An Extendible Metadata Specification for Component-Oriented Networks with Applications to WSN Configuration and Optimization

Elena Meshkova, Janne Riihijärvi, Junaid Ansari, Krisakorn Rerkrai and Petri Mähönen
Department of Wireless Networks, RWTH Aachen University
Kackertstrasse 9, D-52072 Aachen, Germany
Email: {eme, jar, jan, kre, pma}@mobnets.rwth-aachen.de

Abstract— **Currently there is no widely accepted formalized description or metadata that represents networks, their components and interrelations between them. There is also no de facto classification of network parameters, models and services. The network community lacks a metadata specification similar to WSDL that represents networks in clear and unified manner and includes software, hardware and networking parts of the system description. This limits the advance of the frameworks aiming to assist the development of complex non-established network systems, such as wireless sensor, cognitive and heterogeneous networks. Additionally a standard metadata would enhance the interoperability and acceptance of various middleware, service discovery schemes and cross-layer optimization solutions. In this paper we suggest a simple, but comprehensive extensible metadata specification and a corresponding metamodel that allows to describe a wide range of network components and network configurations. We describe metadata and metamodel on the example of wireless sensor networks. In this field the need for standard metadata is particularly pressing due to the high number of available protocol solutions and lack of standardization.**

I. Introduction

Networks are growing more complex every day. In order to create large heterogeneous networked systems developers have to possess not only expert knowledge on networking, but also very specialized knowledge related to particular network types. Nowadays it is enough for a network software developer to know the TCP/IP stack and have a general understanding when it is better to use UDP or open a TCP socket. However, in future ubiquitous heterogeneous environments there will be no single stack of protocols working perfectly for every scenario and every kind of network topology. It will be very expensive to develop a heterogeneous system that incorporates, for example, wireless sensor (WSNs), cognitive and fixed networks, as experts not only on each network type have to be hired, but also specialists on particular network layers and services have to be contacted. An introduction of an expert system can lower the cost of such development as it will be able to give useful hints on the particular type of components to be installed on a network node. The first step towards development of such a system is creation or adoption of data format, or more precisely a *metadata* specification, for describing network components. For services that a network or individual nodes provide, corresponding software and hardware have to be described. The network topology and behavior is to be specified as well. A respective *metamodel* has to be created in order to reflect interrelations between different components of the network.

Besides such a far-reaching example, a metadata is urgently required even today to formally describe emerging networks and their components like WSNs or cognitive networks. At present network parameters and their importance for different network types and scenarios is generally understood by researchers, but the common vocabulary is not established, which has a negative effect on possible collaboration. The absence of a well-established metadata and common vocabulary also slows down acceptance and adoption of various cross-layer designs, service discovery protocols, middleware solutions, systems for runtime optimization and dynamic code loading techniques.

In this paper we suggest a format for the network description that follows component or service-oriented approach. We provide an extendible metadata and a metamodel for this purpose. The applicability of the suggested structures to WSNs is shown and the details on some specific WSN components are given as representative examples. We also argue on suitability of the proposed metamodel and the metadata for both static and runtime configuration and maintenance of networks.

The rest of the paper is organized as follows. In Section II we describe our metadata and metamodel. The main attributes and parameters that characterize a network are presented in Section III. In Section IV we show how the metadata and the metamodel can be used on the example of wireless sensor networks. We conclude the paper with a short summary and the future work directions in Section V.

II. Metadata and Metamodel Specification

In this section we present a metamodel and an extensible metadata specification that are applicable for representing and modeling almost any network following component or service-oriented engineering approaches [1], [2]. *Metadata* can be defined as a structured data that is used to characterize a resource. A metadata may contain information on the content, quality, condition, origin, and other characteristics of data or pieces of information. There exists several well-known standards for defining metamodels and metadata. XML is a typical example of a language for metadata specification, and Web Services Description Language (WSDL) [3] is a well known XML-based language for metadata specification. An example of a non XML-based metadata format is Doxygen [4], which is used to document software modules.

