Robust and Efficient Software Management in Sensor Networks

Wolfgang Schröder-Preikschat, Rüdiger Kapitza, Jürgen Kleinöder, Meik Felser, Katja Karmeier,
Thomas Halva Labella, and Falko Dressler
Dept. of Computer Sciences, University of Erlangen-Nuremberg, Germany
Email: {wosch,kapitza,kleinoeder,felser,karmeier,labella,dressler}@informatik.uni-erlangen.de

***Abstract*— Software deployment and updating of deployed code is a critical topic in the area of wireless sensor networks (WSN). Reasons are unreliable network connectivity, resource limitations of devices, and energy restrictions in general. Consequently, software deployment has to be performed with great care otherwise resources might be wasted or nodes become unavailable due to failed updates. The overall objective for software management in sensor networks is to enable a robust and efficient way to build deployable software images that take into account all the needs from application requirements to node-specific resource restrictions. In this article, we outline the current state-of-the-art for software management in WSN and (network-based) sensor network (re-)programming. We provide insights into three different approaches enabling a more comprehensive management of WSN while challenging the robustness and efficiency of software configurations and reprogramming. Additionally, we outline research challenges in the area of software management and sensor network operation.**

I. Introduction

With the proliferation of wireless sensor networks (WSN) [1], [2] and sensor/actuator networks (SANET) [3], new application domains appear that make efficient use of such networks, for example in the field of habitat monitoring [4] and precision agriculture [5]. Many challenges, such as energy efficiency, security, and self-organization, have been identified in this area [3], [6] including also software management in WSN [7]. An overview to software management techniques in WSN is given by Han and co-authors [8]. They depict a model consisting of three fundamental components: the execution environment at the sensor node, the software distribution protocol in the network, and the optimization of transmitted updates. We concentrate on software management techniques for WSN that are dynamic in terms of availability, mobility, and current application demands. Due to the heterogeneity of employed hardware platforms and the low resources in terms of processing power, available memory, and networking capacities (sensor nodes are usually able to run a single task only) [9], new approaches for efficient software engineering are needed. An overview to the issues that are specific for sensor nodes is provided by Culler et al. [10]. In this work, the necessity for network-oriented software architectures is described. Questions such as how to configure, reconfigure, program, and reprogram networked embedded systems such as sensor nodes are discussed by Handziski and co-workers [11]. Software management for networked embedded systems such as sensor networks has been intensively studied in the last few years [12], [13].

In general, two approaches for software updates in sensor networks have been discussed in the literature: multihop network-based node reprogramming and robot-assisted software management. Work on the first technique was done mainly based on network-centric reprogramming. For example, the Deluge system [14] was developed for reprogramming Mica2 motes. Deluge propagates software update over the ad hoc network and can switch between several images to run on the sensor nodes. An role assignment system was developed at the ETH Zurich [15] to switch between multiple tasks depending on the current requirements. The flexible exchange of software components in TinyOS was investigated at the University of Stuttgart. The developed toolkit FlexCup [16] introduces software engineering methods for sensor node programming. Incremental network (re-)programming was studied by Jeong and Culler [17]. The primary focus of this work was on the delivery of software images over an ad hoc network.

The presented previous work mainly addressed the (re-)programming of homogeneous networks regarding software but—more important—also hardware aspects and concentrates on some aspects of the deployment and update process of sensor software. We believe that future sensor networks will face diverse application demands that require the deployment of heterogeneous sensor nodes. Such requirements occur for example in an habitat monitoring scenario where animals wear very small sensors that collected data that is forwarded to bigger infrastructure sensors forming the real network. Besides application dictated requirements, the use of heterogeneous hardware offers flexibility to reduce costs, enlarge the lifetime, and in general provides extended opportunities. Lastly we believe that heterogeneity might be a by-product of durable sensor networks where nodes are supplemented and failed nodes are replaced. Of course one can assume that the same type of sensor nodes might be still available but as the development constantly moves on this would waste benefits of the next generation sensors that might be smaller, cheaper, and have a longer life-time.

