Service-Oriented Design Methodology for Wireless Sensor Networks: A View through Case Studies

Elena Meshkova, Janne Riihijärvi, Frank Oldewurtel, Christine Jardak and Petri Mähönen
Department of Wireless Networks, RWTH Aachen University
Kackertstrasse 9, D-52072 Aachen, Germany
Email: {eme, jar, fol, cja, pma}@mobnets.rwth-aachen.de

***Abstract*— In this paper we discuss the design methodology based on the service-oriented architecture and agile development principles for wireless embedded and sensor networks (WSNs). This methodology suits particularly well for streamlining and partially automating the design and implementation of complex WSNs. We report results from selected case-studies to test applicability of service-oriented architectures for embedded software. We evaluate the proposed design methodology by studying cases that include the development of three different services for wireless sensor networks that can work together as a part of a complete solution. We specifically comment on the trade-offs that a developer might face while designing and implementing systems. We follow the "best-practices" of the software design methodology and adapt them to the development of both the sensor network services and the sensor networks themselves. The design and implementation cycle includes three stages: the overall solution and architecture design, the protocol and application design, and finally, the implementation. These stages are iterated throughout the life-time of the project. During the design we consider both the abstract definition of user requirements and targeted functionality, and their mapping to the real hardware and software.**

I. Introduction

Wireless sensor networks (WSNs) lie at the crossroads of software, network and embedded engineering. This field requires developers knowledgeable in all of these areas, so that the maximal benefit can be gained from the well-established domains. Otherwise WSNs will inevitably suffer from the danger of re-inventing the things that are already well-known and widely used in the other areas. However, most of the "best practices" can not be directly re-used in WSNs and have to be adapted accordingly to the specifics of this domain. In this paper we suggest a design methodology for wireless sensor networks that is based on the concepts of *service-oriented architecture* (SOA), *agile development methodology* and "best practices" of the network development in the WSN, MANET and peer-to-peer areas. The use of this methodology is illustrated on several use-cases. Additionally, we suggest a framework for partial automation of WSNs that is based on the suggested methodology and is using, among others, Semantic Web instruments.

Originally the term *service-oriented architecture* refers to a logical set that consists of several large software components that together perform a certain task or *service* [1], [2]. SOA is a particularly popular paradigm in the community of the web software developers, for example Web Services [1] utilize this architecture. WSNs can be viewed in part as a reduced copy of Internet, where different nodes or their groups provide different services to the end user. Ideally nodes in WSNs self-organize to provide the required functionality. Web services aim to achieve exactly the same goal on the Internet scale.

Therefore technologies and concepts popular for the Internet can be tried out in the WSN domain. The SOA is one of them. However, this architectural principle can not be used directly. It has to be adapted to additionally include not only large complex services, but also simple ones, like data storage, routing or sensor readings. To a certain extent the terms *component-based* and *service-based* architectures can be used interchangeably in WSNs. One can distinguish between these by defining that the first term refers more to the actual implementation while the second term is used for logical constructions.

WSNs have a large range of applications starting from personal area (PAN) and home environment networks to habitat monitoring (see Table I). These highly diverse scenarios impose different requirements on WSNs and lead to distinct design and implementational decisions. Wireless embedded devices further possess limited energy, memory and processing resources[1]. The major limitation for WSNs is power. In many scenarios sensor nodes are deployed without constant power supply and have to operate on one set of batteries for a considerable amount of time. Therefore, all the factors that contribute to the power usage have to be minimized. Among them are communication over the network, reading/writing to/from the EEPROM, usage of power-hungry peripherals and excessive computations [3]. This leads to *the unique protocol stack and application design for each particular WSN usage scenario*.

Usually WSNs have to be also optimized against a number of other parameters besides the network lifetime. Among them are the maximum allowable data propagation time, the fault-resilience requirements, the overall network cost, the complexity and flexibility of the system, code maintainability, re-usability and comprehensiveness. Depending on the application scenario different parameters of WSN have to be considered (see Table II). Improvement of one parameter often leads to downgrading of others. The balancing of the

[1]For example Berkeley TelosB sensor nodes, or motes, on which we implement our designs have only 48 KB of ROM, 10 KB of RAM, a 16-bit microcontroller and, by default, are powered by two 1.5 Volt batteries.

978-0-7695-3158-8/08 $25.00 © 2008 IEEE
DOI 10.1109/SUTC.2008.43

TABLE I
SOME OF THE APPLICATION SCENARIOS FOR WSNS.

Scenario	Scale	External networks	Mobility	Fault-tolerance	Required lifetime	QoS
PAN	Small	Often	Minimal	None	Medium	Quick
Home network	Small	Always	Minimal	Minimal	Medium to long	Quick
Structure montoring	Medium/ Large	Some-time	Minimal	None	Medium to long	Quick to medium
Disaster monitoring	Large	Often	Minimal	Some	Short to long	Quick
Habitat monitoring	Large	Randomly	Minimal to Large	Some	Long	Medium

TABLE II
COMMON WSN PARAMETERS.

