Sens4U: Wireless Sensor Network Applications for Environment Monitoring Made Easy

Krzysztof Piotrowski
IHP
Frankfurt(Oder), Germany
Email: piotrowski@ihp-microelectronics.com

Steffen Peter
Center for Embedded Computer Systems
University of California, Irvine
Email: st.peter@uci.edu

***Abstract*—The development of wireless sensor network (WSN) or cyber physical systems (CPS) applications is a complex and error prone task. This is due to the huge number of possible combinations of protocols and other software modules, to choose from. Additionally, testing of the chosen configuration and the individual software modules is not a trivial task, especially in case where they are all implemented from scratch. The aim of the Sens4U methodology we present in this paper is to simplify and possibly automate the process of building a WSN application and to simplify its testing. The main idea of our approach is to exploit the modularity of the available libraries in order to speed-up application development done by non-WSN-experts and to solve the real-life problems. The proposed abstraction is very powerful–the modules provide specific functionalities via defined interfaces and can be connected using these according to the application requirements, to create the desired and minimum target configuration. The modularity improves the testability and reuse of components and thus, their reliability and, as a result, the reliability of the target configurations. Further, the Sens4U approach goes beyond pure software generation and supports creating software and hardware configurations. We are currently focusing on environment monitoring scenarios in order to analyze this problem area in the semi-automatic computer aided application logic generalization process. This paper presents the general concept as well as the tool chain that supports the application development done by non-WSN-experts.**

I. Motivation

Developing a wireless sensor network (WSN) or a cyber-physical system (CPS) application is not a trivial task. And this remains true, even if the WSN applications seem simple, i.e., they usually collect some data, process it and perform some actions or deliver the processing results to some specific nodes in the network. However, implementing the distributed application logic and choosing the right data processing algorithms, like network protocols or aggregation algorithms, requires knowledge on WSN programming and a lot of testing effort. To address this issue, the WSN operating systems, like TinyOS [1] or Contiki [2], usually provide kind of modularity and a library base, so that some of the required functionalities do not have to be programmed from scratch. But unfortunately, the persons responsible for application requirement specification usually do not have the low-level WSN knowledge and thus, converting the application needs into the right set of functionalities, is complicated, even if the most of the functionalities are available in the library. Additionally, choosing one module from several providing the same functionality is not a trivial task. Thus, in order to simplify the application development process, there is a need for two stages to support the users. First is the transformation of the non-technical requirements into a set of technical ones and the second is the choice of most suited modules providing the required functionality, based on the requirements. The aim of the Sens4U approach is to address both these issues and to provide a solution that can satisfy the real-life monitoring application requirements defined by the non-technical project partners working in the area of hydrology and biology.

The development of wireless network applications is usually an activity of a closed group of WSN experts. Thus, the aim of our approach is to enable the WSN application development for a wider group of users and especially for the end users of the target application in order to increase their involvement and to improve the quality of the applications, while reducing the development time and costs. We have chosen to support the environment monitoring scenarios as we have experts from this area in the project consortium. These applications allow focusing on their problem space and allow rapid testing of the initial results, but we are convinced that our approach can cover other application areas as well. The goal is also to generate WSN monitoring systems that simply help their users to reduce the effort related to their, sometimes obligatory, monitoring activities.

Taking an example from the C programming as analogy, the transformation of a set of non-technical requirements into technical ones can be compared to a type cast of a void variable onto some given type, like integer. Without the knowledge what can be inside the variable, the result of the cast operation may be more or less successful (or meaningful). Thus, it is reasonable to define the application area frame to be covered by the non-technical requirements, to simplify the transformation task. The advantage of this limitation is that for the limited application area coverage it is possible to define more precise mappings that allow to transform the non-technical application requirements into technical ones more reliably. Following this strategy, the supported application area coverage can be extended by adding new mappings for additional kinds of applications.

