8] and data sharing and reduction in the network's local regions. This serves an efficient method for energy and bandwidth savings by trading increased local computation for reduced radio communication, which is expensive in terms of energy. For example, consider a sensor network application for determining the boundaries of a region of interest in a network. The best approach is having nearby nodes cooperate and compute the boundaries locally, then communicate the results to the base station. This yields considerable energy savings and less communication overhead than having the base station collect data centrally on each sensor.

Abstract regions support operations such as neighbor discovery, in which sensor nodes discover each other's enumeration operator to return the set of nodes participating in the region; and data sharing between the nodes in the region and a reduction operator, whose role is to reduce the amount of data shared in a region by performing operations such as `sum`, `max`, or `min` on the sensors data. Due to the wide range of sensor network applications, Matt Welsh and Geoff Mainland propose various abstract regions implementations: radio and geographic neighborhood, approximate planar mesh, and spanning tree.[28] Each of these implementations is suitable for a subset of sensor networks applications. They have proven successful on University of California, Berkeley, motes running on TinyOS.

EnviroTrack (data-centric)[30] is well suited for embedded tracking applications. EnviroTrack adopts a data-centric programming paradigm called *attributed-based naming through context labels*, where the routing and addressing are based on the requested data's content rather than the target sensor node's identity. As with most of the projects, it's also built on top of TinyOS using compiled NesC programs. Its contribution stems from its convenient, robust interface for application developers geared toward tracking the physical environment. Programmers apply the attributed-based naming by associating user-defined entities (context labels) to real physical targets. With this network-abstraction layer, the programmer declares the environmental characteristics, which define the context label of the object to be tracked. Based on this declaration, all sensor nodes that sense the same declared characteristics (object) are aggregated to track that physical target, such as a car or a fire.

With powerful network-management mechanisms such as lightweight group management and group leader election, EnviroTrack supports the dynamic behavior of the tracked targets, such as mobility. Thus, the system detects and reports any moving target's presence, which is very useful for environmental watch and military applications. EviroTrack is one middleware service among a whole set of other middleware services carried out at the University of Virginia under a major initiative in the sensor network community called VigilNet,[31] an integrated sensor network system for surveillance missions.

Other research. Additional research efforts fit in either category. Recent projects adopting a macroprogramming model include

- *Regiment:* a demand-driven functional language that takes a sensor network as a whole,[32]
- *Abstract Task Graph:* a data-driven programming model incorporating novel extensions for distributed sense-and-respond applications,[33] and
- *Semantic Streams:* a mark-up and declarative query language based on the use of semantic services accessed directly by query specifications.[34]

Projects adopting a local-nodal-behavior programming model include

- *Hood:* similar to abstract regions, Hood shows that a neighborhood abstraction of sensor nodes can convey local behaviors adequately.[35] and

- *Generic role assignment:* programmers assign individual sensor nodes user-defined roles in an application-specific manner.[36] (Due to article length constraints, we left out the implementation details of these projects).

Evaluation framework

Before we evaluate the middleware approaches we've mentioned, we'll first define a framework for evaluation on the basis of the following criteria: heterogeneity, scalability, power awareness, mobility, ease of use, and openness. We've already defined the first four criteria in the first section. Ease of use and openness are also critical in the notion of middleware.

Ease of use

Middleware's abstraction level defines its ease of use. In other words, to what degree does its interface relieve the user from dealing with the complex, low-level APIs of heterogeneous resources by providing an easy-to-use single entity of the whole system? Large-scale sensor network applications usually involve various resource types, which increases the complexity of the code and the programming implementation needed to manage all these heterogeneous resources. For example, environmental monitoring and multimedia involve many heterogeneous resources, including network bandwidth, CPU cycles, memory buffers, and storage. Dealing with this heterogeneity will require abstract resource models to alleviate the complexity of coordinating and adapting these diverse resources. Such models should be uniform and offer a user-friendly interface.

