Tool-supported Methodology for Component-based Design of Wireless Sensor Network Applications

Steffen Peter and Peter Langendörfer
IHP, Im Technologiepark 25, 15236 Frankfurt (Oder), Germany
{*peter, langend*}*@ihp-microelectronics.com*

***Abstract*—A major issue when developing Wireless sensor networks applications is the need for highly specialized knowledge in the field of embedded programming, networking and in the application domain. In order to speed up the development process, a new methodology for WSN application development is required. It needs to provide ready to use building blocks as well as means to map application requirements to technical features provided by these blocks. Last but not least, mechanisms to select appropriate building blocks and to evaluate the system compiled out of these blocks are essentially needed. This paper presents a design flow fulfilling the mentioned features. In a first step, user requirements elicited from a managed catalog are translated to a graph structure. Then, properties of the composed system derived from meta-information of the applied components are evaluated to resolve constraints –representing application requirements and/or features of the target system– in the derived system model. The validity of the methodology, for which the needed tool support has actually been implemented, is shown in an example that illustrates how this approach can propose correct configurations for secure systems as proposed in related work. Since the approach allows designing correct and fine-tuned solutions even for general application requirements we consider it to be a significant step towards improved programmability of WSN nodes.**

***Keywords*-component-based engineering; wireless sensor networks; models**

I. Introduction

Wireless Sensor Networks (WSN) comprise networks of tiny computing devices, called sensor nodes. Equipped with wireless communication the nodes are the basis of large autonomous networks typically used to monitor physical phenomena or to control actuators in the environment. WSNs have become interesting for a broad range of applications reaching from home, environmental and structural health monitoring to medical, military and homeland security applications. These applications possess a diversity of challenging requirements of functional, qualitative but also security-related aspects. The high requirements are opposed by the scarce resources a sensor node can offer. Resources such as energy, processing power and memory are kept as small as possible with the aim of reducing the costs for the network and to increase the lifetime of the typically battery-powered devices. Coping with these severe constraints is one significant design challenge in the developers of such networks. This demands a broad range of experience covering knowledge in distributed algorithms, radio propagation, network protocols, electrical engineering, and embedded programming, among others. Despite or due to this heterogeneity of specialties needed for WSN integration, no structured and tool-supported holistic development flows exist. The pre-dominant way of development for WSNs involves handcrafted code and a trial-and-error methodology based on simulations or experiments. This approach is expensive and complicated to apply even for experts, while it is not suitable for end users or experts in the application domain who are faced with the task of configuring WSNs in practice.

On the other side we see a huge variety of technical approaches and implementations to solve individual challenges of WSNs, ranging from basic services to network protocols or data replication strategies in the network. The conditions, footprints, scalability and side effects of the protocols are barely understandable for experts. To facilitate the merits of these approaches we need a development infrastructure that fosters reusability of such building blocks with low overhead, which however expressively describes the behavior of the components.

This paper presents such a design flow that allows reuse of already existing components and provides an automated way to combine these components into an application defined on a pretty high level of abstraction. This process is controlled by models that assess the suitability of the composed system for the user-given set of high level requirements.

The rest of this paper is structured as follows. After a brief overview on the state of the art in WSN programming in Section II, we present our component-based framework approach in Section III, which outlines the concept and the key components. Section IV then refines the technical details on the required model transformations and property assessment. Section V presents the current state and first results of our evaluation, before we conclude the paper with a summary and a short outlook.

II. Related Work

State of the art in today's WSN development is handcrafting code using available software components that provide a certain degree of hardware abstraction. This is supported by nesC [3], the programming language of TinyOS, the dominating operating system for WSNs. NesC uses abstract

published in CORCS 2012: The 4th IEEE International Workshop on Component-Based Design of Resource-Constrained Systems

components, called modules, which use and provide interfaces. It is an important motivation of our work to reuse the available components available for nesC. To improve the development experience with nesC, graphical user support has been provided. Most mentionable is GRATIS [11], a model-based approach for the component-based development of WSN applications. It bases on a meta-model toolkit for model integrated computing and is able to map TinyOS code to modeling concepts and, thus, provides a set of graphical tools for the development of WSN applications. GRATIS emphasizes interface specifications and thus is a great help for developers of TinyOS applications. However, since GRATIS does not model the actual behavior of the modules, it does not support component selection.