A *metamodel* specifies the constructs and rules required to completely or partially model a domain of interest. The XML-based RDF, and its derivation OWL that are used in the Semantic Web context, are examples of metamodel definition language [5]. The other examples are UML and SDL, languages targeted at object and system modeling, and especially for modeling software and network protocols.

We are not the only ones who target at global metadata and the respective metamodel specification. There exist multiple middleware, service discovery and cross-layer optimization solutions that require a well-defined metadata. The most well-known examples of middlewares include CORBA [6] and Web Services, while in the service discovery these are UPnP [7] and Bluetooth's SDP [8]. The metadata used by CORBA is very expressive, but too bulky to be used for all types of networks. Furthermore, it does not provide the analog of the utility function according to which the network performance can be optimized. For the same reasons the SNMP management information base (MIB) specification also can not be used. MIB is also otherwise more limited with its expression power, since it was designed to store parameter-value pairs instead of metadata capable constructs. OWL-S [9] is a very specific metamodel and applicable only for web services composition.

A. Vocabulary

In order to formally describe a network we need to establish a *vocabulary* or a nomenclature of the domain. Below we describe the fundamental terms of our metadata specification and the corresponding metamodel.

We consider a *service* as a *logical* entity that can provide a simple or a composite *functionality*. A *component* is a *physical* unit that hosts a service. For example AODV is a service that provides a routing functionality and which is implemented in the NST-AODV [10] software component. A network node or even the whole network can also be regarded as separate components that provide composite services such as packet forwarding and environment monitoring. Services are also characterized by *interfaces* which formally define the functionalities that a service provides to the outer world. Each interface is defined by a unique name and a list of passed type-variable pairs.

Each service is characterized by a set of *attributes*. Attributes are used to evaluate the performance of a service and the respective component. Attributes typically cannot be directly adjusted by a user or a programmer, they can only be "measured". *Parameters* are often the directly adjustable variables or constraints in a software module. Typically there are much more paramters rather than attributes in a system. One can also refer to parameters as the lowest level attributes. Example parameters include the transmission power of a node, its hardware characteristics, a particular protocol's header size and various types of timer values. Examples of attributes are the average network delay, the mean datarate generated by a node or the average power consumed by a network protocol as a function of the incoming traffic, hardware and radio parameters. Attributes and parameters are further described in Section III. Generally, an attribute depends on parameters, other attributes and influenced by constraints and preferences. That is

$$A_c = F(\{A_{input}\}, \{P_{netw}\}, \{P_{comp}\}),$$

where A_c stands for a component attribute, F is a *dependency function*, A_{input} is a set of input attributes on which the current component's attributes are dependent, P_{netw} is a set of parameters that characterize a network and P_{comp} is a set of component parameters. There can be both intra- and inter-component attribute dependencies.

Depending on the accuracy of the model we want to build certain attributes, parameters, services and components can be abstracted. For example, if we create a model of a heterogeneous network, we can regard separate nodes as components that provide certain services and deal only with "high level" attributes such as an expected node datarate. On the other hand if we develop a specific software component, for instance a schedule-based contention type MAC protocol, parameters such as RTS and CTS frame sizes, initial and congestion back-off values and clean channel assessment duration start to be important as they influence the major MAC attributes such as power-consumption, delay and throughput of a protocol. Abstractions are also used to simplify a metamodel. For example, in our current work on automatic network service stack composition, we consider the network topology, hardware and software platforms as fixed and do not vary or iterate between them, but rather concentrate only on the choice of optimal software components.

Constraints can be specified either by a user of the system or a component developer. Constraints restrict the pool of services or components considered when performing a certain operation on a network. For example, a typical WSN-related service discovery request is: "report all the sensor nodes that provide temperature readings higher than 20°C". If we regard a separate node as a component within our system and apply a constraint of 20°C on its attribute "temperature reading" we obtain a pool of components (nodes) that satisfy the given request. Constraints are also widely used for static and runtime configuration of the network and its individual components. They can be employed to restrict, for example, the operating system to be installed onto a device, types of peripherals to be used and attributes, like minimum expected network lifetime or maximum allowable network delay. Constraints can also impose security requirements on components employing, for example, the policy-based approach [11].