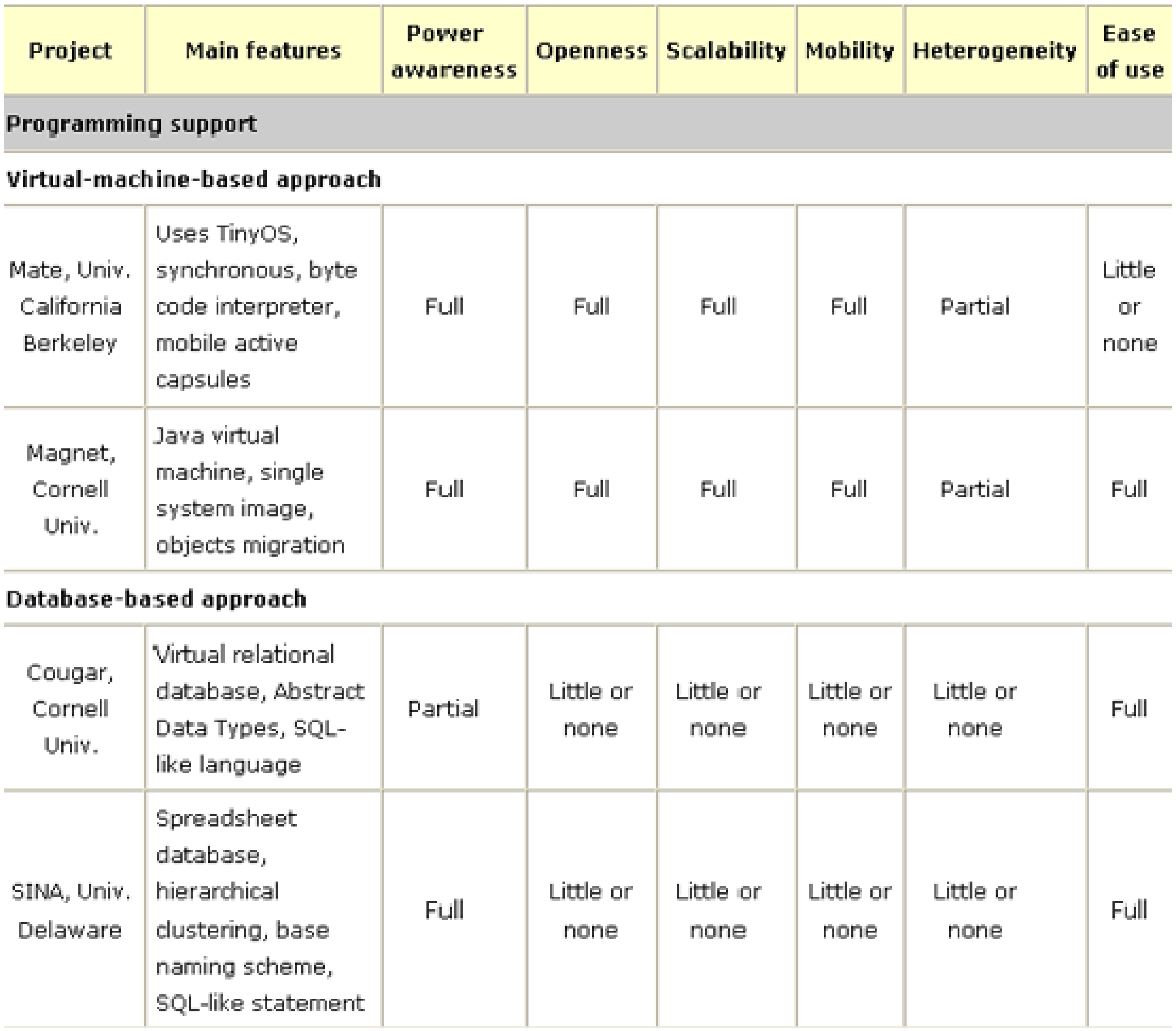
Openness

Openness is the ability to extend and modify the system easily as functional requirements change. Traditional approaches encapsulating and hiding the implementation details result in a "black box" that's difficult to inspect and modify. This should be avoided in middleware design for WSN, where dynamic topology and frequent resource changes are a norm rather than an exception. For example, in devices such as PDA and sensors, which have a limited battery life and limited CPU and memory resources, the system resources should conform to the constraints that the deployment platform and network topology dictate. Resource management in middleware should dynamically adapt to resource availability and other contextual changes. Designing an open middleware prevents stagnation. Standards are essential for open systems and must be continually updated as the environment evolves. Developers must introduce open resource configuration and reconfiguration to achieve the resource-management adaptation that the middleware requires.