The format for the technical requirements derived from the first stage has to be specified in a clear and comprehensive way, so that they can be further transformed into a set of required functionalities and a set of parameters that describe these functionalities. These two sets form the technical extract of the application requirements and are the input for the second stage–the automatic component choice. Here, the most suited components are chosen from the available ones and a list of configurations is created, based on the required functionalities and their parameters. The generated configurations can be ordered by their ranking values that describe how good they fulfill the initial user requirements. Thus, it is very important to provide means to define the allowed connections between the available modules and the ways to judge which component out of several providing the same functionality fulfills the requirements the best. It is also very important to provide means to test if the components provide the promised functionality and quality.

The remaining part of the paper is structured as follows. The following section presents similar approaches that introduce modularity and configurability in WSN applications. After that we present the Sens4U concept followed by the application examples. The paper concludes with the outlook and the future work.

II. Related Work

Research literature is rich on approaches to address the problem space defined in this paper, but most approaches focus on support for the programmer instead of supporting the potential end-user of the systems. However, some of these ideas can be used as inspiration for solutions to be part of the complete approach.

TinyOS [1] with NesC and its concept of modules and interfaces is a significant example of using modularity and reusability in WSN application development. Defined interfaces describe the functionalities and the access methods to these. And interfaces are either used or provided by modules, allowing generating configurations built of modules to represent the software applications. But in the pure NesC, there is no means to judge between two modules that provide the same interface, i.e., which one suits better for the specific application. A variety of other composition methods for WSNs is discussed in [3]. The approaches range from monolithic approaches, such as plain C code, compiled for a certain hardware, over modular approaches, such as Contiki [2], static component models, such as NesC [4], dynamic component models, like openCOM [5], up to service oriented approaches, like MILAN or MiSense [6].

As advanced example of a service oriented approach, REMORA [7] provides high-level and event-driven programming in WSNs through a component-based abstraction. It is a lightweight framework executable on every system software written in the C language. Lorien [8] is another dynamic component composition framework. These frameworks provide valuable contribution for runtime adaptation of WSN software. However, they do not deliver guidelines or methods to chose components that are suitable for the application. They also do not consider hardware variations.

[9] is an event-based publish-subscribe component approach. They follow the publish-subscribe notion on network level, but also inside the node. On network level nodes can provide or use services from other nodes in the network. The authors demonstrated it in a health-net application scenario. Inside the node, specific services, such as tracking or security protocols, can be attached to the middleware. However, the actual middleware and the network protocols are fixed what significantly limits the flexibility.

Hardware support becomes increasingly important. Application of FPGAs to accelerate specific algorithms or to reduce energy consumption is a feasible design option [10]. A component framework for hardware description languages has been proposed in [11].

GRATIS++ (Graphical Development Environment for TinyOS) provides a set of sophisticated graphical tools for the development of TinyOS-based applications. In the latest version hardware and software modules are uniformly interchangeable. Within the hardware/software design space exploration, alternative implementations of particular components can be automatically evaluated and proposed to the user. Beside the graphical representation of TinyOS code, the main advantage of GRATIS is the ability to create and maintain models from real nesC code. Contrary to classical interface description and compatibility checking, GRATIS does not only check static typing of the interfaces, but also respects dynamic aspects of the component interfaces. GRATIS emphasizes interface specifications and thus, is a great help for developers of TinyOS applications. Using the models and linked module source codes GRATIS can also generate actually compilable system code. However, since GRATIS does not model the actual behavior of the modules, it does not support the component selection process. Further, it is limited to TinyOS operating system only.

An important aspect when it comes to user-friendly design is the questions how to derive technical requirements from user needs. In [12] proposed a requirement definition process in which the user can select and define attributes for the intended network out of a given catalog of parameterizable properties. These rather high-level attributes can be then translated into technical attributes following an inference logic.

The application of feature models, which allow users to chose feature combinations as proposed in [13] and [14] are an option to provide improved accessibility to the composition process of WSNs. Users can step-by-step define what features are required and parametrize them, while the underlying model solves dependencies and adds or removes options in accordance to the user's selection. [14] showed this approach for a configurable middleware, while [13] targets the composition of the entire application.