Preferences are non-strict constraints. They can, but do not need to, be followed. Preferences are considered in the case when there are several solutions available that comply with all of the constraints and they can be rated according to the defined preferences. For example a user might specify that

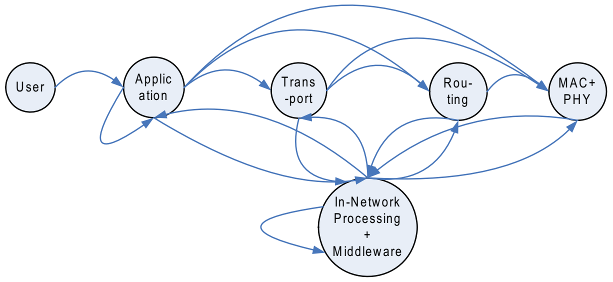

Fig. 1. A sample WSN-specific metamodel.

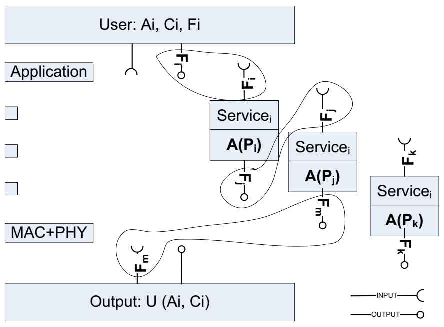

Fig. 2. A sample wiring.

he prefers a longer lifetime of the network over a higher throughput. A component provider may also point out that, if possible, a certain component should be wired only to other components of the same developer (see Section II-B).

Utility function is used to evaluate the performance of a single component, a stack of wired components or a whole network in a specific environment. It is a part of the metamodel. The utility function enables formal, automatic assessment of the component performance and therefore allows the creation of software that can automatically configure networks and choose the corresponding components to be deployed. The utility function also enables the runtime configuration of networks and can be, for example, a part of middleware or cross-layer optimization software. The utility function can be specified either as a mathematical function or as a set of logical constructs, for a example as a tree of `if` statements. The typical example is the utility that defines the trade-off between the delay and the reliability of the packet delivery. The utility function depends on attributes, that in turn take into consideration other attributes, parameters, constraints and preferences. The optimal performance of a component stack can be defined as:

$$\text{OptimalStackPerformance} = \max_{p \in P} \left(\max_{w \in W} \left(U(\{A_j(w,p)\}) \right) \right),$$

where U stands for the utility function, A_j encodes a set of attributes, P is the parameter set and $W = W(\{C_i\}, \{Pr_m\})$ encodes possible wirings[1] that in turn depend on the defined constraints C_i and preferences Pr_m.

A *stack model* is a component of the metamodel. Stack models reflect allowable component wirings using general service interdependencies within a single node or the whole network. Stack models can be arbitrary complex and generic. In Figure 1 we provide a sample generic model for WSNs that allows hierarchical connection of protocol layer modules and addition of extra, “parallel”, components to any basic layer module. An application (a sensor-reading component), an in-network processing module (a network coding component) and a MAC protocol with corresponding PHY-layer form a valid component stack according to this model. Depending on the usage scenario some stack models might become invalid. For example, in case of a multi-hop network a routing or a broadcasting protocol is required therefore the above stack model is not applicable as it allows the omission of the component providing the routing functionality. In similar manner the protocol stack that allows only reactive routing protocols might be specified for the scenarios with medium node mobility.

[1]We have the same notion of wiring as in nesC/TinyOS framework [12].

B. Metamodel

We suggest a very simple extensible metamodel that connects different components together on the basic of their functional compatibility and estimates the effectiveness of their combined performance.

The functional compatibility of components is estimated in two steps. First the components that have complying functionalities are linked together in order to provide desired services. In Figure 2 we show how different components can be wired together on the basic of functional and service compatibility in order to provide a desired functionalities to the user. The component wirings are determined using the stack models that reflect interdependencies of services. Constraints and preferences also effect the component wirings, like “component `A` should always be wired only with component `B`”. This is the usual way to define complex components.

