EURASIP Journal on Wireless Communications and Networking 2005:5, 774–788

A Survey of Application Distribution in Wireless Sensor Networks

Mauri Kuorilehto
Institute of Digital and Computer Systems, Tampere University of Technology, P.O. Box 553, 33101 Tampere, Finland
Email: mauri.kuorilehto@tut.fi

Marko Hännikäinen
Institute of Digital and Computer Systems, Tampere University of Technology, P.O. Box 553, 33101 Tampere, Finland
Email: marko.hannikainen@tut.fi

Timo D. Hämäläinen
Institute of Digital and Computer Systems, Tampere University of Technology, P.O. Box 553, 33101 Tampere, Finland
Email: timo.d.hamalainen@tut.fi

Received 14 June 2004; Revised 23 March 2005

Wireless sensor networks (WSNs) are deployed to an area of interest to sense phenomena, process sensed data, and take actions accordingly. Due to the limited WSN node resources, distributed processing is required for completing application tasks. Proposals implementing distribution services for WSNs are evolving on different levels of generality. In this paper, these solutions are reviewed in order to determine the current status. According to the review, existing distribution technologies for computer networks are not applicable for WSNs. Operating systems (OSs) and middleware architectures for WSNs implement separate services for distribution within the existing constraints but an approach providing a complete distributed environment for applications is absent. In order to implement an efficient and adaptive environment, a middleware should be tightly integrated in the underlying OS. We recommend a framework in which a middleware distributes the application processing to a WSN so that the application lifetime is maximized. OS implements services for application tasks and information gathering as well as control interfaces for the middleware.

Keywords and phrases: ad hoc networking, distribution, service discovery, task allocation, wireless sensor networks.

1. INTRODUCTION

Wireless sensor networks (WSNs) have gained much attention in both public and research communities because they are expected to bring the interaction between humans, environment, and machines to a new paradigm. Despite being a fascinating topic with a number of visions of a more intelligent world, there still exists a huge gap in the realizations of WSNs. In this paper, we define WSNs as networks consisting of independent, collaborating nodes that can sense, process, and exchange data as well as act upon the data content. Compared to traditional communication networks, there is no preexisting physical infrastructure that restricts topology.

WSNs are typically ad hoc networks [1] but there are major conceptual differences. First, WSNs are data-centric with an objective to deliver time sensitive data to different destinations. Second, a deployed WSN is application-oriented and performs a specific task. Third, messages should not be sent to individual nodes but to geographical locations or regions defined by data content [2].

In WSNs quantitative requirements in terms of latency and accuracy are strict due to the tight relation to the environment. In general, the capabilities of an individual sensor node are limited, but the feasibility of WSN lies on the joint effort of the nodes. Thus, WSNs are distributed systems and need distribution algorithms. Another motivation for distribution is the resource sharing. Further, to obtain results, WSN applications typically require collaborative processing of the nodes sensing different phenomena in diverse areas [2].

The main focus of WSN research, as well as wireless ad-hoc network research in general, has been on different protocol layers, reviewed in [2, 3, 4, 5, 6, 7, 8] and on energy efficiency [9, 10]. Recently, issues concerning security,

context-sensitivity, and self-organization have gained more attention [11]. Surveys concerning application layer issues and prototype implementations are fairly limited [4, 12, 13]. Furthermore, proposals implementing distribution are emerging as the complexity of applications increases. These are covered in [2] but the discussion of proposals supporting application distribution is limited to few solutions for distribution control.

In this paper, we focus on four essential distribution aspects in WSNs, namely, *service discovery*, *task allocation*, *remote task communication*, and *task migration*. The service discovery comprises of identifying and locating services and resources required by a client. In homogeneous WSNs, the service discovery is not important but when node platforms and the composition of tasks are heterogeneous, the service discovery is essential. The task allocation specifies a set of sensor nodes, on which the execution of an application task is activated. The remote task communication covers the means for communication between distributed tasks through a wireless communication link. The task migration means the methods for transferring a task executable from a sensor node to another. The algorithms defining the target nodes for migration are included in the task allocation.

Algorithms that are tightly bound to an application are not discussed. The presented distribution aspects are selected due to their generality for different types of WSNs and applications. We omit, for example, data fusion and data aggregation that are beneficial only for applications that gather data to a centralized storage.

In this paper we review the application distribution for WSNs focusing on distribution implemented in *systems software*. By systems software we mean software components providing application-independent services and managing node resources. The proposed solutions vary according to tools provided, requirements placed on the underlying platforms, and targeted applications and environments. However, the current proposals lack an integrated solution providing a distributed operating environment for WSN applications. This approach would lead to a more efficient usage of resources.

This paper is organized in two main parts as follows. The first part describes the basics of objectives, challenges, and systems software solutions of WSNs. In addition, a summary of WSN application proposals is presented in order to define requirements. The second part starting in Section 3 contains the survey of distribution proposals followed by their analysis in Section 4. Finally, conclusions are given in Section 5.

2. OVERVIEW OF WSNs

In order to give an overview of WSN applications, we review some examples and their characteristics. These are listed in Table 1. The selection is mainly based on prototype implementations and thus all the scopes of WSNs might not be represented.

The first column in Table 1 lists the applications and the second classifies them according to the main task. The third column presents the requirements set by the application. The networking requirements in terms of data amount and frequency are defined in the fourth column, while the last column gives the scale and density of the application.

Most of the applications gather, evaluate, or aggregate data from different types of sensors. Major differences are in networking requirements and complexity. Unfortunately, accurate values or limits to these properties are not often reported, which complicates a fair comparison.

The nature of applications listed in Table 1 varies, but at least four main tasks can be identified [28]. *Monitoring* is used to continually track a parameter value in a given location, and *event detection* recognizes occurrences of events. *Object classification* attempts to identify an object or its type and *object tracking* traces movements of an object.

For the presented applications, the "worst-case" WSN would comprise of an extensive number of nodes with varying density and a network topology that constantly changes due to the errors in communication, mobility of nodes, and inactive nodes [3]. To complete complex tasks in the scenario, the application requires distributed processing within the network.

In our view, WSN application quality of service (QoS) is constructed from network lifetime, network load, accuracy of data, and fault tolerance. Network load in this case comprises of the required data latency, throughput, and reliability. WSN protocols and their functions are adapted according to the QoS requirements. Currently, security is a QoS issue that is often omitted in WSNs. The natural reason is that security requires too much resources [2].

For the rest of the paper we define an *environmental monitoring application* that is used for the analysis of the proposed solutions. For clarification, we refer to the application as *EnvMonitor*. The main task of the application is the constant gathering of location-dependent information within a defined area. In addition to the passive monitoring involved in the environmental monitoring applications in Table 1, *EnvMonitor* consists of active monitoring tasks reacting to condition changes in WSN. The passive monitoring data are gathered to a central storage and aggregated during the routing. Active in-network monitoring tasks execute signal processing algorithms locally in order to determine threshold values for temperature and humidity. When a threshold is reached, a set of predefined actions modifying the application QoS and the communication topology taken. The modifications alter the requirements for data composition, accuracy, and latency. The priority of active monitoring tasks precedes passive monitoring.

2.1. Systems software for WSNs

A general-purpose operating system (OS) is an example of systems software. Early WSNs have not included systems software due to scarce resources and simplicity of applications. However, complex applications require systems software because it eases the control of resources and increases the predictability of execution. The heterogeneity of platforms can be hidden under common interfaces provided by the software. Still, the major disadvantages are heavy computation and memory usage.

Table 1: Examples of prototyped applications for WSNs.