One approach to improve programming of WSNs is using powerful abstraction layers. Meta-programming approaches such as [4] provide an easily-accessible interface to the entire network, so that tasks can be described in an interpreted language without the need of deep technical knowledge. Such abstraction approaches provide a fixed middleware and optimize the program running on top of that fixed middleware instead of optimizing the services in the middleware, which limits the space of non-functional properties. More advanced in this respect is Macrolab [5], which allows parametrization of the radio protocols based on environmental properties. An application of our methodology, presented in this paper, to configure the middlewares could further improve the performance and flexibility of meta-programming.

In the domain of embedded systems and automotive systems much work has been accomplished in the area of component-based development to foster reusability of a stock of components [6]. Since embedded systems are similar to WSN systems in several aspects the methodologies and properties of such component composition systems can be valuable also for WSNs. Frameworks like THINK [2] already provide extended tool-support and powerful meta-models to describe formal model behavior. However, a huge part of such environments rely on proprietary operating system functions, and focus on general embedded systems which –due to their functional heterogeneity– do not correspond to the needs of WSNs. Additionally, the focus of the majority of proposed tools to support the developer instead of the actual user limits the direct reusability for us.

III. Development Flow

In this section we describe our development flow on a high abstraction level and introduce the applied views on users, requirements, components, and application. The technical details of the model-based interaction between these entities are addressed in the following section.

A. Development Flow and User Model

Figure 1 shows the general flow of the intended component-based composition-driven design process. The

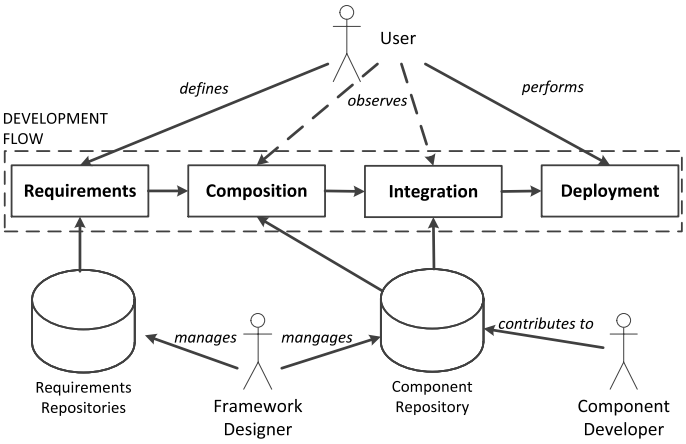

Figure 1. Component-based development flow: The user controls requirements definition and the composition with the aim to deploy the network. Repositories for requirements and composition are managed by framework designer. They add the components developed by Component developers.

process steps focus on the actual development process performed by the user, who can execute the process without expert support. Participants in this process are the user, the framework builder and the component developer. The **User** is the person or organization who will compose and deploy the sensor network. In the future that can be an end user or application domain expert who wants to apply WSNs to accomplish a specific task. Today it is more likely that the user is a sensor network engineer who assembles the network for the actual end user. The **Framework builder** sets the infrastructure and the models and chooses the set of domain-specific components and supported requirements. The **Component developer** provides implementation and description of basic building blocks.

The process starts with requirements defined by the user, chosen from a **Requirements Repository**, which contains a set of selectable requirements. In the composition step components are selected and assembled to a system that promises to satisfy the user's requirements. The selection and assessment process employs a **Component Repository** containing models of components and their properties. Based on the resulting configuration, the actual system is compiled and assembled. Finally, the resulting sensor nodes are equipped with the resulting code images and are deployed at the application site.
Our vision is that ultimately users only define the requirements and finally perform the physical deployment, while composition and integration are executed automatically.

B. Requirements Model

The actual development process starts with the definition of the system requirements. They are the input for the system composition. This is why the requirements inputs have to be correct, complete, and technically usable. In our model requirements may be functional or non-functional

requirements. The former capture the nature of interaction between the component and its environment, while the latter, which are also referred as constraints, may be qualitative restrictions, physical limitations, or equipment constraints. Requirements explicitly comprise the environmental description, which describes properties of the deployment side. In WSNs the explicit definition of the environmental boundary is important as it is the major discriminator between otherwise similar applications.