The second step is to check the possible service wiring combinations for validity on the interface levels, i.e. to ensure that interfaces required by one service are provided by the other. As a result, the pool of valid *component wirings* is obtained. These wirings are further evaluated using the utility function to estimate their performance. Usually we want to find component stacks that maximize the utility function.

In order to minimize a pool of possible component combinations, we propose a *hierarchical wiring* with three levels. The *component-wise wiring* allows to create complex software components from the simple ones. A *node specific wiring* determines a valid combination of the complex or simple components that can be installed on a single network node. During the *network wiring* components reflecting separate nodes are combined together to from a network that functions according to user preferences [13].

Realization of wiring models, component specification validity checks, as well as interface correspondence checks can be realized using rule-based reasoning. Ontology is one of the

```
<Component> <Name> Test Application </Name>
  <CommonData>
      <Parameters>
          <OS> <Name> TinyOS2.x </Name>
            <Version> 2.15.4 </Version> </OS>
         <Footprint>
            <ROM> 3000 </ROM> <RAM> 500 </RAM>
            <Unit>Byte</Unit> </Footprint>
      </Parameters>
      <Hint> <SW Provider> Berkeley </SW Provider>
      </Hint>
  </Common Data>
  <Services>
    <Service>
      <Name> nanoMAC </Name>
      ...............................
      <Parameters>
         <Persistence> <Size>0.1</Size>
         <Ref> Pers </Ref></Persistence>
      </Parameters>
      <Attributes> <Input>
        <Datarate> <Ref>datarate</Ref></Datarate>
      </Input> </Attributes>
      .................................
      <Functions>
         .......................................
            S_nanomac=(G*(b+1)*(1-Pers + Per*
            exp(-a*G)))/(G*(1+(4+Pers)*a + 2*b + c)
            + Pers*exp(-a*G));
         .......................................
      </Functions>
      .................................
      <Attributes> <Output>
        <Datarate> <Ref>S_nanomac</Ref> </Datarate>
      </Output> </Attributes>
    </Service>
  </Services>
</Component>
```

Fig. 3. A Sample metadata in XML format.

techniques that can be used to create lightweight and flexible metamodels, like OWL-S [9], for example.

C. Structure of the Metadata

Next we need to consider a specification of a metadata for a single component. A minimal set of meta-data fields for each component includes: attributes, parameters, provided services and respective interfaces, constraints, priorities and dependency functions.

The *services* section lists the functionalities that a component provides. The service descriptions can be arbitrary complex. It is required that a component defines its functionalities at the highest level. For example *nanoMAC* [14] can be described both as a component that provides a *one-hop transport and/or MAC data transport functionalities*. However, it can further be defined as a *CSMA/CA* and a *scheduled-based* protocol. Additional specification allows to achieve a better and faster wiring, utilizing specific stack models. The service description also contains the list of associated *interfaces*.

The *dependency functions* are needed to estimate attributes. The functions use either mathematical formulas of behavior of certain attributes or reference to a file where the behavior model is stored as an array of values. These values can be obtained either from simulations or real-time deployment measurements. The dependency functions can be specified either inside of other section descriptions, for example inside an attribute section, or separately using a specific tag. In Figure 3 the description of the throughput of nanoMAC protocol is given under the specific dependency functions tag and is referred to in the parameters and the attributes sections. From the same figure it is evident that the metadata fields can appear multiple times in the component specification.

The *parameters* section lists parameters that can be adjusted and affect the component's performance. Each parameter is characterized by the units, the range of accepted values and their granularity. Parameters are also linked to the corresponding variables in the program code.

The *attributes* section lists the attributes that the particular component has an effect on. The attributes, as well as parameters, are also characterized by units. Attributes of one component can be affected by the same attributes from other components, for example the MAC datarate is affected by the datarate of an above-layer component. Therefore, it is necessary to distinguish between the input attribute values and the output ones, and link them to the respective variables in the dependency functions.

