A survey of Middleware for Wireless Sensor Networks

Wouter Horré *Nelson Matthys* *Sam Michiels*
Wouter Joosen *Pierre Verbaeten*

Report CW 498, August 2007



Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

A survey of Middleware for Wireless Sensor Networks

Wouter Horré *Nelson Matthys* *Sam Michiels*
Wouter Joosen *Pierre Verbaeten*

Report CW 498, August 2007

Department of Computer Science, K.U.Leuven

Abstract

Distributed sensor applications emerge as a promising solution to be utilized for complex business scenarios. However, the development and deployment of these applications remains a complex challenge. In this survey we present a two-dimensional classification of middleware technologies needed to realize these complex end-user business scenarios. Subsequently, we discuss currently existing middleware approaches and place them in the classification framework we developed. Finally, based on this classification overview, we identify two issues for which middleware support leaps behind: cross-layer integration and end-to-end integration.

Keywords : sensor networks, sensor middleware
CR Subject Classification : C.2.4, D.2.11

A survey of Middleware for Wireless Sensor Networks

Wouter Horré, Nelson Matthys, Sam Michiels
Wouter Joosen, Pierre Verbaeten
DistriNet Research Group, Dept. Computer Science
Celestijnenlaan 200A, B-3001 Leuven, Belgium

{wouter.horre,nelson.matthys,sam.michiels}@cs.kuleuven.be

Abstract

Distributed sensor applications emerge as a promising solution to be utilized for complex business scenarios. However, the development and deployment of these applications remains a complex challenge. In this survey we present a two-dimensional classification of middleware technologies needed to realize these complex end-user business scenarios. Subsequently, we discuss currently existing middleware approaches and place them in the classification framework we developed. Finally, based on this classification overview, we identify two issues for which middleware support leaps behind: cross-layer integration and end-to-end integration.

Contents

1 Introduction **2**

2 Two dimensions **2**
2.1 End-to-end decomposition 2
2.2 Functional decomposition 3

3 Sensor network operating systems **5**

4 Overview of existing middleware approaches **6**
4.1 Sensor network 6
4.1.1 Technology in the sensor network tier 6
4.1.2 Distribution in the sensor network tier 7
4.1.3 Services in the sensor network tier 9
4.1.4 Management in the sensor network tier 10
4.2 Gateway 12
4.3 Back-end 15

5 Discussion **18**
5.1 Cross-layer integration 18
5.2 End-to-end integration 18

6 Conclusion **19**

7 Acknowledgment **19**

1 Introduction

Sensor networks, combining measurements with computation and wireless communication, emerge as a promising technology that can be applied in a wide variety of application domains, for instance in the domain of control, actuation and maintenance of complex systems, fine-grained monitoring of indoor and outdoor environments, logistics, health care and many others.

In the context of these business applications, sensor networks (and the whole infrastructure) are a reusable asset. They can be deployed for substantial periods of time, during which they can be used for various applications. These applications are however often not known beforehand. Multiple applications, e.g. HVAC monitoring and asset tracking, must be able to run on the same infrastructure at the same time. Multiple users will also be using the same application, for instance the HVAC monitoring may be used by both the HVAC control and an HVAC technician simultaneously.

Distributed applications that exploit the potential of sensor networks rely on a hybrid collection of hardware platforms: (1) wireless and programmable sensor nodes that interact with each other and with their environment, as well as (2) general purpose platforms, for instance application servers, personal computers, personal digital assistants (PDAs), etc. Because of this complexity of platforms and their operational environment, the construction of complete (end-user) applications has become a grand challenge.

In this survey we present a classification of middleware technologies needed to realize these complex end-user business scenarios. Subsequently, we discuss currently existing middleware approaches and place them in the classification framework we developed. The remainder of this text is structured as follows: Section 2 introduces our classification, Section 3 provides a background on sensor network operating systems, existing approaches are presented in Section 4, Section 5 draws some conclusions from the overview and Section 6 concludes this text.

2 Two dimensions

Our classification of middleware technologies needed to realize the end-user business scenarios outlined in Section 1 consists of two dimensions (Figure 1). The first dimension is an end-to-end decomposition. It expresses which parts of the infrastructure are targeted by the middleware. The second dimension decomposes middleware into four layers of functionality, similar to the layering introduced in [114].

2.1 End-to-end decomposition

Section 1 already introduced the hybrid collection of hardware platforms involved in end-to-end business applications. Typical deployments consist of three tiers: (1) the sensor network, (2) the gateway(s) and (3) the back-end. The sensor network consist only of programmable sensor nodes, the back-end is built using general purpose platforms. Between the two, the gateway provides the functionality to bridge them. The gateway functionality can be implemented on sensor network devices, on general purpose platforms, or on a combination of both.

Although the sensor network consists exclusively of programmable wireless sensor network nodes, the nodes in the network can still be very heterogeneous, both in terms of available sensors and in terms of available processing and communication resources. This may thus have implications on the functionality implemented on these nodes, which can be very diverse. It can be as simple as data collection or as complex as multi-node data fusion. If actuators are present in the network, this functionality can also include decision logic to select appropriate actions.

The gateway allows the sensor network and the back-end to interact. It is the bridge between the sensor network devices and the general purpose platforms. The gateway functionality can be provided using many different settings. A traditional base station consists of a single device, connected to a general purpose computer through UART or USB. Another possibility is an overlay of nodes with more resources that are capable of communicating with the back-end via traditional

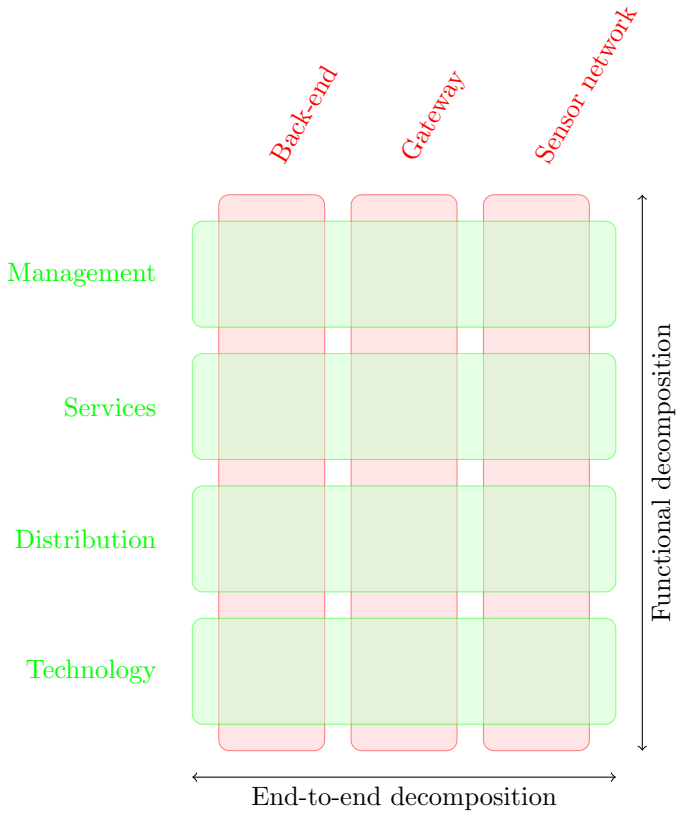


Figure 1: Classification of middleware according to two dimensions

networks. PDAs, smartphones or notebooks that are moving through a sensor network can also act as gateways.

The back-end can also be very heterogeneous. Besides traditional enterprise networks with high-end servers and user terminals, the back-end may also include emerging technologies such as ad hoc networks comprising of PDAs and smartphones. The functionality implemented in the back-end typically ranges from presentation of data to end-users, over archival of data in data warehouses to complex aggregation and correlation of data coming from several sensor networks and other information sources.

Complete end-to-end sensor network applications will comprise functionality in all three tiers. The integration of this functionality is not straightforward. Sensor network middleware must therefore provide support for implementing and integrating functionality of all three tiers.

2.2 Functional decomposition

Orthogonal to the end-to-end dimension, the functional decomposition dimension places middleware functionality in four layers: technology, distribution, services, and management. Note that these layer are conceptual and do not imply a layered implementation. Especially on the resource-constrained sensor nodes, interactions between these conceptual layers are so common that the middleware must take them into account.

The technology layer hides some of the peculiarities of the underlying operating system and hardware. It provides abstractions that make it easier to program a single node. It may also extend or enhance the functionality provided by the operating system to facilitate the development of the other layers.

The distribution layer is similar to the technology layer, but aims at distributed programming. It provides abstractions and mechanisms that ease the implementation of distributed programs.

Besides local and distributed abstractions, middleware also provides reusable services to the application developers. This further eases the development of applications, since the application developer doesn't have to re-implement this recurrent functionality.

The management layer groups all middleware efforts that support co-ordinated management of a (sensor network) deployment. This includes support for starting or stopping services or devices, adding, removing or replacing services, etc. The middleware can support manual management (executing triggers from humans), semi-automated management (some parts are automated, but still human-in-the-loop) or automated management based on e.g. policies or QoS requirements.