Main	Network	Service	Hardware	Software
Cost	Mobility	Datarate	Platform: Chip, Memory	Service
Lifetime	Fault rate	Fuctionality	Platform: Radio, Interface	OS / VM
QoS: Max. extra delay	Topology: Size, Diameter	QoS	Power	RAM + ROM
QoS: Fault-resilence	Topology: Node Distribution, Area		Sensors	Hardware
Services	Network graph			

major WSNs parameters is not a trivial task and leads to the iterative design process. This is one of the peculiarities of the WSNs development, along with the fact that the development teams are usually small and highly qualified. The requirements of the project can also change from initial ones during the project lifetime. These makes the use of the *agile design technologies* [4] appropriate for the WSNs development. This class of methodologies is primary characterized by the short and iterative development cycle, highly motivated and qualified team, and a constant face-to-face interaction with the customer, as well as between the team members. The use of this methodology allows easy adaptation of the project to the changing circumstances and user requirements.

We split our further discussion into five sections. In Section II we briefly describe the major characteristics of the WSN design and discuss on the applicability of SOA and agile technique for these systems. In Section III we present the basic parameters that can be used to characterize WSNs. The overall design process, its challenges and iterations are given in Section IV. In Section V we provide three case studies in the areas of service discovery, in-network processing and fault-resilience to ground our theoretical discussion. The possibilities for the partial automation of the WSN design, the corresponding challenges and gains are discussed in Section VI. Finally, the conclusions are drawn in Section VII.

II. CHARACTERISTICS OF WSN DESIGN. SOA AND AGILE METHODOLOGY APPLICABILITY.

Judging from both our experience, discussions with the colleagues and the research papers we have produced a list of characteristics typical for WSN design, some of which are coming from the software development domain:

- *Minimalism* is essential as a simple and well-defined system usually works the best.
- The requirements to a WSN project can *change* during the development of the project. The refinement of the requirements can be initiated by both users and developers.
- *Precise* solutions are preferable to generalized solutions in WSNs as they allow to optimize the performance of the system which is important in the resource-constraint environment.
- The development of the WSN is an *iterative* process.
- *Rapid prototyping* is essential for wireless embedded networks.
- *Re-use* of the already developed components is desirable, which requires in turn good documentation, careful planning and identification of common parts in the project.
- Before development of the new system the *deficiencies of the old similar ones* are to be identified, as well as the reusable parts.
- The design plans for the WSN include the hardware, network topology, operation system, programming and security issues.
- WSN solutions are widely developed using *prototypes*. The results of the prototyping, as well as following deployment and testing should exhaustively evaluated and recorded to enable the experience sharing and component re-use.
- WSN systems are typically developed by a *small number* (up to 10-15) of *highly qualified developers* and researches. The team members are typically working in a close collaboration and can contribute to each other's work. Therefore the project development steps have to be logical and obvious to each member of the development team in order to increase the efficiency of the work and inspire team-work.

Most of the features of this list map well to the agile software design methodology [4] and therefore it can be chosen as one of the basis on which our methodology can reside. Service-oriented architecture also suits particularly well for WSNs as the development of the whole network can directly be mapped to service, simple or complex. For example, the network itself provides several composite high-level services as area monitoring or manipulation of actuators. Each node also offers complex services like data forwarding or sensor readings. Each node can also be represented as a collection of services that interact with each other. The model-driven architecture (MDA) [5] also seems to be a promising candidate to be used for WSN systems, as there the process of the system development is very well defined and recommended design tools are specified. However, this architecture is more structured and heavy than the SOA and agile methodology combination, and we feel that it need to be adapted to be used for the typically lightweight and rapidly changing WSN projects. We hope to reuse or adapt some of the core MDA features, like meta-data component specification, in our future research on design in WSNs.

We propose an abstract service model for an individual node (Figure 1) that incorporates the following modules: *Application, Service Discovery, Transport, Routing, MAC, Physical,*

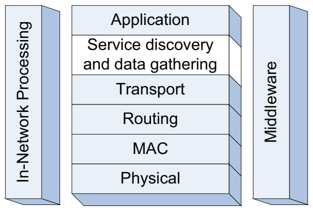


Fig. 1. Basic WSN node structure

In-network processing and Middleware service classes. Each of the above modules can either contain one simple service like a routing protocol or be composed of several complex ones. For example, the in-network processing block can include data aggregation and data recovery modules. The service discovery module might not be required on the mote, if its functionality is already incorporated into the middleware component. The service-oriented architecture not only suits well for wireless-sensor networks, it also maps well to the agile design methodology too, as it allows to realize the iterative principle of the agile design with regular deliveries of small portions of working code, namely individual services.