The *constraints* field specifies the constraints that the current component imposes on other modules. Typically, these are functionality constraints, for example a request for an external security provision or QoS. Constraints section is also used to define complex components. For example a geocasting module requires a geo-routing component.

The *preferences* section defines recommendations on wiring imposed by the current component on other modules. For example this field might contain information like a preference for a certain MAC module.

Attributes, parameters, constraints and preferences fields can be both defined inside the service section metadata specification and outside of it in the general component specification. In the former case, the respective metadata fields refer only to the specific service. In the latter they refer to the whole component. There can be cases when only one of the services suggested by the component is used and therefore only metadata concerning this specific service should be considered. On the other hand, there exist common metadata shared by all the services of the component and it does not make sense to duplicate the specification.

All variables in the metadata can be cross-component references using class-like dotted notation, e.g., $ComponentName.MetadataSection.VariableName$. We propose to specify the metadata using XML, as it is easy to automatically parse and process. The use of default values and templates will also simplify the process of metadata creation. On the resource-constraint devices only the most relevant parts of the metadata need to be reflected using a compressed, for example a token-based, representation.

III. Parameters and Attributes

Attributes characterize the performance of services and respective components. They enable not only the functional wiring of the components, but also the evaluation of the resulting component stack. There are five *global* attributes that are the most important for the user and can be applied to almost any component of the system. These are *latency/delay, throughput, reliability, lifetime* and *economic cost*. Network protocols expose various tunable/adjustable parameters in order to affect these attributes. Each component in the stack has a certain degree of control on the attributes through the parameters. Given the constraints, the metamodel of the

TABLE I
Common attributes and parameters.

	Global	Network		Hardware	
Common for all	Delay	Size	Node specific	Processor	
	Realiblity	Mean path length		Clock frequency	
	Lifetime	Node distribution		Peripherals	
	Datarate	Active/sleep node ratio		Memory	RAM
	Cost				ROM
		Mobility			EEPROM
		Network graph		Interfaces	

	Software		Shared
Common for SW comp.	OS / VM		Average suppl. packet length
	Hardware platform		Suppl. packet length frequency
	Memory	RAM	
		ROM	Avg. header length
		EEPROM	

TABLE II
Layer specific parameters on example of WSNs.

Application	Transport	Routing	MAC+PHY
Sensor readings	MTU	Route re-new timer	Transmit power
Sample rate	Number of retries		Frequency
Accuracy	Timeout timer	Timeout timer	Max. frame size
Power per sample	Window size		Number of retries

complete protocol stack may determine the settings of the parameters of the various individual protocols so as to achieve overall the most optimized attributes. It should be noted that there are certain runtime and deployment time factors that also affect the attributes of a network stack. These may include network topology related parameters (like the node density), amount of the generated traffic, the priorities and the constraints. K. Römer and et al. in [15] have identified some common properties of WSNs. We have extended and generalized this list with respect to our metadata. In Table I we provide a summary and a classification of the most common attributes and parameters that affect the performance of any software component in the networked system. Besides *global, network hardware and software*-related attributes and parameters, one may define the so-called *common protocol* parameters that are applicable to any protocol and include the size of the header and the statistics on the supplementary or overhead control traffic imposed by the component.

Additionally, in Table II, we identify typical parameters for different protocol layer components. By no way is this an exhaustive list as each protocol is unique and has its own set of parameters. Partially, the parameters can be clustered depending on protocol types, as certain parameters are shared by different protocol classes. For instance, all the CSMA/CA MAC protocols have parameters like carrier sensing time, initial and congestion back-off values, packet sizes, number of re-transmissions, etc. that affects throughput, reliability and energy consumption. However, there can be many ways of classification of a particular set of protocols. This is so because functionalities and features of various protocols are typically intermingled and overlapping, metadata associated with a protocol provides a better way of expression and thereby making the use of the functionalities.

IV. Application Scenarios for WSNs

In this section we provide a description of complex scenario where the proposed metadata and metamodel can be used. We consider a framework that allows automatic service composition of WSNs at the pre-installation phase and later fine-tuning during the runtime after the deployment of the network. We also briefly discuss related work.