Application	Type	Requirements	Data amount and frequency	Scale and density
Great Duck Island [14]	Environmental monitoring	Data archiving, Internet access, long lifetime	Minimal, every 5–10 min, 2–4 h per day	32 nodes in 1 km^2
PODS in Hawaii [15]	Environmental monitoring	Digital images, energy-efficiency	Large data amounts, infrequently	30–50 nodes in 5 hectares
CORIE (Columbia River) [16]	Environmental monitoring	Base stations, lifetime	Moderate data amounts, infrequently	18 nodes in Columbia River
Peek value evaluation [17]	Environmental monitoring	Collaborative processing, minimal network traffic	Moderate data amounts, periodically	Case dependent
Flood detection [18]	Environmental monitoring	Current condition evaluation	50 bytes every 30 s	200 nodes 50 m apart
SSIM (artificial retina) [19]	Health	Image identification, realtime, complex processing	Large data amounts, frequently every 200 ms	100 sensors per retina
Human monitoring [20]	Health	Quality of data, security, alerts	Moderate data amounts, depend on the human stress level	Several nodes per human
Mountain rescue [21]	Health	Communication intensive	Large data amounts in high frequency	One per rescuer in mountain area
WINS for military [22]	Military	Target identification, realtime, security, quality of data	Large data amounts, infrequently	Several distant nodes
Object tracking [23]	Military	Collaborative processing, realtime, location-awareness	Large data amounts with high frequency near an object	7 (prototype) nodes in proximity
Vehicle tracking [24]	Military	Identification and coordination, realtime	Large data amounts every 8 s near an object	1024 nodes in 40 km^2
Intelligent input/output [25]	Home entertainment	Communication intensive	Large data amounts with high frequency	One node per input device
WINS condition monitoring [22]	Machinery monitoring	Data aggregation, machinery lifetime projection	Depend on machinery complexity and its current status	Few nodes per machinery
Smart kindergarten [26]	Education	Video streaming, identification, location-awareness	Large data amounts in variable frequencies	Tens of sensors, indoor
Smart classrooms [27]	Education	Context-sensing, data exchange	Large data amounts in random frequency	Several nodes in classroom

The systems software for WSNs implements *single node control* and *network-level distribution control*. The single node control software implements the low-level routines in a node, whereas the network-level distribution control manages application execution within several nodes.

Single node control

The single node control operates on a physical node depicted in Figure 1. A processing unit consists of CPU, storage devices, and an optional memory controller for accessing the instruction memory of the main CPU. A sensing unit consists of sensors and an analog-to-digital converter (ADC). A transceiver unit enables the communication with other sensor nodes. A power unit can be extended by a power generator that harvests energy from environment. Other peripheral devices, like actuators for moving the node and location finding systems, are attached to the node depending on the application requirements [3].

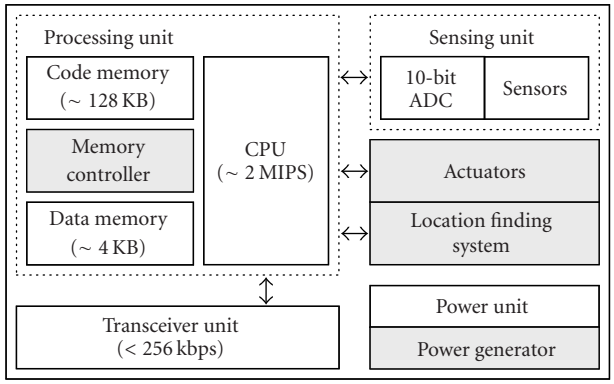

Figure 1: Reference hardware platform architecture of a sensor node.

The reference values in Figure 1 are the resources available in MICA2 mote [29]. The power consumption of a node is in order of mW when active and in order of μW when the node is in sleep. The power unit is typically an AA battery or similar energy source.

The single node control is accomplished by OS or virtual machine (VM). In the reference platform, OS is executed on the main CPU and it uses the same instruction and data memories as applications. Services implemented by OS include scheduling of tasks, interprocess communication (IPC) between tasks, memory control, and possible power control in terms of voltage scaling and component activation and inactivation. OS provides interfaces to access and control peripherals. The interfaces are typically associated with layered software components with more sophisticated functionality, for example a network protocol stack.

Network-level distribution control

Distribution control relies on networking. Figure 2 depicts an example protocol stack for WSN in comparison to two widely utilized stacks, the OSI model [1] and a distributed system in a wireless local area network (WLAN). In a WLAN computer, the TCP/IP stack is used through a sockets application programming interface (API). The WLAN adapter that contains the medium access control (MAC) protocol and the WLAN radio is accessed by a device driver.

There is no unified protocol stack for WSNs and most of the proposed stacks are just collections of known protocol functions. At the moment, the IEEE 1451.5 Wireless Sensor Working Group [30] is standardizing the physical layer for WSNs with an intention to adapt link layers from other wireless standards, for example, Bluetooth [31], IEEE 802.15.4 low-rate wireless personal area network (LR-WPAN) [32], or IEEE 802.11 WLAN [33]. Other types of networks posing common characteristics with WSNs are mobile ad hoc networks (MANETs) [34] targeted to address mobility.

In WSNs, the essential protocol layers are the MAC protocol on the data link layer and the routing protocol on the network layer. The MAC protocol creates a network topology and shares the transmission medium among sensor nodes. The topology in WSNs is either flat, in which all sensor nodes are equal, or clustered, in which communication is controlled by cluster headnodes. The routing protocol allows communication via multihop paths. A transport protocol that implements end-to-end flow control is rarely utilized in WSNs. The middleware layer is equivalent to the presentation layer in the OSI model [1].

For WSNs, the development of a distributed environment requires the consideration of all four distribution aspects. The control actions are taken according to the application QoS. The distribution aspects are typically implemented on the middleware layer on top of OS. Thus, the middleware component can reside in different types of platforms. In addition to OS routines, the middleware utilizes networking interface to implement communication between its own instances on different sensor nodes. Some distribution aspects can also be implemented directly by OS.

3. SURVEY OF DISTRIBUTION PROPOSALS

Numerous technologies for the service discovery and remote task communication are available for computer networks. The task migration is typically a transfer of a binary code image or a Java applet. In computer networks, the task allocation is often not the main concern as resources are sufficient. Even though not directly applicable for WSNs, the computer network technologies define the basic paradigms and algorithms for the application distribution.

Other types of wireless ad hoc networks, like MANETs and Bluetooth, have common characteristics with WSNs. First, communication in these networks is very similar to WSNs. Second, the resource constraints must be considered, even though the limits are looser than in WSNs. For this reason we include technologies proposed for MANETs and Bluetooth in our assessment of WSN proposals.

A distinct categorization of proposed solutions for WSNs cannot be made since a proposal typically present a more complete architecture addressing several distribution aspects. Therefore, we categorize the proposals according to their system architecture to OSs, VMs, middlewares, and stand-alone protocols.

3.1. Architectural paradigms

Figure 3 presents three architectural paradigms for distribution, which are *client-server*, *mobile code*, and *tuple space*. In computer networks, the client-server architecture is applied for the service discovery and remote task communication. It consists of one or multiple servers hosting a set of services and clients accessing these. A directory service is maintained at the server in the service discovery. In the remote task communication, a client outsources a task processing to a server. Two alternatives are available, remote procedure calls (RPCs) and object-oriented remote method invocations (RMIs). As the internal data and state of objects are accessed only through the object interface, RMI achieves better abstraction and fault tolerance. In addition, objects can be cached and moved [35].

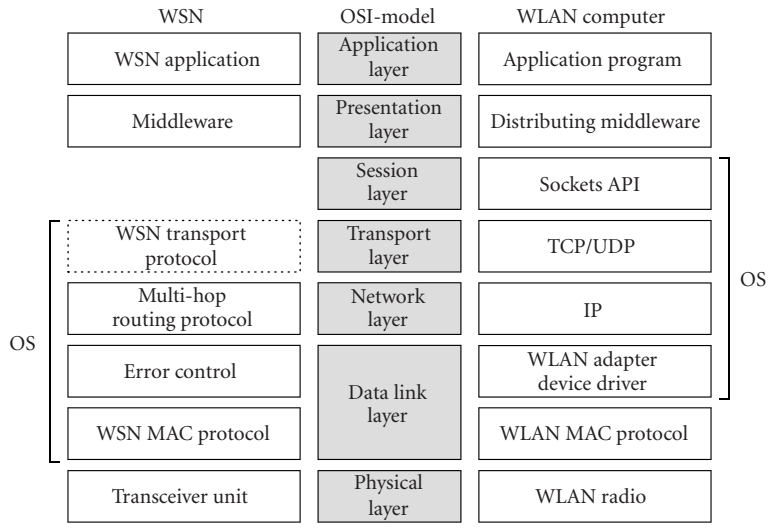

Figure 2: OSI model, WSN, and distributed system in WLAN protocol layers.