Srijan [15], is a graphical toolkit that uses a data-driven macro-programming back-end. Based on an application task graph and some imperative code for specific functions, it automatically generates system structure and program classes that can be customized by the developer. That way the graphically-controlled macro-programming framework organizes data flow

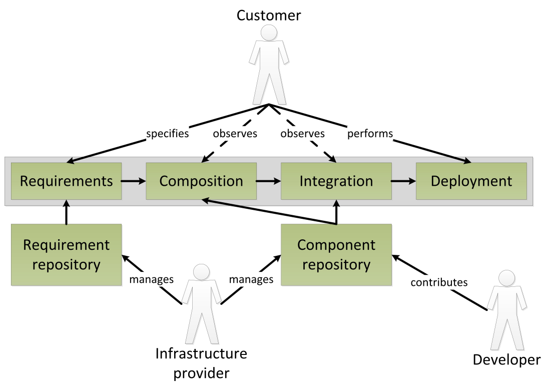

Fig. 1. Component-based development flow and the main user roles

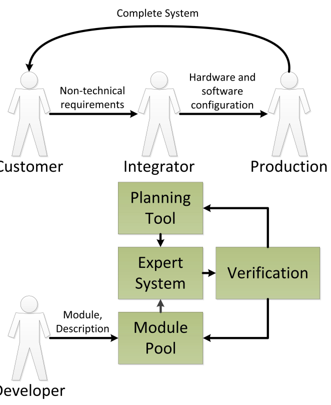

Fig. 2. The detailed tool-chain-oriented development flow and user roles

on the node and in the network, while the developer has to fill in the function blocks.

EBlocks [16] is a graphical tool that promises to allow creation of WSN systems even for non-technicians. The idea of eBlock is to describe rather high-level blocks that can be assembled in a GUI. All eBlocks should be self-explanatory, which needs no training or tutorials. This is also by the extensive graphical development environment that illustrates each block with individual pictures, such as a lamp for a LED. eBlocks comes with a code generator and a simulator that allows testing the behavior of the system. Unfortunately eBlocks is only able to express functional behavior, i.e., QoS, performance or security aspects are not supported.

It is the predominant assumption that performance, energy and memory footprint improve with lower abstraction, while only a few works exist to quantify this effect. Porter et. al. [17] presented a comparison of static and dynamic component models with the result that the dynamic platform needs about 2KB more RAM, 10KB more ROM and a duty cycle overhead of up to 100% (In fact, they argue that on the given scenario the idle time drops from 99.4% to 98.8%).

Each of these above mentioned examples only partially covers the problem space we have defined for our approach. None of them supports both hardware and software composition together with the simplified application logic or data flow definition for non-WSN-experts, as our approach does. Further, our approach aims at defining a powerful module abstraction with testability support. It also combines the low level performance with high level simplicity.

III. The Sens4U concept

This section describes the details of our approach. First, we sketch the design flow and the roles of the users as introduced in [18] and extend this description.

a) Design Flow and User Model: Figure 1 shows the general flow of the intended component-based composition-driven design process as described in [18]. The process steps focus on the actual development process performed by the customer, who can execute the process without expert support. Participants in this process are the customer, the infrastructure provider and the (component or module) developer. The **customer** is the person or organization who will compose and deploy the sensor network. The **infrastructure provider** sets the infrastructure and also chooses and manages the set of available domain-specific components and supported requirements. The **developer** provides the implementation and description of the basic building blocks–modules.

The process starts with the definition of requirements, done by the customer. These are chosen from the **requirements repository**, which contains a set of selectable and parameterizable requirements. In the following step–the composition–the components are selected and assembled to form a system that promises to satisfy the customer's requirements. The selection and assessment process employs the **component repository** containing models of components (modules) together with their properties and implementations. Based on the resulting configuration, the actual system is integrated–compiled and assembled. Finally, the resulting sensor nodes are programmed with the resulting code images and are deployed at the application site.