A. Related Work

Runtime optimization is particularly important for WSNs and has been extensively studied. TinyCubus [16] is a WSN-specific solution and aims at solving runtime optimization of network employing TinyOS system. SNACK [17] focuses on both enabling faster component development for WSNs and runtime optimization of the system. This framework as well as the FiGaRo middleware [18] are the most similar to our current area of research, but they are not as generic as our specification and concentrate only on WSNs. They also do not consider trade-offs between attribute and their optimization using the utility function. The authors do not describe their metadata in detail rather only the corresponding configuration language.

There exist also solutions for WSNs that aim to achieve the auto-configurability of these systems using ontologies, for example [19]. These approaches concentrate mostly on the functional wiring of the components rather than on performance optimization.

B. Pre-Installation

Consider that each WSN software component has metadata attached to it. A user specifies the desired network functionalities, global attribute constraints and preferences. Additionally (in the current development step) he/she specifies the hardware and sensor platforms to be used, the operating system and the network characteristics or topology as given, non-adjustable parameters.

A metamodel for possible component wirings is stored in form of ontology in the knowledge base. If the knowledge base is not populated/updated with the component metadata, the directory storing the relevant files is searched and the metadata is extracted from the files and placed in the knowledge base. Using the ontology and the corresponding reasoning functionalities we obtain a representative pool of possible component wirings, i.e. we rapidly generate a number of compatible component stacks that can be deployed on the given hardware and software platform and would provide the desired functionality. The next step is to evaluate these stacks if they satisfy the user constrains for the given network configuration. We propose to employ some metaheuristic algorithm, for example simulated annealing, to perform a rapid search through the available wirings estimating their utility values and filter out the solutions that are foreseen to work best at the specified network configuration. The use of metaheuristics allows to achieve a fast search speed at the price of accuracy of the final solution, i.e. to obtain a satisfactory good local maximum of the utility function.

In case no appropriate solution is found, the user is notified and the system needs to be updated with new more appropriate network components or the initial conditions should be adjusted. In case of heterogeneous network several node-specific component stacks have to be formed and their combined performance needs to be assessed (which is out of the scope of our current work).

After the system deployment, the statistics on its performance are to be gathered and fed back to the framework. This data is further processed, analyzed and performance assessment of each of the deployed components is fed into the system. This feedback loop ensures constant improvement of the system performance.

C. Runtime Configuration

WSNs usually operate in heterogeneous varying environmental conditions. The system should be robust against changing surroundings or states of the network which can occur unpredictably. Examples include link failures, node damages and topological changes. Sensor nodes need to reconfigure themselves dynamically in order to adapt to the changes, especially without presence of an external or centralized control entity. They form a self-adaptive network in terms of structure and functionality. Components can interact either on an inter-node level, i.e. just inside of one node, or in an intra-node manner forming a distributed system. Both ways a common metadata is required. The access and manipulation of the common metadata, as well as inter-component signaling, can be handled by a dedicated component. For example a Universal Data Access Engine (UDAE) [20] is very well suited for this purpose. There are two possibilities to reconfigure components at runtime which include tuning of parameters which have to be specified using a common metadata and rewiring of components, which can be done according to a specific metamodel. UDAE can be used to efficiently retrieve and tune individual component's parameters in order to provide the information to a cross-layer optimization module that aims at maximizing of the utility function. In worse cases, the optimization module may suggest a node to switch to a more suitable protocol by rewiring components. This decision results from the assessment of the utility function for a different component stack. A set of replaceable components is usually stored in EEPROM. Dynamic rewiring mechanisms depend heavily on the design of operating system. For example, TinyOS [12] requires rebooting of the system when a new component has been loaded while in Contiki [21] load and unload components do not need to reboot the system.

Similar to cross-layer optimization, self-organization protocol also benefits from runtime network configuration. The self-organization module has to adjust parameters and select functionality which are best suited to network changes. For instance, depending on the node failure rate, the network should be able to react autonomously and re-route traffic so that its performance (again measured by the utility function) can be improved.