A problem in practice is that end users and application domain experts are not familiar with the terms typically used in WSN engineering. They can express the requirements precisely in their domain-specific language, but we need a translation stop from the rather high-level, domain-specific requirements to verifiable metrics that are understood within the WSN development process. For this issue we decided for a catalog approach as proposed in [9]. The user chooses attributes from the catalog and parametrizes them. The technical process benefits from the knowledge about the items in the catalog, so that a translation to technical terms is available. The main disadvantage of this method is the lack of freedom for the user to define new requirements. This task, however, can be performed by the framework designer.

C. Component Model

Components in the design flow are the building blocks for the application and may represent software, hardware modules or protocols. We chose a black-box abstraction, as access to the implementation code is not relevant for the composition. Nevertheless, the behavior and qualitative attributes of the individual components is not fixed, as it may vary with attached components and the environment the system is exposed to. Each component provides its function to other components via interfaces. Components using this interface may connect to one provided interface. Additionally to the interfaces, each component is described by a set of meta information expressing properties and constraints of the component. As we will describe later, the meta information are key to assess the properties and suitability of a system under consideration in scope of given requirements and environment.

D. Composition of the Application

As center of the development process, individual components should be combined to one system that satisfies the needs of the user. Basically, this is done by connecting (binding) components via their interfaces. In this process the **Composition** is the model of the system under development, consisting of the components.

The actual application is –similar to other components– one black-box component from the repository that uses other components by interfaces. We decided for this approach because first, the functional heterogeneity of WSN applications is rather limited which means that individual programming is no immediate need, and second, the assumed user is no programmer, but merely selects the general functions and describes their qualities. That makes the composition of the stack of services, which determines the non-functional qualities, to the actual key in the development process.

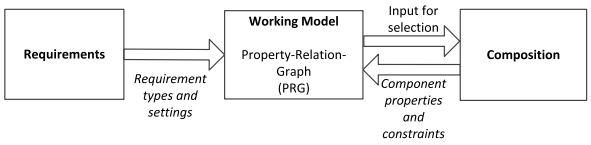

Figure 2. Data flow of the working model: Requirements initialize the WM. The state of the WM is input for the composition which updates the status of the WM.

The composition is represented by the Component-Composition-Graph (CCG), which is a graphical representation of the system of components connected by interfaces. The basic goal within the CCG is to find a composition that is complete and can be integrated. Technically, as discussed in [7], the composition has to comply with the requirements reachability (all components are reachable from the main component), extensiveness (all user-required components and functions are included), completeness (all needed interfaces are provided), and implementability. Only if those conditions are satisfied, the composition is complete and implementable. The assessment of fulfillment of structural requirements in the CCG (reachability, completeness) is straightforward. However not all syntactically complete compositions are free of contextual conflicts. Therefore the properties of the components have to be evaluated. This process is introduced in the following section.

IV. Working Model

The previous section outlined the development flow and introduced its inputs – i.e. requirements and components – and the composed application as output. As binding element for these inputs we apply the concept of a **Working Model (WM)**. It processes both, requirements and system information, to establish a complete model of the development domain. Its integration in the development flow is illustrated in Figure 2. Between the data structures exist the following data flows:

Requirements → WM: The technical requirements parametrize the WM initially. This step is performed once for the system configuration. Thereafter the WM contains all information about the requirements.

WM → Composition: The current status or a subset of the status of the WM is transferred to the (selection algorithm of the) CCG. There the information is applied to execute the component composition. The actual composition process is a systematic search, starting with a set of fixed components and backtracking through possible system configurations, which is not scope of this paper.

Composition → WM: Properties of the component selec-

tion are forwarded from the CCG to the WM. The WM then evaluates the properties of the composition in context of the requirements and the domain-specific model.

The WM can be considered as a rule-book for the composition process. Basically it has two tasks: First, to derive and manage all information in context of the development and usage of the system. This includes technical properties of the system, behavioral aspects, but also description of the application and environmental situation. Second, the WM has to manage conflicts that may occur during the assembly of the system or at runtime. For example if the application scenario demands a specific node distance, but the radio on the considered sensor node is not able to provide the distance in the required quality, it is a conflict that must be discovered within the WM.

To realize these needs, the WM applies a concept of properties and relations between properties to define the model. They are represented in a graph structure named Property-Relation-Graph (PRG).

A. Property-Relation-Graph

The PRG is a powerful abstraction that allows to manage requirements, express system properties, and maintain system models. It is a graph that, most generally, describes properties and relations between properties.