III. WSN Parameters and Trade-offs

Wireless sensor networks are relatively small systems that have a limited amount of expected functionality, namely gathering of the senor reading and enabling actuators in response to sensor readings. WSNs can also be used as a transition network to propagate third-party data. These networks are rarely directly connected to the end-user. More often they are parts of a larger heterogeneous distributed system, which are the "users" for the WSNs. All of the above leads to the possibility to *formalize* the requirements to these systems, as they, though widely diverse, are used for a limited number of purposes. Additionally, these requirements usually come from the professionals who are generally more precise in formulating their needs than the normal end-users. The formalization of the WSN description does not only allow an easier and therefore faster development of WSN systems, but also enables a partial automation of this process.

We suggest a basic set of parameters that need to be specified virtually for any wireless sensor network (Table II). The parameters can be divided into several groups. *Main parameters* characterize overall system performance. These are the most important parameters for the user. *Network parameters* describe the wireless network and behavior of the nodes in it. *Service parameters* define the behavior of each realized service as well as its inputs and outputs. *Hardware and software parameters* describe the devices that can be used to create a WSN; they are used during the implementation stage of the project.

A. Main Parameters

We have identified five basic parameters which are used for the overall WSN description. They are *cost, lifetime, delay, fault-resilience* and *services*. The *overall cost* of the network consists of hardware, development and deployment costs. The deployment costs grow larger when the individual placement of specific nodes is required. It is much cheaper to deploy a homogeneous network that self-organizes depending on the surrounding conditions. However, this will in turn increase the development costs. The decision between the static and the dynamic network configuration is also an important part of an architectural design. From one side a careful study of the target scenario, extensive design efforts and test implementations allow the creation of a static WSN network that requires minimal interference after the deployment. The development of a dynamic network is a more difficult task, but ideally it has to be performed only once. After that the same design can be employed for different scenarios and later the network can self-adjust depending on the surrounding. The examples of *dynamic network configuration* mechanisms that enable network self-organization are virtual machines, middleware frameworks and code updates from EEPROM [6].

The *network lifetime* is primary determined by power consumption of the individual nodes which are mostly battery powered and therefore have a limited lifetime. The network lifetime can be increased, for example, by putting a part of computational burden to the gateway or user devices. Care has to be taken to ensure that the gain from the processing of information on a gateway is higher than the communicational costs of transporting the data to this device. Typically this approach is applicable for small diameter networks.

Maximum additional delay and fault-resilience are two main quality-of-service (QoS) parameters of wireless sensor networks. *Maximum additional delay* specifies the maximum allowable propagation delay between the gateway and the farthest node from it. *Fault-resilience* of the network, i.e. tolerance of the WSN to node failures, outdated and imprecise information, as well as data losses and injection of malicious data, can be improved by employing special fault-resilience mechanisms, such as data back-ups or security add-ons. This approach leads to additional deployment costs and can reduce the network lifetime. The introduction of back-up nodes in the network leads to the increase in fault-resilience of the network and at the same boosts up the hardware and deployment costs. Finally, the *services* parameter contains a list of functionalities that the user expects from the network.

B. Network Parameters

Network parameters allow to describe the sensor network where the application is to be deployed. These parameters include information on the expected mobility, fault rate, minimal required bandwidth, number of nodes in the network, average network diameter, information on the network symmetry and heterogeneity. Some of these parameters, like expected bandwidth are direct user inputs. Some other are generated and adapted during the project deployment and directly influence the main parameters that have to be optimized. The user is free to fix some of them and leave the rest to the developers. For example, the user might want to fix the positions of some nodes and therefore indirectly influence the network size, diameter and heterogeneity parameters. In general the network is modeled and described by the graph. However, if we want to partially automate the WSN generation process first we have to estimate the generalized parameters of the network, like network symmetry, maximum diameter and density. Later, when we will have a limited pool of options, more detailed graph level representation of the network can be considered and evaluated. However, if the WSN design is done manually these processes are typically merged.