Evaluation results

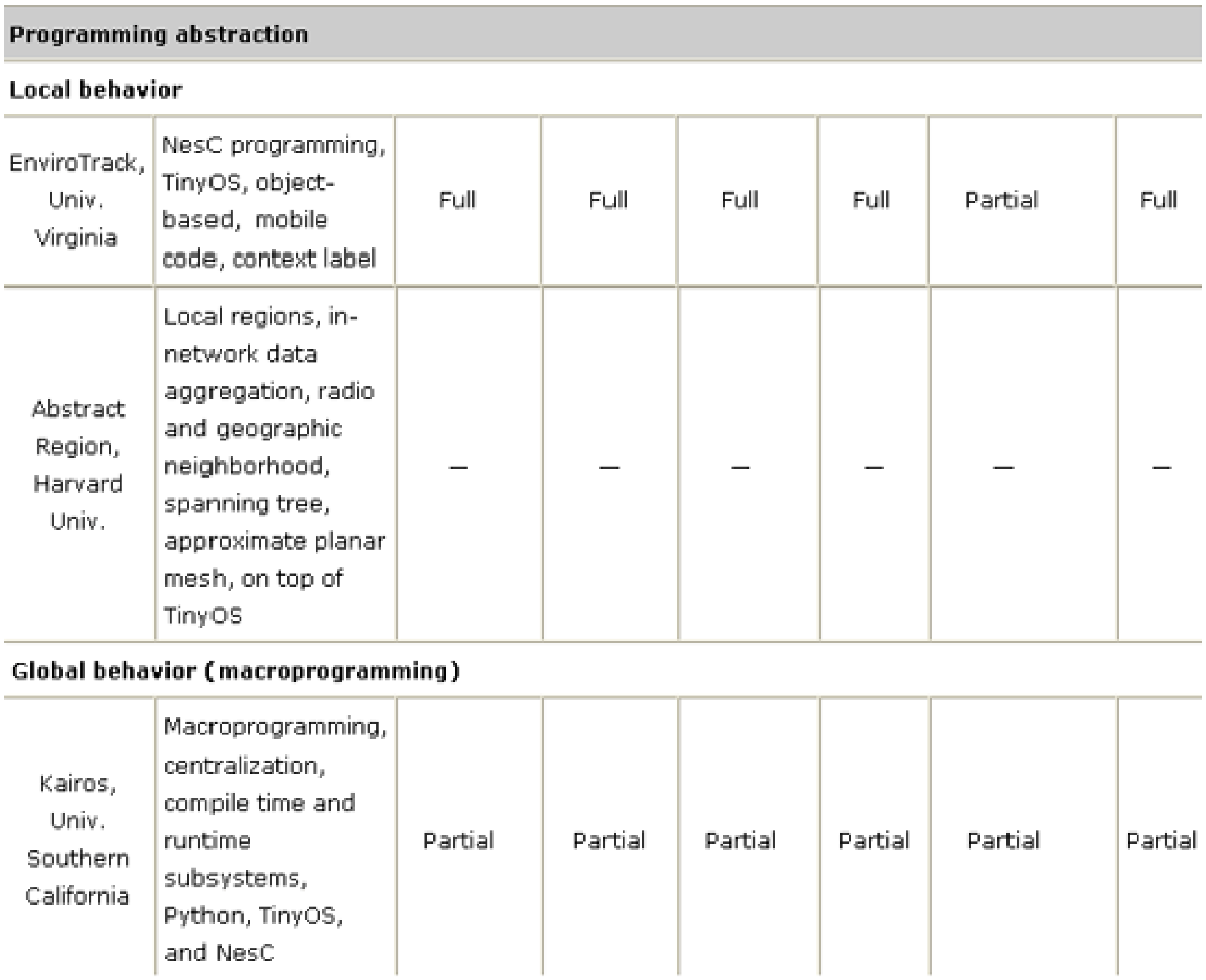
We evaluated each of the middleware by concentrating on how well they meet the framework criteria and the advantages and disadvantages of each approach (see table 1).