An integrated approach has to consider these four levels of abstraction and the interaction between them. These interactions are of various natures, e.g. the way in which services interact will depend on the underlying technology and distribution layers, the management is dependent on which services can be dynamically loaded and updated, etc.

3 Sensor network operating systems

Sensor operating systems are of course not middleware themselves, but they are the underlying technology on which sensor network middleware is built. Because of the resource constraints, it is often the case in sensor networks that the operating system and the middleware are tightly coupled, therefore we consider it useful to include an overview of sensor operating systems in this survey of middleware approaches.

TinyOS [76]
TinyOS is the de-facto standard operating system for wireless sensor networks. It features a mature component model and an event-driven execution model, which are both supported by the nesC language [42]. The component based model has led to the availability of components for various tasks (such as routing, sensor sampling, etc) that can be reused or refined for custom applications. The event-driven execution model enables fine-grained power management and the flexible scheduling needed for tight coupling with the physical world.

Mantis [12]
Mantis is a multi-threaded cross-platform operating system for wireless sensor networks. It implements fine-grained preemptive multi-threading, which enables applications to interleave complex computations with time critical tasks. Mantis is designed to be very flexible through its cross-platform support and testing capabilities across PCs, PDAs and various sensor platforms. Mantis also supports remote management and debugging via a remote shell. This can also be used to set parameters and variables of a running thread. Support for reprogramming a whole Mantis image or one thread within a Mantis system is planned, but not yet completed.

SOS [55]
One motivation for Sensor Operating System (SOS) is dynamic reconfiguration. Therefore SOS consists of a common kernel that implements basic functionality (such as messaging, dynamic memory, module loading and unloading) and loadable modules for implementing more advanced functionality. SOS also tries to provide an expressive system that provides developers with common services (e.g. neighborhood discovery, time synchronization). The execution model of SOS is similar to that of TinyOS, but as SOS is implemented in C, its execution model is, in contrast to TinyOS, not supported by the language constructs of nesC.

Contiki [32]
Contiki is a highly portable, multi-tasking operating system for memory-constrained devices. It consists of an event-driven kernel op top of which applications can be loaded and unloaded at runtime. Contiki processes use protothreads, a lightweight thread-like programming style provided on top of the event-driven kernel. Additionally Contiki supports full fledged preemptive multi-threading via an optional library.

t-kernel [50]
T-kernel is an operating system that aims to provide reliable operating system support to the domain of wireless sensor networks. It provides OS protection, virtual memory and preemptive scheduling. It doesn't rely on hardware support to implement these features. Instead, it uses code modification at load time. T-kernel makes only three assumptions about the hardware:

- Reprogrammable: the system must be able to write the modified code into program memory.
- External read-friendly nonvolatile storage is present
- There is a certain amount of RAM present

4 Overview of existing middleware approaches

4.1 Sensor network

This section elaborates on the existing middleware approaches for sensor networks and places them in the four functional layers from Section 2.2.

4.1.1 Technology in the sensor network tier

The technology layer is divided into two categories: virtual machine approaches and programming abstractions for sensor networks.

Virtual machines: Virtual machines are used in sensor networks for various reasons. One of the most important reasons why this form of programming environments was introduced, is the small size of programs expressed in virtual machine byte code compared to native code. This is true because of the relatively higher level instructions in a virtual machine instruction set.

This observation leads to a trade-off between the cost of updates and the execution overhead. On the one hand, the concise nature of virtual machine code allows more energy efficient code updates. On the other hand, the interpretation of the byte code implies an execution overhead and corresponding energy consumption. Because this trade-off is often application dependent, there is a trend towards customizable virtual machines that allow the programmer to define the border between byte code and native code.

Maté [74, 75]

Maté is a framework to built application specific virtual machines for TinyOS. The framework allows to customize the instruction set of the virtual machine with user provided instructions, hence the name application specific. Maté exploits the fact that a lot of applications, especially within an applications domain, use the same building blocks, but in a different way. The virtual machine instruction set is a way to express how to combine the building blocks. The building blocks are implemented as virtual machine instructions. The applications itself can then be very concise, because they consist of high-level instructions. This makes the dissemination of an update very energy efficient. Maté was the first virtual machine approach built to exploit this finding.

Melete [130]

Melete extends Maté to enhance the support for concurrent applications. In addition to Maté, it enables reliable storage and execution of concurrent applications. It uses dynamic grouping for flexible, on-the-fly deployment of applications based on the contemporary status of the sensor nodes. A group-keyed code dissemination mechanism is used to distribute code selectively and reactively.

DVM [11]

DVM is virtual machine for SOS, based on Maté. Its focus is multi-level software reconfiguration. The combination of DVM and SOS provides on the one hand, reprogramming with native code through the module loading mechanism of SOS, and on the other hand simple reconfigurations through parameter tuning and a virtual machine as a compromise between the flexibility of complete reprogramming and the lightweight updates of parameter tuning.

DAViM [88]

DAViM leverages on the technology introduced by Maté. It provides dynamic operation libraries and multiple virtual machines. It is targeted towards a lightweight service platform for wireless sensor networks. Separation between services and applications is provided by the VM - native code boundary. Dynamic management of available services is provided through the dynamic operation libraries and different users are isolated via the support for multiple virtual machines.

VM⋆ [67, 128]

VM⋆ is a framework that uses fine-grained software synthesis to build a resource-efficient Java virtual machine on a per-application basis. They use an incremental update scheme to allow updates to the application. The virtual machine follows the evolution of the application. In a later stadium, a remote just-in-time compilation framework was built on top of VM⋆. This allows the use of hybrid execution to balance the computation overhead due to interpretation and the code distribution cost of native code.

SwissQM [102]

SwissQM [102] is a virtual machine with a byte code instruction set that is geared towards data acquisition and in-network processing applications. It therefore is an excellent compilation target for higher-level query languages, such as XQuery or SQL.

Programming abstractions: Programming software for sensor nodes using low-level embedded languages like nesC [42] is often very hard. The systems in this section provide higher-level programming abstractions to facilitate writing software for the resource-constrained sensor nodes.

OSM [65]

Many sensor network applications are based on the event-driven paradigm. In languages like nesC, they need to write management code for this event handling. To facilitate this, the Object State Model allows developers to specify the application as a finite state machine, composed of states, transitions, actions and events. OSM provides support for information sharing between actions and makes transitions and actions dependent on the program state. Once the developer specified a finite state machine of its application, it can be transformed into native code through the OSM compiler.

DSN [20]

DSN (Declarative Sensor Network) eases the programming of sensor networks by providing a declarative system. The system consists of a declarative language, a compiler and a run-time system. The language, Snlog, is a dialect of Datalog with extensions to facilitate sensor network programming. The compiler translates the Snlog program to a nesC program using components from a set of compiler library generic templates. The run-time system is based on data flow and is compiled together with the translated Snlog program into a single binary image.

FACTS [121]

FACTS is an event-driven rule-based programming environment introduced to facilitate the programming of complex wireless sensor network applications. All information, ranging from sensor readings to temporary variables, is represented by facts. Rules consist of a predicate over these facts and an action. The action is triggered by the rule engine whenever the predicate becomes true. These actions can manipulate facts (aggregate facts, add or remove facts, etc.) or call native functions that are hooked in the firmware or operating system of the node.

4.1.2 Distribution in the sensor network tier

Programming distributed software using the simple communication mechanisms provided by the operating systems and the technology layer in the sensor network tier is often very complex. The programmer has to deal with a lot of low-level communication details. The systems in this section alleviate this problem by providing a higher-level distributed abstraction that shields this low-level details from the application developer.

Distributed database

The systems in this category consider the sensor network as a distributed database that can be queried for data. The data however, is not stored in tables, but comes directly from the sensors.

Probably the most well known system in this class is TinyDB [83]. It is a query processing system for TinyOS [76] motes. It provides its users a SQL-like interface to specify network-wide

queries. This interface does not only allow one to specify which data to extract from the network, but also at which rate this must happen. Given a query, TinyDB extracts the data from the network, filters it, aggregates it and routes it back to a base station.

Other systems that use the distributed database abstraction are DSWare [79], Cougar [129] and SINA [119].

Tenet [46]

Tenet proposes a tiered sensor network model with a mote tier and a master tier. The low power resource-constrained devices in the mote tier execute only tasks composed from simple *tasklets* from a *tasklet library*. Complex tasks, such as multi-node fusion, are constrained to the master tier, which contains more powerful devices. The actual programming of the sensor network thus takes place in the master tier.

Tenet hides the distributed nature of the mote tier to the applications in the master tier. It offers a simple API that allows to task either one specific mote or all motes. The results of a task are automatically routed back to the master that issued the task.