C. Service Parameters

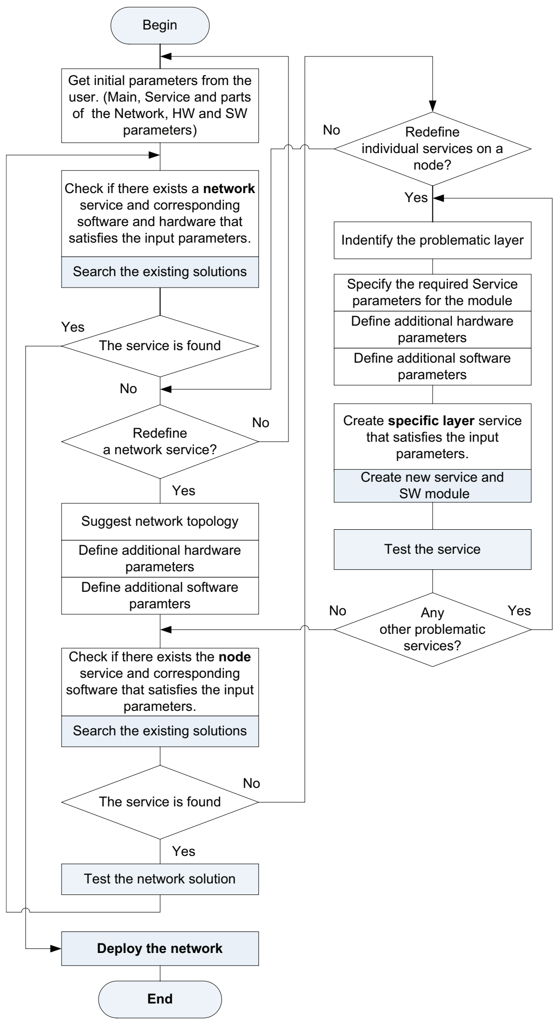
An abstract description of *services* is a key element of the SOA-based WSN system design, as it allows to create non-implementation specific design solutions important in the initial project development stages. Additionally, abstract service descriptions help to formalize the developed WSN, which is important if we want to create heterogeneous systems and automate the wireless sensor network design process. A service description includes the list of *provided functionalities* and *required services*. For example, the data collection service requires an underlying data delivery mechanism and provides the functionality of data gathering. Each service is also characterized by the expected influence on the node *lifetime* and *overhead data rate*, i.e. the amount of data per time unit generated extra depending on the given data rate from the services that utilize this service. For composite distributed services QoS parameters are also important.

D. Software and Hardware Parameters

The *software and hardware parameters* become important in this development stage. The hardware parameters include hardware platform specification (memory, chip, interfaces, radio, power resources) and sensor platform descriptions. The software parameters include the required OS/Virtual machine, needed amount of memory (RAM/ROM), a list of modules that the considered software module requires to operate and requirements to the hardware, like a radio or a specific sensor support.

IV. Design Process

As we have proposed earlier, the WSN design process should be agile, however it should have some structure. We have chosen to loosely follow the standard waterfall model [7]. The waterfall model includes eight stages: *gathering of the requirements, their analysis, the design of the solution, development of the software architecture, development of the code, testing, deployment and post implementation*. For wireless sensor networks most of these stage are also valid. However, the *solution design stage* has to be altered to include the network design parameters and the following choice the network architecture. The development of the software architecture stage in WSNs is changed to the *design of the individual network protocol stacks and applications* that provide the services desired by the user. The stage of the *code development* deals with the mapping of the abstract



Begin
Get initial parameters from the user. (Main, Service and parts of the Network, HW and SW parameters)
Check if there exists a **network** service and corresponding software and hardware that satisfies the input parameters.
Search the existing solutions
Yes
The service is found
No
Redefine a network service?
No
Yes
Suggest network topology
Define additional hardware parameters
Define additional software paramters
Check if there exists the **node** service and corresponding software that satisfies the input parameters.
Search the existing solutions
The service is found
No
Yes
Test the network solution
Deploy the network
End
No
Redefine individual services on a node?
Yes
Indentify the problematic layer
Specify the required Service parameters for the module
Define additional hardware parameters
Define additional software parameters
Create **specific layer** service that satisfies the input parameters.
Create new service and SW module
Test the service
No
Any other problematic services?
Yes

Fig. 2. Generalized WSN design process.

parameters and concepts derived in the previous two steps into the implementation on the real devices. The flowchart corresponding to these three design steps in terms of SOA is given in Figure 2.

After gathering and analysis of the user requirements at first the existing solutions can be checked for the re-use, i.e. if the existing solutions fit in the user constrains and while providing the desirable services. If no appropriate solution is found a new one has to be created.

A. Design of the Solution

In the solution design stage the estimation needs to be made if the parameter constrains specified by the user can be achieved in principle and what corresponding network architecture should be used. The choice of the network architecture is based on the expected network topology and the behavior of the nodes (mobility, fault ratio). Additionally, judging from the user requirements, the priority between the main parameters should be estimated, as in WSNs we always face the trade-off between *cost vs. network lifetime vs. QoS vs. functionality of services*. For example, the increase in the network lifetime can be achieved by attaching a larger battery to the sensor node, i.e. increasing its cost or decreasing QoS or functionality constraints and therefore allowing inaccurate or out-of-date data to be reported.