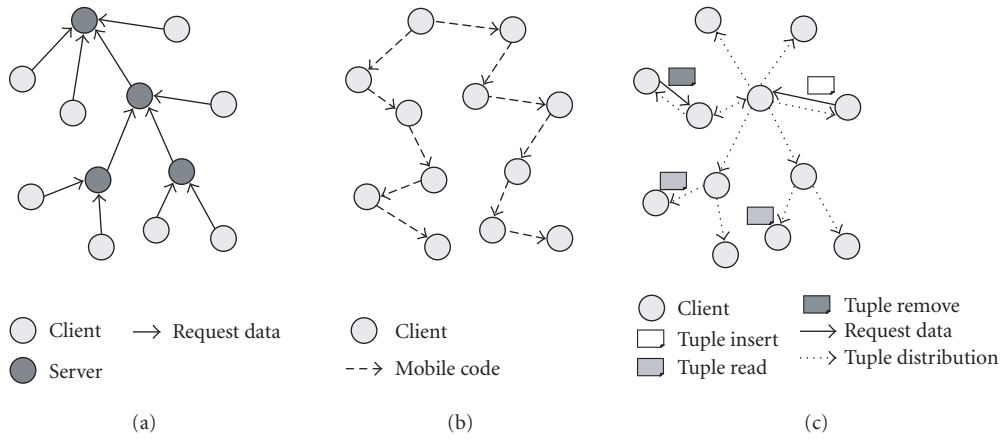

Figure 3: Three architectural paradigms for distribution: (a) client-server, (b) mobile code, and (c) tuple space.

Differences in programming languages and platforms must be hidden in the remote task communication. Stub procedures are generated for this from interface definitions. A stub procedure at the client marshals a procedure call to an external data presentation, which is then unmarshalled back to a primitive form at the server [35].

In the mobile code paradigm, instead of moving data from a client to a server for processing, the code is moved to the data origins, and data are then processed locally. A *mobile agent* is an object that in addition to the code carries its state and data. Furthermore, mobile agents make migration decisions autonomously. They are typically implemented on top of VMs for platform independency [36].

The concept of tuple space was proposed originally in Linda [46] for the remote task communication, but it is applicable also for the service discovery. Tuples are collections of passive data values. A tuple space is a pool of shared information, where tuples are inserted, removed, or read. Data are global and persistent in the tuple space and remain until explicitly removed. In the tuple space, a task does not need to know its peer task, tasks do not need to exist simultaneously, and they do not need to communicate directly.

3.2. Computer networks

Service location protocol (SLP) [47], Jini [48], universal plug and play (UPnP) [49], and secure service discovery service (SDS) [50] implement a client-server architecture service discovery in computer networks. The tuple space is utilized in JavaSpaces [51] on top of Jini and in TSpaces [52]. For the remote task communication, Sun RPC [53] and distributed computing environment (DCE) [54] are well-known RPC technologies. The best-known object-oriented technologies are common object request broker architecture (CORBA) [55], Java RMI [56], and Microsoft's distributed common object model (DCOM) [57]. The mobility of terminals is addressed in Mobile DCE [58], Mobile CORBA [59], and Rover Toolkit [60]. Schedulers for computer clusters implement task allocation within a cluster by allocating tasks to the most applicable resources [61].

Table 2: Implemented distribution aspects in single node proposals.

Proposal	Target network	Resource requirements (CPU/code memory/ data memory)	Service discovery	Task allocation	Remote task communication	Task migration
OS-based architectures						
EYES OS [37]	WSN	1 MHz / 60 KB / 2 KB	Resource requests	Not supported	RPC	Not supported
BTnodes [38]	WSN	8 MHz/ 128 KB/ 64 KB	Tuple space	Not supported	Callbacks	Smoblets
TinyOS [39]	WSN	8 MHz/ 128 KB/ 4 KB	Not supported	Not supported	Active messages	Not supported
BerthaOS [40]	WSN	22 MHz/ 32 KB/ 2,25 KB	Not supported	Not supported	BBS	Binary code
MOS [25]	WSN	8 MHz/ > 64 KB/ > 1 KB	Not supported	Not supported	Not supported	Binary code download
QNX [41]	LAN	33 MHz/ 100 KB/ N/A	Network manager	SMP scheduler	Message passing	Not supported
OSE [42]	LAN	N/A/ 100 KB/ N/A	Hunting service	Not supported	Phantom process	Not supported
VM-based architectures						
Sensorware [17]	WSN	N/A/ 1 MB/ 128 KB	Not supported	Script population specification	Not supported	TCL script migration
MagnetOS [43]	WSN	N/A / N/A / N/A	Not supported	Automatic object placement	DVM [44]	Mobile Java objects
Maté [45]	WSN	8 MHz/ 128 KB/KB	Not supported	Not supported	Not supported	Code capsule update

Distribution technologies designed for computer networks are typically both computation and communication intensive and cannot be implemented on sensor nodes. They are based on the client-server architecture and use detailed specifications for services and interfaces. These technologies do not consider the possible mobility or unavailability of sensor nodes. While mobility is addressed in Mobile DCE, Mobile CORBA, or Rover toolkit, these still rely on the client-server architecture from DCE and CORBA.

3.3. Distribution proposals for WSNs

From systems software proposals for WSNs, OSs and VMs implement the single node control and middleware architectures implement the network-level distribution control. These can be supported by stand-alone protocols that address only a single distribution aspect. We contribute the WSN proposals according to distribution aspects they implement.

OS-based architectures

The distribution aspects implemented in OSs are listed in Table 2. In addition, the second column defines the type of a network OS is targeted for, while the third one gives OS resource requirements. In WSNs, OSs implement a very limited set of services and they are fairly primitive in their nature. As shown in Table 2, the remote task communication is addressed typically by providing a simple method for RPC. The service discovery is rarely implemented in OS but on a higher system services layer that is associated to OS. Tasks migrate as binary code, because OSs do not support code interpreting.

The service discovery is implemented in EYES OS [37] on a distributed services layer above the OS by utilizing resource requests to neighbor nodes. Also Bluetooth smart nodes (BTnodes) [38] implement distribution in system services above a lightweight OS. BTnodes use the tuple space to implement the service discovery. The task allocation is not implemented in any of the proposals.

A client-server type RPC is applied to the remote task communication in TinyOS [39], BerthaOS (for Pushpin nodes) [40], and in EYES OS. In the component-based TinyOS, the handler name of the remote component and required parameters are encapsulated in a TinyOS active message. BerthaOS uses bulletin board system (BBS) for IPC and nodes can post messages also to BBS of a neighbor node. In EYES OS, the basic RPC between neighbor nodes is applied. BTnodes use the tuple space also for information sharing and for sending notifications to callbacks routines.

The task migration as binary code is possible in BetrhaOS and in MultimodAI NeTworks of In-situ Sensors (MANTIS) OS (MOS) [25]. BerthaOS allows the in-network initiation of transfers and checks the code integrity using a simple checksum, but neither it nor MOS considers the vulnerability of the system to malicious code. In BTnodes, precompiled Java classes, smoblets, are able to migrate but they must be executed on more powerful platforms.

Embedded OSs and RealTime OSs (RTOS), like QNX [41] and OSE [42], support service discovery and remote task communication in OS services. In QNX, the network of computers is abstracted to a single homogenous set of resources. QNX uses message passing to implement IPC and hides remote locations in process and resource managers.