Macro-programming

Macro-programming approaches allow a programmer to specify the global behavior of the sensor network, rather than the behavior of each individual node. These macro-programs are then translated into node-level programs by a compiler and are sometimes supported by a run-time system. As the website of MagnetOS [22, 80] states: "The network is the computer." Other examples of macro-programming approaches are Kairos [51], Regiment [104], RuleCaster [13], Abstract Task Graph [10] and COSMOS [9].

Publish/Subscribe

The message-oriented publish/subscribe abstraction has been adapted to meet the resource constraints in the sensor network tier. Costa et al. [25] provide a semi-probabilistic routing strategy to support the publish/subscribe paradigm. Mires [118] uses the publish/subscribe abstraction for both in-node communication between application components and communication between nodes.

Neighborhood abstractions

Hood [126] is a neighborhood programming abstraction, wherein a node can select neighbors according to certain criteria and share state with them. It allows the programmers of distributed algorithms to design their algorithms in terms of the neighborhood abstraction itself, instead of decomposing it into primitives such as communication protocols, neighbor lists, etc.

Abstract Regions [125] is a set of spatial operators that capture local communication within regions of a sensor network. These regions can be defined in terms of radio connectivity, geographic location, ... The Abstract Regions provide interfaces to identify neighbors, share data within a region and perform reductions over shared variables. In addition, these interfaces expose the trade-off between accuracy and resource usage. Applications can tune the energy and bandwidth consumption of the communication mechanisms.

Logical Neighborhoods [99] is a programming abstraction that includes nearby nodes that satisfy certain predicates over the static and dynamic properties. Static properties include available sensors, available processing power, etc. Battery status, sensed values, etc. are examples of dynamic properties. The logical neighborhoods are specified declaratively. This specification also includes a high-level indication of the maximum allowed cost for the communication involved. Logical neighborhood replaces the physical proximity of Hood and Abstract Regions by a higher level, application defined notion of proximity and the low-level QoS tuning of Abstract Regions by a high-level notion of communication cost.

Virtual sensors and actuators [21]

Virtual nodes are an abstraction built on top of Logical Neighborhoods [99]. The value read from a virtual sensor is the aggregate of the values of the sensor nodes within a logical neighborhood. The aggregation is done by a user provided function. A virtual actuator is a way of

distributing commands to all nodes in a logical neighborhood. Virtual sensors and actuators can again be used as part of another virtual sensor or actuator, thus creating hierarchies of virtual sensors and actuators.

TeenyLIME [24]

TeenyLIME provides an API that is based on LIME (Linda in a Mobile Environment). It provides (a variant of) the tuple space abstraction made popular by Linda. In the TeenyLIME model, the tuple space of a node is transiently shared with its one hop neighbors. TeenyLIME's notion of reactions allows an application to register a listener that is executed asynchronously whenever a tuple that matches a given pattern appears in the tuple space.

Mobile code

Several systems allow applications to easily migrate to another node and resume computation there. In SensorWare [16], TCL-like scripts can replicate or migrate themselves to neighboring nodes. Smart Messages [64] are messages that contain data, executable code and routing logic (used during migration). In RASA [14] (Resource-Aware Service Architecture), services consist of code and public and private data. The code and the public data can be migrated to neighboring nodes. Agilla [36] is framework for mobile agents built on top of Maté. Mobile Code Daemons [17] uses object serialization to allow objects to travel along the network.

4.1.3 Services in the sensor network tier

The services listed in this section provide common functionality in a reusable form. They relief the application programmers from the burden of re-implementing recurring functionality so that they can concentrate on the core functionality of their applications.

Code distribution

Code distribution services allow to disseminate code updates to sensor nodes for installation. They rely on mechanisms in the technology layer to perform the installation once the code has arrived at a node. Complete network programming systems often consist of both a code distribution service and the mechanisms to perform the actual installation.

Deluge is the best known network programming system for TinyOS. Its code distribution service builds on Trickle [77], a density-aware, epidemic maintenance protocol. Deluge also provides a bootloader and a mechanism for storing code images in the external flash of a node.

TinyCubus [87] includes a role-based code distribution service in its Configuration Engine. It is able to distribute code to a subset of the nodes fulfilling a particular role. The Topology Manager assigns roles to nodes using Generic Role Assignment [111]. In contrast to Deluge, the Configuration Engine of TinyCubus allows to only distribute part of an image and link it to the already available code using the FlexCup [86] mechanism.

The code distribution in the Application Updater of Impala [82] is able to distribute code in networks with highly mobile nodes. Due to the mobility, the nodes only have sporadic connectivity. As a consequence, the code distribution must handle incomplete updates, contemporaneous updates, etc.

Aggregation and storage

The literature contains a lot of work on aggregation and storage services. Aggregation services offer generic support for calculating aggregates (average, minimum, maximum, etc.) of sensed values. Storage services archive sensed data to make it possible to answer historical queries and observe trends.

The most important challenge these systems face, is to keep the service as energy-efficient as possible. There are several techniques that are used to achieve this: estimation [15], (model-based) prediction [78, 123], multi-resolution [41], using more powerful nodes to store or index the data [30, 115] and storing the data at several nodes to save communication when answering queries [108, 109, 43].

PathDCS [33] focuses on the practical applicability of data-centric storage, it therefore uses only widely available tree routing algorithms. TinyPEDS [45] raises the security of data storage by using encryption.

Time synchronization

Time synchronization in sensor networks [117] may be useful for two main reasons: (1) sensor nodes may cooperate on a task that is time sensitive (e.g. tracking) and (2) it can be used for power saving schemes (i.e. all nodes wake up at the same time). Various time synchronization algorithms with different characteristics can be found in the literature, for instance in [34, 40, 85, 39, 19].

Localization

Localization services can be divided into three categories: (i) scene analysis, (ii) proximity-based localization, and (iii) lateration approaches. Scene analysis requires an accurate analysis of the signal's behavior in the environment (fingerprinting) [63], which is then used to retrieve the location of a sensor node. Proximity-based localization [56, 73] uses the connectivity and distribution of the nodes to locate them. Finally, lateration-based approaches [70, 113, 69, 105] require at least three distance estimations between a pair of sensor nodes to determine their positions.

Tracking

EnviroTrack [1] is an object-based distributed middleware system geared towards tracking the physical environment. It raises the level of programming abstraction for tracking applications by providing a convenient and powerful interface. This interface integrates objects that exist in physical time and space into the computational environment of the application.

4.1.4 Management in the sensor network tier

This section contains middleware that provides support for management in the sensor network tier. The first two groups listed provides support for the management of the software in the context of changes in the environment. The last group assists in the selection of a suitable subset of the sensor nodes to provide high enough quality of the result.

Adapting to changes

The Data Management Framework of TinyCubus [87] automatically selects the best suited implementation of certain system and application components according to application requirements, system parameters and optimization criteria. The Data Management Framework monitors these criteria throughout the lifetime of the system, eventually selecting different components when adaptation becomes beneficial.

The Application Adapter of Impala [82] periodically queries the running application for certain parameters. It then combines these with the system parameters it maintains and examines its switching rules. If a switching rule is satisfied, the Application Adapter switches to a different application. The Application Adapter also tracks the hardware requirements of each application. If a hardware failure affects an application, it will be disabled and if needed a switch to another application will happen.

Gridkit [49, 27]

Gridkit, originally designed for grid environments, has been customized to run on a (fairly resource-rich) gumstix-based [52] sensor platform. The reconfigurator performs co-ordinated distributed reconfigurations based on policies and context information provided by a context engine. Gridkit uses the OpenCOM [26] reflective component model and defines a set of component frameworks that structure the distribution and service layer in order to enable easy reconfigurations by replacing a components.

Quality management

MiLAN [57] is a middleware that allows applications to specify a policy for managing the sensor network and the sensors. MiLAN then selects the best set of sensors to provide the required QoS for the application and configures the network appropriately. In [107], a detailed description of the mechanisms used is given.

MidFusion [6] is a middleware architecture that uses Bayesian theory to support information fusion in sensor networks. It transparently discovers and selects the best set of sensors on behalf of applications. Its decision is based on the required QoS and the QoS that can be delivered by the sensor network.

In the QUASAR [71, 72] project at UC Irvine, a quality-aware sensing architecture is developed. The system provides a query interface that exploits the notion application error tolerance. The system gives an approximate answer, computed from imprecise data.

MASTAQ [58], short for Middleware Architecture for Sensor Applications with Statistical Quality Constraints, allows applications to specify Quality of Information (QoI) requirements. These requirements are specified in terms of statistical metrics. MASTAQ then activates and queries only a subset of the sensors to meet the QoI bound.

4.2 Gateway

As gateways form the bridge between the sensor network and back-end infrastructure, they play an important role in building end-to-end distributed applications. Their main operation includes the aggregation and transferring of data from the sensor network towards the enterprise infrastructure. However, as distributed applications may also integrate gateway devices, there might also be more application-specific functionality installed on gateways.

