Available online at www.sciencedirect.com

ScienceDirect

Ad Hoc Networks 6 (2008) 909–935

www.elsevier.com/locate/adhoc

Rapid design and evaluation framework for wireless sensor networks

Mauri Kuorilehto [a,*], Marko Hännikäinen [b], Timo D. Hämäläinen [b]

[a] *Nokia Technology Platforms, Visiokatu 3, FIN 33720, Tampere, Finland*
[b] *Tampere University of Technology, Institute of Digital and Computer Systems, Korkeakoulunkatu 1, FIN 33720, Tampere, Finland*

Received 29 October 2004; received in revised form 12 July 2006; accepted 21 August 2007
Available online 11 September 2007

Abstract

The diversity of applications and typically scarce node resources set very tight constraints to Wireless Sensor Networks (WSN). It is not possible to fulfill all requirements with a general purpose WSN, for which reason the rapid development of application specific WSNs is preferred. We present a new framework called WIreless SEnsor NEtwork Simulator (WISENES) for the design, simulation, and evaluation of WSNs. The target WSN is designed in Specification and Description Language (SDL), simulated in WISENES, and implemented on target platform either through automatic code generation or manually. The high-level WSN model is back-annotated with the measured values from a real platform. In this way, very accurate WSN simulations can be performed with a rapid design cycle. WISENES itself has been verified with TUTWSN (Tampere University of Technology Wireless Sensor Network) and ZigBee protocols. The MAC protocol of ZigBee was designed in two weeks from scratch by one designer, which shows the effectiveness of WISENES. For accuracy comparison, the results show 6.7% difference between the modeled and measured TUTWSN prototype power consumption. WISENES hastens the evaluation of new protocol and application configurations, especially for the large scale and long-term WSN deployments.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Wireless Sensor Network (WSN); Network simulation; Medium access control (MAC); Energy efficiency

1. Introduction

Wireless Sensor Network (WSN) applications are diverse ranging from toys to military systems. Typical challenges for WSN are large scale, constantly changing network topology, and error prone communications, while in WSN nodes processing and storage capacities, as well as energy resources are limited. Most often WSNs are demanded to be robust against environmental strains, and able to autonomously recover from error situations. Further, depending on the applications and the interaction with environment, time synchronization and security requirements can be strict [1,2].

Opposite to general expectations, an all-purpose WSN is not a reasonable goal, since it is impossible

[*] Corresponding author. Tel.: +358 71 800 8000; fax: +358 3 3115 4561.
E-mail addresses: mauri.kuorilehto@nokia.com (M. Kuorilehto), marko.hannikainen@tut.fi (M. Hännikäinen), timo.d.hamalainen@tut.fi (T.D. Hämäläinen).

1570-8705/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.adhoc.2007.08.003

to meet all the real life constraints simultaneously. Instead, WSN protocol layers and their configuration parameters must be tailored to meet the specific application requirements. However, the design space is very large and makes the design automation the most important challenge for real working WSNs. A designer simply cannot handle all the parameters, functions, and their complicated dependencies without a tool support.

Prototyping can be applied to a single node functionality and small scale WSN testing. However, prototypes are not applicable for verifying the operation of e.g. a thousand-node network during a five year deployment. Even moderate sized networks benefit from extensive simulations, but the accuracy of simulation is very important. According to our experiences on real WSNs, the smallest and a minor-looking issue might cause severe changes for example in the network power consumption. Therefore, the accuracy of the design and performance estimations is not an option but essential for any real WSN.

Several legacy computer network simulators exist for the testing and modeling of communication protocols, but they omit WSN specific aspects. Proposed WSN simulators vary in their implementation, scale, and in the accuracy and coverage of the modeling of node platforms, protocols, and real world phenomena. Common features are the models for dedicated platforms, sensing, and wireless networking. However, none of the previous WSN simulators offers a complete and seamless design flow from abstract sketching to the real implementation.

Our WIreless SEnsor NEtwork Simulator (WISENES) framework is the first tool that enables the design, simulation, implementation and evaluation of WSNs with measured back-annotated information. WISENES is targeted to the design of deployable, real WSN networks. The main difference to the other proposed frameworks is that there is no need to carry out a separate high abstraction WSN modeling project and another development project for the actual implementation. Instead, WISENES supports all phases in the design flow. However, if preferred, WISENES can also be used for the plain simulations like other WSN simulators. In all cases, WISENES eases the assessment of the protocol and application interoperability, and the evaluation their applicability for different sensor node platforms.

The key benefit of WISENES is that the evaluation of protocols, applications, and their different configurations is carried out starting from the design phase. The framework defines rules and interfaces for a designer to the protocol stack and application implementation. The functionality, type, or composition of the protocols is not limited by the framework. Sensor nodes, transmission medium, and inspected phenomena are modeled separately in the WISENES framework. The WSN protocols and applications are implemented in Specification and Description Language (SDL) [3]. The models of high abstraction level SDL are compiled to executables used for both simulation and final implementation. Unlike in the other WSN simulators, target node platforms are not restricted to a specific pre-defined platform.

WISENES has been tested and its own performance evaluated with large TUTWSN (Tampere University of Technology Wireless Sensor Network) [4] and ZigBee networks [5]. For the evaluation of WISENES accuracy, real and simulated TUTWSNs are compared. However, the comparison of different WSNs themselves is not the primary scope of this paper.

This paper is organized as follows. Section 2 discusses the related work in the area of WSN simulation and presents the comparison of WISENES with the other WSN simulators. WISENES design is presented in Section 3 and Section 4 introduces the WISENES framework. The use of WISENES for TUTWSN and ZigBee protocol implementation is presented in Section 5. Section 6 gives the evaluation of WISENES, and the TUTWSN and ZigBee simulation results. Finally, conclusions are given and future work projected.

2. Related work

Legacy computer network simulators, such as ns-2 [6], GloMoSim [7], Qualnet [8], OPNET [9], OMNeT++ [10], Scalable Simulation Framework (SSF) [11], and J-Sim [12] enable the simulation of wireless network behavior and protocol stack operation but lack accounting for WSN characteristics. This is overcome in the simulators proposed specifically for WSNs, which we have categorized to *networking oriented* and *sensor node* simulators. The networking oriented simulators model the transmission medium in detail and are more suitable for the large scale WSN simulations. The sensor node simulators mainly simulate the operation of a single node but implement a lightweight communication model. Currently, there exist eleven relevant proposals for the networking oriented WSN simulators and six proposals targeting to the sensor node modeling. These are compared with WISENES in Section 2.3.

2.1. Networking oriented simulators for WSN

Most of the proposed networking oriented simulators are based on the legacy computer network simulators. SensorSim [13] and Naval Research Laboratory's (NRL) sensor network simulator [14] extend ns-2 with general WSN features. sQualnet [15] is built on top of Qualnet and Simulator for Wireless Ad-hoc Networks (SWAN) [16] is based on SSF. SENSIM [17] and simulation template for EYES [18] utilize OmNet++ environment, while J-Sim sensor simulator [19] adds WSN features to its parent simulator. VisualSense [20] is an extension to Ptolemy II [21], Prowler [22] utilizes MATLAB, and H-MAS [23] and SENSE [24] implement custom simulation environments.

The most realistic transmission media and protocol stacks are available in SensorSim, NRL simulator, sQualnet, and J-Sim sensor simulator. While the first two rely on the models available in parent simulators, the last two include also a set of WSN protocols. VisualSense has several models for transmission medium, which vary in their accuracy. SWAN and Prowler include abstracted transmission media and lowest layer protocol models that estimate the network operation. SENSIM, EYES simulator, H-MAS, and SENSE have error free transmission medium models, in which the signal propagation is dependent only on the transmission range. Simple protocol stacks are available for SENSIM, H-MAS, and SENSE. In VisualSense and EYES simulator, the protocol stack is implemented by the designer.

A separate sensing channel containing also the sensed targets is used for phenomena modeling in SensorSim, NRL simulator, sQualnet, and J-Sim sensor simulator. VisualSense has also a dedicated channel for modeling the propagation of different phenomena. Of the other related simulators, SWAN models catastrophic plume dispersion, and H-MAS generates random sensor readings. Prowler, SENSIM, EYES simulator, and SENSE do not support phenomena sensing.

Concerning the node platform capabilities, only the power consumption is accounted in the related simulators. SensorSim, sQualnet, SENSE, SENSIM, and J-Sim sensor simulator have detailed power models, which consider battery discharge rate and relaxation. NRL simulator and EYES simulator use linear battery models, in which the maximum energy capacity is available independent on the discharge rate. Prowler and VisualSense estimate the power consumption based on activity, whereas in SWAN and H-MAS power consumption is not taken into account.

The simulated code is applicable directly for hardware platforms in SensorSim, sQualnet, J-sim sensor simulator, and partly in VisualSense. In SensorSim, simulated SensorWare applications are compatible with custom hardware platforms [25]. The other three simulators enable the execution of applications, and sQualnet also higher layer protocols, on Berkeley motes [26] on top of well-known WSN Operating System (OS) TinyOS [27].

VisualSense uses a graphical notation for the design and supports a combination of different Models of Computation (MoC) of Ptolemy II. Abstracted application scripts can be simulated also in Prowler.

2.2. Sensor node simulators

Most of the proposed sensor node simulators are targeted to TinyOS motes. Complete TinyOS system can be simulated with TinyOS SIMulator (TOSSIM) [28], ATmel EMUlator (ATEMU) [29], and TinyOS Scalable Simulation Framework (TOSSF). TOSSF itself is an extension to SWAN [30]. SENS [31] supports only TinyOS application simulation. EmSim implements a simulation environment for custom Em* Linux applications and protocols [32]. Sensor Network Asynchronous Processor (SNAP) [33] is a hardware emulator, which connects several processors on a Network-on-Chip (NoC).

The TOSSIM transmission medium model is directed graphs with individual bit error rates, whereas in ATEMU and EmSim the transmission medium is error free accounting only the transmission range. TOSSF utilizes a transmission medium model from SWAN and in SENS transmission medium and networking protocols are combined into a simple model. In SNAP, NoC models the transmission medium. Protocol stacks in TOSSIM, ATEMU, and TOSSF are dependent on TinyOS configuration. EmSim and SNAP implement simple protocols for the system testing.

Phenomena sensing is modeled in TOSSF by the plume dispersion model of SWAN. TOSSIM and EmSim retrieve a sensor reading from an external source. SENS incorporates a separate environment model that supports sensing and actuating. In SNAP and ATEMU phenomena are not modeled. None of the simulators models power consumption.

The applications from all sensor node simulators, and protocols from other than SENS, can be directly mapped to the hardware platforms. However, the platform is restricted to a specific one.

2.3. Comparison of WISENES with related simulators

The comparison of WISENES and the other WSN simulators is summarized in Table 1. The comparison is based on the public information available about each simulator and the possible parent simulator engine. If exact scalability information is not available, the simulator is assessed according to the largest reported simulations.

The scope of input parameterization is vital when comparing the configurability of simulators to different kinds of platforms, protocols, and applications. Also, the availability of Graphical User Interfaces (GUI) and the type of information output by the simulator are accounted in the comparison. The possibility to use simulated protocols and applications for the final implementations on physical platforms defines the applicability of a simulator as a complete design and development environment.

The term *accurate results* denotes a very close correspondence of the simulation results to the real world measurements with physical WSN prototypes. Accurate results in full-scale simulations need to combine at least realistic models for communications (application, transmission medium, transceiver unit, and low-level communication protocols) and node platform (energy, memory, peripheral I/O, and computation). The node state changes, peripheral activation, and leakage currents contribute to the accuracy of energy consumption. The memory allocation and thread scheduling in simulator depend on the accuracy of the OS model of the simulator.

As shown in the table, most of the simulators are capable of simulating WSN scenarios consisting of thousands of nodes. This is an acceptable limit for the current WSNs, but in future the capability to simulate networks with in order of magnitude larger scale is required.

Major differences between the simulators are in input and output parameterization. Although sensor node simulators emulate a single node platform in detail, they do not allow the testing and evaluation of applications and protocols on other types of sensor nodes. Further, they omit the modeling of node power consumption. From the simulators, WISENES incorporates the most comprehensive modeling of node platforms, which allows a detailed description for virtually any real node platform. In addition to power consumption, which is considered also in several other simulators, WISENES accounts memory and computation capacities. From the related simulators, the possibility to define different platforms is available only in sQualnet, but its modeling is not as detailed as that of WISENES.

Most of the simulators visualize network topology and key parameters through GUI. The event information is gathered to trace or log files with varying level of detail. From the related simulators, ns-2 based simulators output extensive trace files. Compared to the other simulators, WISENES outputs extensive event and data statistics but in addition also detailed information about the node resource usage. Further, in WISENES a designer can add own log items e.g. to obtain the values of a desired parameters during the simulated period.