The local managers interact with a network manager that handles name resolution. OSE uses stub procedures, referred to as phantom processes, for the remote task communication. A phantom process uses a link handler to communicate with the peer phantom process on the remote node. The remote node is discovered by a hunting system service that broadcasts service requests to the network.

From these proposals, QNX and OSE offer a distributed environment for applications, but they require more efficient sensor node platforms. Their resource requirements shown in Table 2 do not contain all the components required for the implementation of the distributed environment. The resource requirements set by other OSs are in the same order of magnitude. All the proposed OS architectures implement the single node control over the application tasks of *EnvMonitor*. The most applicable environment for *EnvMonitor* is available in BTnodes, where the tuple space implements service discovery and callbacks and smoblets support in-network distributed processing.

VM-based architectures

Compared to OSs, VMs offer hardware platform independency and substitute the lack of hardware protection by the protection implemented in code interpreters. The distribution aspects, target network, and required resources of VM architectures are categorized in Table 2. As shown, the mobile code is a common approach to distribution, whereas service discovery is not supported.

The task allocation is supported by Sensorware [17] and MagnetOS [43]. The population of tool command language (TCL) scripts in Sensorware is specified in the scripts themselves. MagnetOS utilizes automatic object placements algorithms that adaptively attempt to minimize communication by moving Java objects nearer to the data source. The remote task communication is addressed only in MagnetOS that relies on distributed VM (DVM) [44]. DVM abstracts network of computers to a single Java VM (JVM).

As depicted in Table 2, the mobile code is a TCL script in Sensorware, a custom bytecode capsule in Maté [45], and a Java object in MagnetOS. The size of the TCL scripts and especially the Maté code capsules is small compared to the size of Java objects. In Maté that operates on top of TinyOS a new code capsule is sent in TinyOS active messages to all nodes.

From the proposed solutions, Sensorware and MagnetOS implement task migration and task allocation, whereas in Maté only the latest code version is updated to all nodes. Implementation of MagnetOS on sensor nodes is not possible, Sensorware sets considerable requirements for underlying platforms, and Maté is implemented to very resource constrained nodes.

Like OSs, these proposals implement the single node control for *EnvMonitor*. From these proposals, Sensorware is the most suitable for *EnvMonitor* due to its migration, allocation, and task coprocessing capabilities. However, the control for these actions must be implemented by the application scripts.

Middleware architectures

Middleware architectures implement a higher abstraction level environment for applications. Generally, three different approaches in WSN middlewares can be identified. First, a middleware coordinates the task allocation based on the application QoS. Second, WSN is abstracted to a database that supports query processing. Third, a middleware controls application processing in the network based on the current context of surrounding environment. The context depends on the location, nearby people, hosts, and devices, and the changes in these over time [62]. The target network and distribution aspects for proposals are listed in Table 3.

Application QoS is applied for controlling the task allocation in the configuration adaptation of the middleware linking applications and networks (MiLAN) [20], in the resource management of the cluster-based middleware architecture for WSNs [63], and in QoSProxies of the QoS-aware middleware for ubiquitous and heterogeneous environments [64]. The cluster-based middleware and MiLAN adapt also the network topology. The QoSProxy selects an application configuration matching available resources and makes resources reservations to guarantee the specified QoS for that configuration. Both MiLAN and QoS-aware middleware adopt service discovery protocols from computer network solutions. QoS-aware middleware requires a more powerful platform than the other two.

A database approach is taken in sensor information and networking architecture (SINA) [24], in TinyDB [65] on top of TinyOS, and in Cougar [66]. In SINA, database queries are injected to network as sensor querying and tasking language (SQTL) [71] scripts. These scripts migrate from node to node depending on their parameters. The task allocation in SINA is implemented by a sensor execution environment (SEE), which compares SQTL script parameters to node attributes and executes script only if these match. In TinyDB and Cougar, the task allocation is implemented by a query optimizer that determines energy-efficient query routes. The query plans generated by the query optimizer are parsed in the nodes and then executed accordingly. TinyDB supports also event-based queries that are initiated in-network on the occurrence of an event.

Application adaptation based on the current context is performed by Linda in a mobile environment (LIME) [67], mobile agent runtime environment (MARE) [21], and reconfigurable context-sensitive middleware (RCSM) [27]. Service discovery is implemented by the tuple space in LIME and MARE. RCSM uses a custom RKS [68] protocol that reduces communication by advertising services only if they can be activated in the current context and potential clients are in the vicinity. LIME implements task allocation by reactions added to tuples. The MARE control manages nearby mobile agents and allocates tasks to the agents. RCSM ADaptive object containers (ADC) activate tasks in an appropriate context.

The tuple space in LIME and MARE is used also for the remote task communication. LIME supports also location-dependent recipient identification. RCSM utilizes RCSM

Table 3: Implemented distribution aspects in middleware and stand-alone protocol proposals.

Proposal	Target network	Service discovery	Task allocation	Remote task communication	Task migration
Middleware architectures					
MiLAN [20]	WSN	SLP, Bluetooth SDP	Configuration adaptation	Not supported	Not supported
Cluster-based middleware [63]	WSN	Not supported	Resource management	Not supported	Not supported
QoS-aware middleware [64]	MANET	SLP/Jini/SDS	QoSProxy	Not supported	Not supported
SINA [24]	WSN	Not supported	Attribute matching in SEE	Not supported	SQTL scripts
TinyDB [69]	WSN	Not supported	Query optimizer, event-based queries	Not supported	Not supported
Cougar [66]	WSN	Not supported	Query optimizer	Not supported	Not supported
LIME [67]	MANET	Tuple space	Context reaction	Tuple space	Mobile Java objects
MARE [21]	MANET	Tuple space	MARE control	Tuple space	Mobile Java objects
RCSM [27]	MANET	RKS [68]	Adaptive object containers	R-ORB	Not supported
Stand-alone protocols					
GSD [69]	MANET	Service groups	Not supported	Not supported	Not supported
Bluetooth SDP [31]	Bluetooth	Clients and servers	Not supported	Not supported	Not supported
Task migration in [70]	WSN	Not supported	Not supported	Not supported	Edit scripts

context-sensitive object request broker (R-ORB) that adapts basics from CORBA ORB. Both LIME and MARE utilize mobile agents implemented as Java objects for the task migration.

Unlike OSs and VMs, most of the middleware architectures implement the network-level distribution control but do not address the single node control. Middlewares relying on the application QoS specification address mainly task allocation, but leave other aspects to external components. The database abstraction is applicable to a certain type of applications, like *EnvMonitor*, but the expressivity of the SQTL scripts in SINA, the event-based queries in TinyDB, and especially the query processing capabilities in Cougar do not support complex in-network processing. As can be seen from Table 3, context-aware proposals cover distribution aspects extensively. They implement extensive environment for *EnvMonitor* but their resource requirements are too high for sensor nodes.

Stand-alone protocols

The environment provided by OSs, VMs, or middleware architectures can be supported by stand-alone protocols implementing dedicated functions. We do not cover WSN MAC and routing protocols but focus on protocols that implement any of the four distribution aspects. The protocols and their target networks are listed in Table 3.

The group-based service discovery protocol (GSD) for MANETs [69] and the Bluetooth service discovery protocol (SDP) [31] implement the service discovery. In GSD, terminals advertise their services and nearby service groups within the distance of n hops. Service requests are forwarded towards the service provider based on group advertisements. A Bluetooth terminal maintains information about its services in an SDP server. Searching and querying for existing services are performed by an SDP client that queries one server at a time.

An approach for minimizing the transferred binary code size on the task migration is proposed in [70]. The proposal transmits only the differences between the existing and the new code. The algorithm is adopted from the *diff* command of UNIX.

These protocols can be used as separate components for *EnvMonitor*, but none of them provides a complete environment. GSD is communication intensive due to the multi-hop advertisements. Bluetooth SDP does not support broadcast queries, which restricts its applicability in large WSNs. The task migration proposed in [70] cannot be initiated in WSNs due to the complexity of the algorithm and the lack of integrity checking.