Table 1. Approaches to Wireless Sensor Network Middleware.



Project	Main features	Power awareness	Openness	Scalability	Mobility	Heterogeneity	Ease of use
Programming support							
Virtual-machine-based approach							
Mate, Univ. California Berkeley	Uses TinyOS, synchronous, byte code interpreter, mobile active capsules	Full	Full	Full	Full	Partial	Little or none
Magnet, Cornell Univ.	Java virtual machine, single system image, objects migration	Full	Full	Full	Full	Partial	Full
Database-based approach							
Cougar, Cornell Univ.	Virtual relational database, Abstract Data Types, SQL-like language	Partial	Little or none	Little or none	Little or none	Little or none	Full
SINA, Univ. Delaware	Spreadsheet database, hierarchical clustering, base naming scheme, SQL-like statement	Full	Little or none	Little or none	Little or none	Little or none	Full

DsWare, Univ. Virginia	Data service, event detection, SQL-like statement, real-time aspects	Full	Partial	Partial	Little or none	Little or none	Full
TinyDB, Univ. California Berkeley	Virtual database, SQL-like statement, Semantic Routing Trees	Full	Partial	Partial	Partial	Partial	Full
Modular programming approach							
Imapala, Princeton Univ.	Mobile agents, binary instructions, asynchronous, Application Finite State Machine, autonomic approach	Full	Full	Full	Full	Little or none	Full
Application-driven approach							
Milan, Univ. Rochester	High-level concerns, QoS requirement, network protocol stack	Partial	Full	Full	Little or none	Little or none	Full
Message-oriented middleware							
Mires, Univ. Pemambuco	NesC programming, TinyOS, message-oriented, publish-subscribe, asynchronous, active messages	Full	Full	Full	Partial	Partial	Full



Programming abstraction							
Local behavior							
EnviroTrack, Univ. Virginia	NesC programming, TinyOS, object-based, mobile code, context label	Full	Full	Full	Full	Partial	Full
Abstract Region, Harvard Univ.	Local regions, in-network data aggregation, radio and geographic neighborhood, spanning tree, approximate planar mesh, on top of TinyOS	—	—	—	—	—	—
Global behavior (macroprogramming)							
Kairos, Univ. Southern California	Macroprogramming, centralization, compile time and runtime subsystems, Python, TinyOS, and NesC	Partial	Partial	Partial	Partial	Partial	Partial

Mate

Mate aims to provide better interaction and adaptation to the ever-changing nature of sensor networks. It supports openness and scalability by using active messages to update the network protocols and parameters by injecting new capsules. Mate is small and expressive and has concise programs that are resilient to failure. It makes the network dynamic, flexible, and easily reconfigurable. Mate gives a *user-land*—an interface or a system for a user—supplemented by the VM, hence it supports heterogeneity and provides efficient network and sensor access. Mobility is addressed by using various ad hoc routing protocols and protocol updates. However, in terms of energy power and awareness, Mate is only suitable for sleepy applications that are in low-duty cycle most of the time. For complex applications, it's wasteful because of its instructions' interpretation overhead. Also, in its current state, Mate comprises only architecture and byte codes, making it hard to use. A higher-level language and programming model for application development are needed.

MagnetOS

With its 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. Programs are written in a single Java program. The runtime system is then responsible for code partitioning and object placement through the network so the energy consumption is minimized. Supporting openness and scalability, Magnet supplies a dynamic runtime support on each node and service for application monitoring and object creation, invocation, and migration. A suite of ad hoc protocols, such as Ad Hoc On-Demand Distance Vector, provide support for mobility and smart robust object migration policies. Net pull and Net center makes the system more power aware. MagnetOS is quite easy to use, because it stems from a Java implementation. However, it supports only partial heterogeneity, since it uses the Java VM. This factor also introduces a considerable overhead on its instructions, making it unsuitable for motes class sensor networks. A new native Java VM is currently in development.