Therefore, this paper advocates for a more comprehensive development and management process of sensor software that addresses software and hardware heterogeneity at the level of application, middleware and operating system. This includes

the introduction of software engineering and management technologies such as feature models and product lines that help to handle application but also hardware diversity. Starting from this point it is possible to determine very fine grained software changes and use this as a basis for code generation and code deployment. As software in general evolves over time due to ongoing development its code footprint usually rises. Even so the use of the software management technologies alleviates this process, there might be constellations where the software that has to be deployed will not fit on some or all of the targeted sensors. In these cases we propose an assistance concept that determines off-line that the software exceeds the available storage. So instead to simply deploy new functionality it is decided which parts of the software can be deployed on different nodes that either provide the code on demand or simply provide the demanded functionality as service.

The main contributions of this paper can be summarized as follows. Besides a motivation for software and hardware heterogeneity in future sensor networks, we discuss three approaches that enable a more comprehensive management of WSN challenging the robustness and efficiency of software configurations and reprogramming. In particular, we discuss feature models, profile-based reconfiguration, and assistance operations. These specific methodologies will enable WSN to become more reliable for the application even if single systems might fail or do not provide sufficient resources. Additionally, we provide a list of future research challenges in the area of software management and sensor network operation.

The remainder of this paper is structured as follows. Section II gives a brief introduction to feature models and its applicability to handle software and hardware heterogeneity of sensor software. Section III outlines how software changes lead to find grained code generation and deployment. Section IV discusses concepts to handle cases when software exceeds the storage of a target node. Section V poses open question regarding software management of sensor networks. Finally, Section VI concludes the paper.

II. Feature Models and Feature Trees

Feature modeling is understood as "the activity of modeling the common and the variable properties of concepts and their interdependencies and organizing them into a coherent model referred to as a *feature model*." [18] Goal is to come up with directives for and a first structure of a design of a system that meets the requirements and constraints specified by the features.

Common is a graphical representation of the feature model in terms of a *feature diagram*. The diagram is of tree-like structure (see figure 1), with the nodes referring to specific feature categories. Four feature categories are defined: *mandatory*, *optional*, *alternative*, and *or*. A feature diagram describes the options and constraints that shall exist within a system. It models the variable and fixed properties of a family of software and hardware assets which implement that system.

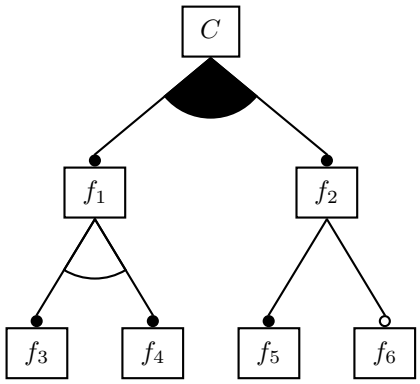

Fig. 1. Feature diagram: f_1 and f_2 are or-features of concept C, f_3 and f_4 are alternative features of f_1, and f_2 implies a mandatory feature f_5 and an optional feature f_6.

The diagram shown in figure 1 describes a specific concept C, e.g. the process management subsystem of an operating system. If concept C gets to be included in the final system configuration, then any non-empty subset of features from the set $\{f_1, f_2\}$ of or-features is also included. The *feature set* with respect to C at this level of abstraction is $\{f_1, f_2, \{f_1, f_2\}\}$. If feature f_1 is present, one feature from the set $\{f_3, f_4\}$ of alternative features must be included. Thus, the feature set of f_1 consists of either f_3 or f_4. If feature f_2 is selected, mandatory feature f_5 must and optional feature f_6 may be included in the final configuration.

This technique allows for a compact and precise specification of interdependencies of functional as well as non-functional properties of fairly complex systems. Basing on a tool which aids the construction process of a feature model and supports the mapping of features to implementations, automated generation of highly specialized software systems becomes possible [19].

Prerequisite for robust and efficient maintenance of sensor-node software is a methodology that allows one to keeping track about the various software "snippets" on the various nodes. At the very beginning, a (more or less) static description of the software and/or node variants is required in order to identify which of the software components goes to which nodes. Such a description is manifested in the structure of the feature diagram for a given application scenario.