TinyLIME [29]

TinyLIME is a middleware built on top of LIME (Linda in a Mobile Environment) and thus provides (a variant of) the tuple space model made popular by Linda. Originally, TinyLIME is built for settings where mobile gateways (laptops, PDAs, ...) move through a sensor network. All sensor nodes in reach of such a gateway are represented by a LIME agent running on the gateway. This agent places data from the nodes in the tuple space, where it can be read by other LIME agents.

Agimone [54]

Agimone is an integration layer between Agilla [36] en Limone [35]. Agilla and Limone are both mobile code approaches. Agilla is designed for sensor networks, Limone targets IP-based ad hoc networks. Agimone allows Agilla agents to migrate to another sensor network over IP-based networks by encapsulating them in Limone agents.

Janus [31]

Janus provides flexible and lightweight access to sensor network resources from external networks. Janus focuses on the functions a particular sensor network provides rather than the data the sensor network contains. This allows for service composition by combining several of these functions. Janus provides RPC-like semantics, however most of the RPC functionality is present on the client in order to keep the RPC server inside the sensor network as lightweight as possible. The RPC system uses XRP (eXtensible Resolution Protocol) [47] for communication between the server (XRP engine) and the client (XRP agent).

SwissQM Gateway [101]

The intelligent gateway for SwissQM compiles high-level query languages, such as SQL and XQuery, on the fly to the SwissQM instruction set. It also combines the queries of several users into one network query [100]. Back-end systems can connect to the gateway using several interfaces to submit queries or user-defined functions. The gateway has, amongst others, a SOAP-interface and a R-OSGi interface.

SONGS [81] / Semantic Streams [127]

SONGS (Service-Oriented Network proGramming of Sensors) is a semantic service framework for a tiered sensor network architecture. The lowest tier corresponds to the sensor network. In the tier we called the gateway tier in Section 2.1, SONGS distinguishes between two (logical) tiers: field servers and gateways. A field server converts sensor data from raw platform specific formats to an open and directly usable format, aggregates data, attaches a semantic interpretation to data, etc. Gateways are used by users (or back-end applications) to interact with the sensor network infrastructure. They contain e.g. a planner that converts user requests into tasks that can be executed by the field servers.

In SONGS, users submit semantic queries to the gateways, where a planner translates these into a set of logic rules that is then solved by an inference engine to produce a service composition [127]. These service dependency graphs are then used as input for the embedding step. In this step the services are instantiated, connected and assigned to a field server. Afterwards this assignment is sent to the field servers in a tasking description language. The field servers execute the task, while continuously monitoring the execution constraints. If one of the constraints is violated and renders the task assignment invalid, the field server tries to solve the problem locally or, if that fails, informs the planner.

JDDAC [94]

JDDAC (Java Distributed Data Acquisition and Control) is a Java-based platform that reduces complexity by providing "last mile" functionality between Java applications and the real world. A system built using JDDAC is based on a multi-tiered architecture and contains two types of nodes: probes and servers. Probes run on Java SE or Java ME devices, so typically on devices in the gateway tier (except for the Sun SPOT device [98], which is a mote type Java ME device). The primary task is to collect data and do simple local processing. The servers typically run on Java EE platforms. They do more sophisticated data processing, enable data persistence and provide users with a console. Both at the probes and the server, functionality is implemented in so-called F-Blocks. These functional blocks are combined in a data flow network by wiring them together using a publish/subscribe or a client-server interaction style. The wiring of the blocks is dynamically adaptable.

GSN [2, 4, 3]

The vision of a "Sensor Internet" is the motivation behind the GSN (Global Sensor Network) middleware. GSN provides a container that manages virtual sensors. These virtual sensors abstract away the details of the underlying sensor networks. GSN accesses hardware via wrappers which are specific for the underlying sensor technology. GSN provides, for instance, an IEEE 1451 [62] wrapper that implements plug-and-play support for IEEE 1451 compliant sensors. Additionally, the container provides some common services, such as storage and access control. The virtual sensors in a GSN container are accessible in several ways, such as via a web interface or via Web Services [120].

OSGi [7]

The OSGi specification defines a standardized, component oriented, computing environment for embedded networked devices that is the foundation of an enhanced service oriented architecture. Software components can be installed, updated, or removed on the fly without ever having to disrupt the operation of the device. Software components are service libraries or applications that can dynamically discover and use other components. Key features of the OSGi service platform are the management of locally, as well as remotely, installed software components, secure execution of software components, cooperation and interoperability assurance of software components, etc.

R-OSGi [110]

R-OSGi is a distributed middleware platform that extends the ordinary OSGi standard to incorporate support for distributed module management. R-OSGi looks to the developer like a standard module management tool for his application. However, at deployment time R-OSGi can be used to easily turn the application into a distributed application by simply indicating where the different modules should be deployed.

IBM Edge Server Software [112]

The IBM Edge Server Software is an OSGi-based software platform, residing at the gateway level. IBM's Service Management Framework (SMF) [59] forms the underlying technology, which is IBM's implementation of the OGSi R3 specification. The platform can be customized by several services, such as filtering and aggregation, that are implemented as bundles. Finally, the platform transforms raw sensor data into the MQtt-protocol [61] format, which can be easily interpreted by IBM's WebSphere [60] back-end solutions. This technology is IBM's move towards the integration of sensor networks with their back-end infrastructure.

Oracle Sensor Edge Server Software [106]

The Oracle Edge Server Software is developed to transform sensor data into meaningful business events. The software resides at the gateway level and provides a solution for the aggregation, filtering, analyzing, etc. of data. The software supports technology like Java Message Services (JMS) [53], HTTP, and Web Services [120], which can be used to better support the integration of the sensor network into the back-end infrastructure. Finally, the Oracle software can also be

used to manage certain infrastructure or applications on the gateway, e.g. adding a new devices and updating applications on the gateway.

NORS and N-RSA [122]

The Nokia Remote Sensing Platform (NORS) is based on the Nokia Remote Sensing Architecture (N-RSA) for distributed applications with mobile phones as key elements. The mobile phone is an intelligent gateway that is meant to act on and transfer sensor data to repositories. N-RSA introduces a layered architecture, where various application components, implementing different specific functionality, are residing on top of a middleware layer. The middleware layer consists of several components, responsible for the registration, retrieval, access, aggregation and storage of sensor data through well-defined protocols.

IrisNet [44]

Intel's IrisNet focuses on a world-wide sensor web where millions of distributed heterogeneous sensors and powerful Internet-connected servers are available to users for querying as a single unit. The two main challenges for a wide-area sensor web are the collection of widely distributed data and the answering of queries over the collected data. IrisNet provides tools that automate these two processes. The IrisNet architecture is a two-tier architecture: the sensing nodes and the nodes that implement storage functionality for the sensed data. The first are called Sensing Agents (SA) and implement a generic data acquisition interface for accessing their sensors, while the latter implement a distributed database for organizing the collected data and hence are called Organizing Agents (OA). IrisNet allows to dynamically adapt and guarantee a safe execution of services (called senselets) on each SA. These senselets mainly perform filtering and forwarding of filtered sensor data towards a nearby OA, however, they can also share intermediate results with each other. This OA then routes the received data towards the appropriate set of OAs inside the distributed database (e.g. data coming from a people detection sensor is routed to the security office's OA). IrisNet's SAs are written in C/C++, while the OAs are written in Java.

4.3 Back-end

Back-end systems are mainly used to interpret and store sensor data. As back-end systems can be very heterogeneous themselves, they may mainly include powerful computing platforms running (parts of) enterprise applications, as well as ordinary desktop or portable computers. Communication between these platforms can also be very heterogeneous, ranging from fast Ethernet connections, over fixed wireless networks, to mobile ad hoc networks.

Java SE [96] and Java ME [95]
Sun's Java SE and Java ME are both Java platforms that provide a particular subset of libaries suitable for a broad family of devices. The Java SE platform aims at desktop computing, while the Java ME platform aims at providing a certified collection of Java APIs for the development of software for small, resource-constrained devices such as cell phones, PDAs, and other types of embedded devices capable enough to run the Java ME platform. The Java platform itself is not specific to any one processor or operating system. It consists of an execution engine (called a virtual machine) and a compiler with a set of standard libraries that are implemented for various hardware and operating systems so that Java programs can run identically on all of them.

Java EE [93]
Java EE is a Java software platform that offers a component-based approach to develop and deploy enterprise applications on a multi-tier distributed model. Enterprise Java Beans (EJBs) are Java EE's core, because they are used to implement server-side business logic and concerns like persistence, transactional integrity, and security. An EJB has an implementation together with two mandatory interfaces, and resides in a container on the application server. EJBs may use and co-operate with services offered by the container. These services include, for instance, persistence, naming, messaging, security in general, etc. Deployment support for EJBs in business applications is standardized through the Java EE's deployment descriptors.