V. Conclusions and Future Work

In this paper we have provided a metadata specification that allows to describe and model various types of networks using the component or service-oriented approach. We have showed how components can be described with different levels of details. We have also provided the metamodel that allows to estimate the compatibility of different components and to estimate the effectiveness of the system where a certain stack of components would operate with respect to the objective function. Currently we are working on the creation of the automatic service composition framework for WSNs.

Acknowledgment

The authors would like to thank the RWTH Aachen University and the DFG for providing financial support through the UMIC excellence cluster. This work has also benefited from partial support from European Commission through ARAGORN and WASP projects.

References

[1] O. Nierstrasz and L. Dami, "Component-Oriented Software Technology," *Object-Oriented Software Composition*, pp. 3–28, 1995.

[2] M. Papazoglou and D. Georgakopoulos, "Service-Oriented Computing," *Communications of the ACM*, vol. 46, no. 10, pp. 25–28, 2003.

[3] R. Chinnici *et al.*, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language," *W3C Working Draft*, vol. 3, 2004.

[4] D. Van Heesch, "User Manual for Doxygen 1.2. 10," 2001.

[5] G. Antoniou and F. van Harmelen, "Web Ontology Language: OWL," *Handbook on Ontologies*, 2004.

[6] S. Vinoski, "CORBA: integrating diverse applications within distributedheterogeneous environments," *Communications Magazine, IEEE*, vol. 35, no. 2, pp. 46–55, 1997.

[7] M. Jeronimo and J. Weast, *UPnP Design by Example: A Software Developer's Guide to Universal Plug and Play*. Intel Press, 2003.

[8] S. Avancha, A. Joshi, and T. Finin, "Enhanced service discovery in Bluetooth," *Computer*, vol. 35, no. 6, pp. 96–99, 2002.

[9] D. Martin *et al.*, "Bringing Semantics to Web Services: The OWL-S Approach," in *Proc. of SWSWPC*, San Diego, CA, USA, July 2005.

[10] C. Gomez *et al.*, "Adapting AODV for IEEE 802.15. 4 Mesh Sensor Networks: Theoretical Discussion and Performance Evaluation in a Real Environment," in *Proc. of WoWMoM*, Buffalo-Niagara Falls, New York, USA, June 2006.

[11] G. Myles, A. Friday, and N. Davies, "Preserving privacy in environments with location-based applications," *Pervasive Computing, IEEE*, vol. 2, no. 1, pp. 56–64, 2003.

[12] P. Levis *et al.*, "TinyOS: An Operating System for Sensor Networks," *Ambient Intelligence*, 2005.

[13] E. Meshkova *et al.*, "Service-Oriented Design Methodology for Wireless Sensor Networks: A View through Case Studies," in *Proc. of IEEE SUTC 2008 (accepted)*, Taichung, Taiwan, June 2008.

[14] J. Haapola, "NanoMAC: a distributed MAC protocol for wireless sensor networks," in *Proc. of the FWCW*, Oulu, Finland, October 2003.

[15] K. Romer and F. Mattern, "The design space of wireless sensor networks," *IEEE Wireless Communications*, vol. 11, no. 6, pp. 54–61, 2004.

[16] P. Marron *et al.*, "TinyCubus: a flexible and adaptive framework sensor networks," in *Proc. of EWSN*, Istanbul, Turkey, January 2005.

[17] B. Greenstein, E. Kohler, and D. Estrin, "A sensor network application construction kit (SNACK)," Baltimore, MD, USA, November 2004.

[18] L. Mottola, G. Picco, and A. Sheikh, "FiGaRo: Fine-Grained Software Reconfiguration for Wireless Sensor Networks," January 2008.

[19] G. Fuchs, S. Truchat, and F. Dressler, "Distributed Software Management in Sensor Networks using Profiling Techniques," in *Proc. of Comsware*, Delhi, India, January 2006.

[20] K. Rerkrai *et al.*, "UDAE: Universal Data Access Engine for Sensor Networks," in *submitted to SECON*, San Francisco, USA, June 2008.

[21] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proc. of LCN*, Tampa, Finland, November 2004.