Once setup properly, the feature diagram lists software components (a) common to every node of the sensor network, (b) common to a subset of nodes, or (c) destined for a very specific node, only. That is to say, all the common and variable (software) properties of a sensor network are summarized by means of a single, concise, and unambiguous description. Based on this initial knowledge about the (options of the) logical distribution of the software components across the sensor network, the software deployment process is directed accordingly: a set of explicitly or implicitly selected features refers to those software components that need to be uploaded to the various sensor nodes. Note that this approach gives

structure only to the decision process on *what* to upload, and not on *how* to proceed with all the subsequent communication activities in an efficient way.

III. Profile-matching for Code Generation

The basic ideas of profile-based reconfiguration are summarized in the following. We rely on a (set of) server system for the decision-making process, code generation, and node reprogramming. A global goal is assumed to describe the application requirements. Such a goal could state the need to have specific applications available in different parts of the sensor network. This example refers to the often discussed coverage problem in sensor networks [20], [21]. Often, mobile systems are employed to handle this issue [22], [23]. Compared to these approaches, we support the need of multiple concurrent applications at the same time and include the issue of reprogramming. Nevertheless, we do not discuss details of the decision process that is adequately investigated by other groups [20], [24], [25].

Our developed profiling (or profile-matching) concept consists of two parts:

1) A definition of profiles that characterize a software service, e.g. software modules, and such profiles that characterize environments, i.e. platforms on which services can be offered, e.g. sensor nodes.
2) A definition of profile-matching rules defining how these platforms can be reconfigured with these services. The word reconfiguration stands here in general for any new software configuration (in the sense of loading new software).

In the following, we use the following notations: *NP* for node profile, *AP* for application profile, *MP* for module profile, and *xP** means at least one profile of type *x*. The primary goal of profile matching is to create all possible combinations of executable source code. Again, we use a straightforward terminology for the definitions. *(NP, AP*, MP*)* means that on the node described by *NP* the applications described by *AP** with the modules described by *MP** can be installed. Each module or application can be realized using different source files.

For code generation and reprogramming, we rely on an external server responsible for control and management. In our example, we focus on code management for Mica2 motes running TinyOS[1]. The server system performs the dynamic source code selection and generation. Figure 2 shows the activity diagram for creating a code binary. One static input corresponds to the code templates for the generation of the wiring, the node profiles, and the configuration, another to the source code of the modules (nesC files). The dynamic inputs are the current configuration and the matching profiles. The goal is to create a binary that runs on the node described by *NP* and contains all applications and modules described by *AP** and *MP**.

[1]see `http://www.tinyos.net/`

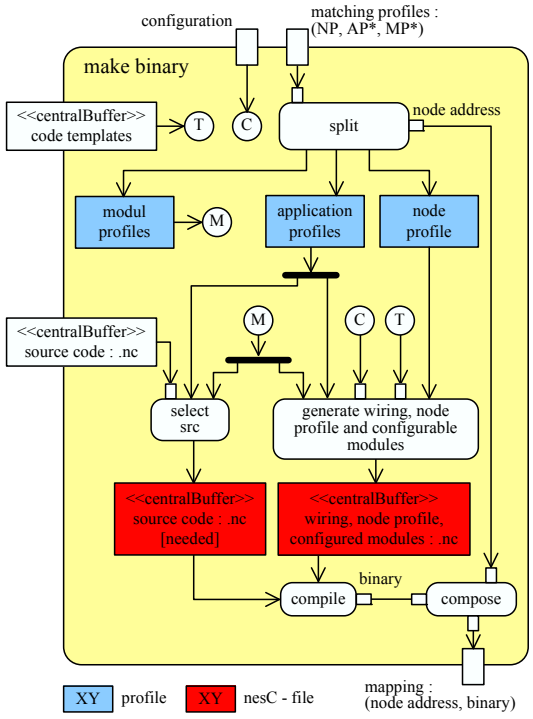

Fig. 2. Activity diagram for code generation.