.NET [91, 89], .NET Compact Framework [90], and .NET Micro Framework [92]
Microsoft's .NET framework is based on a Common Language Runtime (CLR) and a large class library to develop distributed applications. The CLR acts as an application virtual machine and manages the execution of programs written for the .NET framework. Each program written for the .NET framework is compiled into a platform-neutral language, called the Common Intermediate Language (CIL). This CIL is then interpreted by the CLR, which compiles or translates the CIL to a platform-specific machine code. The class library provides support for common programming requirements in a wide application domain. The Microsoft .NET framework can be tailored to several device classes, including the full version of the .NET framework, the .NET Compact framework for mobile phones and PDAs, and the .NET Micro framework for resource-constrained embedded devices.

CORBA [116]
The Common Object Request Broker Architecture (CORBA) is an OMG standard, implemented in several programming languages. CORBA wraps program code in bundles, containing information about the capabilities of the code and how to invoke it. To achieve this, CORBA uses an Interface Definition Language (IDL) to describe a software component's interfaces. This IDL is then mapped to specific programming languages. Through this IDL, and with RPC-semantics, CORBA enables a uniform communication between distributed software components, written in different programming languages, by making an abstraction of the physical distribution of each component. Hence, in CORBA, all interactions between distributed components appear to be local. Finally, CORBAservices are the OMG's specification of several basic services that almost every object in an application needs. These services include security, naming, support for transaction processing, support for the object's life cycle, and other functionalities.

JINI [124]
JINI is a Java-based middleware to support service-oriented architectures for highly dynamic,

adaptive, and large-scale distributed systems. JINI enables access to 'services' (which can be programs, other services, or even users) over any type of network for any type of platform. Together with a set common services for distributed systems, such as naming, service lookup, communication, etc., JINI allows management of services by the ability to dynamically load and run code on their hostplatforms.

JXTA [84] and JXME [97]

JXTA is a set of open protocols to enable P2P communication between different clients. JXTA creates a virtual overlay network and connects peers by allowing them to find each other and to let them communicate using their own protocols. The JXTA protocol is based on XML-messages and thus provides communication pipes between different different devices, running their own protocols and programs, implemented in any programming language on any platform.

As the JXTA code base is mainly too large to run on resource-constrained devices, JXME is a lightweight implementation of JXTA on top of Java ME, suitable for resource-constrained devices.

Linda [18] and Lime [103]

Linda is a model of coordination and communication among several parallel processes operating upon objects stored in and retrieved from a shared, virtual, associative memory. The Linda model is implemented as a coordination language in which several primitives operating on ordered sequences of typed data objects, tuples, are added to a sequential language, and a logically global associative memory, called a tuplespace. Processes can then store and retrieve tuples from this shared tuplespace.

Linda in a Mobile Environment (Lime) is a model and middleware supporting the development of applications that exhibit the physical mobility of hosts, logical mobility of agents, or both. LIME adopts the coordination perspective inspired by the work on the Linda model. The context for computation, represented in Linda by a globally accessible persistent tuple space, is refined in LIME to transient sharing of the identically named tuple spaces carried by individual mobile units. Tuple spaces are also extended with a notion of location and programs are given the ability to react to specified states.

UPnP [37]

Universal Plug and Play (UPnP) is a distributed open networking architecture to enable communication between any pair of devices under the command of a controlling device in the network. UPnP is based on a set of common communication protocols and standards and it is device, operating system and language independent. The UPnP architecture supports zero-configuration, "invisible networking" and automatic discovery for many device categories from a range of vendors; any device can dynamically join a network, obtain an IP address, announce its name, convey its capabilities upon request, and learn about the presence and capabilities of other devices. A device can also leave the network smoothly without leaving any unwanted state behind.

Jadabs [38]

Java ADhoc Application BootStrap (JADABS) implements a lightweight AOP service-oriented architecture on top of OSGi and JXTA for resource-constrained devices. Services are deployed in lightweight containers and can be (un)loaded when needed. It focuses on ubiquitous interconnections of devices, which could require certain services to be deployed under certain conditions in time.

Chedar [8] and Mobile Chedar [68]

The Cheap Distributed Architecture (Chedar) is a peer-to-peer middleware built on top of Java. Chedar provides an API for P2P application developers, allowing them to perform operations on resources in the P2P network. Each Chedar node keeps a XML-based database of locally and remotely available resources, where one can perform operations on. Mobile Chedar is an extension of the Chedar middleware for mobile ad hoc networks. Mobile Chedar is more suitable to be deployed on resource-constrained devices since it is built on top of Java ME and uses bluetooth

instead of WiFi to connect to other peers.

Proem [66]

Proem is a Java-based middleware platform for mobile peer-to-peer computing. The Proem middleware consists of three parts: an application runtime environment, a set of middleware services, and protocol stack for communication. In Proem, each application is called a peerlet, and is managed by the peerlet engine. Peerlets can be dynamically added to or removed from the peerlet engine. The set of middleware services provides common functionality to allow distributed applications (peerlets) share resources and particular events with each other, while the protocol stack defines the syntax and semantics to enable communication between peers.

5 Discussion

From our overview in Section 4, it can be seen that there has already been done a lot of work in all distinct areas of our taxonomy. However, in our opinion, there are two important issues that are not adequately addressed in the state-of-the-art: cross-layer integration and end-to-end integration. Cross-layer integration has to do with issues that crosscut the categories along the functional decomposition axis. End-to-end integration deals with crosscutting issues along the other axis.

5.1 Cross-layer integration

Cross-layer integration is already common in traditional middleware systems. As can be seen from the overview, Java EE and .NET are both active in the four conceptual layers along the functional decomposition axis. This cross-layer integration has two main advantages. First, it makes the middleware as an entirety easier to understand, use and maintain. Adopting a cross-layer approach implies structuring the four layers and their interactions in an overall framework. Such a framework is easier to understand than an ad hoc composition of functionality from the four layers. Second, a cross-layer approach makes it easier to address concerns that crosscut the conceptual layers. For example, enforcing security has consequences on the four conceptual layers we identified. Implementing such a concern is a lot easier if the four layers are structured in an overall framework.

In the gateway and sensor network tier, cross-layer integration is not yet mature. There is already some work done, but this has not yet reached the level of integration in traditional middleware.

Impala [82] includes work in three of the four conceptual layers. It includes technology for installation and storage of code updates, a code distribution service and an Application Adapter that adapts the running application to the current state of the node and to changes in its environment (e.g. number of direct neighbors, etc.). These various parts are integrated in a middleware architecture. Although Impala has some cross-layer features, these are rather limited. For example, there is only one service (the code distribution) and there isn't support for adding others.

TinyCubus [87] consists of three main parts, a data management framework, a configuration engine and a cross-layer framework. The data management framework adapts both system and application components according to certain criteria. The configuration engine provides the technology and the code distribution service needed to support reprogramming of the nodes. The code distribution service uses application knowledge about the sensor topology to optimize its behavior. The sharing of this knowledge is possible through the cross-layer framework. This framework eases cross-layer interactions and provides a state repository to facilitate cross-layer data sharing.

Gridkit [27] uses component frameworks to structure the technology, distribution and services layers. In the technology layer, Gridkit's overlay framework structures the implementation of network overlays. The interaction framework provides a structured approach to distributed interaction styles. Gridkit also includes frameworks for various services, such as resource management and resource discovery. These frameworks are built on top of the reflective component model OpenCOM [26] to enable reasoning over the structure of the system and facilitate fine grained adaptation.

Although the potential benefits of cross-layered approaches for sensor networks have been identified earlier [48, 5], our overview shows there are only few efforts to tackle cross-layering using a structured approach. Implementing cross-layer interactions in an ad hoc manner may lead to complex, hard to maintain systems. Therefore we believe a solid approach for engineering cross-layered sensor network systems is vital.

5.2 End-to-end integration

An end-to-end integrated approach takes a broader look on the combination of the sensor network, gateway and back-end tiers. It tries to provide a uniform approach across the different tiers. Of

course, such an approach can built upon the work already done in the three tiers separately. We believe the integration of existing work in the different tiers into one overall framework can add significant value and that the integrated whole is worth more than the sum of its parts.

Initial efforts in the direction of an integration across the three tiers can already be found in the literature. The RUNES component model [23] is implemented on devices in the three tiers. Gridkit [27] was originally designed for back-end systems but has been customized to also run on gateway and sensor devices [49]. These efforts create the possibility to use the same platform across the different tiers, but work on the actual integration is lacking.

In industry, there are similar observations: the Java ME edition of the Java Technology has been introduced in the sensor network tier by Sun Microsystems. Their Sun SPOT (Small Programmable Object Technology) [98] is a mote class device which runs a Java ME compliant Java virtual machine. The series of Microsoft .NET platforms has been extended with a .NET Micro Framework and Crossbow Technology Inc. is shipping Imote2 devices preloaded with this framework [28].

These efforts are valuable and need to be continued towards a complete end-to-end integration. We believe this integration is a prerequisite for widespread adoption of sensor networks in complex business applications. Cost-effective application development is only feasible if developers can built their applications on top of one platform that allows them to implement functionality in the three tiers.