The simulated code is directly applicable for node platforms in sensor node simulators. In networking oriented simulators, two approaches are taken for the final implementation. Either the simulated protocols and applications are converted to executables for node platforms, or an existing code library is used for emulating a node in a large scale simulation. In the latter, the node implementation already exists, and only the configuration parameters for the final implementation are acquired by the simulations. WISENES is the only simulator that supports both of these. The SDL generated C from WISENES with a custom lightweight kernel is also applicable for the resource constrained sensor nodes [34].

A rapid protocol evaluation for a specific application is possible only if the simulator protocol stack is modular and its layers interchangeable. Most of the simulators that descend from a legacy computer network simulator incorporate a modular protocol stack. In WISENES, the protocol layers communicate through pre-defined interfaces, which allow the replacement of simulated protocols at the different layers. Graphical design of protocols and applications is possible only in WISENES and VisualSense. In both simulators the high abstraction level designs can also be embedded to node platforms. However, WISENES provides significantly more accurate results compared to VisualSense.

Table 1
Comparison of WISENES and existing WSN simulators

Simulator	Simulator engine	Scalability	Simulator input	Simulator output	Final implementation	Benefits	Deficiencies
WISENES	Extended telelogic TAU SDL	~10,000 [34]	Nodes, protocols, applications, mediums (XML)	GUI, log files, (data, energy, memory, CPU, errors)	SDL code generation/C modules directly	Graphical design, accurate results, modular, scalability, back-annotation	Sensing channel
Networking oriented simulators							
SensorSim	ns-2	~2000 [17]	Power model, protocols (TCL)	ns-2 nam UI, trace files (data, energy, errors)	Applications for SensorWare, ns-2 protocols	Accurate results, variety of protocols (ns-2), modular	No memory and CPU modeling
sQualnet	Qualnet	~10,000	Nodes, traffic, protocols (scripts)	Qualnet Visualizer, statistics files (data, energy)	nesC applications directly	Accurate results, integration to HW, modular, scalability	No memory and CPU modeling in simulator
NRL simulator	ns-2	~2000 [17]	Nodes, protocols, sensing (TCL)	ns-2 nam UI, trace files (data, energy, errors)	ns-2 protocols	Variety of protocols (ns-2), modular	No memory and CPU modeling
SWAN	DaSSF	~10,000	Nodes, plume dispersion (DML)	GUI, system printouts (data counters, delay)	WiroKit routing protocol directly	Scalability	Inaccurate medium model, no node and sensing modeling, no modularity
SENSIM	OmNet++	~5000	Protocols (ini-file (for OmNet++))	OmNet++ GUI (data)	None	Modular	No sensing, memory, and CPU modeling
EYES simulator	OmNet++	<1000	Protocols (ini-file (for OmNet++))	OmNet++ GUI (data,errors)	None	Modular	Inaccurate energy and medium modeling, no memory and CPU models
J-Sim sensor simulator	J-Sim	>1000 [35]	Protocols (script)	Text output, GUI possible, (data)$^{(1)}$	Applications directly	Modular	No GUI, no memory and CPU modeling
VisualSense	Ptolemy II	~100	MoC configurations (Ptolemy II)	Ptolemy II GUI (topology, node information)	Algorithms integrated to TinyOS [36]	Graphical design, algorithm integration	No protocol stack, nodes modeled by power model
Prowler	MATLAB	<1000	Application (script)	GUI (data)	None	MATLAB for algorithm testing	Inaccurate protocol, node, and medium modeling
H-MAS	Custom	>100	Nodes, protocols (text)	GUI, event files (data)	None		Inaccurate protocol and medium modeling, no modularity, scalability
SENSE	Custom	~1000	Topology, traffic (script)	None$^{(2)}$	None	Modular	No output, inaccurate medium modeling, no node and sensing modeling

(*continued on next page*)

Table 1 (*continued*)

Simulator	Simulator engine	Scalability	Simulator input	Simulator output	Final implementation	Benefits	Deficiencies
Sensor node simulators							
TOSSIM	Custom	~10,000	TinyOS code	TinyViz, debug (data, node)	Directly	Applicability of code for mores	No energy modeling, same code for all nodes
ATEMU	Custom	~100	TinyOS code network, nodes (XML)	XATDB debugger (debug data)	Directly	Applicability of code for mores	No sensing and energy models, scalability
SENS	Custom	~10,000	Node profiles (text)	GUI, text files (data, energy, errors)	Applications directly	Scalability	Only for applications, node and medium models
TOSSF	DaSSF (SWAN)	~10,000	TinyOS code, SWAN parameters	SWAN output$^{(3)}$	Directly	Scalability	Inaccurate medium model, no node and sensing modeling
Em* EmSim	Custom	<100	Simulation case (script) [37]	Debug traces (node)	Directly	Applicability of code	Inaccurate medium model, no node and sensing models, scalability
SNAP	FPGA Emulator	~100$^{(4)}$	Configuration (for FPGA)	FPGA debug interface (node)	Directly	Emulation	Inaccurate medium model, no sensing modeling, scalability

(1) J-Sim outputs simulation related data to an "instrument channel", to which user can implement a custom UI.
(2) No information about output is given, only results given show simulation times and simulator memory consumption.
(3) TOSSF I/O is not specified, but we assumed SWAN I/O due to the relation.
(4) SNAP emulators can be connected to increase scalability, but no evaluation is given.

Generally, the networking oriented simulators are more suitable for the network-wide evaluation, whereas the strength of the sensor node simulators lies in the testing and optimization of a single node operation. From the related simulators, SensorSim and sQualnet implement the most comprehensive simulation environment. Compared to WISENES, their battery and sensing models are currently more accurate. The distinctive features of WISENES are the complete design flow from the high abstraction level graphical models to the final node implementation, the accurate full-scale simulations with configurable protocol stack and node platform models, and the back-annotation of performance information from real platforms.

3. Designing WSNs with WISENES

WISENES defines the rules and interfaces for the WSN design and provides a library that contains existing protocols and a set of implementations for known functions. A reference WSN protocol stack used in WISENES is depicted in Fig. 1 in correspondence to the OSI model [38].

The key layer for WSN topology creation, channel access, and power management is the Medium Access Control (MAC) protocol. The topology in WSNs is either clustered or flat. In a clustered topology, nodes are grouped to clusters, in which a central cluster headnode coordinates the networking and associated subnodes. In a flat topology all nodes are equal. A routing protocol creates multi-hop paths for the end-to-end communication. A middleware layer hides the heterogeneities of underlying protocol stacks and node platforms from applications [1].

The composition of the protocol stack is network and application dependent, thus all layers are not mandatory in WISENES. The transceiver unit at the physical layer is part of the WISENES framework and the lowest layer protocol sends data to the transmission medium through its interface.

Multiple WSN applications can be simulated simultaneously in WISENES. Applications are designed as a set of tasks communicating together. Tasks initiate sensing, perform data processing and aggregation, and initiate data transfers. The application layer, which implements a host environment for application tasks, is a part of the WISENES framework.

Applications are implemented either in detail using SDL or by a task graph. The task graph is a simple data dependency graph described by the simulator input parameters. In addition to the task data dependencies, it defines the task activation frequencies, and task sensing and data characteristics. This approach enables the testing of different types of applications with minimum effort.

3.1. WISENES input and output

The input parameter and output result groups of WISENES are summarized in Fig. 2. WISENES input parameters are defined using eXtensible Markup Language (XML), each parameter set having a dedicated file with a pre-defined structure.

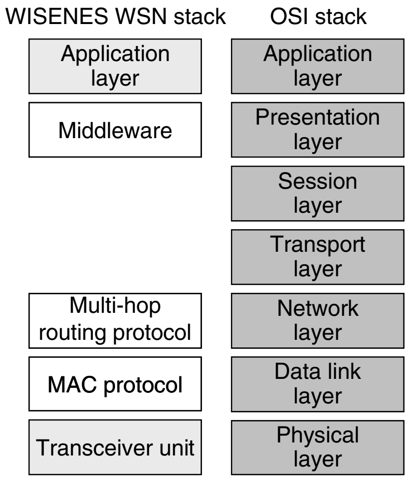


Fig. 1. WISENES WSN and OSI model protocol stacks.

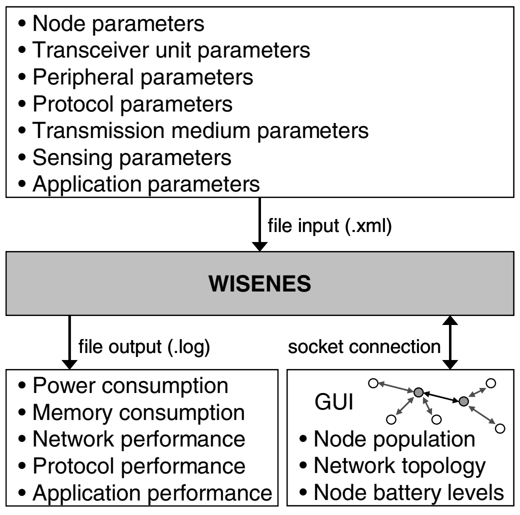


Fig. 2. WISENES input parameter groups and output results.

Table 2
WISENES input parameters and their types

Parameter	Type	Description
Node parameters		
Execution capacity	Integer (MIPS)	Instructions CPU can execute in a second
Instruction memory size	Integer (instruction)	Available instruction memory
Data memory size	Integer (byte)	Available data memory
Power unit capacity	Float (mAh)	Initial energy capacity in node battery
Active state power consumption	Float (μW)	Consumed power in active state
Sleep state information	[Float, float] array (ms, μW)	Required idle time before activation and power consumption in sleep state
Harvesting capacity	Float (μW)	Power node can harvest from environment
Transceiver unit(s)	Integer array (identifier)	Transceiver unit(s) on the node
Peripherals	Integer array (identifier)	Peripherals attached to the node
Node coordinates	Float array (m)	Three-dimensional coordinates for node (x, y, z)
Transceiver unit parameters		
Throughput	Integer (bps)	Transceiver unit data rate
Start-up transient period	Float (ms)	Start-up transient time before receiver/transmitter is ready
Data loading information	[Integer, float] pair (bps, μW)	Throughput and power consumption during transceiver to CPU communication
Receiver power	Float (μW)	Power consumption while receiver is active
Transmit power levels	[Integer, float] array (dBm, μW)	Available transmit powers and corresponding power consumption
Carrier sensing power	Float (μW)	Power consumption during carrier sensing operation (if available in transceiver)
Peripheral parameters		
Type	Constant string (identifier)	Defines the peripheral type, e.g. sensor, ADC, location-finding system, mobilizer
Phenomena	Integer array (identifier)	Phenomena the peripheral is related to, if sensor
Relations	Integer array (identifier)	Relations to other peripherals
Activation time	Float (ms)	Time the peripheral is active once activated
Activation power	Float (μW)	Power consumption when peripheral is active
Protocol parameters (for each protocol layer)		
Instruction memory consumption	Integer (instruction)	Instruction memory required by the protocol layer
Data memory consumption	Integer array (byte)	Protocol static and dynamic data memory consumption
Transmission medium parameters		
Signal attenuation curve	Float array (constant)	Define signal attenuation curve coefficients (k, b, v)
Minimum PER	Float (constant)	Minimum PER in the transmission medium
Sensing parameters		
Active phenomena	Integer array (identifier)	Phenomena that can inspected
Limits	Float array (dependent unit)	Lower and upper limits separately for each sensed phenomenon value
Application parameters		
Task activation interval	Float (ms)	The interval between task activations
Task data activation	Integer (constant)	After how many activations task initiates a data transfer
Task data amount	Integer (byte)	The amount of data sent by the task
Task data relations	Integer array (identifier)	Task to which the data is sent
Task peripheral relations	Integer array (identifier)	Peripherals required by the task
Instruction memory consumption	Integer (instruction)	Task instruction memory consumption
Data memory consumption	Integer array (byte)	Task static and dynamic data memory consumption
Task executed operations	Integer (constant)	Executed operations per task activation
Task population	Integer array (identifier)	A list of node identifiers defining the nodes, in which the task binary is located

Node parameters are given in two separate files. The first defines the capabilities of node platforms, and the other per-node platform type and node coordinates.

WISENES outputs information in two forms. Detailed information about the simulated WSN and nodes is collected to log files. Each protocol and data event during a simulation is logged with the parameters that define the cause and the consequence of the event. Log data are written to per-node directories, each protocol layer, application, and a control instance modeling OS routines having a dedicated *.log*-file. During an active simulation run, the progress of a simulation is illustrated through GUI presenting the node population, network topology, and node energy level.