For *small-scale* networks the cost and fault-resilience often come in the first place. The health-monitoring system is one example. Here the network life-time is not usually the major issue and in PANs it is relatively easy to recharge the devices. However, the fault of a single device might lead to heavy consequences as the change in the patient's state can be missed. Cost is an issue in non-critical applications such as smart home and sport training scenarios. The *plain architecture*, where all the nodes have the same basic functionality, is well suited to such scenarios. Additionally, small scale networks can benefit the most from the back-end devices that connect WSNs to external networks. The small network diameter allows these networks to put computational load on the back-end devices, such as gateways, at a low communication cost.

For *medium and large networks* more complex role specific architectures suit the best, as it was already demonstrated by the Internet community [8]. By *role-specific architecture* we mean any network configuration where nodes play different roles. In this category, among others, fall the *client-server* and *cluster-based* architectures. The nodes in a WSN generally perform either the *basic* role, i.e. they gather sensor data, or the *processing* role. The processing role can include a variety of functionalities like data aggregation, data back up, cluster-head operation and other *in-network processing* activities. The use of roles for large networks can significantly boost the network lifetime keeping the overall cost of the network relatively low. However, introducing a hierarchy into the network can lead to longer network response times or affect the network stability, as the failure of a cluster-head can cause the whole surrounding network to malfunction. A hierarchical architecture is also not suitable for application scenarios with medium or high mobility due to the increased cost of maintaining the hierarchy.

B. Protocol Stack and Application Design

After the general decisions have been made concerning the solution architecture, protocols and applications residing on the individual nodes are to be designed. Stack formulation is as well as the system design in general, is an iterative process, first we define the top services, than go to the lower level services. The shared, cross layer-services are defined as they are required by the hierarchical services, or at the end to adjust the main parameters. For example, distributed coding saves some energy at the expense of additional complexity, i.e. development cost. Additionally, if we want to further optimize the network performance by assigning different roles to nodes, like a data aggregator or a cluster head, we should either assign these roles statically for which a detailed knowledge of a network topology is desirable or use a dynamic role assignment mechanism, for example the one proposed in [9].

The major trade-off at this design stage is *simplicity vs. efficiency vs. flexibility*. By *simplicity* we understand the architectural simplicity of an application, as well as the simplicity of the protocol message flow and on-node processing. *Efficiency* refers to ability of the specific application or protocol to fulfill the goal. *Flexibility* reflects the re-usability of the design for other projects and also to the ability of the WSN to adapt to the changing environment.

The network lifetime can be improved by reducing the network traffic, which requires the increase of the network protocol efficiency. Techniques that help to reduce the network traffic include, but are not limited to *caching, piggybacking* and *in-network processing*. The use of each of these techniques involves different trade-offs. Caching is used to both increase the fault-resilience of the network and decrease the network traffic. Caching is employed, for example, in routing and service discovery. When applying caching one should maintain a careful balance between the amount of messages injected into the network used to update the cache and the age of the cache, i.e. the probability to have stale information there. This balance is application specific as, for example, in fairly static networks routing information can be updated much more rarely than in highly mobile networks. *Piggybacking* allows to attach additional data to bypassing network packets, thus potentially decreasing the number of transmitted packets. The trade-off here is between the increase in the packet size and, therefore the decrease in the probability of a packet delivery, and the number of messages generated. It is well known that the larger the packet transmitted over wireless network is the more chances that it will be corrupted. However, it is cheaper in terms of power to transmit the data in a large packet rather than in several small packets [10]. This is strictly true only if we do not consider possible retransmission.

The messaging trade-off involves two different data propagation techniques: *reactive* vs. *proactive*. If the data is propagated *reactively* through the network it is sent after the user request. The *proactive* data propagation is initiated by the data accumulating device, in form of an advertisement or an event notification. Proactive data transmissions are used, for example, in case of emergency notifications. They are also used as supporting messaging in, for example, routing protocols needed to obtain the addresses of neighboring nodes. Reactive data propagation is typically cheaper in terms messaging as packets are generated only when it is necessary.

Most of the supporting functionality of WSNs can be defined as *in-network processing* that was already briefly discussed in the previous section. The classic examples of in-network processing are data aggregation, cluster-head functionality, data storage and compression. In-network processing can be activated on some or all of the network nodes. This allows to save the individual power and computational resources and sometimes leads to better routing. This functionality can therefore increase the network lifetime, the efficiency of the network, and its flexibility. However, the mismanaging of in-

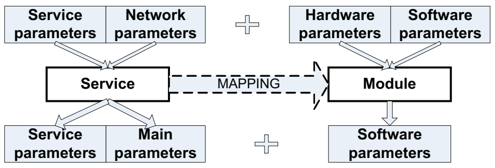


Fig. 3. Mapping services to components.

TABLE III

PERFORMANCE CHARACTERISTICS FOR DISTRIBUTED SOURCE CODING.