`<split>` extracts the information of the profiles and provides it for further processing. `<select src>` selects the source code, which is described by the *AP*s and *MP*s (please remark the unique mapping) and puts it into a temporary buffer. `<generate wiring, node profile and configurable modules>` generates the dynamic nesC files depending on the current configuration and the different combinations of *AP*s and *MP*s, and puts them into another temporary puffer. Code templates can be used to provide generic functionalities or static segments that are node-specific and must be prepared appropriately before compiling. `<compile>` compiles all the nesC files. `<compose>` maps the resulting binary with the corresponding node address.

An example is depicted in figure 3. Necessary code fragments, i.e. software modules that do not need further adaptation, are compiled to the final sensor application. A special fragment is the base system. Similar to a middleware solution, it provides necessary standard functionality such as the algorithms for profile exchange and network-based node reprogramming. Additionally, code templates can be used representing code that must be adapted according to the local needs. For example, sensor calibration can take place by adapting reference values in such code templates.

In the system that we developed in our lab [26], code fragments for TinyOS programs are written in nesC. Specific profiles as discussed previously are connected to these fragments in order to describe functionality and utilization. In order to generate a binary that runs on the nodes described by its node profile, corresponding nesC modules are extracted

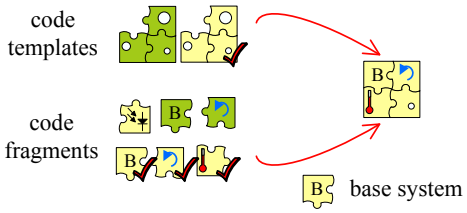

Fig. 3. Code selection and complication.

from the software repository on the dedicated server and provided to the compiler. The structure of TinyOS programs requires some additional handling in combination with the selection of source files. First, the wiring between the modules must be defined. Based on the available descriptions, templates can be used for an unambiguous wiring. Secondly, some parts of the nesC code have to be adapted to different hardware configurations. We also allow to generate nesC code on demand using code templates. Such templates are filled with variables and algorithms depending on the current context, i.e. the environmental conditions. This procedure can be used to calibrate sensor readings. A template and a configuration defined by a profile will be substituted to a configurable software module that is adapted to a particular hardware configuration. In a final step, the node profile is transformed to a nesC file that can be compiled to a new binary. This binary reflects the application profile and corresponds to the actual hardware capabilities.

IV. Assistance

Software updates independent of their granularity as well as the technique to apply them are performed in a push-based manner as new information denoted by software or policies is deployed onto nodes trigged by external entities like developers or researchers. In some scenarios not all demanded functions can be stored on a node at the same time. This might be the case, if the required function set is very large or more likely a consequence of software evolution. In the later case further development of software obviously leads to an extended functions set that usually requires more storage. Sensor networks are thought to be deployed in less developed environments like rain forests, glaciers, or other inaccessible or unpredictable environments as motivated by several habitat monitoring applications. Additionally, such networks will have an operation time of at least several hours up to years that will rise in the near future. Therefore, we believe software evolution will become a key factor.

To address this development and to ensure future extensibility and adaptiveness of sensor software, we anticipate—without the extensive application of hardware—pull-based extension mechanisms have to be provided. A possible solution offers an assistance concept that dynamically utilizes functions provided by neighboring nodes. The basic idea behind such a mechanism is the observation that some functions are seldom needed and therefore do not have to be permanently located at each demanding node. Instead, these functions can be located at different nodes that are contacted during runtime.

An assistance-concept has to be provided with very low overhead and in general, it should be, at least during implementation time, transparent to the developer. So the development process remains unchanged. Instead, at deployment time it has to be determined if the target system can handle the whole application and system software at its full function set. If this is not the case, a semiautomatic approach composed by code analysis and developer knowledge has to identify possible candidates for relocation. The code analysis thereby identifies code fractions that are self-contained in a sense that only some other parts of the code access these fractions. Then, the developer has to decide whether a candidate can be relocated, as she should know whether there are timing constraints, or if the function is often accessed so an externalization is not suitable. In a second step, the selected candidates are removed from the binary and replaced by very small code sequences acting as stubs. Of course this is only one way to implement the identification process that targets late stages of development and deployment. Other ways to tackle this problem might be the introduction of annotations at development time to mark possible relocation candidates or other approaches targeting earlier stage of development like custom programming constructs.