Input parameters at each group and their types are presented in detail in Table 2. Node, transceiver unit, and peripheral parameters define the capabilities of sensor node platforms. Protocol related input parameters define the static memory consumption for each protocol, while the rest of the characteristics are specified in the protocol SDL implementation. Transmission medium parameters define its modeling and sensing parameters active phenomena. An application task graph and an initial application task population are described in the application parameters.

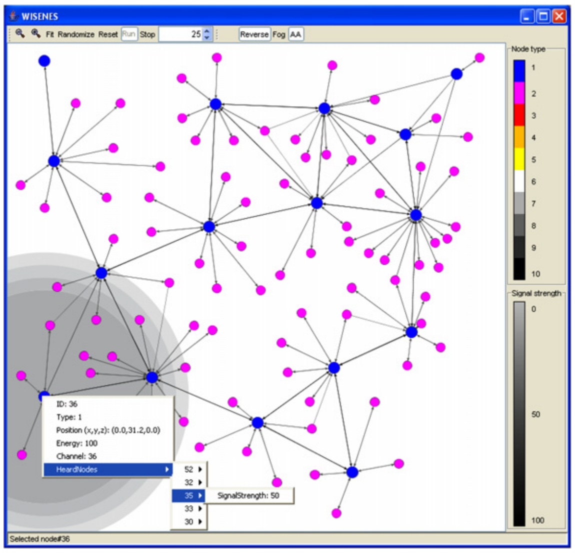
Table 3 shows the output results and their types. As the events are logged as they occur, the momentary values of the presented results are stored into the logs. Depending on the type of the result, the values are also accumulated or averaged to obtain an overall knowledge about the system behavior.

3.2. WISENES user interfaces

WISENES has two UIs for controlling and monitoring simulation runs. Simulations are started and controlled through a command line interface. The progress of the simulation and the topology of a simulated WSN are visualized in WISENES GUI. GUI is implemented in Java using Java foundation classes Swing packages [39]. The communication between GUI and WISENES is implemented by a socket interface. A screenshot of GUI visualizing a hundred-node TUTWSN simulation is depicted in Fig. 3.

Table 3
WISENES output results and their types

Result	Type	Description
Power consumption (final and variation during time)		
Node, total	Float (μW)	Total power consumption in node
CPU	Float array (μW)	Power consumption in CPU in execution and different sleep states
Transceiver unit	Float array (μW)	Transceiver unit power consumption in sending, receiving, and scanning
Peripherals	Float (μW)	Peripheral power consumption separately for each
Protocols	Float array (μW)	Power consumption in the execution of different protocols (network vs. node)
Application tasks	Float array (μW)	Power consumption in the execution of application tasks (network vs. node)
Memory consumption (final and variation during time)		
Application tasks, instruction	Integer array (instructions)	Instruction memory consumption in application tasks per node
Application tasks, data	Integer array (byte)	Static and dynamic data memory consumption in application tasks per node
Protocols, data	Integer array (byte)	Static and dynamic data memory consumption in protocols per node
Network performance (averaged and variation during time)		
Throughput	Integer array (bps)	Throughput per link in the network
Delays	Float array (ms)	Transfer delays per link
PER	Float array (constant)	Packet error rate per link
Collisions	Integer	Number of collisions in a node
Protocol performance (averaged and variation during time)		
Delays	Float array (ms)	Delays due to buffering and control in different protocol layers
Buffering	Integer array (constant)	Buffer lengths in the protocol
Throughput utilization	Float array (constant)	Utilized vs. available throughput
Application performance (averaged and variation during time)		
Delays	Float array (ms)	Communication delays between tasks
Activation accuracy	Float array (ms)	The variance in task activation times vs. to expected activation times
Data coherence	Float array (constant)	The accuracy and sufficiency of provided data



ID: 36
Type: 1
Position (x,y,z): (0.0,31.2,0.0)
Energy: 100
Channel: 36
SignalStrength: 50

Fig. 3. WISENES GUI screenshot from a hundred-node TUT-WSN simulation.

Different node roles, in this case cluster headnodes and subnodes, are shown by different colors. Remaining energy levels and the received signal strengths of the active communication links are presented in a pop-up window when a node is selected. By double clicking a node, its transmission ranges with different transmit powers are illustrated. Furthermore, nodes can be moved by dragging and dropping them in GUI.

3.3. WISENES tools

A set of Tool Command Language (TCL) scripts is implemented for WISENES initiation and result handling. The initiation scripts facilitate a simulation case construction from different protocols and a random node population generation for large simulation cases. The possible relations between different types of platforms can be given as a parameter to the population generation script, e.g. in order to force ten low power nodes in the vicinity of a more powerful one.

The result handling scripts facilitate power consumption, data packet tracing, and link utility evaluations. The power consumption information is gathered from individual nodes and a listing defining detailed power characteristics for different components in each node is created. The packet tracing tracks the hops of a packet from its source to the destination node, and determines the delays on different hops and protocol layers. The link utility evaluates activity and congestion of node to node connections.

4. WISENES framework

The *WISENES framework* consists of models for transmission medium, sensing channel, sensor node, and of a central simulation control that manages simulations and handles simulator Input/Output (I/O). Networking protocols and node platform modeling are embedded to the sensor node model.

In addition to application tasks and protocols described by the designer, SDL is used for the implementation of the WISENES framework. The tool used for SDL development is Telelogic [40] TAU SDL Suite [41], version 4.5. The SDL suite uses a graphical notation for SDL design, and provides tools for simulation, integration, and implementation.

4.1. SDL introduction

SDL is used for designing systems ranging from general software to embedded applications. MoC in SDL is parallel communicating Extended Finite State Machines (EFSM). SDL hierarchy has multiple levels, of which the system level consists of a number of blocks that clarify the representation. They can be recursively divided into sub-blocks. The behavior of a block is implemented in processes described by EFSMs. The representation of a process can be simplified by implementing a part of the functionality in a procedure. Blocks and processes can be implemented using the type concept of SDL, which allows their instantiation. These type definitions can be included with other type definitions to SDL packages that facilitate modular system design. The maximum number of instantiated blocks must be defined at a compile time, whereas processes can be created dynamically during runtime [42].

Processes in a same or in different blocks communicate by asynchronous signals that can carry any number of parameters. Each process has an infinite First-In-First-Out (FIFO) buffer for incoming signals. Signal routes define which type of signals a process can send and to which processes. An outgoing signal is routed according to the signal route or a Process IDentifier (PID). Communication between processes can also be executed synchronously by calling remote procedures, which are exported on the process interface [42].

Due to its formality, SDL can be automatically converted for example to C source code, which can then be used to make an executable application or simulation. Telelogic TAU SDL simulation engine supports discrete event simulations and real-time simulations. In WISENES we utilize discrete event simulation, in which events are processed and handled in the order of occurrence. This makes the time concept fully parallel and avoids an active waiting during the idle times.

Environment functions are needed for the interaction between SDL and its execution environment. Dedicated functions are defined for environment initialization, unloading, signal output, and signal input. The output function is called when a signal is sent from the SDL system to the environment. Because a method for interruption is absent in SDL, the input function must be polled for receiving signals from the environment. In Telelogic TAU SDL simulation engine, the input function is called after every transaction, which is an execution flow from a state to another triggered by an incoming signal. An SDL procedure can be substituted by an external function, in a case where SDL lacks expressivity or a more efficient implementation is desired. Both environment and external functions are implemented in C for WISENES.

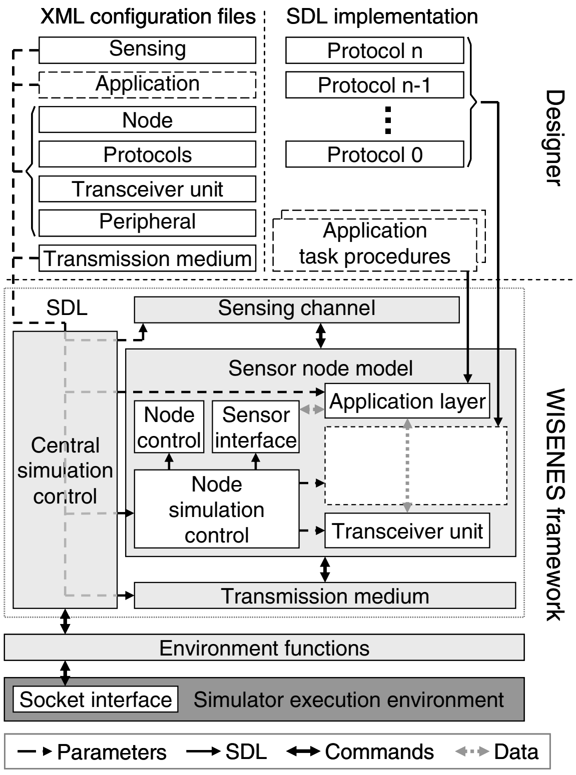


Fig. 4. The architecture of WISENES instantiation.

4.2. WISENES instantiation

The instantiation of WISENES is depicted in Fig. 4. The designer selects the protocols from the library or implements new ones in SDL and integrates them to the WISENES framework. The upper and lower interfaces of the protocol stack are the pre-defined interfaces to the application layer and transceiver unit, respectively. Application functionality is either implemented as SDL procedures or described by a task graph.

The protocol stack consists of data link, network, and middleware layers that are instances of block types implemented in SDL packages. The interfaces between the layers are fixed, but a layer can be bypassed, i.e. a network layer can communicate with the application layer at its upper interface. The internal implementation of layers is not restricted in any way.

Node platforms are parameterized in the XML configuration files that are parsed by *Central Simulation Control*. The parameters are passed to *Sensor Node* model. Node coordinates are relayed also to *Transmission Medium* and *Sensing Channel*, both of which have also dedicated parameters.

The interfacing of WISENES GUI is implemented in environment functions that maintain the socket connection. Information to GUI is updated only periodically in order to lessen communication. A data structure that defines sensor node parameters is sent to the environment functions as a signal parameter and parsed to the socket.

The SDL system of WISENES framework is illustrated in Fig. 5. The framework consists of SDL blocks that implement the main functional models and the central simulation control. The sensor node is a dynamic block of type *Node_Type*, and the number of its instances is specified by *NODE_COUNT*. The figure depicts also the signal routes between the blocks and a dedicated signal route to the environment.

4.3. Central simulation control

The central simulation control initiates the WISENES framework, controls active simulation

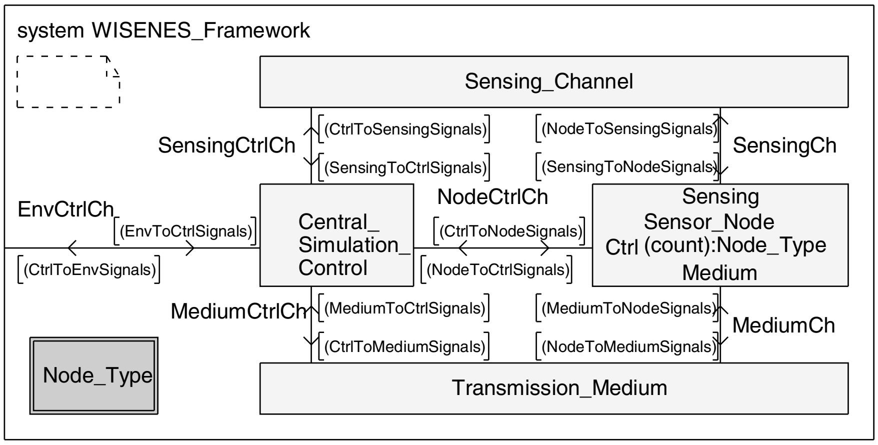


Fig. 5. The SDL system level of WISENES framework.

runs, and performs event logging. The information gathering for the control and logging is implemented in remote procedures that are presented with their parameters in Table 4.

The input parameters of *UpdateNodeInformation* procedure specify the node identifier, current battery level, role, and the connectivity to other nodes. The node role defines whether a node is a headnode, a subnode, or a sink. Sensor nodes call the procedure whenever any of the parameter values changes. The information is relayed to GUI and utilized for determining whether the simulation end condition is satisfied. The end condition is set by the designer and it defines a simulation time limit, a percentage of dead nodes, or e.g. a limit for an application task activation count.

Distinct remote procedures for event and data packet logging are exported for each protocol layer, application tasks, and for framework components. Their parameters vary depending on the layer. Logs are stored in dedicated data structures and written to files either periodically when WISENES memory consumption exceeds a pre-defined limit, or at the end of the simulation.

Table 4
The remote procedures exported by the central simulation control

Procedure	Parameters
UpdateNodeInformation	NodeId, NodeRole, BatteryState, TransceiverUnit, Connectivity
LogXxxEvent	NodeId, EventName, Cause, Consequence, ...
LogXxxData	NodeId, PacketId, DataLength, DataAction, ...