4. ANALYSIS OF PROPOSALS

A comprehensive comparison of the proposals is problematic due to the diversity of platforms, applications, and implementations. However, the requirements for each distribution aspect are similar, which makes their assessment possible. In the analysis, we concentrate on the proposals targeted for WSNs.

Table 4: System testing and validation environments for distribution proposals.

Proposal	Test environment	Simulation and testing tools	Prototype platform	Result accuracy	Published results
OS-based architectures					
TinyOS [39]	Prototype	TOSSIM [72]	Motes	Accurate	Component sizes, OS routine delays, computation costs
BerthaOS [40]	Prototype	None	Pushpin	None	Functionality mentioned
EYES OS [37]	None	None	None	None	None
MOS [25]	Prototype	PC emulator XMOS [25]	Nymph	Moderate	Memory and power consumption, test application performance results
BTnodes [38]	Prototype	None	Micro-size BTnodes	Moderate	Component sizes, energy consumption
VM-based architectures					
Sensorware [17]	Prototype	SensorSim [73]	Linux IPAQ	Accurate	Framework size, execution delays, energy consumption
MagnetOS [43]	Windows/Linux JVM	Custom packet-level simulator	PC	None	Internal algorithm comparison in simulator
Maté [45]	Prototype	TOSSIM [72]	TinyOS mote	Accurate	Bytecode overhead, installation costs, code infection performance
Middleware architectures					
MiLAN [20]	None	None	None	None	None
Cluster-based middleware in [63]	Algorithm simulation	Custom simulator	None	None	Heuristic resource allocation, algorithm performance
Qos-aware middleware in [64]	None	None	None	None	None
SINA [24]	Simulations	GloMoSim [74]	None	Poor	SINA networking overhead, application performance
TinyDB [65]	Simulations, prototype	Custom environment	TinyOS mote	Accurate	Query routing performance in simulations, sample accuracy and sampling frequency in prototypes
Cougar [66]	None	None	None	None	None
LIME [67]	JVM	None	PC	Poor	Approximations about Java code size
MARE [21]	JVM	None	PDA	Poor	Service discovery performance
RCSM [27]	Prototype	None	PDA with custom hardware	RCSM poor, RKS accurate	RCSM memory consumption, RKS size, communication, energy consumption
Stand-alone protocols					
GSD [69]	Simulations	GloMoSim [74]	None	Poor	Influence of internal parameters on service discoverability
Task migration in [70]	PC	None	Tested in EYES nodes	Accurate	Algorithm performance, influence of internal parameters

4.1. Testing and validation of WSN proposals

Discussed WSN architectures vary in their complexity and requirements. In order to provide a scope for the assessment of proposals, their testing and validation environments are presented in Table 4. The test environment is presented in the second column. The simulation and testing tools and prototype platforms identify the proposal validation tools and test platform. The published results and their accuracies are listed in the last two columns.

Generally, prototypes exist for the single node architectures and their results are accurate including information required for comparison. Instead, on the middleware layer, proposals are evaluated by simulations or not at all. The simulation results are inaccurate as they compare only the internal algorithms and do not give any information for a general comparison. Of course, exceptions exist in both cases.

Even though some of the presented results in Table 4 are accurate and their scope is adequate, the direct comparison

Table 5: Characteristics of technologies implementing service discovery.

Technology	Communication	Scalability	Fault tolerance	Requirements	Benefits (pros)	Problems (cons)
Resource requests	Requests to neighbors	Restricted to neighbors	Broadcasted to all neighbors	Resource declaration	One-hop communication	Scalability
Tuple space	Tuple operations	Balancing between memory and scale	Redundant information	Memory pool in each node	Source and target independency	Communication/ memory load
Network manager	Name resolution requests to manager	Local manager area, but extensible	Possibly redundant network managers	Resource managers, register to manager	Scalability due to naming	Name resolution, communication load
Hunting service	Broadcast hunt service requests	Not restricted	Lost services can be rehunted	Remote service identification	Lightweight after initiation	First hunt latency and communication load
Bluetooth SDP	Peer-to-peer link	Only nearby nodes one at a time	Service information only in the host	Bluetooth protocol stack	Querying for available services	Scalability, no broadcast
RKS	Advertises for potential clients	Only to nearby clients	Advertisements when context and clients applicable	Context definitions for services	Advertisements	Scalability
GSD service groups	Service and group advertisements	n-hop diameter, but groups span wider	Redundant information	Service registration	Request routing based on group advertisements	Communication load (both advertisements and requests used)

of distribution performance is not possible. The prototype platforms vary in their efficiency, the simulators in their accuracy, and the test applications in their requirements and functionality. As the area is evolving rapidly, generally accepted benchmarks would ease the comparison of the proposals. However, the definition of general-enough benchmarks for WSNs is difficult due to their application-specific nature.

4.2. Comparison of technologies

We classify the technologies for each distribution aspect separately. The classification dimensions for a technology are *communication mechanism*, *scalability* to large WSNs, *fault tolerance*, and *requirements* that must be met before the technology can be used. For each technology, we also assess its pros and cons in general. These dimensions offer tools for the evaluation of the robustness and applicability of a technology for different kinds of WSNs and applications.

Service discovery

The classification of the service discovery technologies in the proposals according to the defined dimensions is presented in Table 5. From the presented solutions, all but the tuple space and GSD rely on client-server architecture. Still, the network manager is the only centralized server. In general, two problems can be identified from the proposals. They either have a restricted scalability or require intensive communication.

The client-server technologies that are limited to nearby nodes do not scale to large WSNs. GSD and the tuple space both scale to large networks but they require more communication for locating a service. However, in both technologies the communication load can be decreased by increasing the number of hops, to which the service information is distributed. This increases the communication during the initiation but reduces it during the discovery, with the cost of increased memory consumption.

Task allocation

The technologies that implement a mechanism for the task allocation and the characteristics of each technology are listed in Table 6. As peer-to-peer communication is not needed in all the technologies, the communication mechanism is replaced by a more general outlining of the taken approach. As shown in Table 6, the variance of technologies is greater than in the service discovery. As most of the technologies are middleware layer implementations, the main reason for the variance is the three different approaches taken at that layer.

The most promising approach is the task allocation based on application QoS. It does not restrict the implementation of tasks nor rely on the surrounding context. Instead, it enables the adaptation of application operations depending on the current application requirements. The application requirements can be adjusted depending on the output of the application itself, which makes the technologies adaptive to changing conditions. Generally, application-QoS-based technologies require a central control for the task allocation, but a distributed control lacks similar adaptability.

Remote task communication

From the remote task communication technologies classified in Table 7, most utilize traditional RPC or RMI that are tailored for resource constrained environments. The tuple space and callbacks, which also utilize tuple space, are the only exceptions.

In general, the technologies either are restricted in their scalability or burden memory and communication resources. The problem in RPC and RMI technologies is the requirement for a client to know the server. In the tuple space and callbacks this is not required. In the callbacks, the message

Table 6: Characteristics of technologies implementing task allocation.