Once a function is relocated to some other node and it is invoked at its original target location a stub is executed and the assistance-concept takes over execution. In general, we anticipate two kinds of assistance: remote execution and on-demand update. In the first case, similar to a traditional remote invocation system, parameters are serialized and transferred to a neighboring node. In most cases, these functions are assumed to be stateless so an arbitrary node has to be found offering the required functionality. If the function is state-full, additional actions have to be taken. This includes not only the transfer of call parameters but also of extended context information. In case of on-demand updates, the removed function is redeployed at runtime so that not a service providing node is required but a node that offers a suitable implementation. Depending on the function and its dependencies, it is integrated by one of the deployment approaches that have earlier been outlined. Of course this might require to replace some of the currently hosted code. In general, a resource management concept at the level of code layout and deployment is required that takes not only only single nodes into account but also multiple neighboring nodes forming bigger entities. Both kinds of assistance require nodes that provide the dynamic lookup and selection of code or functionality as it can not be expected that suitable nodes are known at deployment time.

Taking this approach one step further, it might be reasonable to widen the scope not only to the management of storage but also other resources like memory or energy. For example, nodes that have less remaining battery power or are known to have greater energy consumption can explicitly outsource functions to other nodes. Candidates for such an approach might be nodes that due to topological reasons take special

roles in scope of the routing process and therefore have greater energy demand.

V. Research Challenges

As outlined in this paper, we assume that sensor networks will face heterogeneity in terms of software and employed hardware. This leads to extended complexity concerning software development, deployment, and, finally, management. Together with other limitations and requirements, we identified the following research challenges for software management in sensor networks:

- *Code maintenance*. How do we have to store code particles and code fragments? How can we identify features of application and system code?
- *Generation of executables*. How do we map code to heterogeneous hardware? How can we include node specific parameters, e.g. for calibration issues?
- *Reprogramming*. How can we transport (parts of the) executables to the dedicated sensor nodes? How can we ensure reliable reprogramming?
- *Assistance functions*. How do we define helper systems for fault tolerance? How can we spread functionality over multiple sensor nodes?

As mentioned before, sensor network applications and wireless sensor networks themselves must deal with several constraints while achieving their goals. These constraints arise inherently due to the nature of either wireless networks or mobile ad hoc networks:

- *Mobility of nodes*. Commonly it is believed that sensor networks being stationary, nowadays, mobility is a mayor concern.
- *Size of the network*. Size has much larger impact compared to infrastructure networks.
- *Density of deployment*, ranging from very high to quite sparse, application domain dependent.
- *Energy constraints*, are much more stringent than in fixed or cellular networks, in certain cases the recharging of the energy source is impossible.

Considering these constraints, the research challenges as stated before can be mapped to particular requirements that are usually present in the context of WSN. Table I summarizes this mapping.

Several approaches exist to address the discussed heterogeneities as well as the typical limitations and restrictions in sensor networks. The first category concentrates on software engineering techniques and comprises the following recommendation:

- Software for sensor networks should be implemented using software engineering techniques for modularization, thereby leading to better manageability but of course also extensibility. Possible candidates are as proposed feature models but also the application of aspect-oriented programming (AOP) techniques seems to be promising.
- Rigorous adaptation of software engineering techniques to the demands of the sensor network domain. This includes, for example, extended tool support to map feature changes to small and specialized deployment units or more general to support the selection of a deployment method and strategy.

Modularly implemented WSN software builds a useful foundation for the efficient and resource aware deployment as code changes and updates can be clearly identified. However the deployment of custom and maybe node specific software onto heterogeneous hardware has certain extended requirements:

- Software deployment has to be efficient in terms of network utilization, storage, and processing requirements as sensor networks are in most cases resource restricted. Heterogeneity amplifies this challenge and existing deployment techniques have to be verified for their usability, this may lead to completely new approaches primarily targeting heterogeneity.
- Reliability is a crucial point for the deployment of software as failed updates may cause unresponsive nodes, that at the extreme can destabilize a whole network. Here, heterogeneity introduces even more pitfalls, so mechanism have to be considered that ensure a save and robust deployment process.