4.4. Transmission medium

The transmission medium model provides the connectivity between sensor nodes. It is implemented as an SDL process that redirects signals from a source to the destination sensor node SDL blocks. Sensor nodes register their node identifier and transceiver unit PID to the transmission medium for enabling the data redirecting. Due to the nature of SDL data typing, transmitted data are separately copied for each destination node.

The signal propagation in the transmission medium is based on the transceiver unit dependent signal attenuation. The curve S defining the Packet Error Rate (PER) is

$$S = kd - \left(b + \frac{P}{v}\right), \tag{1}$$

where d is the distance between the source and destination nodes (m), and P is the transmit power (dBm). Constants k, b, v are derived from the measured signal attenuation curve. PER for a packet is

$$\text{PER} = \left\{ \begin{array}{ll} 1, & \text{if } S \geqslant 1 \\ S, & \text{if } L < S < 1 \\ L, & \text{if } S \leqslant L \end{array} \right\}, \tag{2}$$

where L is the lower limit for the PER.

In order to realistically model the hidden node problem and collisions, S is calculated separately for each node within the coverage of the transmission. The transmit power is specified by the source node. After the PER evaluation, a random number $0 \leqslant r < 1$ is generated again separately for each node. A transmission is successful, if $r > \text{PER}$. If

$S > 1$, the signal attenuates during the transmission, otherwise the signal is relayed to the destination. If $S < 1$ and $r < \text{PER}$, the unsuccessful transmission is indicated to the recipient but the copying of the actual data is not performed.

A delay during a transmission is calculated by dividing the packet length by the transceiver unit throughput. The delay of the signal propagation in the medium is omitted. Thus, a packet is relayed to the destinations immediately after the transfer delay.

4.5. Sensing channel

The sensing channel model simulates physical phenomena. Similarly to the transmission medium, the sensing channel utilizes node coordinates. Each phenomenon is modeled separately with individual propagation characteristics. The propagation depends also on the media in the vicinity.

Our current sensing channel implementation generates random stimuli for phenomena, except for the location queries that return node coordinates. The upper and lower limits are defined for each phenomenon. Currently simulated phenomena are temperature, humidity, vibration, sound, luminance, and location information. The selected approach is applicable for environmental monitoring, but a more detailed sensing channel must be implemented for e.g. object tracking applications.

4.6. Sensor node

The sensor node SDL block implementing the node model is depicted in Fig. 6. On the sensor node model, *Physical layer*, *Sensor Interface*, *Application Layer*, and *Node Control* blocks are part of the WISENES framework, while the instantiated protocol layers are selected or implemented by the designer. The signal routes between the protocol layers are for data communications, whereas the signals sent from the node control to the other blocks are for the initiation and shutdown.

4.6.1. Physical layer

A transceiver unit process at the physical layer models the hardware and its device driver. The process implements the interface to the transmission medium, performs collision detection, and models the internal delay in a transceiver unit. Depending on the modeled hardware, additional features such as Cyclic Redundancy Check (CRC) [38] for error detection or an algorithm for encryption are incorporated to the model. No real CRC or encryption calculation is performed, but the data consistency is validated from the parameters of the signal received from the transmission medium. The upper interface of the transceiver unit defines signals for the lowest protocol layer. Dedicated signals are declared for the transceiver unit control, transmitter or receiver enabling, carrier sensing, data sending, send confirmation, and data reception indication.

Transferring data to or from the transceiver unit causes delay, because a transceiver unit is typically a distinct hardware component on a node platform. A delay, which is calculated by dividing the data length in bits by the interface bit rate, is generated when data is loaded to or from transceiver unit.

4.6.2. Sensor interface

The sensor interface block implements a process that models the Analog-to-Digital Converter (ADC) and sensor operations. When an application task initiates sensing, it sends a signal to the sensor interface process. The process activates the sensor and other required peripherals (e.g. ADC for sampling the analog sensor output) for that phenomenon. The sensor interface process acquires a value from the sensing channel by signal exchange. The operation delay depends on the associated sensor and possibly on the ADC sampling frequency, which are defined in the input parameters. The used peripherals are reserved during the operation.

4.6.3. Application layer

The application layer consists of a process that implements the scheduling of application tasks. This approach is selected to facilitate the task scheduling when they are implemented as SDL procedures. When an application is described as a task graph, the application layer process emulates the execution of tasks. In this case no real functionality apart from the sensing and data transfer initiation is implemented.

The application task procedures are implemented by the designer. They define the functionality of the tasks, while the task state control and scheduling are implemented in the application layer process. Task state is *running* when it is executed, *ready* when it is ready for execution but another task is running, or *wait* when the task requires either a timer, data, or sensor event to occur before running. Supported scheduling algorithms are round robin and static priority scheduling.

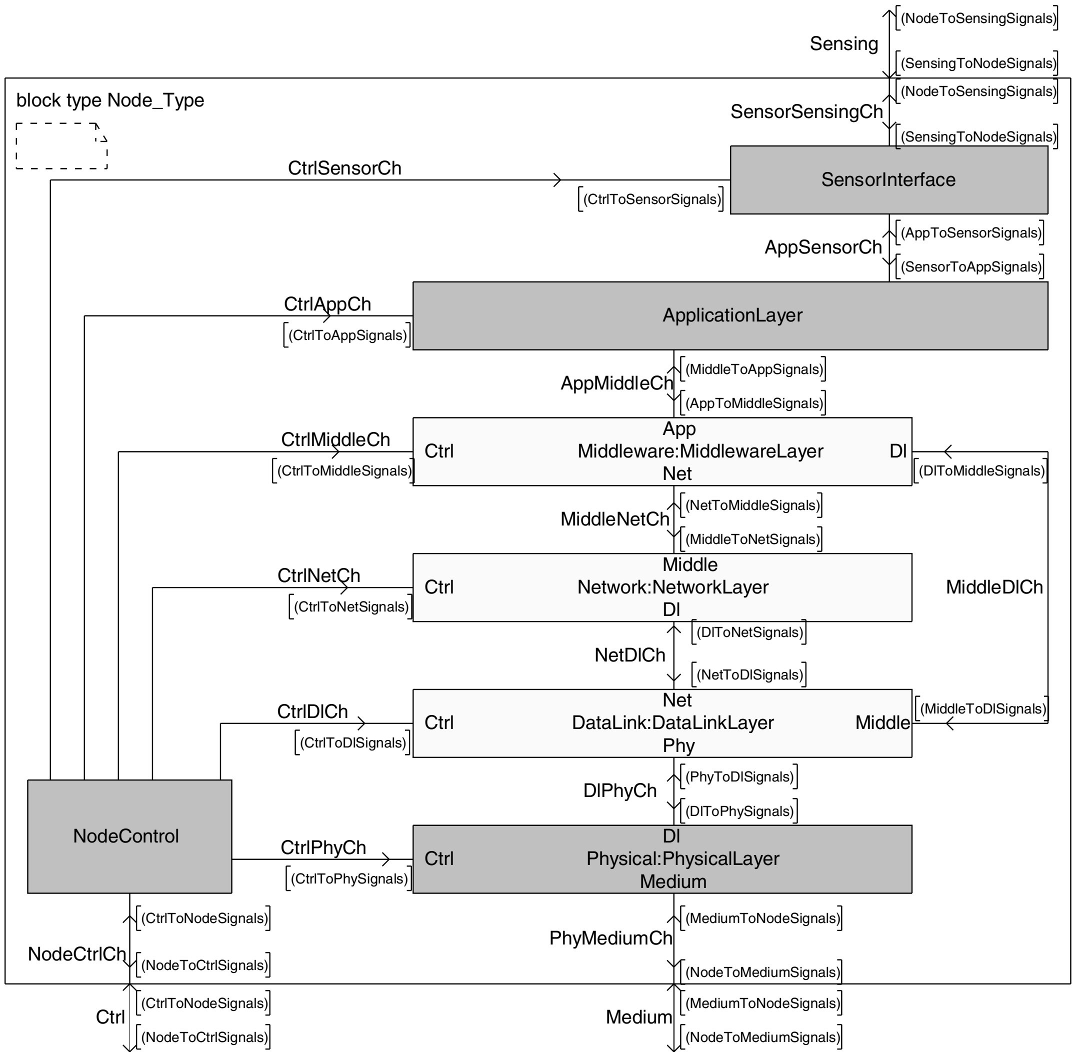


Fig. 6. The SDL block type for WISENES sensor node.

When a task is ready and scheduled for processing, the application layer process calls the procedure that implements the task. The event that moved the task to the ready state and the payload associated to the event are given in the parameters of the procedure. When the task completes its next transition it enters to the wait state. The waited event is returned to the application layer process. In occurrence of an event, all tasks waiting for it are set to the ready state.

4.6.4. Node control

The node control block consists of two processes, *Node control* and *Node simulation control*. The node control process implements OS routines in WISENES. The possible states of a sensor node and the actions triggering node state transitions are depicted in Fig. 7. When the node is in an *active* state, its Central Processing Unit (CPU) and transceiver unit are powered. Transceiver unit dependent states *receiving* and *transmit* are substates of the active state. The node enters to a *transceiver sleep* state, when its transceiver unit is not needed during a constant period. When there is nothing to process on CPU, the node is set to a sleep state that depends on the length of the inactive period. The periods and corresponding sleep states are defined in the input parameters. Tasks can be executed and sensing activated when the node is in the active state, one of its substates, or in the transceiver sleep state.

The remote procedures exported by the node control for implementing OS routines are presented

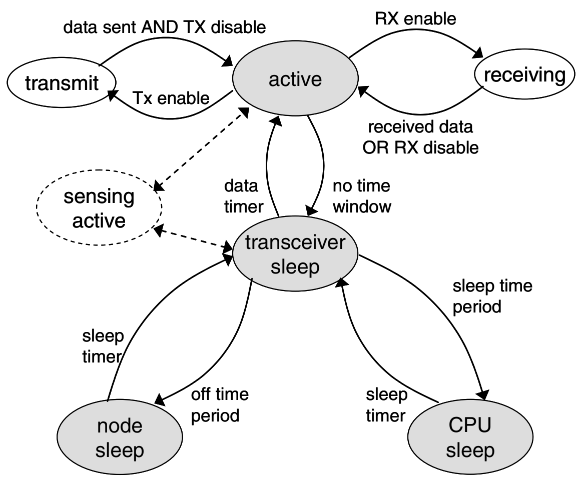


Fig. 7. Sensor node states and state transitions in WISENES.

in Table 5. Periodically activated protocols and application tasks call *SetNextActiveTime* procedure to indicate their next wakeup time. When all protocols and application tasks are inactive, the node control sets the node to a sleep state. The memory management for application tasks and protocols is implemented in *ReserveMemory* and *FreeMemory* procedures. Protocols and applications reserve CPU time slots by calling *Execute* procedure. The executed operations are given as a parameter. A protocol may check remaining energy resources of the node by calling *GetCurrentBatteryLevel* procedure.

The *Node simulation control* implements a per-node interface to the central simulation control and models the power consumption of node platform. In the initiation, the node simulation control relays node parameters in signal parameters to the node control and to the different layers. During an active simulation run, the node simulation control gathers GUI related information from the node and passes it to the central simulation control. Remote procedures that implement the gathering are presented in Table 5. *UpdateNodeRole* and *UpdateNodeConnectivity* must be called from protocols that possess the required information.

The node power consumption modeling is implemented in the node simulation control by a linear battery model, in which the component power consumption is independent of the battery discharge rate. The remote procedures related to the power consumption are called only from the SDL processes that are part of the WISENES framework. The sensor interface process indicates a peripheral activation by calling *ConsumePeripheralPower* procedure. Procedures *ConsumeTransceiverSendPower* and *ConsumeTransceiverReceivePower* are called by the transceiver unit process when a transmitter and a receiver are activated, respectively. The node control marks the node state transitions by calling *SetNodeState* and indicates the execution by calling *MarkActiveExecution*. *GetCurrentBatteryLevel* procedure in the node control calls *GetRemainingEnergy* to determine the battery level.

The power consumption in a sensor node can be divided into a very detailed level, as peripherals, protocols, and application tasks can be identified when they indicate their activation by calling a dedicated remote procedure. The power consumption by the transceiver unit can be split between the transmission and reception, while CPU power consumption can be assigned to the different states, and to the application tasks, protocols, and device drivers.

The harvesting of energy from the surroundings is modeled in the node simulation control, if such a peripheral is available at the node platform. The generated energy is randomized between 0 and the harvesting capacity limit specified in the input parameters. When a sensor node runs out of energy, the node simulation control removes the node from the transmission medium and indicates this to the central simulation control.