Technology	Approach	Scalability	Fault tolerance	Requirements	Benefits (pros)	Problems (cons)
SMP scheduler	Scheduling of tasks to free resources	Not restricted	Redundant	High-speed bus, shared memory	Efficiency and transparency	Inapplicable requirements
Script population specification	Specification in migrating scripts	Not restricted	Multiple copies in network	Control in application scripts	No control required	Expressivity of specification
Automatic object placement	Activating and moving objects near to source	Not restricted	Multiple agents available	Object placement algorithms	Reduced data communication	Complexity
Configuration adaptation	Mapping tasks to available resources	Not restricted	Changes active nodes adaptively	Feasibility analysis, state updates	Application QoS consideration	Control communication
Resource management	Heuristic algorithm balancing load [75]	Restricted to a cluster	Continuous allocation	Control messages	Network lifetime maximizing	Algorithm complexity
QoSProxy	Component and service adaptation for resources and application QoS	Network-wide in small networks	Adaptation according to conditions	Application QoS specification	QoS adaptation dynamically to available resources	Server required, complexity and communication
Attribute matching in SEE	Matching script attributes to node parameters locally	Not restricted	Multiple copies in network	Accurate attribute specifications	Local late binding	Restricted expressivity
Query optimizer	Optimizing query routing to network	Optimization in gateway node	Redundancy in queries	Disseminated query plans	Only required set of nodes activated	Networking load of query plans
Event-based queries	Initiate query on occurrence of event	Not restricted	Possibly several event detectors	Event identification capability	In-network reaction	Loading of event source node
Context reaction	Reactions on tuples and executed on matching context	Reaction restricted to a location	Redundancy in tuple space	Location identifying	Task executed only when its context is applicable	Scalability
MARE control	Nearby agents form an execution environment	Restricted to nearby agents	Possible redundancy	Agent managers controlling agents	Agent cooperation in complex tasks	Scalability
Adaptive object containers	ADC activates tasks in correct context	Not restricted	Possible redundancy	Context interface specifications	Only applicable tasks activated	Complex context specifications

is sent to a registered callback function whenever the value of a tuple changes. The tuple space does not support such interests on tuples. Like in the service discovery, the communication and memory load of the tuple space are adjustable.

Task migration

The technologies for the task migration are summarized in Table 8. Most of the technologies rely on the mobile agents due to their fault tolerance and smaller physical size. Three technologies rely on binary code in order to lessen the computation load caused by the agent interpreting.

In order to use binary code in the task migration, the possible errors during transfers and malicious attacks must be managed. The edit script generation algorithm is too complex to be executed in nodes, thus making it inapplicable for dynamic WSNs. From the VM approaches, the TCL and SQTL scripts and Maté bytecode capsules are more lightweight than Java objects because of the complexity and memory requirements of JVM.

4.3. Suitability assessment

Generally, the OS and VM proposals support the remote task communication and the task migration but leave the task allocation and the service discovery to an application or other external components. On the contrary, middleware approaches concentrate on implementing the task allocation, leaving other aspects for the tuple spaces or some legacy protocols. MARE and LIME are the only proposals that cover all distribution aspects. However, the utilization of JVM and the distributed tuple space requires resources that are not generally available in current WSN platforms.

We assess the applicability of the proposals for *EnvMonitor*. For a fair comparison, we separately compare the approaches for node platforms with enough resources, and then for platforms with limited resources defined in Figure 1. The main aspect considered in the assessment is the completeness of the operating environment provided for the application. In a complete environment, the application does not need to consider its distributed nature but the distribution is handled by the systems software. Further, the adap-

Table 7: Characteristics of technologies implementing remote task communication.

Technology	Communication implementation	Scalability	Fault tolerance	Requirements	Benefits (pros)	Problems (cons)
Active messages	Remote handler, data encapsulation	Not restricted	N/A	Awareness of remote handler	Mapping to TinyOS event model	Handler name in ASCII
BBS	Message posting to neighbor BBS	Restricted to neighbor nodes	Message posted to all neighbors	Neighbor posting enabled by sender	One-hop communication	Scalability, memory load
EYES OS RPC	N/A	Restricted to neighbors	N/A	N/A	One-hop communication	Scalability
Callbacks	Callback registered to a tuple	Restricted to nodes sharing tuple space	Callback registered only in one node	Shared tuple space between nodes	Callback fired only on an event	Fault tolerance
Message passing	Custom networking (QNet) operations	Not restricted	Possibility for redundant messages	Name resolution	Mapping to local IPC	Network naming overhead
Phantom process	Messages sent by link handler	Not restricted	Possible secure channels	Created channel for communication	Mapping to local IPC	Required handshaking, communication load
DVM	Invocation redirection	Not restricted	N/A	Compile time script modification	Seamless IPC between objects	Communication and processing load
Tuple space	Tuple operations	Not restricted	Redundant	Shared tuple space between nodes	Distributed in space and time	Communication/ memory load
R-ORB	Message-oriented communication	Requires nearby recipient	Activated when link available	Context sensing	Activated only in applicable context	Scalability

Table 8: Characteristics of technologies implementing task migration.

Technology	Communication	Scalability	Fault tolerance	Requirements	Benefits (pros)	Problems (cons)
Binary code	Binary code after negotiation	Only to one neighbor at a time	Simple checksum	Initiated by the binary code itself	Runtime initiation	Scalability, bit errors, binary size
Binary code download	Binary code from workstation	No in-network initiation	No protection	User initiates downloads	Possibility to update OS components	Errors, binary size, user interaction
Smoblets	Java applet modules	Execution only in laptops/PDAs	Java interpreter protection	Efficient platforms	Complex processing outsourcing	Executed only in efficient nodes
TCL script migration	TCL scripts	The scale specified in scripts	TCL interpreter protection	Injected to network by a user	Dynamic migration, small size of scripts	Complex population specifications
Mobile Java objects	Objects on top of JVM	Not restricted	Interpreter protection	Event initiating mobilization	Scalability	Communication and processing load
Code capsule updates	Small capsules in one active message	Script populated to all nodes in network	Maté interpreter protection	Injected to network by a user	Small size of scripts	No controlled migration
SQTL scripts	Custom query scripts	The scale specified in scripts	SEE interpreter protection	Injected to network by a user	Small size of scripts	Communication cost in broadcast
Edit scripts	Scripts containing changes to old code	No in-network initiation	Erroneous/missing scripts requested from neighbors	Generation of edit scripts in workstation	Small size of scripts	Complexity, no in-network operation

tivity of the proposals to changing conditions and the task allocation for extending network lifetime are emphasized.

For resource rich environments, MARE is the most suitable environment. The sensing and aggregation tasks in *EnvMonitor* can be allocated by the MARE control, and the active monitoring tasks can be implemented as mobile agents that are activated on demand.

For typical WSN platforms, MARE is not applicable due to its resource requirements. On the other hand, BTnodes fit to the restricted resources. The callbacks can be used to implement active monitoring tasks in *EnvMonitor*. The only aspect that is not supported by BTnodes is the task allocation so that the load is balanced between nodes.

4.4. Recommendations

From the systems software proposals for WSNs, OS and VM technologies implement the single node control and separate solutions for application distribution. The middleware proposals are applicable to the network-level distribution control. However, we argue that in WSNs, OS and middleware layers must be integrated to provide sufficient services within the constraints set by applications and platform resources.

In this kind of an approach, OS and middleware are inside the same framework so that information about OS internals and network topology is applicable to the middleware layer. Thus, this approach minimizes extra computation required for interfacing OS routines and communication due to the control signaling. Further, the middleware layer is aware of the influences of its actions at both the single node and network level. This awareness can be beneficial in the network-level power management and in the balancing of node loading.

For a sufficient environment for *EnvMonitor*, OS must implement a preemptive scheduling of tasks, a memory and power management, and a local IPC. The memory control should support static and dynamic memory and maintain information about available memory. We recommend the usage of a message-passing IPC because it is easily extended to the remote task communication. This kind of a general-purpose OS can be implemented on limited resources as shown in [25].

In addition to the local services, OS informs the middleware about the node energy and storage consumption, network role, associations, and nearby nodes and routes. An internal interface for the middleware to control tasks, power states, and network is implemented in OS. When all distribution aspects are implemented on the middleware layer, the components are able to utilize the information from each other more efficiently.

For service discovery we recommend the tuple space, since the pure client-server architecture is too static for WSNs. The resource and communication load of the tuple space can be diminished by selectively distributing tuple storing to nodes that use the tuple data and by dividing tuples to two-level hierarchies similar to GSD. The nodes that need a tuple for their operation can be identified with the support of task allocation. By sending only service group tuples to the distant nodes, less memory is needed but requests for tuples can still be routed accurately.

For the task allocation, the current application-QoS-based middleware proposals implement sufficient technologies. However, simpler algorithms that require less control communication should be used, even with the cost of accuracy.