Impala

Impala provides application adaptation at runtime using a suitable architecture model and ensures security against unfortunate programming errors. The adapter agent assures mobility, openness, and scalability by switching between different protocols and modes of operation depending on the applications and network conditions. Impala uses a novel autonomic approach supplied by the AFSM mechanism to choose and switch between adequate protocols. The key to energy efficiency is for sensor node applications to be as modular as possible to support updates—small mobile agents that introduce only little transmission overhead and energy consumption. However, Impala doesn't support heterogeneity in terms of hardware platforms, since it's being destined to run only on Hewlett-Packard/ Compaq iPAQ Pocket PC handhelds running Linux. So, its applications are limited.

Cougar

The Cougar database approach is very suitable for large sensor collections and offers an easy-to-use database query system for different network operations. However, it uses valuable resources to transfer large amounts of raw data from sensor nodes to the database server. The potential risk for communication links failure in a large-scale sensor network makes Cougar's power awareness limited. Also, it falls short in addressing issues such as openness and scalability, nodes mobility, and hardware heterogeneity, since the dynamic nature of large-scale sensor networks poses a problem for the centralized optimizer that Cougar uses to maintain a global knowledge of the network.

TinyDB

The declarative query system is easy to use. The user is relieved form dealing with low-level APIs in the sensor node, where a network is seen as a single entity and the distribution issues are hidden. The middleware uses TinyOS, which offers development tools for heterogeneity. It also includes support for grouped aggregation queries using the aggregation function. This yields considerable bandwidth and energy savings by reducing the amount of data that must be transmitted through the network. However, TinyDB maintains a network-wide structure using a spanning tree, and it sends queries to all nodes. This yields to some scalability problems since most sensor network applications are local-behavior oriented, where only few nodes are participating at any given time. Openness also is only partially supported because adding a new utility will require modifying the query processor on all sensor nodes. Sensor node mobility partially uses Semantic Rooting Trees (SRTs),[20] but in the event of quick changes, the maintenance cost is too prohibitive.

SINA

In addition to its easy-to-use query-processing database, SINA incorporates two low-level robust mechanisms that offer an advantage over Cougar: a hierarchical clustering of sensors for energy savings, scalability, and some mobility; and an attribute-based naming, making it easy to access sensor data. However, the middleware requires the application layer to provide more services and falls short in full support for openness and hardware heterogeneity. It also maintains a fixed global-network structure, which is unsuitable for scalability and highly dynamic applications.

DsWare

DsWare's interface is similar to conventional database systems, so it's easy to use. It also provides a set of services so that applications don't have to implement their own application data-service. The group management and scheduling components handle stable and dynamic groups of nodes, which either leave or join groups. DsWare can easily tackle failure as long as enough sensors remain in the area to provide valid measurement. Thus, the approach addresses openness and scalability only partially, since highly dynamic applications will require updating the sensor database in each sensor. Furthermore, the middleware includes a data-centric service that processes and aggregates the data and then sends only the result to the sink. This process offers considerable power and bandwidth savings. However, DsWare in its present state doesn't provide solutions for heterogeneity and mobility.

Milan

Milan's application-driven network management is well suited to application adaptation, and it effectively tackles the challenges of openness and scalability. Its easy-to-use approach concentrates on high-level abstractions, selecting the optimal feasible solution and trading off application requirements with network cost—that is, it places emphasis on extending the application's runtime rather than on efficient sensor power use. The middleware uses service-discovery protocols such as Service Discovery Protocol or Service Location Protocol to discover new nodes and notice when nodes become inaccessible. However, because of the tight coupling with the application, the middleware lacks support for operating systems and hardware heterogeneity. In addition, Milan currently doesn't address mobility.

Mires

Mires successfully demonstrates that conventional, easy-to-use message-oriented middleware is applicable in WSN middleware. Its asynchronous communication model uses a publish-subscribe mechanism that confers considerable energy savings. Because Mires is built on top of TinyOS, it supports hardware heterogeneity and openness. TinyOS's event-based and message-oriented nature makes it a suitable foundation on which to build publish-subscribe middleware. Mires addresses scalability and mobility only partially, since the multi-hop routing protocol that Mires uses doesn't include a resource-discovery mechanism. Furthermore, its impact on the network's overall performance must be evaluated.