Table 5
The remote procedures exported by the node control and node simulation control

Procedure	Parameters
Node control	
SetNextActiveTime	CallerId, ActivationTime, TransceiverUnitControl
ReserveMemory	CallerId, Type, Amount
FreeMemory	CallerId, Type, Amount
Execute	CallerId, OperationCount
GetCurrentBatteryLevel	*Returns* CurrentLevel
Node simulation control	
UpdateNodeRole	NodeRole
UpdateConnectivity	NearbyNodeIds, NearbyNodeSignalStrengths
ConsumePeripheralPower	PeripheralId, ActivationTime
ConsumeTransceiverSendPower	SendBytes
ConsumeTransceiverReceivePower	Type, ReceiverActivationTime
SetNodeState	State
MarkActiveExecution	CallerId, OperationCount
GetRemainingEnergy	*Returns* RemainingEnergy

4.7. Prototype mapping in WISENES

The accurate parameterization of sensor node platforms in the XML configuration files and the detailed capability modeling in the WISENES framework lead to the simulation results that correspond to those obtained from real node platforms. We refer the parameterization of hardware platforms and their modeling in the simulator to as *prototype mapping*.

The exact modeling of node and peripheral state changes, and the detailed specification of the node power characteristics in the input parameters result in realistic mapping of the power consumption. For a fine-grained power consumption mapping of protocol and application execution, the number of executed operations given as a parameter to *Execute* procedure must be estimated during the initial design phase. For the further evaluation, benchmarking information from the prototype measurements is utilized.

The memory and processing capacities of the node platforms are defined in the input parameters. The static instruction and data memory usage of protocols and applications is parameterized, whereas the dynamic data memory consumption depends on the data buffering and control constructs stored within each protocol layer. The processing capacity is controlled by *Execute* procedure in the node control.

The delays in the data transmissions are modeled in the transmission medium and in the transceiver unit. This approach considers the transfer delay and the internal processing delay of the transceiver unit, but omits the signal propagation delay in the transmission medium.

4.8. Simulation of node code implementations in WISENES

In addition to the simulation of node models that are composed of protocols implemented in SDL, WISENES contains a *prototype emulation environment*, which allows the simulation of low-level C implementations that are directly applicable for node prototypes. Similarly to sQualnet [15], the C code implementation is integrated above the data link layer by a dedicated network layer SDL block. This block consists of a process that redirects the per-node signals received from the data link layer to the emulation environment and vice versa, and implements the timer concept for the emulation.

When an SDL block in WISENES is created for a node, the prototype emulation environment creates a simulator host OS thread for the node and associates the thread to the node id. On the reception of a signal from the data link layer, the SDL block calls a signaling function from the prototype emulation environment. The emulation environment redirects the signal to the correct thread and converts the signal parameters to a function call or to a message for the final C implementation. After the subsequent processing is completed, the SDL block calls a query function at the emulation environment. The query function gathers the events caused by the received signal and passes them back to the SDL block. The indications related to timers are handled similarly.

The current version of the prototype emulation environment in WISENES supports TUTWSN C code implementations. The required effort for porting the emulation environment core to support other platforms is minimal. Only the interface provided by the emulation environment for the C code needs to be adapted.

5. WISENES use-cases: TUTWSN and ZigBee

WISENES is used for the design and evaluation of two use-cases: proprietary TUTWSN and a standard ZigBee network. In both, the design starts from scratch and ends up to extensive performance simulations. Prototype platforms for both cases are parameterized using the XML configuration files and the protocols are designed and implemented in SDL.

5.1. TUTWSN implementation

TUTWSN is a very energy efficient WSN targeted to low data rate applications, such as environmental and industrial monitoring [4]. TUTWSN consists of a configurable full feature protocol stack, a family of physical node platforms, and several GUIs for the network management and visualization.

5.1.1. TUTWSN protocol stack

The TUTWSN protocol stack in WISENES consists of a middleware, a multi-hop routing protocol at the network layer, and a MAC protocol at the data link layer. TUTWSN MAC is an energy efficient clustered protocol that minimizes the time spent in a receiving state per a node. Time Division

Multiple Access (TDMA) is used for the intra-cluster communication, whereas neighboring clusters operate on different frequency channels. The control signaling, not the data transfers, for the inter-cluster communication is performed on a dedicated network signaling channel.

At the beginning of each access cycle a cluster headnode broadcasts an active network beacon to the network signaling channel. Other headnodes utilize the beacon for the multi-hop routing, whereas subnodes listen the channel only when they are searching for a cluster for the association. Neighbor discovery times are shortened by idle network beacons that are sent during the inactive period of the access cycle. An idle network beacon indicates the time of the next active network beacon.

The communication within a cluster is performed during a superframe consisting of cluster beacons, aloha slots for contention, and reservation data slots. The inter-cluster data transfers are made during the superframe of the recipient headnode. Cluster beacons start a superframe by informing the associated nodes of the allocated reservation data slots and access cycle timing. Subnodes listen the beacons but sleep the rest of the access cycle, unless they have data to process. In the contention slots, subnodes and neighbor headnodes send occasional data and slot reservation requests to the cluster headnode. The reservation data slots are allocated for periodical data transfers. Each slot consists of an uplink for data sending to the headnode and of a downlink for acknowledgements and data sending to the associated node.

The TUTWSN MAC protocol in WISENES implements a self-organizing cluster creation algorithm, and intra-cluster and inter-cluster communication. For the inter-cluster communication, the cluster access cycle timing can be adapted according to the routing protocol needs. In order to minimize the delay, the start time of the own access cycle is adjusted so that the superframe is completed right before the start of the access cycle of the next hop cluster. The length of a slot is 20 ms, uplink and downlink being both 10 ms. The access cycle length, and the number of contention and reservation data slots are varied.

The flooding routing protocol is implemented for the multi-hop path creation. Route requests are broadcast to the network until a path to the destination is found. In order to avoid unnecessary communication, the route requests are identified so that duplicates can be discarded. Further, the neighboring information from the MAC layer is utilized. If a node knows a valid route to the destination, it initiates a response. Each node stores only the address of the next hop and the total hop count for each destination.

The TUTWSN middleware layer controls the application task hosting in sensor nodes and abstracts the communication between tasks. The middleware keeps track of the neighbor nodes, which host tasks that are related those tasks hosted by the node itself. When a task sends data to another task, the middleware redirects the data to the correct node.

5.1.2. TUTWSN prototype platform

The TUTWSN prototype used in the simulations is depicted in Fig. 8. The main component on the prototype is a 2 MIPS Xemics XE88LC02 [43] MicroController Unit (MCU) consisting of a CoolRisc 816 processor core, a 16-bit ADC, 22 KB program memory, and 1 KB data memory. 2.4 GHz NordicVLSI nRF2401 transceiver unit [44] on the prototype supports 250 kbps and 1 Mbps data rates with transmit power adjustable between −20 and 0 dBm. A 16-bit CRC error detection is implemented in the transceiver unit. For the environmental monitoring, the prototype has an integrated MAX6607 temperature sensor. A 0.22 F capacitor is used as the energy storage for the prototype.

The mapping between the TUTWSN prototype and the WISENES sensor node model is depicted in Fig. 9. The TUTWSN protocol stack for the prototype is implemented in C and executed on Xemics MCU. In WISENES, it is implemented in SDL on the sensor node model. Application tasks run on top of the protocol stack. A lightweight OS that is implemented in C controls the scheduling and power management on the prototype. In WISENES, its functions are modeled by the node control.

The nRF2401 transceiver unit and its device driver are implemented by the transceiver unit process

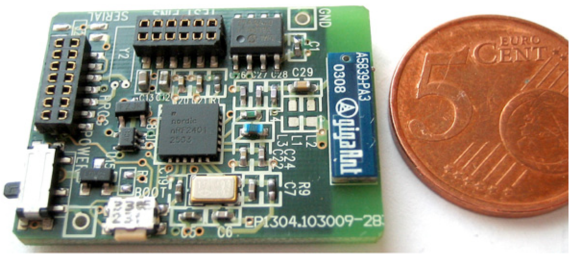


Fig. 8. TUTWSN Xemics prototype.

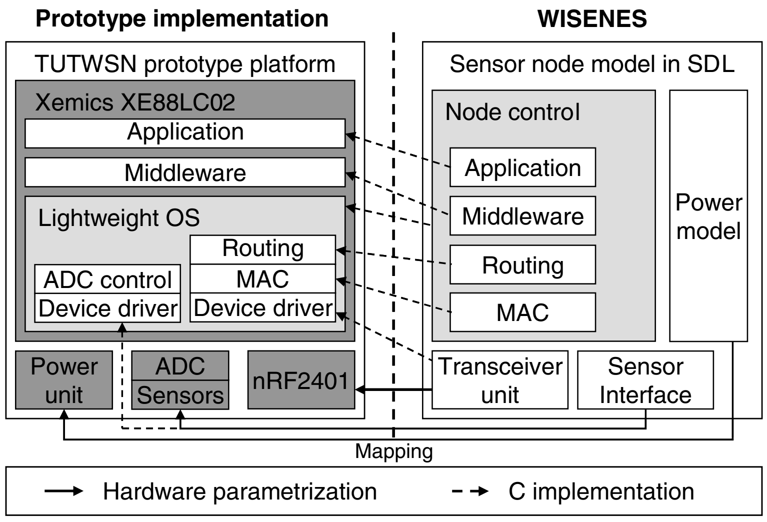


Fig. 9. The prototype mapping between the WISENES sensor node model and TUTWSN prototype.

in WISENES. The sensor interface process models the sensors, ADC, and their device drivers. The power model in the node simulation control simulates the power unit and the power consumption of the hardware components.

Fig. 10 gives an example of the XML configuration parameters that specify the presented TUTWSN prototype for WISENES. A dedicated parameter set for a sensor node is assembled from the node type parameters (a), transceiver unit parameters (b), peripheral parameters (c), and node parameters (d). The transceiver unit dependent signal attenuation curve constants for nRF2401 are following: $k = 0.2385$, $b = 2.7$, $v = 18.0$, and $L = 0.03$. These are obtained by measuring PER for different distances and deriving the values from the results.

5.2. ZigBee implementation

ZigBee [5] is a networking architecture targeted to low-cost and low-power monitoring and control applications. Although not directly designed for WSNs, many of its characteristics have encouraged its use also in WSN scenarios. The topology in a ZigBee network can be either star or mesh. In the star topology, a single *ZigBee coordinator* controls the whole network, whereas in the mesh topology, nodes communicate directly through peer-to-peer links. A special type of a mesh network is a cluster-tree topology, in which a coordinator starts the network but other coordinators (referred also to as *ZigBee routers*) can extend the network. In the following, we concentrate on the cluster-tree topology, because it is the only alternative for the low-power WSNs and the closest to the TUTWSN use-case.

5.2.1. ZigBee protocol stack

The ZigBee protocol stack defined in the specification consists of a MAC protocol, network layer, and application layer. WISENES implements a full featured version of the IEEE 802.15.4 Low-Rate Wireless Personal Area Network (LR-WPAN) MAC protocol [45] for ZigBee, and a simplified version of the ZigBee networking layer. The same middleware layer as in TUTWSN is used on top of the ZigBee protocols.