For the remote task communication we recommend a simple approach that marshals the local message passing IPC to network packets. The remote nodes are identified by the service discovery. To make the delivery of a packet reliable, acknowledgements must be used. This is more lightweight than the tuple space, and the fault tolerance does not depend on the available recipients.

From our perspective, the task migration is required only in very dynamic applications, like object tracking. These applications require a VM-based environment. In OSs, the communication cost of the large binary transfers is extensive. Thus, the task migration should only be used when extremely necessary. The transfers must be protected with checksums and digital signatures, even though these are resource consuming.

We recommend also the usage of *virtual clusters*. A virtual cluster may follow the physical topology or it can be a set of adjacent nodes that have elected a single control entity. By storing detailed tuple information and performing task allocation within the boundaries of a virtual cluster, the communication and memory load can be diminished.

5. CONCLUSIONS

Our survey of WSN applications and their distribution shows that, despite many proposals, no common benchmarks nor detailed, large-scaled experiments have been published. The research seems to focus either on node implementations or theoretical work on distinct aspects, such as routing algorithms, without a realistic relation to physical platforms.

The systems software proposals are still evolving. Currently, they implement technologies and algorithms for application distribution but lack an approach combining a distributing middleware layer to OS providing a single node control. This kind of an approach is needed in order to implement a distributed operating environment, which supports application QoS and extends network lifetime, for resource scarce sensor nodes.

REFERENCES

[1] W. Stallings, *Data & Computer Communications*, Prentice-Hall, Englewood Cliffs, NJ, USA, 6th edition, 2001.

[2] J. A. Stankovic, T. E. Abdelzaher, C. Lu, L. Sha, and J. C. Hou, "Real-time communication and coordination in embedded sensor networks," *Proc. IEEE*, vol. 91, no. 7, pp. 1002–1022, 2003.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Commun. Mag.*, vol. 40, no. 8, pp. 102–114, 2002.

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[5] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," *Elsevier Ad Hoc Network Journal*, vol. 3, no. 3, pp. 325–349, 2005.

[6] H. Karl and A. Willig, "A short survey of wireless sensor networks," Tech. Rep. TKN-03-018, Technical University Berlin, Berlin, Germany, 2003, available: http://www.tkn.tu-berlin.de/publications.

[7] D. Chen and P. K. Varshney, "QoS support in wireless sensor networks: a survey," in *Proc. International Conference on Wireless Networks (ICWN '04)*, pp. 227–233, Las Vegas, Nev, USA, June 2004.

[8] S. Tilak, N. B. Abu-Ghazaleh, and W. B. Heinzelman, "A taxonomy of wireless micro-sensor network models," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, no. 2, pp. 28–36, 2002.

[9] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava, "Energy-aware wireless microsensor networks," *IEEE Signal Processing Mag.*, vol. 19, no. 2, pp. 40–50, 2002.
[10] A. J. Goldsmith and S. B. Wicker, "Design challenges for energy-constrained ad hoc wireless networks," *IEEE Wireless Communications*, vol. 9, no. 4, pp. 8–27, 2002.
[11] D. Remondo and I. G. Niemegeers, "Ad hoc networking in future wireless communications," *Computer Communications*, vol. 26, no. 1, pp. 36–40, 2003.
[12] N. Xu, "A survey of sensor network applications," available: http://enl.usc.edu/~ningxu/papers/survey.pdf.
[13] C.-Y. Chong and S. P. Kumar, "Sensor networks: evolution, opportunities, and challenges," *Proc. IEEE*, vol. 91, no. 8, pp. 1247–1256, 2003.
[14] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proc. International Workshop on Wireless Sensor Networks and Applications (WSNA '02)*, pp. 88–97, Atlanta, Ga, USA, September 2002.
[15] PODS, A Remote Ecological Micro-sensor Network Project website, http://www.pods.hawaii.edu.
[16] CORIE website, http://www.ccalmr.ogi.edu/CORIE.
[17] A. Boulis, C.-C. Han, and M. B. Srivastava, "Design and implementation of a framework for efficient and programmable sensor networks," in *Proc. 1st International Conference on Mobile Systems, Applications, and Services (MobiSys '03)*, San Francisco, Calif, USA, May 2003.
[18] P. Bonnet, J. Gehrke, and P. Seshadri, "Querying the physical world," *IEEE Pers. Commun.*, vol. 7, no. 5, pp. 10–15, 2000.
[19] L. Schwiebert, S. K. S. Gupta, and J. Weinmann, "Research challenges in wireless networks of biomedical sensors," in *Proc. 7th ACM International Conference on Mobile Computing and Networking (MobiCom '01)*, pp. 151–165, Rome, Italy, July 2001.
[20] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, and M. A. Perillo, "Middleware to support sensor network applications," *IEEE Network*, vol. 18, no. 1, pp. 6–14, 2004.
[21] M. Storey, G. S. Blair, and A. Friday, "MARE: resource discovery and configuration in Ad hoc networks," *J. Mobile Networks and Applications*, vol. 7, no. 5, pp. 377–387, 2002.
[22] H. O. Marcy, J. R. Agre, C. Chien, L. P. Clare, N. Romanov, and A. Twarowski, "Wireless sensor networks for area monitoring and integrated vehicle health management applications," in *Proc. AIAA Guidance, Navigation, and Control Conference and Exhibit*, Portland, Ore, USA, 1999, Collection of Technical Papers. Vol. 1 (A99-36576 09-63).
[23] K. Römer , "Tracking real-world phenomena with smart dust," in *Proc. 1st European Workshop on Wireless Sensor Networks (EWSN '04)*, pp. 28–43, Berlin, Germany, January 2004.
[24] C.-C. Shen, C. Srisathapornphat, and C. Jaikaeo, "Sensor information networking architecture and applications," *IEEE Pers. Commun.*, vol. 8, no. 4, pp. 52–59, 2001.
[25] H. Abrach, S. Bhatti, J. Carlson, et al., "MANTIS: system support for MultimodAl NeTworks of In-situ Sensors," in *Proc. 2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA '03)*, pp. 50–59, San Diego, Calif, USA, September 2003.
[26] M. B. Srivastava, R. R. Muntz, and M. Potkonjak, "Smart kindergarten: sensor-based wireless networks for smart developmental problem-solving enviroments," in *Proc. 7th ACM International Conference on Mobile Computing and Networking (MobiCom '01)*, pp. 132–138, Rome, Italy, July 2001.
[27] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta, "Reconfigurable context-sensitive middleware for pervasive computing," *IEEE Pervasive Computing*, vol. 1, no. 3, pp. 33–40, 2002.
[28] NIST Wireless Ad hoc Networks Project website, http://www.antd.nist.gov/wahn_ssn.shtml.
[29] MICA2 data sheet, available: http://www.xbow.com.
[30] IEEE P1451.5 Wireless Sensor Working Group website, http://grouper.ieee.org/groups/1451/5.
[31] Bluetooth Special Interest Group, "Bluetooth specification, version 1.1," February 2001.
[32] *Wireless Medium Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPAN)*, IEEE Standard 802.15.4, 2003.
[33] *Wireless LAN Medium Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Standard 802.11, 1999.
[34] IETF Mobile Ad-hoc Networks Working Group website, http://www.ietf.org/html.charters/manet-charter.html.
[35] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design*, Addison-Wesley, Boston, Mass, USA, 3rd edition, 2001.
[36] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding code mobility," *IEEE Trans. Software Eng.*, vol. 24, no. 5, pp. 342–361, 1998.
[37] P. J. M. Havinga, "System architecture specification," EYES Project Deliverable 1.1, available: http://www.eyes.eu.org/dissem.htm.
[38] J. Beutel, O. Kasten, F. Mattern, K. Roemer, F. Siegemund, and L. Thiele, "Prototyping wireless sensor networks with BTnodes," in *Proc. 1st European Workshop on Wireless Sensor Networks (EWSN '04)*, pp. 323–338, Berlin, Germany, January 2004.
[39] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," in *Proc. 9th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '00)*, pp. 94–103, Cambridge, Mass, USA, November 2000.
[40] J. Lifton, D. Seetharam, M. Broxton, and J. Paradiso, "Pushpin computing system overview: a platform for distributed, embedded, ubiquitous sensor networks," in *Proc. 1st International Conference on Pervasive Computing (Pervasive '02)*, pp. 139–151, Zurich, Switzerland, August 2002.
[41] QNX website, http://www.qnx.com.
[42] OSE website, http://www.ose.com.
[43] R. Barr, J. C. Bicket, D. S. Dantas, et al., "On the need for system-level support for ad hoc and sensor networks," *ACM SIGOPS Newsletter on Operating Systems Review*, vol. 36, no. 2, pp. 1–5, 2002.
[44] E. G. Sirer, R. Grimm, A. J. Gregory, and B. N. Bershad, "Design and implementation of a distributed virtual machine for networked computers," in *Proc. 17th ACM Symposium on Operating Systems Principles (SOSP '99)*, pp. 202–216, Kiawah Island, SC, USA, December 1999.
[45] P. Levis and D. Culler, "Maté: a tiny virtual machine for sensor networks," in *Proc. 10th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '02)*, pp. 85–95, San Jose, Calif, USA, October 2002.
[46] D. Gelernter, "Generative communication in linda," *ACM Trans. Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112, 1985.
[47] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service location protocol, version 2," IETF, RFC 2608, June 1999.
[48] *Jini Architecture Specification*, version 2.0, Sun Microsystems, June 2003, available: http://www.sun.com/software/jini/specs.
[49] *UPnP Device Architecture*, Microsoft Corporation, June 2000, available: http://www.upnp.org/resources/documents.asp.
[50] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, "An architecture for a secure service discovery