The vision is that ultimately customers only define the requirements and finally perform the physical deployment, while composition and integration are executed automatically. We want to realize this vision by applying the now proposed development tools (see Figure 2) that implement the above sketched concept and are going to be realized within the Sens4U project. In the new, tool-chain-oriented flow, we explicitly split the customer expertise into the application domain expertise owned by the actual **customer** and the, at least basic, WSN expertise owned by the **integrator**. The aim here is to provide the real-life applicability of the approach and thus, to introduce the integrator role to support the customer in requirement specification.

The customer explains the target application to the integrator and together they identify the most important features of the application that are further provided to the **planning tool** as input. They specify the non-technical requirements–the target deployment area coverage, the kind of communication obstacles in that area, the required sensing capabilities, the spacial placement of these measurement units, the required data flow and processing, together with the required security,

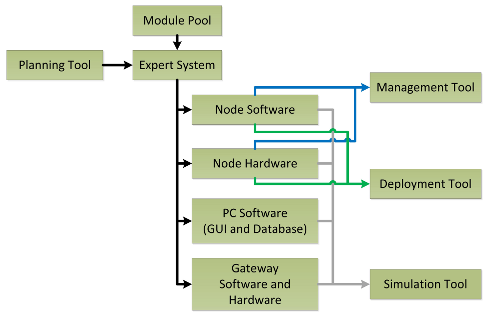

Fig. 3. The tool chain and the split of output configuration data

reliability and maintainability features. The planning tool generates the set of technical requirements containing the required functionalities and their required parameters that are further forwarded to the **expert system**. Here, the hardware and software configurations are generated, based on the available modules in the **module pool** and the above mentioned extract from the technical requirements.

The configurations can be verified in simulation before sending them for production. In case of negative verification result or in case of negative composition result the actions can be twofold, the application requirements provided to the planning tool may be relaxed or a request for developing new modules may be issued to a developer, e.g., if none of the available modules provides some functionality with the requested parameters. Of course, these actions define a trade-off between the quality and costs of the final application.

The expert system generates a set of configurations, each with a ranking score specifying the extent of requirement fulfillment. The customer can choose one of these configurations. Accepted configuration is sent for production, i.e., the hardware is produced and programmed with the software. A final configuration consists of several parts, describing the subsystems of the application (see Figure 3). Currently, due to the broad coverage of hardware and software in our abstraction, we do not define the detailed specification on the production process. For that reason we introduced the **production** role. It represents all the actions necessary to prepare the output configuration data for production. And this actions can be further automated as well.

During the deployment, the sensor network can be further verified with respect to communication conditions by using the **deployment tool**. This tool analyses the actually achieved network topology and the quality of links between individual nodes in the sensor network. If the quality of links is not sufficient, additional repeater nodes may be installed to improve the connectivity. The deployment tool requires some parts of the configuration, in order to be able to communicate with and evaluate the deployed network (see Figure 3).

After the deployment, the network can be monitored and managed using the **management tool**. This tool allows monitoring all the parameters of each individual sensor node in the network as well as changing their parameters on-the-fly. This tool also allows forcing special conditions to verify the network robustness as well as allows fine tuning the already deployed nodes, e.g., to reduce their energy consumption or to increase the transmission power.

b) The Module concept: Practically, a module in the pool is an abstraction of (an implemented) functionality or set of functionalities. And each functionality is defined by an interface that specifies the ways the functionality is to be used together with the parameters describing the functionality. Thus, a module is a self-contained building block with a coherent functionality and well defined interfaces so that it can be deployed independently and is subject to composition by third parties. Modules can represent software modules, such as functions, implemented classes, implemented algorithms, or services, but also hardware modules with diverse granularity, including hardware platforms (sensor nodes), hardware components (chips, sensors, memory extensions), or implementable hardware description modules (HDL). So, the granularity of a component can be very fine (one mathematical operation) or coarse (an entire platform). It is also allowed that modules combine other modules together and by that abstract some larger functionalities. All components are stored in the module pool/component repository.