Component	RAM [bytes]	ROM [bytes]	Energy, R=0.36 [µJ]	Energy, R=0.72 [µJ]
Encoding	330	34	0.82	0.53
Decoding	562	558	27.61	4.35
Tracking	620	26	237.57	237.57

network processing might lead to the faster power-losses by the hosting nodes and cause additional messaging.

C. Implementation Considerations

In the implementation phase of WSN development the abstract description of services have to be mapped into the real code running under a certain operating system on the specific device (Figure 3). The developer has to find a compromise between *complexity*, *comprehensiveness* and *efficiency* of the code[2]. The complexity of the code leads to poor code reusability and maintainability. There is not much that can be done about complexity. Certain technologies can contribute to the code comprehensiveness. These are the *component-base* approach, the use of suitable programming *abstractions* and the clear *naming* of software primitives [11].

The size of the memory footprint of the code and the processing burden contribute heavily to the efficiency of a WSN application. One more trade-off worth mentioning in this section is the trade-off between implementation complexity and protocol efficiency. There exist several possibilities to make the network communication more efficient at the cost of additional on-node processing. *Data compression and encoding* as well as *cross-layer optimization* are such methods.

V. CASE STUDIES

In this section we provide several case studies based on our own research experience that show the applicability of the proposed design methodology for development of a complete WSN system, as well as individual services. With these examples we aim at highlighting the complex tradeoffs one has to consider in the overall systems design. Only with a flexible design methodology such as outlined above can all these tradeoffs and their impact on the overall application performance be estimated accurately.

A. Distributed Source Coding as In-Network Processing

Distribute Source Coding belongs to the class of the in-network processing service. It is one of the techniques that allows to extend the network lifetime of WSNs, especially in the case of high spatial node densities. This high density induces spatial correlations between the measurements of individual sensor nodes. A direct consequence of these correlations is that the sensor readings between neighboring sensor nodes are highly redundant. The Distributed Source Coding approach seeks to exploit the spatial correlations and is fundamentally based on the Slepian-Wolf theorem [12]. The basic idea behind it is the compression of multiple correlated sensor readings from sensor nodes in vicinity. These sensor nodes do not communicate with each other and send their compressed readings to a central sensor node which performs the joint decoding [13], [14]. Distributed Source Coding has two operation modes in time-varying environments: the entropy tracking phase and the compression phase. In the tracking phase the conditional entropy of the sensor readings is periodically estimated and then used for determining the code rate. In the compression phase, whereas the source nodes compress their readings at the assigned rates, we save energy and capacity due to reduced packet sizes. Based on the results obtained from our experiments, the energy consumed for the transmission of 1 bit is about $E = 180\,\text{nJ}$ and the energy and memory consumed due to our Distributed Source Coding scheme is listed in Table III. Depending on the underlying correlation structure significant net energy savings can be achieved. The low memory consumption of the corresponding software components makes this approach also applicable for resource-poor environments.

The prolonged network lifetime is achieved at the cost of the increase in the systems complexity (as the nodes are required to play specific roles) and by moving the computational complexity and load from source nodes toward gateways. Very low additional deployment costs are expected since the software components can easily be loaded to hardware platforms, especially in case of a code propagation mechanism being used. There is no need to later on change the software components in case the set of suitable codes is identified during the application design phase. Finally, the system can run unattended since the entropy tracking algorithm analyses the environment in terms of varying correlations and thus Distributed Source Coding can not be observed from the user perspective.

B. DISC as a Mechanism to Increase Fault-Resilience of a WSN

DISC [15] is a distributed data storage and collection mechanism for WSNs which protects the information from malicious destruction of parts of the network and therefore increases the fault-resilience of the system. The system is designed for medium and large-scale networks and requires a hierarchical architecture from the WSN. Nodes are arbitrarily located in a monitored area divided into identified regions by the mean of existing ID configuration protocol. In each region a node is dynamically elected as a cluster head using

[2]In this paper we consider the TinyOS operating system for our experiments, though most of the conclusions drawn are OS-independent.

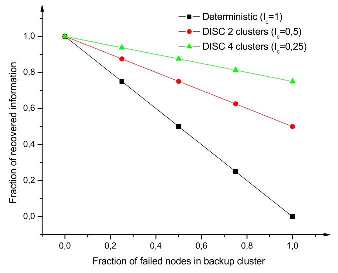


Fig. 4. The fraction of recovered information per cluster in case of malicious destruction of nodes.

a low-energy cluster formation algorithm such as PANEL [16]. The network life-time is divided into the *time periods*, epochs, and the data collected in one cluster is stored in the nodes of a randomly chosen neighboring cluster serving as a backup cluster. In order to keep track of the stored data, each cluster head maintains *traces* identifying the stored data. A trace is a single-element Bloom filter [17] computed from the time epoch of the data aggregation, the region identifier of the aggregator and the type of data. When there is need to backtrack the information from a physically destroyed region within a particular period of time, the correspondent traces are created, combined into Bloom filters and sent in a request message. Using geographical routing this request is routed to the neighboring regions of the destroyed area where the requested data is stored.