The ZigBee MAC in the cluster-tree topology networks uses a beacon-enabled channel access. A ZigBee network is started and its parameters defined by a ZigBee coordinator. When joining to a network, a ZigBee router starts transmitting beacons on the same communication channel with the parameters advertised by its parent. The beacon is followed by a Contention Access Period (CAP) for Carrier Sense Multiple Access (CSMA) data transfers. Periodic data transfers between the ZigBee coordinator and a child device can be made in guaranteed time slots during a Contention Free Period (CFP) that follows CAP in the superframe of the ZigBee coordinator. The length of the access cycle (beacon period) and the superframe (CAP period) depends on the constants *BeaconOrder* and *SuperframeOrder* that are defined by ZigBee coordinator.

```
<node_type id=“1”>
  <name>TUTWSN Xemics Node</name>
  <type>FFU</type>
  <cpu_info>
    <capacity>2000000</capacity>
    <code_memory_inst>8000</code_memory_inst>
    <data_memory_byte>1024</data_memory_byte>
  </cpu_info>
  <battery>
    <voltage_V>3.0</voltage_V>
    <capacity_mAh>0.57</capacity_mAh>
    <efficiency>0.8</efficiency>
    <harvest_uW>0</harvest_uW>
  </battery>
  <state_info>
    <state name=“active” ms=“0” uW=“1350”/>
    <state name=“node sleep” ms =“3” uW=“19”/>
  </state_info>
  <transceiver_unit id=“1”/>
  <peripheral_info>
    <peripheral id=“1” count=“1”/>
    <peripheral id=“2” count=“2”/>
  </peripheral_info>
</node_type>
```

a

```
<transceiver_unit id=“1”>
  <name>nRF2401</name>
  <throughput_bps>1000000</throughput_bps>
  <rssi capability=“NO”/>
  <data_load_info bps=“230400” uW=“1310”/>
  <receiver_info transient_ms=“0.25” uW=“43600”/>
  <transmitter_info>
    <transient_ms>0.25</transient_ms>
    <tx_power_levels>
      <tx_power dBm=“-20” uW=“18700”/>
      <tx_power dBm=“-10” uW=“23500”/>
      <tx_power dBm=“-5” uW=“26400”/>
      <tx_power dBm=“0” uW=“29300”/>
    </tx_power_levels>
  </transmitter_info>
  <carrier_sense_info capability=“NO”/>
</transceiver_unit>
```

b

```
<peripheral id=“1” phenomenon=“NONE”>
  <name>ADC</name>
  <dependency id=“0”/>
  <active_period ms=“0.5” uW=“15”/>
</peripheral>
<peripheral id=“2” phenomenon=“TEMPERATURE”>
  <name>Temperature sensor</name>
  <dependency id=“1”/>
  <active_period ms=“0.1” uW=“10”/>
</peripheral>
```

c

```
<node_list count=“5”>
  <node id="1“ type=“2”>
    <coordinates x=“0.0” y=“0.0” z=“0.0”/>
  </node>
  <node id=“2“ type=“1”>
    <coordinates x=“7.23” y=“3.4” z=“0.72”/>
  </node>
  <node id=“3“ type=“1”>
    <coordinates x=“8.4” y=“-5.21” z=“1.03”/>
  </node>
  <node id=“4“ type=“1”>
    <coordinates x=“15.4” y=“-8.53” z=“1.76”/>
  </node>
  <node id=“5“ type=“1”>
    <coordinates x=“22.4” y=“0.21” z=“0.93”/>
  </node>
</node_list>
```

d

Fig. 10. TUTWSN prototype: (a) node type, (b) transceiver unit, (c) peripheral and (d) node parameters for WISENES.

Each child device communicates with its parent coordinator during CAP. All transactions must be completed before the end of CAP. Before a transmission, a random back-off period is waited. After the back-off, the state of the transmission medium is assessed. If the medium is busy, the back-off procedure continues. If a coordinator has data pending for its child devices, it indicates the identifiers of these child devices in the beacon. When a device receives a beacon listing its address, it sends a data requests to the coordinator, after which the transmission of data is made.

The ZigBee networking layer in WISENES implements a mechanism for joining and leaving the network, the algorithms for the cluster-tree topology formation, and data routing from devices to the ZigBee coordinator. The routes are created according to parent associations, i.e. each ZigBee router sends data targeted to the ZigBee coordinator to its parent. The continuous neighbor discovery and the support for other network topologies are not implemented. Further, the short address assignment and superframe scheduling algorithms benefit the simulator addressing and timing information.

5.2.2. ZigBee prototype platform

The prototype platform used in WISENES for the ZigBee evaluation is constructed from two separate components. The MCU module of the prototype is 2 MIPS PIC18LF4620 MCU [46] with 64 KB of program and 4 KB of data memory, an integrated 10-bit ADC, and an interface to MAX6607 temperature

```
<node_type id=“1”>
  <name>ZigBee PIC node</name>
  <type>FFU</type>
  <cpu_info>
    <capacity>1000000</capacity>
    <code_memory_inst>32768</code_memory_inst>
    <data_memory_byte>3986</data_memory_byte>
  </cpu_info>
  <battery>
    <voltage_V>3.3</voltage_V>
    <capacity_mAh>2100</capacity_mAh>
    <efficiency>0.91</efficiency>
    <harvest_uW>0</harvest_uW>
  </battery>
  <state_info>
    <state name=“active” ms=“0” uW=“3010”/>
    <state name=“node sleep” ms =“2” uW=“30”/>
  </state_info>
  <transceiver_unit id=“2”/>
  <peripheral_info>
    <peripheral id=“1” count=“1”/>
    <peripheral id=“2” count=“2”/>
  </peripheral_info>
</node_type>
```

a

```
<transceiver_unit id=“2”>
  <name>CC2420</name>
  <throughput_bps>250000</throughput_bps>
  <rssi capability=“YES”/>
  <data_load_info bps=“460800” uW=“1310”/>
  <receiver_info transient_ms=“0.22” uW=“54820”/>
  <transmitter_info>
    <transient_ms>0.22</transient_ms>
    <tx_power_levels>
      <tx_power dBm=“-15” uW=“28300”/>
      <tx_power dBm=“-10” uW=“31400”/>
      <tx_power dBm=“-5” uW=“37600”/>
      <tx_power dBm=“0” uW=“46400”/>
    </tx_power_levels>
  </transmitter_info>
  <carrier_sense_info capability=“YES” uW=“54200”/>
</transceiver_unit>
```

b

Fig. 11. ZigBee prototype: (a) node type and (b) transceiver unit parameters for WISENES.

sensor. The transceiver unit is 2.4 GHz Chipcon CC2420 transceiver [47], which is IEEE 802.15.4 standard compliant. The transceiver support 250 kbps data rate and the transmit power is adjustable between −24 dBm and 0 dBm. A CR123A lithium battery is used as an energy storage.

Although the prototype is assembled with two distinct components, the resulting platform is realistic. MCU has enough resources for the ZigBee implementation. The power consumptions of the components are measured separately, but they are combined in WISENES. The transceiver unit parameters are measured using a Chipcon SmartRF CC2420DK Development Kit [47]. The constants for the transmission medium are: $k = 0.08$, $b = 3.4$, $v = 6.0$, and $L = 0.05$. The WISENES XML configuration parameters describing the prototype node type (a) and transceiver unit (b) are depicted in Fig. 11.

6. WISENES evaluation

The design of a new protocol and the deployment of a complete WSN protocol stack in WISENES are straightforward for the designer due to the hierarchical structure of SDL and the modularity of the WISENES framework. The graphical design based on state machines is well-suited for the protocol modeling. The modeled protocols are internally independent of WISENES, but an external interface for the adaptation of each layer to the WISENES framework is required. Interface templates are provided for the designer.

For the usage evaluation, the full feature ZigBee MAC protocol was implemented according to the specification by a single designer within two working weeks. The designer was familiar with the WISENES interfaces, but still the development cycle was faster than expected. The SDL description attached to the IEEE 802.15.4 standard was too incomplete to be used in WISENES.

The full implementation of the ZigBee MAC protocol in WISENES consists of three processes having totally 31 different states. In addition, 75 SDL procedures are implemented in order to avoid redundant implementations and clarify the description. The number of state transitions in the implementation is 163. Among the transitions, there are in total 1817 divergent execution paths. The implementation of TUTWSN MAC protocol in WISENES consists of two processes that have totally 21 states and 32 procedures. The number of state transitions is 112, but there are still 2642 divergent execution paths.

The simulation and evaluation capabilities of WISENES are assessed by simulating TUTWSN and ZigBee networks. The application used in the simulations is an environmental monitoring application. All nodes in WSN observe the temperature in their vicinity. Sensed data are aggregated in cluster headnodes (coordinator) and routed for a further processing to the sink node (ZigBee coordinator). The application is implemented by describing it as a task graph.

The evaluated aspects in WISENES are the performance of the simulator, applicability for the large scale simulations, and the accuracy of the prototype mapping. All simulations are executed on a workstation with a 2.8 GHz Pentium4 processor with 1 GB of memory and running Windows XP SP2. WISENES memory consumption is limited to 150 MB, meaning that gathered log data are written to files when the limit is exceeded.

6.1. WISENES performance

TUTWSN simulations are repeated five times with 10, 100, and 1000 of nodes. The node population is randomly generated for each simulation run. Monitored aspects are the correct functionality and performance. The initial energy capacity of a TUTWSN node is set twenty times larger than specified in Fig. 10 in order to obtain required lifetime. The TUTWSN access cycle in simulations is 10 s, and each node measures temperature once in an access cycle and sends the result towards the sink node. Headnodes aggregate the subnode readings to a single data packet.

The presented WISENES performance is the time elapsed for the simulation of a single node for a 24-h period. During that time, a node initiates 8640 data packets. The resulting time is obtained by dividing the overall network simulation time by the node count. The elapsed per-node simulation times with and without GUI are depicted in Fig. 12. In addition, the time consumed on writing the log data to files is presented separately.

The operations on a single node remain similar regardless of the node count. Hence, the simulation time per a node should be at the same level in all cases. However, as depicted, the time per node increases as the node count increases. This is caused by the increased time consumption in the processing of longer lists in both Telelogic TAU SDL Suite simulation engine and WISENES framework components. The communication with GUI is time consuming but the penalty is independent of the node count. The increase in the simulation time is caused by the environment function polling and the large amount of node information passed through the socket interface.

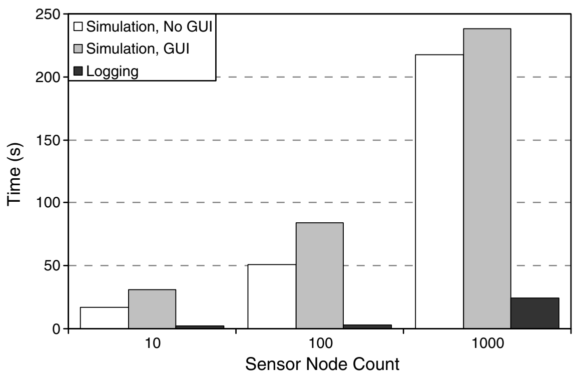


Fig. 12. Time required for the simulation of a single node in WISENES for a 24-h period with varying total number of nodes.

As depicted in Fig. 12, the time consumed on log writing with 1000 node simulations is ten times longer than in the other cases. A reason for this is not accurately known, but it may be due to the fragmentation of the storage disk, because the size of the logged data in this case is over 12 GB.

6.2. Prototype mapping results

For the evaluation of the prototype mapping accuracy, a similar configuration, illustrated in Fig. 13, is constructed for both WISENES and prototypes. Subnodes S1 and S2 perform sensing and send the data to headnode H1 once in every access cycle. Headnode H1 aggregates data and sends them to the sink node through headnode H2. The number of contention slots is four, reservation data slots are limited to eight, and idle network beacons are sent every 250 ms. The access cycle length is 1, 2, 5, and 10 s. Cluster scanning is avoided by using static access cycle timings.

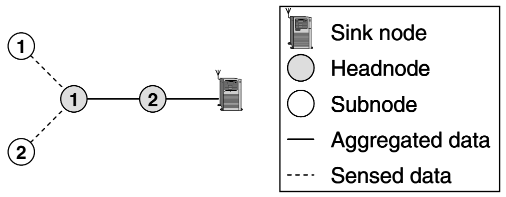


Fig. 13. Static topology for the prototype mapping test case.

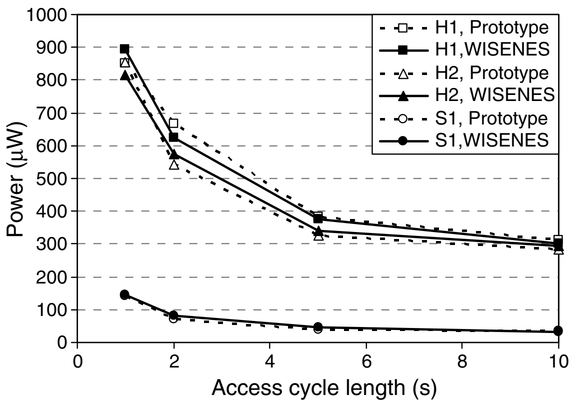


Fig. 14. Modeled WISENES and measured TUTWSN prototype power consumptions in the prototype mapping test case.

Fig. 14 presents the simulated power consumption from WISENES in contrasts to the measured prototype power consumption. Generally, the power consumption in subnodes is minimal compared to that of headnodes. The margin between the two headnodes is not considerable, because the only difference in their activity is one active reservation data slot. When compared to the prototype measurements, the WISENES results are very accurate. The overall average difference is 6.73%. The results are more accurate for headnodes, average difference being 4.0% for H1 and 4.8% for H2. Due to the very low activity, the modeling in subnodes is more inaccurate, as the average difference in case of S1 is 11.34%. The main reason to this is the differences in the timing models of WISENES and the prototypes.

Other aspects related to prototype mapping are delay and throughput. As mentioned, WISENES models delays in the transmission medium and transceiver units accurately. Only the delay of signal propagation is omitted, but it is negligible in short distances when compared to the other delays in transmissions. Moreover, the main causes for the delays on WSNs are higher layer protocol buffering and channel access. Thus, the verification of the delay mapping is omitted.