Last but not least, many (if not the majority of) use cases are faced with the situation where application demands exceed the resources provided by individual sensors. A graceful handling of such situations demands for extended technologies that have to be tightly coupled with the proposed software engineering technologies and deployment mechanisms:

- The provision of these techniques like remote execution or on-demand installation itself poses a challenge if performed in a resource and efficient manner.
- Remote execution as a basic technology becomes useless if required services are not provided by neighboring nodes or at least sensors that can be reached within a short distance. Ensuring such properties demands for node spanning resource management, supported by offline planning as part of the deployment process.

A big challenge is to get all these approaches integrated to provide users with a *workbench* supporting development, evolution, and maintenance of forthcoming WSN.

VI. Conclusion

In this paper, we discussed the need to robust and efficient software management solutions for use in wireless sensor networks. Above all, the motivation for new approaches and solutions arises due to emerging hardware and software heterogeneities. Such differences are caused by the need to use one WSN installation for multiple purposes depending, for example, on the current time or environmental conditions. Limitations of sensor nodes in terms of resource restrictions (memory, storage, processing) do not allow to install all possible software modules to all available sensor nodes. Thus, heterogeneity is even further increased.

We identified several research challenges in the domain of sensor network software management and node reprogramming. We also proposed three different approaches that can

	code maintenance	generation of executables	reprogramming	assistance
reliability	Yes	-	Yes	Yes
security	-	-	Yes	Yes
network utilization	Yes	-	Yes	Yes
system resource utilization	Yes	Yes	-	Yes
reprogramming performance	Yes	Yes	Yes	-
fault tolerance	Yes	-	-	Yes
hardware heterogeneity	Yes	Yes	Yes	Yes
software heterogeneity	Yes	Yes	-	Yes

TABLE I

MAPPING OF RESEARCH CHALLENGES TO TYPICAL APPLICATION REQUIREMENTS IN WSN

help to address the discussed challenges. First, feature models help to identify relationships between different modules and to (semi-) automatically select the functions that are really needed in a particular scenario. Secondly, a profile-matching based technique to select code fragments as well as code templates for generating executable binaries eases the node and application specific code generation. Finally, assistance models allow to spread functions over a set of available nodes in order to either enhance the fault tolerance or to distribute applications to multiple nodes.

REFERENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, pp. 393–422, 2002.

[2] D. Culler, D. Estrin, and M. B. Srivastava, "Overview of Sensor Networks," *Computer*, vol. 37, no. 8, pp. 41–49, August 2004.

[3] I. F. Akyildiz and I. H. Kasimoglu, "Wireless Sensor and Actor Networks: Research Challenges," *Elsevier Ad Hoc Network Journal*, vol. 2, pp. 351–367, October 2004.

[4] Y. Guo, P. Corke, G. Poulton, T. Wark, G. Bishop-Hurley, and D. Swain, "Animal Behaviour Understanding using Wireless Sensor Networks," in *31st IEEE Conference on Local Computer Networks (LCN): 1st IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006)*, Tampa, Florida, November 2006, pp. 607–614.

[5] A. Baggio, "Wireless sensor networks in precision agriculture," in *ACM Workshop on Real-World Wireless Sensor Networks (REALWSN 2005)*, Stockholm, Sweden, June 2005.

[6] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks," in *ACM/IEEE International Conference on Mobile Computing and Networking*. Seattle, Washington, USA: ACM, August 1999, pp. 263–270.

[7] G. Fuchs, S. Truchat, and F. Dressler, "Distributed Software Management in Sensor Networks using Profiling Techniques," in *1st IEEE/ACM International Conference on Communication System Software and Middleware (IEEE/ACM COMSWARE 2006): 1st International Workshop on Software for Sensor Networks (SensorWare 2006)*, New Dehli, India, January 2006, pp. 1–6.

[8] C.-C. Han, R. Kumar, R. Shea, and M. Srivastava, "Sensor Network Software Update Management: A Survey," *ACM International Journal on Network Management*, vol. 15, no. 4, pp. 283–294, July 2005.

[9] C. Margi, "A Survey on Networking, Sensor Processing and System Aspects of Sensor Networks," University of California, Santa Cruz," Report, February 2003.