Kairos

Kairos represents the next step in sensor network programming by adopting a macroprogramming model. At its early stage of development, Kairos doesn't relieve the programmers from having to understand the effect that structuring programs in a particular way has on performance, so it's partially easy to use. Kairos also supports scalability, openness, and power awareness only partially, because applications don't have a full control of the underlying runtime resources for predictability, resource management, and performance. However, it addresses mobility fully, supporting robust mechanisms for node localization and routing.

Abstract regions

This approach by itself is only a suite of general-purpose communication primitives for developing middleware services for sensor network applications. It doesn't constitute complete middleware. Therefore, an evaluation based on our framework is impossible. We included this approach to show the potential it offers as a foundation for WSN middleware.

EnviroTrack

EnviroTrack is data-centric middleware that uses the popular, easy-to-use object-oriented model. Because its main scope is environmental tracking, mobility is a key concern, which its group membership management module addresses at runtime. The preprocessor module interprets user directives and allows appropriate middleware calls at compile time, fully supporting openness and scalability. The middleware is built on TinyOS, which provides adequate development tools, making it power aware and heterogeneous. However, its performance is based on only a very small-scale implantation at its early development stage. Additionally, EnviroTrack isn't general purpose, because it's specifically designed for environmental tracking applications.

Conclusion

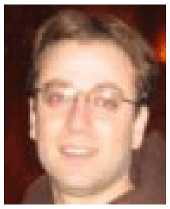
Most of the projects we've mentioned are at an early stage, focusing on developing algorithms and components of WSN middleware. The design of a middleware layer for sensor networks that fully meets the challenges is now open to discussion. One primordial issue is to satisfy application QoS requirements while providing a high-level abstraction that addresses sensor node heterogeneity. Another crucial challenge is developing an easy-to-use, expressive programming interface while meeting different sensor network application challenges, such as limited hardware resources and QoS requirements. Middleware using autonomic computing could provide more robustness, reliability, and self-management. At this point, it's unclear whether successful network management and adequate programming abstractions will stem from the known paradigms of the work we surveyed, or if all-new abstractions and approaches must emerge to specifically meet WSN goals.

References

1. I.F. Akyildiz , et al., "A Survey on Sensor Networks,"*IEEE Communications Magazine,* vol. 40, no. 8, 2002,pp. 102-114.
2. C. Chong and S.P. Kumar , "Sensor Networks: Evolution, Opportunities, and Challenges,"*Proc. IEEE,* vol. 91, no. 8, 2003, pp. 1247-1256.
3. D. Culler , D. Estrin and M. Srivastava , "Overview of Sensor Networks," http://doi.ieeecomputersociety.org/10.1109/MC.2004.93, *Computer,* vol. 37, no. 8, 2004,pp. 41-49.
4. Q. Jiang and D. Manivannan , "Routing Protocols for Sensor Networks,"*Proc. 1st IEEE Consumer Comm. and Networking Conf.* (CCNC 04), IEEE Press, 2004,pp. 93-98.
5. S. Hadim and N. Mohamed , "Middleware for Wireless Sensor Networks: A Survey," to appear in *Proc. 1st Int'l Conf. Comm. System Software and Middleware* (Comsware 2006), IEEE CS Press, 2006.
6. A. Cerp , et al., "Habitat Monitoring: Application Driver for Wireless Communication Technology,"*Proc. ACM SIGCOMM Workshop Data Comm.,* ACM Press, 2001, pp. 20-41.
7. D. Estrin , et al., "Next Century Challenges: Scalable Coordination in Sensor Networks,"*Proc. 5th Ann. ACM/IEEE Int'l. Conf. Mobile Computing and Networking* (MobiCom 99), ACM Press, 1999,pp. 263-270.
8. W.B. Heinzelman , et al., "Middleware to Support Sensor Network Applications,"*IEEE Network,* vol. 18, no. 1, 2004,pp. 6-14.
9. B. Krishnamachari , D. Estrin and S. Wicker , "Impact of Data Aggregation in Wireless Sensor Networks," http://doi.ieeecomputersociety.org/10.1109/ICDCSW.2002.1030829, *Proc. 22nd Int'l Conf. Distributed Computing Systems* (ICDCSW 02), IEEE CS Press, 2002, pp. 575-578.
10. D. Chen and P.K. Varshney , "QoS Support in Wireless Sensor Networks: A Survey,"*Proc. Int'l Conf. Wireless Networks* (ICWN 04), CSREA Press, 2004, pp. 227-233.
11. A. Perrig , J. Stankovic and D. Wagner , "Security in Wireless Sensor Networks,"*Comm. ACM,* vol. 47, no. 6, 2004, pp. 53-57.
12. P. Levis and D. Culler , "Mate: A Tiny Virtual Machine for Sensor Networks,"*Proc. 10th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS-X), ACM Press, 2002, pp. 85-95.
13. P. Levis , D. Gay and D. Culler , "Bridging the Gap: Programming Sensor Networks with Application Specific Virtual Machines," submitted to *Proc. 6th Symp. Operating Systems Design and Implementation* (OSDI 04), 2004.
14. D. Gay , et al., *nesC 1.1 Language Reference Manual,*May 2003, http://nescc.sourceforge.net / papers/nesc-ref.pdf (pdf).
15. J. Hill , et al., "System Architecture Directions for Networked Sensors,"*Proc. 9th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS-IX), ACM Press, 2000, pp. 93-104.
16. D. Gay , et al., "The nesC Language: A Holistic Approach to Networked Embedded Systems,"*Proc. ACM Sigplan Conf. Programming Language Design and Implementation* (PLDI 03), ACM Press, 2003,pp. 1-11.