The fault-tolerance provided by DISC comes at the price of higher complexity and a slight decrease of the network lifetime. The complexity of the WSN increases due to the use of a cluster-based architecture and the need for services such as geographical routing and cluster head election mechanism. Since our network follows a hierarchical architecture, cluster heads execute the heavy work of computing the collected data and sending it to be stored. We therefore try to preserve a homogeneous level of energy consumption for all the nodes by using a dynamic election of cluster heads. In comparison with mechanisms that store data locally on the node itself, DISC offers a higher resiliency for the data at a price of a slight increase in energy consumption due to the remote storing.

The DISC is a very flexible mechanism that allows to balance the amount of recovered data and the amount of resources spent. The amount of recovered information depends on the number of neighboring clusters that serve as backup clusters for one region. The DISC's efficiency grows with the density of clusters. Figure 4 shows the amount of recovered information depending on the fraction of failed nodes in the cluster in the case of malicious destruction of nodes. The usage of memory and processing resources can be adjusted depending on the tolerable probability of locating false data, that comes due to the natural false positive probability in Bloom filters. For example, in order to decrease the consumption of RAM by a factor of three, we logically form one Bloom filter

TABLE IV

MEMORY FOOTPRINT OF THE SERVICE DISCOVERY PROTOCOLS.

Resources	nanoSLP simplified	nanoSLP full	nanoSQL
RAM [Bytes]	320	411	655
ROM [Bytes]	1653	2505	4512
Lines of code (protocol + data storage)	700 + 700	950 + 700	3400 + 1300

from every three traces, which leads to approximately 9% increase in the false-positive probability of locating false data in the WSN.

C. Protocol Design and Implementational Trade-offs on Example of Service Discovery Protocols

We illustrate the protocol design and implementation trade-offs using the example of two service discovery (SD) protocols: nanoSLP [18] and nanoSQL. These general purpose discovery protocols differ in their expressive power and therefore are efficient to different range of applications. The nanoSLP protocol is an adapted version of the Service Location Protocol [19] for WSNs and is capable only of data discovery, not data management. Therefore it can only be used for SD purposes. The nanoSQL protocol uses a simplified version of SQL and allows both sophisticated data discovery and data management. The range of possible application for this protocol are not limited to SD. The nanoSQL can also be used for cross-layer optimization and as a part of a middleware. Both protocols function in a peer-to-peer fashion and, in principle, can the flexibility to be used on any network architecture. The protocols support proactive and reactive messaging and require an information storage service where the sensor nodes can register their services and data.

In nanoSLP each service is characterized by its name and attributes. We have realized two versions of the protocol. The first protocol version allows users to conduct complex searches using multiple AND, OR and comparison operators and discover separately both services and attributes. The second version is simplified further and is limited only to a single comparison operator and service searches. The comparison of the footprints of both protocol versions given in Figure IV shows that there is 56% increase in the ROM consumption and approximately 30% in the RAM size. The complexity of the code also increased significantly (by nearly 30%) due to more sofisticated parsing needed. The reader should note that the memory footprint of the protocols include both the data storage and the protocol modules, as we have used different data storages in case of the SLP- and SQL- based parsers.

The nanoSQL protocol is a new multipurpose protocol capable of both sophisticated service discovery and data management. The implemented support for both the service discovery and data gathering functionalities allows to save network traffic, via use of piggybacking. The SQL syntax was already used in WSNs before (see, for example, [20], [21]) however the implemented parsers did not allow as high flexibility as ours. The nanoSQL is capable of forming complex expressions, nested SQL statements, built-in aggregation functions and support for a wide range of data types. The expressiveness of nanoSQL comes at the price of the increase in the ROM footprint (80%) and the tripled code size. The growing functionality of the SD protocol and the corresponding parser leads not only the the increase of the memory footprint and code size, it also automatically leads to more complex component interfaces. For nanoSQL module we have more than fifteen interfaces, which certainly makes it more difficult to use this component.

VI. Design Automation Possibilities

The design process of WSNs can be partially automated due to their relative simplicity. The development of such a tool is a subject of our on-going work. The proposed SOA-based agile WSN methodology suits well for at least partial automation using Semantic Web [22] approach developed for web services composition. However, substantial differences exist in the process of how computers and humans can reach the satisfactory design decision. The computer tends to make decisions by iterating through a number of possible solutions where the number of possibilities searched are regulated by pre-defined rules. Humans take shortcuts using their experience and intuition. For example, a part of the application development process is the mapping of the abstract requirements and descriptions to the real world software and hardware. At this stage human-dependent parameters such as developer experience with one or another software and hardware platform influence the decision. People tend to develop in those environments they are familiar with, as additional learning curve is a time and resource consuming process.