6 Conclusion

This document presented a classification and an overview of middleware technologies in the context of complex end-to-end business application involving sensor networks. The classification consists of two dimensions: an end-to-end decomposition and a functional decomposition. The end-to-end decomposition covers three deployment tiers: the back-end infrastructure, the sensor network and the gateway(s) between both. The functional decomposition highlights four conceptual layers of functionality found in middleware.

We have also provided an overview of existing approaches and classified them according to our classification. This overview showed two issues for which middleware support leaps behind: cross-layer integration and end-to-end integration. Cross-layer integration deals with concerns that crosscut the conceptual layers along the functional decomposition axis. End-to-end integration set a step back and takes a wide angle view on the whole infrastructure. It integrates approaches in the three tiers and addresses concerns that crosscut multiple tiers.

7 Acknowledgment

Wouter Horré is a Ph.D. fellow of the Research Foundation - Flanders (FWO). Sam Michiels is a Postdoctoral Fellow of the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT).

References

[1] Tarek F. Abdelzaher, Brian M. Blum, Qing Cao, Y. Chen, D. Evans, J. George, S. George, Lin Gu, Tian He, Sudha Krishnamurthy, Liqian Luo, Sang Hyuk Son, Jack Stankovic, Radu Stoleru, and Anthony D. Wood. Envirotrack: Towards an environmental computing paradigm for distributed sensor networks. In *ICDCS*, pages 582–589, 2004.

[2] Karl Aberer, Manfred Hauswirth, and Ali Salehi. A middleware for fast and flexible sensor network deployment. In *Very Large Data Bases (VLDB)*, 2006.

[3] Karl Aberer, Manfred Hauswirth, and Ali Salehi. Infrastructure for data processing in large-scale interconnected sensor networks. In *Mobile Data Management (MDM)*, 2007.

[4] Karl Aberer, Manfred Hauswirth, and Ali Salehi. Zero-programming sensor network deployment. In *Next Generation Service Platforms for Future Mobile Systems (SPMS)*, 2007.

[5] Ian F. Akyildiz and Ismail H. Kasimoglu. Wireless sensor and actor networks: research challenges. *Ad Hoc Networks*, 2(4):351–367, October 2004.

[6] H. Alex, M. Kumar, and B. Shirazi. Midfusion: middleware for information fusion in sensor network applications. In *Intelligent Sensors, Sensor Networks and Information Processing Conference*, pages 617–622, 2004.

[7] OSGi Alliance. About the OSGi Service Platform, technical whitepaper, revision 4.1, June 2007.

[8] A. Auvinen, Mikko Vapa, Matthieu Weber, Niko Kotilainen, and Juori Vuori. Chedar: peer-to-peer middleware. In *IPDPS*, 2006.

[9] Asad Awan, Suresh Jagannathan, and Ananth Grama. Macroprogramming heterogeneous sensor networks using cosmos. In *Proceedings of EuroSys 2007*, March 2007.

[10] Amol Bakshi, Viktor K. Prasanna, Jim Reich, and Daniel Larner. The abstract task graph: a methodology for architecture-independent programming of networked sensor systems. In *EESR '05: Proceedings of the 2005 workshop on End-to-end, sense-and-respond systems, applications and services*, pages 19–24, Berkeley, CA, USA, 2005. USENIX Association.

[11] Rahul Balani, Chih-Chieh Han, Ram Kumar Rengaswamy, Ilias Tsigkogiannis, and Mani Srivastava. Multi-level software reconfiguration for sensor networks. In *ACM Conference on Embedded Systems Software (EMSOFT)*, October 2006.

[12] Shah Bhatti, James Carlson, Hui Dai, Jing Deng, Jeff Rose, Anmol Sheth, Brian Shucker, Charles Gruenwald, Adam Torgerson, and Richard Han. Mantis os: an embedded multithreaded operating system for wireless micro sensor platforms. *Mob. Netw. Appl.*, 10(4):563–579, 2005.

[13] Urs Bischoff and Gerd Kortuem. Rulecaster: A macroprogramming system for sensor networks. In *Proceedings of the OOPSLA '06 Workshop on Building Software for Sensor Networks*, October 2006.

[14] Jan Blumenthal and Dirk Timmermann. Resource-aware service architecture for mobile services in wireless sensor networks. *icwmc*, 0:34, 2006.

[15] Athanassios Boulis, Saurabh Ganeriwal, and Mani B. Srivastava. Aggregation in sensor networks: an energy-accuracy trade-off. *Ad Hoc Networks*, 1(2-3):317–331, 2003.

[16] Athanassios Boulis, Chih-Chieh Han, and Mani B. Srivastava. Design and implementation of a framework for efficient and programmable sensor networks. In *MobiSys '03: Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 187–200, New York, NY, USA, 2003. ACM Press.

[17] Richard R. Brooks and Thomas E. Keiser. Mobile code daemons for networks of embedded systems. *IEEE Internet Computing*, 8(4):72–79, 2004.

[18] Nicholas Carriero and David Gelernter. Linda in context. *Commun. ACM*, 32(4):444–458, 1989.

[19] Shujuan Chen, Adam Dunkels, Fredrik Österlind, Thiemo Voigt, and Mikael Johansson. Time synchronization for predictable and secure data collection in wireless sensor networks. In *The Sixth Annual Mediterranean Ad Hoc Networking WorkShop*, pages 165–172, June 2007.

[20] David Chiyuan Chu, Lucian Popa, Arsalan Tavakoli, Joseph M. Hellerstein, Philip Levis, Scott Shenker, and Ion Stoica. The design and implementation of a declarative sensor network system. Technical Report UCB/EECS-2006-132, EECS Department, University of California, Berkeley, October 16 2006.

[21] Pietro Ciciriello, Luca Mottola, and Gian Pietro Picco. Building virtual sensors and actuators over logical neighborhoods. In *MidSens '06: Proceedings of the international workshop on Middleware for sensor networks*, pages 19–24, New York, NY, USA, 2006. ACM Press.

[22] Cornell University. The MagnetOS operating system, July 2007. http://www.cs.cornell.edu/People/egs/magnetos/.

[23] Paolo Costa, Geoff Coulson, Richard Gold, Manish Lad, Cecilia Mascolo, Luca Mottola, Gian Pietro Picco, Thirunavukkarasu Sivaharan, Nirmal Weerasinghe, and Stefanos Zachariadis. The runes middleware for networked embedded systems and its application in a disaster management scenario. In *Proceedings of the 5th IEEE International Conference on Pervasive Computing and Communications (PERCOM07)*, March 2007.

[24] Paolo Costa, Luca Mottola, Amy L. Murphy, and Gian Pietro Picco. Teenylime: Transiently shared tuple space middleware for wireless sensor networks. In *Proceedings of the First International Workshop on Middleware for Sensor Networks (MidSens'06)*, November 2006.

[25] Paolo Costa, Gian Pietro Picco, and Silvana Rossetto. Publish-subscribe on sensor networks: a semi-probabilistic approach. In *Mobile Adhoc and Sensor Systems Conference*. IEEE, November 2005.

[26] Geoff Coulson, Gordon S. Blair, Paul Grace, Ackbar Joolia, Kevin Lee, and Jo Ueyama. A component model for building systems software. In *Proceedings of IASTED Software Engineering and Applications (SEA'04)*, Cambridge, MA, USA, November 2004. ACTA Press.

[27] Geoff Coulson, Paul Grace, Gordon Blair, Wei Cai, Chris Cooper, David Duce, Laurent Mathy, Wai Kit Yeung, Barry Porter, Musbah Sagar, and Wei Li. A component-based middleware framework for configurable and reconfigurable grid computing. *Concurrency and Computation: Practice and Experience*, 18(8):865–874, July 2006.

[28] Crossbow Technology Inc. Wsn imote2 .builder kit. http://www.xbow.com/Products/-productdetails.aspx?sid=267, July 2007.

[29] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A.L. Murphy, and G.P. Picco. Mobile data collection in sensor networks: The TinyLime middleware. *Journal of Pervasive and Mobile Computing*, 4(1):446–469, December 2005.

[30] Peter Desnoyers, Deepak Ganesan, and Prashant Shenoy. Tsar: a two tier sensor storage architecture using interval skip graphs. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 39–50, New York, NY, USA, 2005. ACM Press.

[31] Adam Dunkels, Richard Gold, Sergio Angel Marti, Arnold Pears, and Mats Uddenfeldt. Janus: An architecture for flexible access to sensor networks. In *Proceedings of First International ACM Workshop on Dynamic Interconnection of Networks (DIN'05)*, Cologne, Germany, 2005.

[32] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN'04)*, pages 455–462, Washington, DC, USA, 2004. IEEE Computer Society.

[33] Cheng Tien Ee, Sylvia Ratnasamy, and Scott Shenker. Practical data-centric storage. In *Proceedings of the Third Symposium on Networked Systems Design and Implementation (NSDI)*, pages 325–338, May 2006.

[34] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Oper. Syst. Rev.*, 36(SI):147–163, 2002.