A **Property** in our context is a characteristic or quality that describes an attribute of the system, or of a part of the system. Basically, a property is everything that can be expressed by a value. That concerns all sorts of attributes to describe the system, its components, its requirements, or its environment. Properties can represent various data types, such as numbers, strings, or sets.

Typically, properties influence and depend on other system attributes. For example the setting of an application environment affects the required security strength. Another example is the required radio sending power that may depend on the distance of the nodes. For this purpose operations on properties are needed. We term these operations **Relations**. The semantic of relations is similar to a line of code in a standard imperative programming language. For example the operation a:=b+c, which, in a programming language, assigns the value of the addition of variables b and c to the variable a, is in our context a relation that expresses the value of the property a as function of the properties b and c. Similar to classic operations, Relations can be Assignments or Constraints. Assignments define a property as function on other system properties. An assignment is for instance the formula `Cost_of_System:=Cost_of_Nodes*Number_of_Nodes`, which assigns the results of the multiplication of `Cost_of_Nodes` and `Number_of_Nodes` to the property `Cost_of_System`. Constraints are boolean functions on system properties. A comparison is for example `Cost_of_System <1500` which returns true only if `Cost_of_System` is smaller than 1500. A PRG is free of conflicts if all constraints result as true.

With this basic concept the PRG realizes the main needs of the WM: it can express all sorts of system information, and it evaluates the constraints of the graph and thus checks all conflicts. In the following subsection we describe how requirements and composition properties are expressed as PRGs that can be translated to the WM.

B. Requirement Transformation

As introduced in Section III-B, requirements are initially provided by the user and transformed to technical terms. In all stages of this process they are represented as PRGs. The expression of requirements as properties and relations is quite natural. As part of the catalog a requirement is one parametrized property data type with a set of implicit relation data types. For instance, the requirement that the radio range must be above a specific distance can be expressed with two properties `min. distance` and `actual distance`, and the constraint relation `min. distance<actual distance`, while the former may be set by the user and the latter is a property of the composition. These properties and relations initialize the WM, for which it is the task to resolve the constraints. Equally intuitive is the translation from high-level user requirements to technical requirements. Assignments, as they can be expressed in PRGs, extend the requirement space, and translate catalog requirements to technical WM inputs.

C. Component Property Transformation

After the initialization of the WM with the PRGs from the requirements, the PRG of the WM remains with many unsatisfied constraints (as the distance constraints). These constraints should be solved with properties from the composition. In the example it means finding a system composition for which the actual distance satisfies the constraint. For this, our component model uses meta-information to describe properties and relations of the individual components. This means, each component contains its own PRG that –during the composition– will be added to the current WM.

Component PRGs contain information about properties that are available if the component is used (like `actual distance=500`), but also new constraints needed by the component that add to the existing requirements (e.g. `MicroController==MSP430`, which defines the needed microcontroller).

In the simplest case the meta information of the components describe propositional global properties. For them the properties of the system correspond to the summation of the individual component properties. However, not all properties can be mapped directly. Instead, most properties of a system are influenced by the binding of the components within the composition.

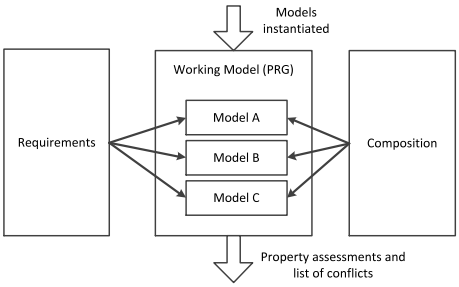

Figure 3. The PRG of the WM consists of sub-frameworks for specific property models. Like the global PRG they are parametrized by requirements and evaluate the composition.

The assessment of the composed properties is a non-trivial issue as for example discussed in [8], and global aggregation heuristics as proposed in [10] are from limited use. For these derived properties, we apply an approach in which the component PRGs are evaluated only locally. This follows the notion that properties of each component are influenced by the properties of the connected components. Neighbored components may process the inputs as part of their component PRGs. The understanding of the component boundaries and its interrelations should be available by the component developer who describes the properties. Ultimately, the top application component propagates these properties to the WM as, in this model, it is the interface to the environment.