17. R. Barr , et al., "On the Need for System-Level Support for Ad hoc and Sensor Networks,"*Operating Systems Rev.,* vol. 36, no. 2, 2002, pp. 1-5.
18. T. Liu and M. Martonosi , "Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems,"*Proc. ACM SIGPLAN Symp. Principles and Practice of Parallel Programming* (PPoPP 03), 2003,pp. 107-118.
19. P. Bonnet , J. Gehrke and P. Seshadri , "Towards Sensor Database Systems,"*Proc. 2nd Int'l Conf. Mobile Data Management* (MDM 01), 2001, pp 314-810.
20. S.R. Madden , M.J. Franklin and J.M. Hellerstein , "TinyDB: An Acquisitional Query Processing System for Sensor Networks,"*ACM Trans. Database Systems,* vol. 30, no. 1, 2005, pp. 122-173.
21. C. Srisathapornphat , C. Jaikaeo and C. Shen , "Sensor Information Networking Architecture," http://doi.ieeecomputersociety.org/10.1109/ICPPW.2000.869083, *Proc. Int'l Workshop Parallel Processing,* IEEE CS Press, 2000, pp. 23-30.
22. S. Li , S. Son and J. Stankovic , "Event Detection Services Using Data Service Middleware in Distributed Sensor Networks,"*Proc. 2nd Int'l Workshop Information Processing in Sensor Networks* (IPSN 03), LNCS 2634, Springer, 2003,pp. 502-517.
23. E. Souto , et al., "A Message-Oriented Middleware for Sensor Networks,"*Proc. 2nd Int'l Workshop Middleware for Pervasive and Ad-Hoc Computing* (MPAC 04), ACM Press, 2004, pp. 127-134.
24. Q. Han and N. Venkatasubramanian , "Autosec: An Integrated Middleware Framework for Dynamic Service Brokering," http://dsonline.computer.org/ portal/pages/dsonline/ past_issues/0107/features/han0107.html *IEEE Distributed Systems Online,* vol. 2, no. 7, 2001.
25. C. Fok , G. Roman and C. Lu , "Mobile Agent Middleware for Sensor Networks: An Application Case Study,"*Proc. 4th Int'l Conf. Information Processing in Sensor Networks* (IPSN 05), IEEE Press, 2005,pp. 382-387.
26. L. St. Ville and P. Dickman , "Garnet: A Middleware Architecture for Distributing Data Streams Originating in Wireless Sensor Networks," http://doi.ieeecomputersociety.org/10.1109/ ICDCSW.2003.1203560, *Proc. 23rd Int'l Conf. Distributed Computing Systems Workshops* (ICDCSW 03), IEEE CS Press, 2003, pp. 235-241.
27. Chih-Chieh Han , et al., "A Dynamic Operating System for Sensor Nodes,"*Proc. Int'l Conf. Mobile Systems, Applications, and Services* (MobiSys 05), ACM Press, 2005,pp. 163-176.
28. R. Gummadi , et al., "Macro-programming Wireless Sensor Networks Using Kairos,"*Proc. Int'l Conf. Distributed Computing in Sensor Systems* (DCOSS 05), LNCS 3560, Springer, 2005, pp. 126-140.
29. M. Welsh and G. Mainland , "Programming Sensor Networks Using Abstract Regions,"*Proc. 1st Usenix/ACM Symp. Networked Systems Design and Implementation* (NSDI 04), 2004,pp. 29-42.
30. T. Abdelzaher , et al., "EnviroTrack: Towards an Environmental Computing Paradigm for Distributed Sensor Networks," http://doi.ieeecomputersociety.org/10.1109/ ICDCS.2004.1281625, *Proc. 24th Int'l Conf. Distributed Computing Systems* (ICDCS 04), IEEE CS Press, 2004,pp. 582-589.
31. Tian He , et al., "VigilNet: An Integrated Sensor Network System for Energy-Efficient Surveillance," to be published, *ACM Trans. Sensor Networks,*2006, http://www-users.cs.umn. edu /~tianhe/Research/Papers/VigilNet-TOSN.pdf (pdf).