[35] Chien-Liang Fok, Gruia-Catalin Roman, and Gregory Hackmann. A lightweight coordination middleware for mobile computing. In *COORDINATION*, pages 135–151, 2004.

[36] Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. In *ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 653–662, Washington, DC, USA, 2005. IEEE Computer Society.

[37] UPnP Forum. UPnP Device Architecture. Technical report, UPnP Forum, July 2006.

[38] Andreas Frei and Gustavo Alonso. A dynamic lightweight platform for ad-hoc infrastructures. In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pages 373–382, Washington, DC, USA, 2005. IEEE Computer Society.

[39] Saurabh Ganeriwal, Deepak Ganesan, Hohyun Shim, Vlasios Tsiatsis, and Mani B. Srivastava. Estimating clock uncertainty for efficient duty-cycling in sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 130–141, New York, NY, USA, 2005. ACM Press.

[40] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 138–149, New York, NY, USA, 2003. ACM Press.

[41] Deepak Ganesan, Deborah Estrin, and John Heidemann. Dimensions: why do we need a new data handling architecture for sensor networks? *SIGCOMM Comput. Commun. Rev.*, 33(1):143–148, 2003.

[42] David Gay, Philip Levis, Robert V. von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. In *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 1–11, New York, NY, USA, May 2003. ACM Press.

[43] Constantinos Georgiou, Evangelos Kranakis, Ricardo Marcelín-Jiménez, Sergio Rajsbaum, and Jorge Urrutia. Distributed dynamic storage in wireless networks. *International Journal of Distributed Sensor Networks*, 1(3-4):355 – 371, 2005.

[44] Phillip B. Gibbons, Brad Karp, Yan Ke, Suman Nath, and Srinivasan Seshan. Irisnet: an architecture for a worldwide sensor web. *IEEE Pervasive Computing*, 2(4):22–33, October-December 2003.

[45] Joao Girao, Dirk Westhoff, Einar Mykletun, and Toshinori Araki. Tinypeds: Tiny persistent encrypted data storage in asynchronous wireless sensor networks. *Ad Hoc Netw.*, 5(7):1073–1089, 2007.

[46] Omprakash Gnawali, Ben Greenstein, Ki-Young Jang, August Joki, Jeongyeup Paek, Marcos Vieira, Deborah Estrin, Ramesh Govindan, and Eddie Kohler. The tenet architecture for tiered sensor networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (Sensys)*, Boulder, Colorado, November 2006.

[47] Richard Gold, Per Gunningberg, and Christian Tschudin. A virtualized link layer with support for indirection. In *FDNA '04: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pages 28–34, New York, NY, USA, 2004. ACM Press.

[48] Andrea J. Goldsmith and Stephen B. Wicker. Design challenges for energy-constrained ad hoc wireless networks. *IEEE Wireless Communications*, 9(4):8–27, August 2002.

[49] Paul Grace, Geoff Coulson, Gordon Blair, Barry Porter, and Danny Hughes. Dynamic reconfiguration in sensor middleware. In *Proceedings of the First International Workshop on Middleware for Sensor Networks (MidSens)*, New York, NY, USA, November 2006. ACM Press.

[50] Lin Gu and John A. Stankovic. t-kernel: providing reliable os support to wireless sensor networks. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 1–14, New York, NY, USA, 2006. ACM Press.

[51] Ramakrishna Gummadi, Omprakash Gnawali, and Ramesh Govindan. Macro-programming wireless sensor networks using kairos. In *DCOSS*, pages 126–140, 2005.

[52] Gumstix Inc. Gumstix website. http://www.gumstix.com/, July 2007.

[53] Kim Haase. *Java Message Service API Tutorial*. SUN Microsystems, 2002.

[54] Gregory Hackmann, Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu. Agimone: Middleware support for seamless integration of sensor and ip networks. In *DCOSS*, pages 101–118, 2006.

[55] Chih-Chieh Han, Ram Kumar, Roy Shea, Eddie Kohler, and Mani Srivastava. A dynamic operating system for sensor nodes. In *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 163–176, New York, NY, USA, 2005. ACM Press.

[56] Tian He, Chengdu Huang, Brian M. Blum, John A. Stankovic, and Tarek Abdelzaher. Range-free localization schemes for large scale sensor networks. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 81–95, New York, NY, USA, 2003. ACM Press.

[57] Wendi B. Heinzelman, Amy L. Murphy, Hervaldo S. Carvalho, and Mark A. Perillo. Middleware to support sensor network applications. *IEEE Network*, 18(1):6–14, 2004.

[58] Inseok Hwang, Qi Han, and Archan Misra. Mastaq: A middleware architecture for sensor applications with statistical quality constraints. In *PERCOMW '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 390–395, Washington, DC, USA, 2005. IEEE Computer Society.

[59] IBM. Introduction to the IBM Service Management Framework. Technical report, IBM, May 2003.

[60] IBM. IBM websphere software, July 2007. http://www.ibm.com/websphere.

[61] IBM. MQ Telemetry Transport, July 2007. http://www.mqtt.org.

[62] IEEE. IEEE standard for a smart transducer interface for sensors and actuators - network capable application processor (NCAP) information model (IEEE std 1451.1-1999), 1999.

[63] Kamol Kaemarungsi and Prashant Krishnamurthy. Modeling of indoor positioning systems based on location fingerprinting. In *INFOCOM*, 2004.

[64] Porlin Kang, Cristian Borcea, Gang Xu, Akhilesh Saxena, Ulrich Kremer, and Liviu Iftode. Smart messages: A distributed computing platform for networks of embedded systems. *Comput. J.*, 47(4):475–494, 2004.

[65] Oliver Kasten and Kay Römer. Beyond event handlers: programming wireless sensors with attributed state machines. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 7, Piscataway, NJ, USA, 2005. IEEE Press.

[66] Gerd Kortuem. Proem: a middleware platform for mobile peer-to-peer computing. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):62–64, 2002.

[67] Joel Koshy and Raju Pandey. VM*: Synthesizing scalable runtime environments for sensor networks. In *Proceedings of the Third International Conference on Embedded Networked Sensor Systems (SenSys'05)*, San Diego, CA, USA, November 2005. ACM Press, New York, NY, USA.

[68] N. Kotilainen, M. Weber, M. Vapa, and J. Vuori. Mobile chedar: A peer-to-peer middleware for mobile devices. In *PERCOMW '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 86–90, Washington, DC, USA, 2005. IEEE Computer Society.

[69] YoungMin Kwon, Kirill Mechitov, Sameer Sundresh, Wooyoung Kim, and Gul Agha. Resilient localization for sensor networks in outdoor environments. In *ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 643–652, Washington, DC, USA, 2005. IEEE Computer Society.

[70] Koen Langendoen and Niels Reijers. Distributed localization in wireless sensor networks: a quantitative comparison. *Comput. Networks*, 43(4):499–518, 2003.

[71] Iosif Lazaridis, Qi Han, Xingbo Yu, Sharad Mehrotra, Nalini Venkatasubramanian, Dmitri V. Kalashnikov, and Weiwen Yang. Quasar: Quality-aware sensing architecture. *SIGMOD Record*, 33(1):26–31, March 2004.

[72] Iosif Lazaridis and Sharad Mehrotra. Approximate selection queries over imprecise data. In *International Conference on Data Engineering (ICDE)*, March 2004.

[73] Loukas Lazos and Radha Poovendran. Serloc: Robust localization for wireless sensor networks. *ACM Trans. Sen. Netw.*, 1(1):73–100, 2005.

[74] Philip Levis and David Culler. Maté: a tiny virtual machine for sensor networks. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 85–95, New York, NY, USA, 2002. ACM Press.

[75] Philip Levis, David Gay, and David Culler. Active sensor networks. In *Proceedings of the 2nd USENIX/ACM Symposium on Network Systems Design and Implementation (NSDI)*, May 2005.

[76] Philip Levis, Samuel Madden, David Gay, Joseph Polastre, Robert Szewczyk, Alec Woo, Eric A. Brewer, and David E. Culler. The emergence of networking abstractions and techniques in tinyos. In *Proc. 1st Symposium on Networked Systems Design and Implementation (NSDI '04)*, pages 1–14, March 2004.

[77] Philip Levis, Neil Patel, David E. Culler, and Scott Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI 2004)*, pages 15–28, San Francisco, CA, USA, April 2004. USENIX Association.

[78] Ming Li, Deepak Ganesan, and Prashant Shenoy. Presto: Feedback-driven data management in sensor networks. In *Proceedings of the Third Symposium on Networked Systems Design and Implementation (NSDI)*, pages 311–324, May 2006.

[79] Shuoqi Li, Sang Hyuk Son, and John A. Stankovic. Event detection services using data service middleware in distributed sensor networks. In *IPSN*, pages 502–517, 2003.

[80] Hongzhou Liu, Tom Roeder, Kevin Walsh, Rimon Barr, and Emin Gün Sirer. Design and implementation of a single system image operating system for ad hoc networks. In *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 149–162, New York, NY, USA, 2005. ACM Press.