WSNs do not utilize the full bandwidth available on their transceiver units but only a small fraction of it. The throughput depends on the channel access method and PER. In WISENES, PER is derived from the transceiver unit dependent measurements and the implemented channel access methods are identical. Hence, the further verification of the throughput mapping is also omitted.

6.3. TUTWSN simulation results

A network of thousand nodes is simulated in order to evaluate the applicability of TUTWSN in large scale. The number of contention slots is set to four, reservation data slots to eight, and idle network beacons are sent every 250 ms. The access cycle length is 1, 2, 5, and 10 s. The environmental monitoring application is again activated once in an access cycle.

6.3.1. Power consumption

The average power consumptions for five arbitrary selected headnodes and subnodes are depicted in Fig. 15a and b respectively. The scale in Fig. 15a is approximately eight times larger than in Fig. 15b. The power consumptions of the transceiver unit, peripherals, power unit, and MCU in sleep and active states are presented separately.

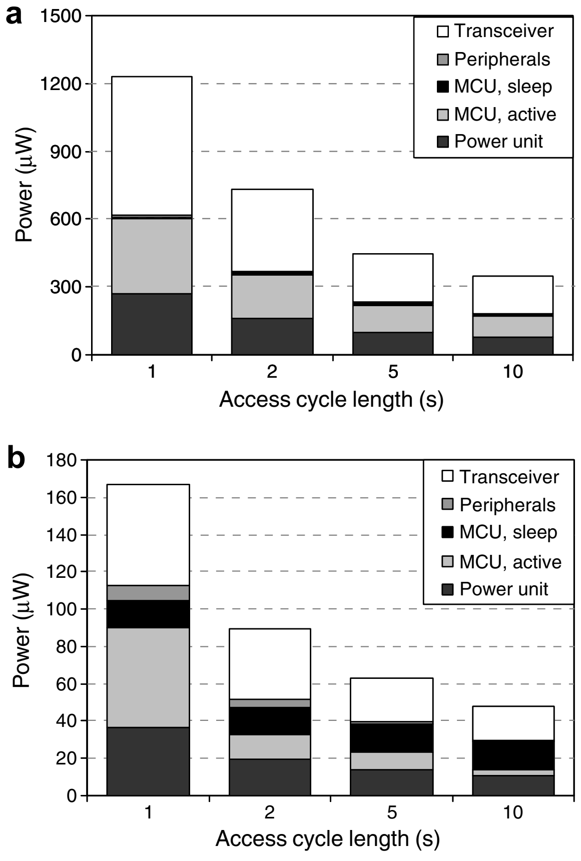


Fig. 15. The power consumptions of different components in TUTWSN: (a) headnode and (b) subnode in a thousand-node network.

As shown, the transceiver unit is the dominating power consumer in both headnodes and subnodes. With shorter access cycles, the share of the transceiver unit in the power consumption is more dominant. The headnodes spend considerably more time with CPU active, due to added processing and active waiting while receiving data. The subnodes spend most of the time in the sleep states.

Compared to the power consumptions in the prototype mapping case, the presented results are averagely 28.7% larger for subnodes and 22.4% larger for headnodes. This is mainly due to the scanning required for the network topology creation and maintenance, which were omitted in the prototype mapping case. Further, headnodes have more active reservation data slots.

6.3.2. TUTWSN lifetime

The lifetimes of TUTWSNs with different access cycle lengths are presented in Fig. 16. The lifetimes are shown for both a case where a node acts as a headnode until it runs out of energy, and a case where a headnode deliberately releases its duty when its remaining energy level is first 50% and then 10%. The network lifetime is considered as the time until 50%r 20% of the nodes are left. In the first case, measurement data with reasonable accuracy can be obtained from WSN, while in the latter case the accuracy suffers but the network is still capable of providing routes to sink node.

The changing of the cluster headnode balances the load between the nodes in the network. This lengthens the time until the first node runs out of energy. Yet, the time between the first and the last node running out of energy is minimal. As shown in Fig. 16, the lifetime of the network until half of the nodes are remaining is considerably longer when the headnode duty is circulated. Instead, with lifetime consideration of 20% of nodes remaining, the network longevity is better if the headnode is not changed.

If a node operates as a subnode continuously, lifetimes are 5.2, 9.9, 21.2, and 34.2 h for 1, 2, 5, and 10 s access cycles respectively. The reason for the short lifetimes is the extremely limited capacity of the capacitor being the energy storage at the nodes. For comparison, with a 1 cm^3 non-rechargeable lithium battery, the obtained lifetimes for a subnode are 96, 182, 391, and 631 days for 1, 2, 5, and 10 s access cycles respectively [48].

6.3.3. Delay and throughput

Fig. 17a depicts the communication delays for different number of hops from a source to the sink node. The delays are measured after the cluster access cycle timings have been adapted and stabilized. For one and two second access cycles, the delays are acceptable. For an environmental moni-

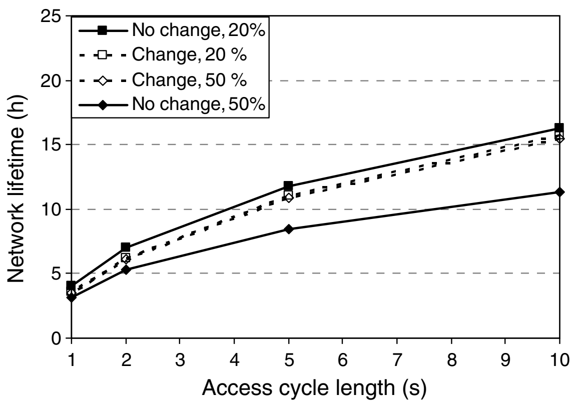


Fig. 16. TUTWSN lifetimes in different conditions.

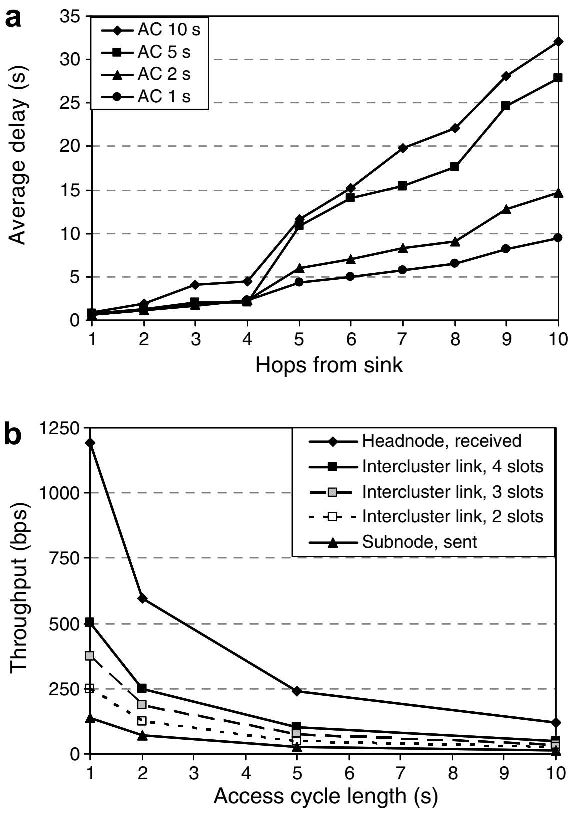


Fig. 17. TUTWSN: (a) packet delays to sink and (b) throughputs.

toring application, the delays are adequate even with the longer access cycle lengths. A per hop delay is quite independent of the distance from the sink node but the nodes in which several data flows converge cause congestion.

Throughputs for a subnode and a headnode, as well as for the active inter-cluster links with different number of reservation data slots are depicted in Fig. 17b. A subnode has one active uplink for the headnode. The headnode throughput is its incoming throughput. The number of the reservation data slots for an inter-cluster link is varied between two and four, depending on the available slots. Obviously, the throughput decreases as the access cycle length increases. This is acceptable, as the access cycle length is typically adapted according to the application requirements.

6.3.4. TUTWSN adaptability

The adaptability of TUTWSN is evaluated by simulating unexpected error situations. An unrecoverable error is simulated at the cluster headnode. The times elapsed until the network is reconfigured are depicted in Fig. 18 for the different access cycle lengths. The subnodes do not start the self-organizing cluster creation algorithm immediately, as a cluster beacon may be lost due to a packet error. Thus, the reconfiguration time depends on the limit of the missed cluster beacon back-off counter. A case, in which the cluster headnode is able to inform about its state, is given as a reference.

As can be seen, the reconfiguration time is almost directly proportional to the access cycle length. The reconfiguration delay is not considerable with the short access cycles, whereas the delay is over half a minute for the longer access cycles.

6.4. ZigBee simulation results

The applicability of ZigBee to WSN applications is evaluated by simulating the ZigBee protocols with the defined environmental monitoring application. In the simulations, the beacon order of ZigBee MAC protocol is varied from 6 to 10, which results to the access cycles of 0.98, 1.97, 3.93, 7.86, and 15.73 s. We refer these to as 1, 2, 4, 8, 16 s access cycles for clarity. The superframe order is set to 2, thus the length of the active period consisting of the beacon and CAP is 61.44 ms.

6.4.1. Small scale ZigBee network

The prototype for ZigBee is modeled from the components, the characteristics of which are measured individually. Therefore, we do not compare the simulated results to physical deployment measurements. For a fair comparison, we use the same statically defined network configuration for ZigBee, which is presented for TUTWSN in Section 6.2. The application in the ZigBee simulation is identical to that of TUTWSN simulations.

The power consumption results from the ZigBee simulations are depicted in Fig. 19. The difference between ZigBee device and coordinator power consumptions is considerably bigger than the same difference between the corresponding TUTWSN nodes. This is caused by the active listening of the complete CAP, which is also the reason for the identical results of both coordinators. Compared to TUTWSN, the power consumption of a device is averagely two times and that of a coordinator averagely 3.5 times larger than the power consumption of the corresponding TUTWSN node.

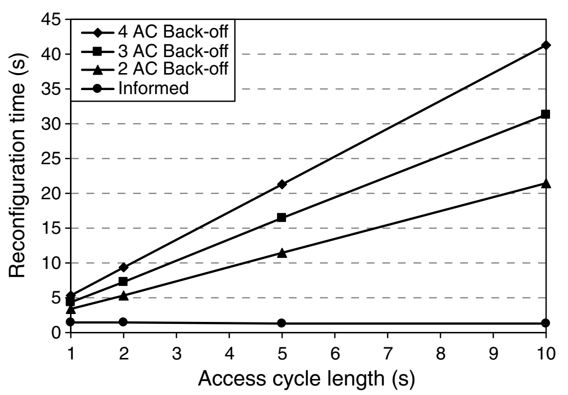


Fig. 18. The reconfiguration times of TUTWSN on an unexpected error situation.

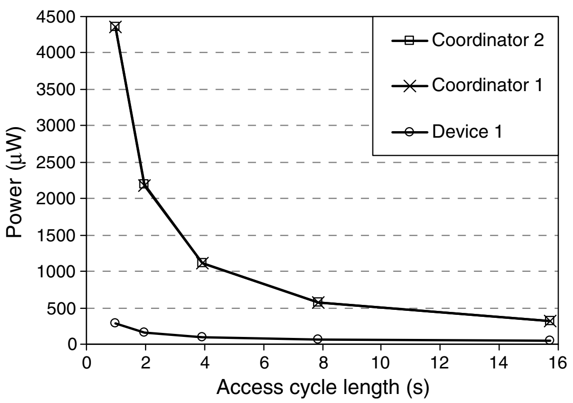


Fig. 19. The power consumption in a small scale ZigBee network.

6.4.2. Large scale network

In the large scale ZigBee simulations, a network that consists of 100 nodes is simulated. The size of the simulated network is restricted by the usage of the same communication channel throughout the network. In dense networks, the distribution of coordinator superframes so that they do not overlap is problematic. In the simulations, 35 coordinator capable devices and 65 reduced function devices are distributed randomly to a 60×60 m area. Due to the congestion, the parameters of the environmental monitoring application are changed so that each coordinator stores the temperatures received from the devices over two access cycles. The results are then aggregated to a single data packet, which is routed towards the ZigBee coordinator.

The power consumptions of the different components in coordinators and devices are depicted in Fig. 20a and b respectively. The access cycle length is varied between 1 and 16 s and the results are averaged over five arbitrarily selected coordinators and devices. The scale in Fig. 20a is 18 times larger than in Fig. 20b. Compared to results presented in Fig. 19, the power consumption of a coordinator is quite similar, whereas in the case devices the power consumption depicted in Fig. 20b is considerable larger for longer access cycles. This is caused by the network scanning, which is required for topology creation and reformation in case of errors. A node scans for a complete access cycle when searching for a network.