As already mentioned, each interface has its associated parameter set that further specifies the detailed attributes of this functionality provided using this interface. This parameters are specified while defining the interface and they specify the qualitative and quantitative aspects of the functionality implemented by a given module, i.e., the values of the parameters are module related. If a module provides an interface, it has to have defined values of all parameters associated with that interface. This allows comparing several modules that provide a given functionality to choose one that suits the requirements the best.

As part of the system assembly process, modules are aggregated, in order to combine their functionalities into the target system. Similar to the well-known composition concepts [5], [4], modules are composed by connecting their interfaces. A module that provides a specific interface may be connected to another module that requires the same, or a compatible, interface. This implies that both modules can communicate via this shared interface.

The association of interfaces and parameters also helps to improve the testability of modules. The quality of the functionality a module promises can be verified before using a module in a target system. This is the task of the infrastructure provider to verify the quality of modules that are added to the component repository/module pool. Additionally, one module that uses an interface can be configured according to the values of the parameters of a second module providing the interface, before these two modules are integrated.

The result of the composition is a structural description of the system in form of a graph of components connected via interfaces. This graph is the blueprint for the system integration. An example application graph is shown in Figure 4.

c) The application logic: One of the most tricky problems in the semi-automatic application development for non-

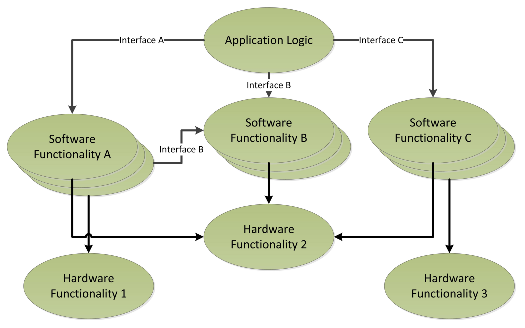

Fig. 4. The graph representing an example modular application

experts is the abstraction of the distributed application logic and the required data flow in the target application. Depending on the required WSN expertise there are several solutions, starting from implementing the application logic from scratch. More elegant solutions for non-experts can be realized by some kind of macro-programming, either by some scripting means or by composition of modules available in the module pool, representing the chosen sensors and specific operations on the data they generate. As a natural consequence of the modular character of our concept, we decided to choose the latter solution. Thus, the application is also constructed using modules that are connected with each other via interfaces. Further, we plan to employ our data handling middleware tinyDSM [19] for data sharing and processing. The middleware module will be configured according to the customer requirements and connected to the sensor modules that provide the measured data to be processed.

IV. The application area

In this section we outline how our design flow can be applied to two scenarios in the domain of environmental monitoring. In the project it was intended that biologists, geologists and hydrologists could define and set up the WSNs required to accomplish their tasks. It was a first goal to analyze the complexity of the transformation of the non-technical requirements into technical ones.

As this transformation mainly focuses on specifying the defined features of the application, we want to identify all the features required to describe applications from the chosen application areas. The result of this study defines the application description framework that we will validate and further improve with application examples that do not fit into the current framework, if we find those.

Thus, as already mentioned, the methodology tries to define frames into which the customers have to put their application requirements in order to generate the set of technical requirements. In any case, currently this process of requirement specification is not automatic, it has to be done by a human, i.e., the customer who knows the requirements for the monitoring application. This person can be supported by the integrator, who knows about the defined methodology for WSN application development, but not necessarily is an expert in low level WSN development.

a) Hydrologic and geologic monitoring: We have chosen two environment monitoring scenarios in order to catch their most important aspects. The first scenario is the hydrology and geology monitoring example that involves a large number of sensor nodes covering a large monitoring area, where each sensor is equipped with relatively similar and small set of sensors. The aim of the application is to monitor the chosen area with respect to the hydrological and geological parameters from the air and soil. The sensors generate small data pieces (measurements), but relatively frequent and the data from all sensor nodes has to be provided live with small delay to the operator in front of the PC. Further, the temporal correlation of the individual measurements is important. Other important aspects to be defined are the way the data is consumed and the networking requirements. The first specifies, if the data is pre-processed within the sensor network or if it is provided in raw to the central station. The networking requirements specify the way the data is transported between the nodes.