We are currently developing on an expert system that will allow to design both complete WSN systems, as well as individual services, automate documentation and feedback gathering process for different components. As a first step we are implementing an ontology-based knowledge base that stores information about different WSN services and can estimate their compatibility and suggest possible component wiring. On the network side we are using a middleware with SD capabilities that is inspired by SQL syntax to gather the required parameters about the individual service functioning, as well as of performance of the overall system.

VII. Conclusions and Future Work

Constructing a WSN that fulfills a particular task, as well as designing and implementing the corresponding applications and protocols is not a trivial task. A developer has numerous decisions to take on the architectural, protocol and application design and implementational levels. In this paper we have proposed a design methodology that relies on service-oriented architecture and agile methodology and adapts this techniques for the WSN use. We have provided case studies that showed the applicability of this methodology to various application scenarios. Additionally we have proposed a list parameters that most of the WSNs can be described with and discussed the major trade-offs WSNs face. We have also developed a basic abstract SOA-based model of an individual node. Our next goal is to develop a framework that will assist developers in designing and implementing wireless embedded and sensor networks.

Acknowledgment

We thank DFG (Deutsche Forschungsgemeinschaft) and RWTH Aachen University for partial financial support through UMIC-Excellence cluster. The work was also supported by European Union (COMANCHE project). We would also like to thank K. Rerkrai, E. Osipov and M. Popa for fruitful discussions.

References

[1] C. Schroth and T. Janner, "Web 2.0 and SOA: Converging concepts enabling the internet of services," *IT Professional*, vol. 9, no. 3, pp. 36–41, 2007.

[2] M. P. Papazoglou, "Service-oriented computing: Concepts, characteristics and directions," in *Proc. of WISE*, Washington, DC, USA, 2003.

[3] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *Proc. of the IPSN*, NJ, USA, 2005, p. 48.

[4] R. Martin, *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2003.

[5] J. Miller *et al.*, "Model Driven Architecture (MDA)," Object Management Group, Draft Specification ormsc/2001-07-01, July 2001.

[6] S. Brown, "Updating software in wireless sensor networks: A survey," Dept. of Computer Science, National Univ. of Ireland, Maynooth, Tech. Rep., 2006.

[7] I. Sommerville, *Software Engineering*. Addison Wesley, 2006.

[8] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like P2P systems scalable," in *Proc. of the SIGCOMM '03*. NY, USA: ACM Press, 2003, pp. 407–418.

[9] C. Frank and K. Römer, "Algorithms for generic role assignment in wireless sensor networks," in *Proc. of SENSYS*, San Diego, California, USA, November 2005.

[10] V. Raghunathan and C. Srivastava, "Energy-aware wireless microsensor networks," *Signal Proc. Mag., IEEE*, vol. 19, no. 2, pp. 40–50, 2002.

[11] D. M. Jones, "The New C Standard (sentence 787)," http://www.coding-guidelines.com/cbook/sent787.pdf [last visited 20.09.2007], 2005.

[12] D. Slepian and J. Wolf, "Noiseless coding of correlated information sources," *IEEE Trans. on Information Theory*, vol. 19, no. 4, pp. 471 – 480, 1973.

[13] S. S. Pradhan and K. Ramchandran, "Distributed source coding using syndromes (DISCUS): design and construction," *IEEE Trans. on Information Theory*, vol. 49, no. 3, pp. 626 – 643, 2003.

[14] J. Chou, D. Petrovic, and K. Ramachandran, "A distributed and adaptive signal processing approach to reducing energy consumption in sensor networks," in *Proc. of INFOCOM'03*, San Francisco, USA, March 2003.

[15] C. Jardak *et al.*, "Distributed Information Storage and Collection in WSNs," in *Proc. of MASS*, Pisa, Italy, October 2007.

[16] L. Buttyan and P. Schaffer, "Panel: Position-based aggregator node election in wireless sensor networks," in *Proc. of MASS*, Pisa. Italy, October 2007.

[17] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[18] Z. Shelby *et al.*, "NanoIP: The Zen of Embedded Networking," in *Proc. of ICC'03*, Seattle, Washington, USA, May 2003.

[19] E. Gutman and et al., "RFC 2608: SLPv2: A service location protocol," June 1999.

[20] K. Rerkrai *et al.*, "Unified Link-Layer API Enabling Portable Protocols and Applications for Wireless Sensor Networks," in *Proc. of ICC*, Glasgow, UK, June 2007.

[21] S. M. *et al.*, "TinyDB: an acquisitional query processing system for sensor networks," *ACM Transactions on Database Systems*, vol. 30, no. 1, pp. 122–173, 2005.

[22] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 284, no. 5, pp. 28–37, 2001.