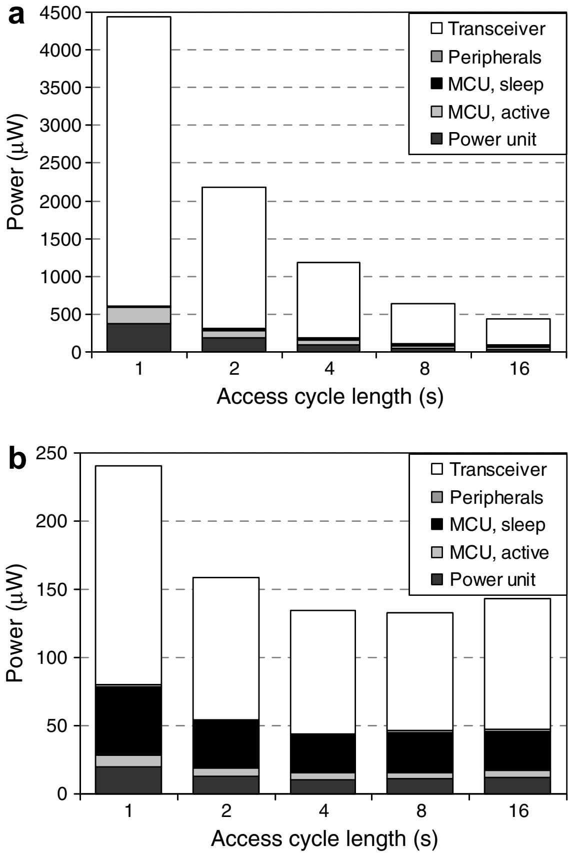


Fig. 20. The power consumptions of different components in ZigBee: (a) coordinator and (b) device in a hundred-node network.

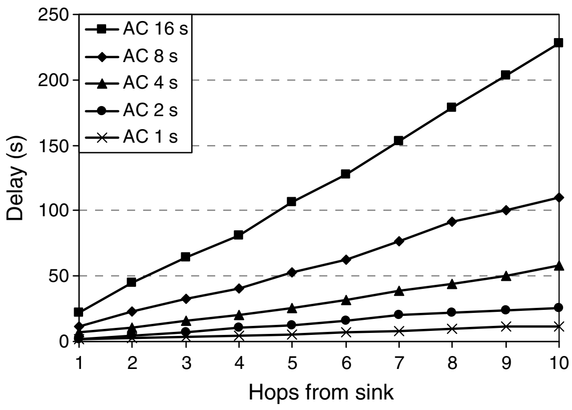


Fig. 21. ZigBee packet delays to the ZigBee coordinator.

In comparison to TUTWSN, the differences are similar to those presented in Section 6.4.1, except for the device vs. subnode power consumption with longer access cycles. While in TUTWSN the scanning times and the energy required for network maintenance diminishes as the access cycle length increases, the effect is opposite in ZigBee. Further, the proportional share of the transceiver unit on the power consumption is significantly larger in ZigBee than in TUTWSN.

The delays in data routing towards the ZigBee coordinator with the different access cycle lengths are depicted in Fig. 21. As in the case of TUTWSN, with 1 and 2 s access cycles the delays are moderate, but with the longer access cycles the delay becomes unacceptable. TUTWSN outperforms ZigBee in this case. The reason for this is that in ZigBee the start times of the access cycles are not delay optimized as in TUTWSN.

7. Conclusions

The large design space of WSNs cannot be managed without a complete tool for the WSN design, configuration, and evaluation. This paper presents

WISENES, which is the first tool that supports the graphical design, simulation, final implementation, and evaluation of WSNs within a single framework. The WISENES framework enables a modular design of WSN protocols and applications. Different platforms and protocols are evaluated in order to obtain an optimal configuration for a specific application. The back-annotation of the measured performance information from physical node platforms improves the accuracy of the simulation results.

The implementation of TUTWSN and ZigBee networks shows that the design of protocols and their performance evaluation in WISENES is fast. The graphical state machine based notation is explicit and designer friendly. The TUTWSN and ZigBee network simulations prove the applicability and performance of WISENES for the simulations of large networks. Further, the simulated performance results correspond to those of real physical platforms.

Our main future work is projected on the development of a more accurate sensing channel model and mobility support. At the moment, the WISENES framework is not publicly available, but we are planning to open it as an online web-based WSN design service. We are also considering open source SDL tools for the design.

References

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

[2] J.A. Stankovic, T.F. Abdelzaher, L. Chengyang, S. Lui, J.C. Hou, Real-time communication and coordination in embedded sensor networks, Proceedings of the IEEE 91 (2003) 1002–1022.

[3] SDL Forum Society Homepage, http://www.sdl-forum.org/, visited in November 2005.

[4] M. Kohvakka, M. Hännikäinen, T.D. Hämäläinen, Ultra low energy wireless temperature sensor network implementation, in Proc. 16th annual IEEE Int. Symp. on Personal Indoor and Mobile Radio Communications, 2005, pp. 187–200.

[5] ZigBee Specification, http://www.zigbee.org/, visited in October 2005.

[6] The Network Simulator – ns-2, http://www.isi.edu/nsnam/ns/, visited in November 2005.

[7] X. Zeng, R. Bagrodia, M. Gerla, GloMoSim: a library for parallel simulation of large-scale wireless networks, in: Proc. 12th Workshop on Parallel and Distributed Simulations, 1998, pp. 154–161.

[8] Qualnet Network Simulator, http://www.qualnet.com/, visited in November 2005.

[9] OPNET Modeler,http://www.opnet.com/products/modeler/home.html, visited in November 2005.

[10] OMNeT++ Community Site, http://www.omnetpp.org/index.php, visited in November 2005.

[11] Scalable Simulation Framework, http://www.ssfnet.org/, visited in November 2005.

[12] J-Sim Homepage, http://www.j-sim.org/, visited in November 2005.

[13] S. Park, A. Savvides, M.B. Srivastava, Simulating networks of wireless sensors, Proc. Winter Simulation Conference 2001 (2001) 1330–1338.

[14] I. Downard, Simulating sensor networks in ns-2, http://pf.itd.nrl.navy.mil/nrlsensorsim/, visited in November 2005.

[15] sQualnet: A Scalable Simulation Framework for Sensor Networks website, http://nesl.ee.ucla.edu/projects/squalnet/, visited in November 2005.

[16] J. Liu, L.F. Perrone, D.M. Nicol, M. Liljenstam, Simulation modeling of large-scale ad-hoc sensor networks, in: Proc. 2001 Simulation Interoperability Workshop, (2001).

[17] C. Mallanda, A. Suri, V. Kunchakarra, S.S. Iyengar, R. Kannan, A. Durresi, S. Sastry, Simulating wireless sensor networks with omnet++, http://csc.lsu.edu/sensor_web/simulator.html, visited in November 2005.

[18] S. Dulman, P. Havinga, A simulation template for wireless sensor networks, IEEE Int. Symp. on Autonomous Decentralized Systems, 2003, fast abstract.

[19] A. Sobeih, W.-P. Chen, J.C. Hou, L.-C. Kung, N. Li, H. Lim, H.-Y. Tyan, H. Zhang, J-Sim: A Simulation and Emulation Environment for Wireless Sensor Networks, http://www.j-sim.org/, visited in November 2005.

[20] P. Baldwin, S. Kohli, E.A. Lee, X. Liu, Y. Zhao Modeling of sensor nets in ptolemy II, in: Proc. 3rd Int. Symp. on Information Processing in Sensor Networks, 2004, pp. 359–368.

[21] Ptolemy II, http://ptolemy.eecs.berkeley.edu/ptolemyII/, visited in November 2005.

[22] G. Simon, P. Völgyesi, M. Maróti, Á. Lédeczi, Simulation-based optimization of communication protocols for large-scale wireless sensor networks, Proc. IEEE 2003 Aerospace Conference 3 (2003) 1339–1346.

[23] B.C. Mochocki, G.R. Madey, H-MAS: a heterogeneous, mobile, ad-hoc sensor network simulation environment, in: Proc. 7th Annual Swarm Users/Researchers Conference, 2003.

[24] G. Chen, J. Branch, M.J. Pflug, L. Zhu, B. Szymanski, SENSE: A sensor network simulator, http://www.cs.rpi.edu/~cheng3/sense/, visited in October 2005.

[25] A. Boulis, C.C. Han, M.B. Srivastava, Design and implementation of a framework for efficient and programmable sensor networks, in: Proc. 1st Int. Conf. on Mobile Systems, Applications, and Services, 2003, pp. 801–805.

[26] Crossbow Technology, http://www.xbow.com/, visited in November 2005.

[27] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, System architecture directions for networked sensors, Proc. 9th ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems (2000) 94–103.

[28] P. Levis, N. Lee, M. Welsh, D. Culler, TOSSIM: accurate and scalable simulation of entire TinyOS applications, Proc. 1st ACM Conf. on Embedded Networked Sensor Systems (2003) 126–137.

[29] M. Karir, J. Polley, D. Blazakis, J. McGee, D. Rusk, J.S. Baras, ATEMU: a fine-grained sensor network simulator, in

Proc. 1st IEEE Int. Conf. on Sensor and Ad Hoc Communication Networks, 2004.

[30] L.F. Perrone, D.M. Nicol, A scalable simulator for TinyOS applications, Proc. Winter Simulation Conference 2002 (2002) 679–687.

[31] S. Sundresh, K. Wooyoung, A. Gul, SENS: a sensor, environment and network simulator, in: Proc. 37th Annual Simulation Symposium, 2004, pp. 221–228.

[32] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, D. Estrin, Em*: a software environment for developing and deploying wireless sensor networks, CENS Technical Report 0034, 2003. http://research.cens.ucla.edu/, visited in November 2005.

[33] C. Kelly IV, V. Ekanayake, R. Manohar, SNAP: a sensor network asynchronous processor, Proc. 9th Int. Symp. on Asynchronous Circuits and Systems (2003) 24–35.

[34] M. Kuorilehto, M. Kohvakka, M. Hännikäinen, T.D. Hämäläinen, High level design and implementation framework for wireless sensor networks, in: Proc. Embedded Computer Systems: Architectures, Modeling, and Simulation, 2005, pp. 384–393.

[35] Evaluation of J-Sim, http://www.j-sim.org/comparison.html, visited in November 2005.

[36] VisualSense Homepage, http://ptolemy.eecs.berkeley.edu/visualsense/, visited in November 2005.

[37] Getting Started with the EmStar Simulator, http://cvs.cens.ucla.edu/emstar/tut/emsim.html, visited in November 2005.

[38] W. Stallings, Data and Computer Communications, sixth ed., Prentice-Hall, 2001.

[39] The swing tutorial, http://java.sun.com/docs/books/tutorial/uiswing/, visited in August 2005.

[40] Telelogic Homepage, http://www.telelogic.com/corp/, visited in September 2005.

[41] Telelogic TAU SDL Suite, http://www.telelogic.com/corp/products/tau/sdl/index.cfm, visited in September 2005.

[42] Specification and description language (SDL), ITU-T Recommendation Z.100, http://www.itu.int/ITU-T/studygroups/com17/languages/, visited in June 2006.

[43] XE88LC02 Sensing Machine, Data Sheet, http://www.xemics.com/, visited in September 2005.

[44] Nordic VLSI nRF2401, Data Sheet, ver 1.0, http://www.nvlsi.no/, visited in September 2004.

[45] Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), IEEE Standard 802.15.4, 2003.

[46] PIC18 Data Sheet, http://www.microchip.com/, visited in October 2005.

[47] Chipcon CC2420 Data Sheet, http://www.chipcon.com/, visited in October 2005.

[48] S. Roundy, P.K. Wright, J. Rabaey, A study of low level vibrations as a power source for wireless sensor nodes, Elsevier Computer Communications 26 (11) (2003) 1131–1144.

Mauri Kuorilehto (M.Sc.’01, TUT) received the M.Sc. in Computer Science from Tampere University of Technology (TUT), Finland. He is currently pursuing his Ph.D. in the Institute of Digital and Computer Systems at TUT. His research interests include network simulation, operating systems, and application distribution in wireless sensor and ad hoc networks.

Marko Hännikäinen (M.Sc.’98, Ph.D.’02, TUT) acted as a research scientist and a project supervisor at the Institute of Digital and Computer Systems at TUT in 1998–2007, and was nominated to professor in 2007. He co-directs the DACI research group, concentrating on wireless sensor networks, high abstraction design tools, and novel web applications.

Timo D. Hämäläinen (M.Sc. ’93, Ph.D.’97, TUT) acted as a senior research scientist and project manager at TUT in 1997–2001. He was nominated to 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.