32. R. Newton and M. Welsh , "Region Streams: Functional Macroprogramming for Sensor Networks,"*Proc. 1st Int'l Workshop Data Management for Sensor Networks* (DMSN 04), ACM Press, 2004, pp. 78-87.
33. A. Bakshi , et al., "The Abstract Task Graph: A Methodology for Architecture-Independent Programming of Networked Sensor Systems,"*Proc. 2005 Workshop End-to-end, Sense-and-Respond Systems, Applications and Services* (EESR 05), Usenix, 2005,pp. 19-24.
34. K. Whitehouse , et al., *Semantic Streams: A Framework for Declarative Queries and Automatic Data Interpretation,* tech. report MSR-TR-2005-45, Microsoft Research, 2005.
35. S. Madden , et al., "TAG: A Tiny AGgregation Service for Ad-hoc Sensor Networks,"*Proc. 5th Symp. Operating Systems Design and Implementation* (OSDI 02), ACM Press, 2002,pp. 131-146.
36. K. Romer , et al., "Generic Role Assignment for Wireless Sensor Networks,"*Proc. ACM SIGOPS European Workshop,*2004, pp. 7-12.



Salem Hadim is a PhD student at the Stevens Institute of Technology. His research interests include middleware, software systems, multimedia networking, autonomic computing and enabling technologies, and wireless ad hoc sensor networks. He received an MS degree in computer engineering from the Stevens Institute of Technology. Contact him at the Dept. of Electrical and Computer Eng., Stevens Institute of Technology, Hoboken, NJ 07030; shadim@stevens.edu.

Nader Mohamed is an assistant professor of electrical and computer engineering at Stevens Institute of Technology. His research interests include middleware, sensor networks, software systems, cluster and Grid computing, and systems integration. He published more than 30 articles in journals and conferences in these areas. He received a Ph.D. in computer science from the University of Nebraska-Lincoln. Contact him at the Dept. of Electrical and Computer Eng., Stevens Institute of Technology, Hoboken, NJ 07030; nmohamed@stevens.edu.

Related Links

- DS Online's Middleware Community, http://dsonline.computer.org/middleware
- "Sensor Networks and Grid Middleware for Laboratory Monitoring", http://doi.ieeecomputersociety.org/10.1109/E-SCIENCE.2005.73
- "A Coordination Middleware for Wireless Sensor Networks", http://doi.ieeecomputersociety.org/10.1109/ICW.2005.5
- "Towards Developing Sensor Networks Monitoring as a Middleware Service", http://doi.ieeecomputersociety.org/10.1109/ICPPW.2004.1328056
- "Works in Progress: The 2nd International Middleware Doctoral Symposium", cms:/dsonline/2006/02/o2003.xml

Cite this article: Salem Hadim and Nader Mohamed, "Middleware Challenges and Approaches for Wireless Sensor Networks," *IEEE Distributed Systems Online*, vol. 7, no. 3, 2006, art. no. 0603-o3001.