service," in *Proc. 5th International Conference on Mobile Computing and Networking (MobiCom '99)*, pp. 24–35, Seattle, Wash, USA, August 1999.

[51] *JavaSpaces Service Specification*, version 2.0, Sun Microsystems, June 2003, available: http://www.sun.com/software/jini/specs.

[52] T. J. Lehman, A. Cozzi, Y. Xiong, et al., "Hitting the distributed computing sweet spot with TSpaces," *Computer Networks*, vol. 35, no. 4, pp. 457–472, 2001.

[53] R. Srinivasan, "RPC: remote procedure call protocol specification version 2," IETF, RFC 1831, August 1995.

[54] The Open Group Portal to World of DCE, website, http://www.opengroup.org/dce.

[55] Object Management Group website, http://www.omg.org.

[56] *Java RMI specification*, Sun Microsystems, 1997–2003, available: http://java.sun.com/products/jdk/rmi/reference/docs.

[57] M. Horstmann and M. Kirtland, "DCOM architecture," Microsoft Corporation, July 1997, available: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/dcom.asp.

[58] A. Schill and S. Kümmel, "Design and implementation of a support platform for distributed mobile computing," *Distributed Systems Engineering*, vol. 2, no. 3, pp. 128–141, 1995.

[59] S. Adwankar, "Mobile CORBA," in *Proc. 3rd International Symposium on Distributed Objects and Applications (DOA '01)*, pp. 52–63, Rome, Italy, September 2001.

[60] A. D. Joseph, J. A. Tauber, and M. F. Kaashoek, "Mobile computing with the rover toolkit," *IEEE Trans. Comput.*, vol. 46, no. 3, pp. 337–352, 1997.

[61] Supercluster.org website, http://www.supercluster.org.

[62] B. Schilit, N. Adams, and R. Want, "Context-aware computing applications," in *Proc. Workshop on Mobile Computing Systems and Applications (WMCSA '94)*, pp. 85–90, Santa Cruz, Calif, USA, December 1994.

[63] Y. Yu, B. Krishnamachari, and V. K. Prasanna, "Issues in designing middleware for wireless sensor networks," *IEEE Network*, vol. 18, no. 1, pp. 15–21, 2004.

[64] K. Nahrstedt, X. Dongyan, D. Wichadakul, and L. Baochun, "QoS-aware middleware for ubiquitous and heterogeneous environments," *IEEE Commun. Mag.*, vol. 39, no. 11, pp. 140–148, 2001.

[65] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "The design of an acquisitional query processor for sensor networks," in *Proc. 22nd ACM SIGMOD International Conference on Management of Data (SIGMOD '03)*, pp. 491–502, San Diego, Calif, USA, June 2003.

[66] Y. Yao and J. Gehrke, "The Cougar approach to in-network query processing in sensor networks," *ACM SIGMOD Record*, vol. 31, no. 3, pp. 9–18, 2002.

[67] A. L. Murphy, G. P. Picco, and G.-C. Roman, "LIME: a middleware for physical and logical mobility," in *Proc. 21st International Conference on Distributed Computing Systems (ICDCS '01)*, pp. 524–533, Phoenix, Ariz, USA, April 2001.

[68] S. S. Yau and F. Karim, "An energy-efficient object discovery protocol for context-sensitive middleware for ubiquitous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 11, pp. 1074–1085, 2003.

[69] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin, "GSD: a novel group-based service discovery protocol for MANETS," in *Proc. 4th International Workshop on Mobile and Wireless Communications Network (MWCN '02)*, pp. 140–144, Stockholm, Sweden, September 2002.

[70] N. Reijers and K. Langendoen, "Efficient code distribution in wireless sensor networks," in *Proc. 2nd ACM International Workshop on Wireless Sensor Networks and Applications*, pp. 60–67, San Diego, Calif, USA, September 2003.

[71] C. Jaikaeo, C. Srisathapornphat, and C.-C. Shen, "Querying and tasking in sensor networks," in *Proc. 14th International Symposium on Aerospace/Defense Sensing, Simulation, and Control*, vol. 4037 of *Proc. SPIE's*, pp. 184–197, Orlando, Fla, USA, April 2000.

[72] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: accurate and scalable simulation of entire TinyOS applications," in *Proc. 1st ACM Conference on Embedded Networked Sensor Systems (SenSys '03)*, pp. 126–137, Los Angeles, Calif, USA, November 2003.

[73] S. Park, A. Savvides, and M. B. Srivastava, "Simulating networks of wireless sensors," in *Proc. Winter Simulation Conference (WSC '01)*, pp. 1330–1338, Arlington, Va, USA, December 2001.

[74] X. Zeng, R. Bagrodia, and M. Gerla, "GloMoSim: a library for parallel simulation of large-scale wireless networks," in *Proc. 12th Workshop on Parallel and Distributed Simulations (PADS '98)*, pp. 154–161, Banff, Alberta, Canada, May 1998.

[75] Y. Yu and V. K. Prasanna, "Energy-balanced task allocation for collaborative processing in networked embedded systems," in *Proc. ACM SIGPLAN Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '03)*, pp. 265–274, San Diego, Calif, USA, June 2003.

Mauri Kuorilehto received the M.S. degree in 2001 from Tampere University of Technology (TUT), Finland. He is currently pursuing his Ph.D. degree and acting as a Research Scientist in the DACI Research Group, the Institute of Digital and Computer Systems at TUT. His research interests include wireless sensor and ad hoc networks concentrating on distributed processing, operating systems, and network simulation.

Marko Hännikäinen received the M.S. degree in 1998 and the Ph.D. degree in 2002 both from Tampere University of Technology (TUT). Currently he acts as a Senior Research Scientist in the Institute of Digital and Computer Systems at TUT, and a Project Manager in the DACI Research Group. His research interests include wireless local and personal area networking, wireless sensor and ad hoc networks, and novel web services.

Timo D. Hämäläinen received the M.S. degree in 1993 and the Ph.D. degree in 1997 both from Tampere University of Technology (TUT). He acted as a Senior Research Scientist and Project Manager at TUT during 1997–2001. He was nominated to be Full Professor at TUT, Institute of Digital and Computer Systems in 2001. He heads the DACI Research Group that focuses on three main lines: wireless local area networking and wireless sensor networks, high-performance DSP/HW-based video encoding, and interconnection networks with design flow tools for heterogeneous SoC platforms.