Further, the security and energy aspects like application lifetime or inter-maintenance period as well as the distances between nodes are to be defined.

b) Biologic monitoring: The second scenario is much different. The aim of the application is to monitor the number of bats living in their shelters and to identify the species. This task done by people requires lots of effort and special training, i.e., every place where the bats are located has to be observed by a human, who is waiting several hours during the night for the bats to fly out of their stay and counts them at that moment. The things are even more complicated, if there are several co-located places, where the bats stay. In this case it may happen that the bats migrate from one shelter to another and thus, each location has to be observed by one person. A sensor network can be deployed, to simplify this task. There are three main sensor kinds in this scenario; the motion detector, the camera to take the video of the flying bats and the ultrasonic recorder to record their sound for identification. As the recorded data is not transmitted over the sensor network, the two latter sensor kinds can be actually regarded as actuators, as they are triggered by the nodes they are attached to.

The sensor nodes equipped with the motion detectors trigger the nodes equipped with the video cameras and ultrasonic recorders to switch them on, enable the recording for several minutes and turn them off again. It is enough to send a notification to the user that a recording was triggered and that the data is available on the memory cards in the recording devices. Depending on the deployment site, there may be several motion detectors and several cameras and ultrasonic recorders in the network. It is necessary to specify their dependencies, i.e., which motion detector controls which actuators.

The biology monitoring application scenarios involve a small amount of sensor nodes, each with an individual set of sensors. In such scenarios, the right correlation of the sensor values that represent some phenomenon and their temporal correlation are the most important aspects. In the bat monitoring scenario each phenomenon is captured by a single sensor, i.e., the camera captures the number and the ultrasonic recorder identifies the species.

Again, the usual aspects like the security and energy aspects like the required application lifetime as well as the distances between nodes have to be considered.

c) First conclusions: The results of our first analysis of the abstraction possibilities in the monitoring applications are very promising. Actually, almost every environment monitoring activity can be realized by a set of sensors, that provide data to be either transmitted (on-line) or to be analyzed afterwards (off-line). The main difficulty is to express the data correlation, processing and flow in a way that it can be used by non WSN experts. But exactly this issue can be addressed by introducing the role of the integrator in the application development, i.e., the basic knowledge on the application description is not required from the customers.

V. Outlook and future work

This paper presents the current state of the research on the methodology and tools to simplify and automate application development for environment monitoring that is realized in the Sens4U project. The aim of the project is to provide a working demonstrator as a proof of concept, based on the mentioned application areas.

The core **contribution** of this paper is the introduction of a component-based design flow for WSNs that:

- supports the non-WSN-experts in the specification of they real-life target system requirements,
- provides a module description framework to support computer aided application development and module testing,
- allows creating application specific/optimized software and hardware configurations,
- does not only suggest software compositions, but also suggests hardware components with diverse granularity (platform, chip, module) and functionality (accelerators, sensors, actuators) to fulfill the given task,
- utilizes an user and process model that emphasizes the applicability by non-WSN-experts.

By this, our approach goes beyond pure software development. Further, this concept, due to the powerful abstraction and the tool chain can be easily applied for embedded system application development in general.

In the next steps we are going to finalize the methodology for the requirement transformation rules and investigate algorithms to generate configurations according to the given requirements as well as the ways to rate the resulting configurations.

We will also further investigate the ways to simplify the definition of the application logic. This includes the adaptation of the tinyDSM middleware in order to obtain a configurable module in the pool.

Further steps involve analysis of other application scenarios and areas to extend and verify the methodology.

Acknowledgements

This work was partially funded by the German government under grant 03WKP26A and by the National Science Foundation under NSF grant number 1136146.

References

[1] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer *et al.*, “Tinyos: An operating system for sensor networks,” *Ambient Intelligence*, pp. 115–148, 2005.

[2] A. Dunkels, B. Grönvall, and T. Voigt, “Contiki - a lightweight and flexible operating system for tiny networked sensors,” in *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, Tampa, Florida, USA, Nov. 2004.