[81] Jie Liu and Feng Zhao. Towards semantic services for sensor-rich information systems. In *2nd International Conference on Broadband Networks*, pages 44–51, October 2005.

[82] Ting Liu and Margaret Martonosi. Impala: a middleware system for managing autonomic, parallel sensor systems. In *PPoPP '03: Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 107–118, New York, NY, USA, 2003. ACM Press.

[83] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[84] Nico Maibaum and Thomas Mundt. Jxta: A technology facilitating mobile peer-to-peer networks. In *MOBIWAC '02: Proceedings of the International Workshop on Mobility and Wireless Access*, page 7, Washington, DC, USA, 2002. IEEE Computer Society.

[85] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49, New York, NY, USA, 2004. ACM Press.

[86] Pedro José Marrón, Matthias Gauger, Andreas Lachenmann, Daniel Minder, Olga Saukh, and Kurt Rothermel. Flexcup: A flexible and efficient code update mechanism for sensor networks. In *Proceedings of the Third European Workshop on Wireless Sensor Networks (EWSN 2006)*, pages 212–227, February 2006.

[87] Pedro José Marrón, Andreas Lachenmann, Daniel Minder, Jörg Hähner, Robert Sauter, and Kurt Rothermel. TinyCubus: A flexible and adaptive framework for sensor networks. In *Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN)*, pages 278–289, January 2005.

[88] Sam Michiels, Wouter Horré, Wouter Joosen, and Pierre Verbaeten. Davim: a dynamically adaptable virtual machine for sensor networks. In *Proceedings of the First International Workshop on Middleware for Sensor Networks (MidSens)*, November 2006.

[89] Microsoft. An architecture for distributed applications on the internet: Overview of microsoft's .net platforms. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 90.1, Washington, DC, USA, 2003. IEEE Computer Society.

[90] Microsoft. Microsoft .NET Compact Framework, July 2007. http://msdn.microsoft.com/mobility/netcf/.

[91] Microsoft. Microsoft .NET Framework, July 2007. http://www.microsoft.com/net/.

[92] Microsoft. Microsoft .NET Micro Framework, July 2007. http://msdn2.microsoft.com/en-us/embedded/bb267253.aspx.

[93] Sun Microsystems. Simplified guide to the java 2 platform, enterprise edition. Technical report, SUN Microsystems, 1999.

[94] Sun Microsystems. Java distributed data acquisition and control. https://jddac.dev.java.net/, June 2007.

[95] Sun Microsystems. Java micro edition. http://java.sun.com/javame/, July 2007.

[96] Sun Microsystems. Java standard edition. http://java.sun.com/javase/, July 2007.

[97] Sun Microsystems. JXME, July 2007. https://jxta-jxme.dev.java.net/.

[98] Sun Microsystems. Sun SPOT world. http://www.sunspotworld.com/, July 2007.

[99] Luca Mottola and Gian Pietro Picco. Programming wireless sensor networks with logical neighborhoods. In *InterSense '06: Proceedings of the first international conference on Integrated internet ad hoc and sensor networks*, page 8, New York, NY, USA, 2006. ACM Press.

[100] René Müller and Gustavo Alonso. Efficient sharing of sensor networks. In *Proceedings of the 3rd IEEE International Conference on Mobile Ad-hoc and Sensor Systems 2006*, October 2006.

[101] René Müller, Gustavo Alonso, and Donald Kossmann. Swissqm: Next generation data processing in sensor networks. In *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research*, January 2007.

[102] René Müller, Gustavo Alonso, and Donald Kossmann. A virtual machine for sensor networks. In *Proceedings of EuroSys 2007*, March 2007.

[103] Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. Lime: A coordination model and middleware supporting mobility of hosts and agents. *ACM Trans. Softw. Eng. Methodol.*, 15(3):279–328, 2006.

[104] Ryan Newton and Matt Welsh. Region streams: functional macroprogramming for sensor networks. In *DMSN '04: Proceeedings of the 1st international workshop on Data management for sensor networks*, pages 78–87, New York, NY, USA, 2004. ACM Press.

[105] Dragos Niculescu and Badri Nath. Ad hoc positioning system (aps). In *Proceedings of GLOBECOM*, November 2001.

[106] Oracle. Oracle sensor edge server. Technical report, Oracle, February 2006.

[107] Mark A. Perillo and Wendi Rabiner Heinzelman. Sensor management policies to provide application qos. *Ad Hoc Networks*, 1(2-3):235–246, 2003.

[108] K. Shashi Prabh and Tarek F. Abdelzaher. Energy-conserving data cache placement in sensor networks. *ACM Trans. Sen. Netw.*, 1(2):178–203, 2005.

[109] Michael Rabbat, Jarvis Haupt, Aarti Singh, and Robert Nowak. Decentralized compression and predistribution via randomized gossiping. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 51–59, New York, NY, USA, 2006. ACM Press.

[110] Jan Rellermeyer, Gustavo Alonso, and Timothy Roscoe. R-osgi: Distributed applications through software modularization. In *To appear in 8th International Middleware Conference (Middleware 2007)*, November 2007.

[111] Kay Römer, Christian Frank, Pedro José Marrón, and Christian Becker. Generic role assignment for wireless sensor networks. In *Proceedings of the 11th ACM SIGOPS European Workshop*, pages 7–12, September 2004.

[112] Sean Rooney, Daniel Bauer, and Paolo Scotton. Edge server software architecture for sensor applications. In *SAINT '05: Proceedings of the The 2005 Symposium on Applications and the Internet (SAINT'05)*, pages 64–71, Washington, DC, USA, 2005. IEEE Computer Society.

[113] Andreas Savvides, Heemin Park, and Mani B. Srivastava. The bits and flops of the n-hop multilateration primitive for node localization problems. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 112–121, New York, NY, USA, 2002. ACM Press.

[114] Douglas C. Schmidt. Middleware for real-time and embedded systems. *Communications of the ACM*, 45(6):43–48, 2002.

[115] Bo Sheng, Qun Li, and Weizhen Mao. Data storage placement in sensor networks. In *MobiHoc '06: Proceedings of the seventh ACM international symposium on Mobile ad hoc networking and computing*, pages 344–355, New York, NY, USA, 2006. ACM Press.

[116] Jon Siegel. Omg overview: Corba and the oma in enterprise computing. *Commun. ACM*, 41(10):37–43, 1998.

[117] F. Sivrikaya and B. Yener. Time synchronization in sensor networks: a survey. *IEEE Network*, 18(4):45–50, July-August 2004.

[118] Eduardo Souto, Germano Guimaraes, Glauco Vasconcelos, Mardoqueu Vieira, Nelson Rosa, Carlos Ferraz, and Judith Kelner. Mires: a publish/subscribe middleware for sensor networks. *Personal Ubiquitous Comput.*, 10(1):37–44, 2005.

[119] Chavalit Srisathapornphat, Chaiporn Jaikaeo, and Chien-Chung Shen. Sensor information networking architecture. In *2000 International Workshop on Parallel Processing (ICPP 2000)*, Toronto, Canada, August 2000.

[120] Michael Stal. Web services: beyond component-based computing. *Commun. ACM*, 45(10):71–76, 2002.

[121] Kirsten Terfloth, Georg Wittenburg, and Jochen Schiller. Facts - a rule-based middleware architecture for wireless sensor networks. In *First International Conference on Communication System Software and Middleware (Comsware 2006)*, pages 1–8, January 2006.

[122] Dirk Trossen and Dana Pavel. N-RSA high level system architecture. Technical report, NOKIA, October 2006.

[123] Daniela Tulone and Samuel Madden. Paq: Time series forecasting for approximate query answering in sensor networks. In *EWSN*, pages 21–37, 2006.

[124] Jim Waldo. The jini architecture for network-centric computing. *Commun. ACM*, 42(7):76–82, 1999.

[125] Matt Welsh. Exposing resource tradeoffs in region-based communication abstractions for sensor networks. *SIGCOMM Comput. Commun. Rev.*, 34(1):119–124, 2004.

[126] Kamin Whitehouse, Cory Sharp, David E. Culler, and Eric A. Brewer. Hood: A neighborhood abstraction for sensor networks. In *MobiSys*, 2004.

[127] Kamin Whitehouse, Feng Zhao, and Jie Liu. Semantic streams: A framework for composable semantic interpretation of sensor data. In *EWSN*, pages 5–20, 2006.

[128] Ingwar Wirjawan, Joel Koshy, Raju Pandey, and Yann Ramin. Balancing computation and code distribution costs: The case for hybrid execution in sensor networks. Technical Report CSE-2006-35, Department of Computer Science, UC Davis, 2006.

[129] Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, 2002.

[130] Yang Yu, Loren J. Rittle, Vartika Bhandari, and Jason B. LeBrun. Supporting concurrent applications in wireless sensor networks. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 139–152, New York, NY, USA, 2006. ACM Press.