[10] D. Culler, J. Hill, P. Buonadonna, R. Szewczyk, and A. Woo, "A Network-Centric Approach to Embedded Software for Tiny Devices," in *First International Workshop on Embedded Software (EMSOFT 2001)*, Tahoe City, CA, USA, October 2001.

[11] V. Handziski, J. Polastrey, J.-H. Hauer, C. Sharpy, A. Wolisz, and D. Cullery, "Flexible Hardware Abstraction for Wireless Sensor Networks," in *2nd European Workshop on Wireless Sensor Networks (EWSN 2005)*, Istanbul, Turkey, February 2005.

[12] B. Hurler, H.-J. Hof, and M. Zitterbart, "A General Architecture for Wireless Sensor Networks: First Steps," in *4th International Workshop on Smart Appliances and Wearable Computing*, Tokyo, Japan, March 2004, pp. 442–444.

[13] D. Martin, A. Cheyer, and D. Moran, "The Open Agent Architecture: a framework for building distributed software systems," *Applied Artificial Intelligence*, vol. 13, no. 1/2, pp. 91–128, 1999.

[14] A. Chlipala, J. Hui, and G. Tolle, "Deluge: Data Dissemination for Network Reprogramming at Scale," Tech. Rep., 2004.

[15] C. Frank and K. Römer, "Algorithms for Generic Role Assignment in Wireless Sensor Networks," in *3rd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, San Diego, CA, USA, November 2005.

[16] M. Gauger, "Dynamic Component Exchange in TinyOS (Dynamischer Austausch von Komponenten in TinyOS)," Master's Thesis (Diplomarbeit), University of Stuttgart, April 2005.

[17] J. Jeong and D. Culler, "Incremental Network Programming for Wireless Sensors," in *First IEEE International Conference on Sensor and Ad hoc Communications and Networks (IEEE SECON)*, June 2004.

[18] K. Czarnecki and U. W. Eisenecker, *Generative Programming—Methods, Tools, and Applications*. Addison-Wesley, 2000.

[19] D. Beuche, H. Papajewski, and W. Schröder-Preikschat, "Variability management with feature models," *Science of Computer Programming*, vol. 53, no. 3, pp. 333–352, Dec. 2004.

[20] X. Bai, S. Kumary, D. Xuany, Z. Yunz, and T. H. Lai, "Deploying Wireless Sensors to Achieve Both Coverage and Connectivity," in *Seventh ACM International Symposium on Mobile Ad Hoc Networking and Computing (ACM MobiHoc 2006)*, Florence, Italy, May 2006, pp. 131–142.

[21] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. B. Srivastava, "Coverage Problems in Wireless Ad-hoc Sensor Networks," in *IEEE Infocom 2001*, Ankorange, Alaska, USA, April 2001.

[22] B. Liu, P. Brass, O. Dousse, P. Nain, and D. Towsley, "Mobility Improves Coverage of Sensor Networks," in *ACM MobiHoc 2005*, Urbana-Champaign, IL, May 2005.

[23] M. A. Batalin and G. S. Sukhatme, "Coverage, Exploration and Deployment by a Mobile Robot and Communication Network," in *International Workshop on Information Processing in Sensor Networks*, Palo Alto, USA, April 2003, pp. 376–391.

[24] V. Mhatre, C. Rosenberg, D. Kofman, R. Mazumdar, and N. Shroff, "A Minimum Cost Heterogeneous Sensor Network with a Lifetime Constraint," *IEEE Transactions on Mobile Computing*, vol. 4, no. 1, pp. 4–15, January 2005.

[25] G. Xing, X. Wang, Y. Zhang, C. Lu, R. Pless, and C. Gill, "Integrated Coverage and Connectivity Configuration for Energy Conservation in Sensor Networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 1, no. 1, pp. 36–72, August 2005.

[26] Z. Yao, Z. Lu, H. Marquardt, G. Fuchs, S. Truchat, and F. Dressler, "On-demand Software Management in Sensor Networks using Profiling Techniques," in *ACM Second International Workshop on Multi-hop Ad Hoc Networks: from theory to reality 2006 (ACM REALMAN 2006), Demo Session*, Florence, Italy, May 2006, pp. 113–115.