[3] K. Wan, D. Hughes, K. Man, and T. Krilavicius, “Composition challenges and approaches for cyber physical systems,” in *Networked Embedded Systems for Enterprise Applications (NESEA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–7.

[4] D. Gay, P. Levis, R. Von Behren, M. Welsh, E. Brewer, and D. Culler, “The nesC language: A holistic approach to networked embedded systems,” in *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, 2003.

[5] G. Coulson, G. Blair, P. Grace, F. Taiani, A. Joolia, K. Lee, J. Ueyama, and T. Sivaharan, “A generic component model for building systems software,” *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 1, p. 1, 2008.

[6] K. Khedo and R. Subramanian, “A service-oriented component-based middleware architecture for wireless sensor networks,” *Int J Comput Sci Netw Secur (IJCSNS)*, vol. 9, no. 3, pp. 174–182, 2009.

[7] A. Taherkordi, F. Loiret, R. Rouvoy, F. Eliassen *et al.*, “Optimizing sensor network reprogramming via in-situ reconfigurable components,” *ACM Transactions on Sensor Networks*, vol. 9, no. 2, pp. 1–37, 2013.

[8] B. Porter and G. Coulson, “Lorien: a pure dynamic component-based operating system for wireless sensor networks,” in *Proceedings of the 4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks*. ACM, 2009, pp. 7–12.

[9] J. Martínez, M. Familiar, I. Corredor, A. García, S. Bravo, and L. López, “Composition and deployment of e-health services over wireless sensor networks,” *Mathematical and Computer Modelling*, vol. 53, no. 3, pp. 485–503, 2011.

[10] A. de la Piedra, A. Braeken, and A. Touhafi, “Sensor systems based on fpgas and their applications: A survey,” *Sensors*, vol. 12, no. 9, pp. 12 235–12 264, 2012.

[11] P. Völgyesi, J. Sallai, Á. Lédeczi, P. Dutta, and M. Maróti, “Software development for a novel WSN platform,” in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Sensor Network Applications*. ACM, 2010, pp. 20–25.

[12] F. Oppermann and S. Peter, “Inferring technical constraints of a wireless sensor network application from end-user requirements,” in *Mobile Adhoc and Sensor Networks (MSN), 2010 Sixth International Conference on*. IEEE, 2010, pp. 169–175.

[13] P. Boonma and J. Suzuki, “Moppet: A model-driven performance engineering framework for wireless sensor networks,” *The Computer Journal*, vol. 53, no. 10, pp. 1674–1690, 2010.

[14] F. Delicato, L. Fuentes, N. Gámez, and P. Pires, “Variabilities of wireless and actuators sensor network middleware for ambient assisted living,” *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, pp. 851–858, 2009.

[15] A. Pathak, Q. Zhou, and V. K. Prasanna, “Srijan: A graphical toolkit for wsn application development,” in *The 4th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS ’08)*, 2008.

[16] S. Cotterell, R. Mannion, F. Vahid, and H. Hsieh, “eBlocks: an enabling technology for basic sensor based systems,” in *Proceedings of the 4th international symposium on Information processing in sensor networks*, 2005, pp. 58–es.

[17] B. Porter, U. Roedig, F. Taiani, and G. Coulson, “A comparison of static and dynamic component models for wireless sensor networks,” in *Proceedings of the The First International Workshop on Networks of Cooperating Objects (CONET2010), Stockholm, Sweden*, 2010.

[18] S. Peter and P. Langendrfer, “Tool-supported methodology for component-based design of wireless sensor network applicationss,” in *CORCS 2012- The 4th IEEE International Workshop on Component-Based Design of Resource-Constrained Systems*, 2012.

[19] K. Piotrowski, P. Langendoerfer, and S. Peter, “tinyDSM: A highly reliable cooperative data storage for Wireless Sensor Networks,” in *Collaborative Technologies and Systems, 2009. CTS’09. International Symposium on*. IEEE, 2009, pp. 225–232.