D. Models

While many technical terms can be described and resolved solely between requirements and composition, complex properties need additional modeling and a general rule set. Such rule set determines how the properties have to be specified in the components description, how the requirements of these properties are formulated and how the information is processed. As illustrated in Figure 3, within the WM various property models can be loaded to represent specific aspects of the development. Such aspects may be memory consumption, energy consumption, expected lifetime and security assessment. Like the global PRG, the sub-models are parametrized by the requirements and assess the composition based on the properties in the components and their binding. As example of a heuristic model we implemented a memory model that receives the amount of available memory from meta information of the hardware components and checks this constraint against the needed amount of RAM and ROM for each software component. If the software is too large, it causes a constraint violation that requires a reconfiguration. Another model, evaluating security and energy properties of compositions is shown as part of the evaluation in the following section.

V. Implementation and Evaluation

For evaluation of the presented methodology we implemented a tool set consisting of three programs supporting the tasks of the three roles in the development process:

- a tool to enter the meta information of the components (for the component developer),
- a tool to manage the models including the requirement translations (for the framework designer),
- front-end for users to define application requirements, observe the composition process, and as result obtain a proposal for the composed application.

In the moment we do not support automatic assembly and compilation. The proprietary description language for components, requirements, and properties is XML-based.

For validation of the methodology we used the tools set to reproduce the results of the currently most advanced user-driven security composer for WSNs discussed in literature. Cionca [1] proposed a tool selecting algorithms that suite high level user requirements. In one example, for a considered hospital every room has a WSN deployed on patients to monitor vital parameters. For each patient there is a mobile cluster of sensor nodes. The data should be reported every second for a runtime of up to one month while standard nodes with two AA batteries are used.

For this example the reasoning process described in the paper ruled out all protocols which are not suitable for a public environment or the hierarchical network topology. Based on the lifetime (one month) and the communication period (1 second) the number of packets over the lifetime is computed to disable protocols that overflow earlier. Finally, the energy consumption for the expected packet count is computed and compared with the energy from the batteries.

In our methodology consisting of the three modules requirement translation, component composition, and working model we could reproduce the results in the following way:
Requirements: To replicate the inputs of the Cionca tool, additional inputs are needed to enable the definition of the network parameters, allowing a user to enter the application type and network parameters such as number of nodes, topology, available energy, needed lifetime and communication frequency. As part of the requirement translation, assignments (as PRG) for derived technical properties such as security environment and harmonized units are added.
Components: We added all components discussed in [1] to our component repository and added interfaces for the protocol category they represent. The main application component uses all the interfaces. The constraints and properties for each component are entered as meta information. For instance, for the "Server Shared KPD" approach the relational properties are expressed by the two relations:

$$Environment <= Hostile$$
$$Topology == Centralized\, OR\, Topology == Hierarch$$

In case the component should be used in the selection process, both constraints must be true. Additionally, meta information for the computation of the maximal packet count and the energy assessment are included.

Working Model: In this example the working model only contains a minimalistic energy model, computing the needed energy based on the number of packets and comparing the result to the available energy. The number of packets as well as the available energy are contributed by the requirements, the energy per packet comes from the used components. Apart from that, the initial working model is empty.

Selection Process: At the beginning of the selection process it will be initialized with the requirements selected by the user as described above. The requirement translation extends the requirements to the technical requirements, as the security degree and the number of packets. During the composition process the properties and constraints of selected components are included in the working model. In case it causes conflicts, either due to not fitting environment or topology, or a missed constraint in the energy model, the component is deselected and the search continues with other components until a suitable composition is found. The process runs without additional user interaction and results in the system composition as proposed in the original work.

VI. Conclusions and Outlook

In this paper we presented the idea, the technical mechanisms, and first results of a practical evaluation of a novel methodology for the component-based design of wireless sensor network applications. As novelty in this process, the gap from the high level requirements to the low level technical implementations is bridged by the concept of the working model (WM), which represents the semantics of the entire development process in a single data structure. This data structure is parametrized by the requirements which are translated from the user inputs on one side, and the derived information from the composition on the other side. Evaluating conflicts of constraints, the WM considers a composition as suitable if the models do not indicate conflicts between requirements and system composition.

The evaluation has shown that the approach already works in practice for smaller examples. However, in our experiments, we impersonated all roles in the process. We defined the meta-information of the components, we assembled the framework and eventually we were the user defining the requirements and compiling the results. While we are convinced that the approach works for developers and users, this has to be validated in practice with third party developers and actual users in the future. Further, as future work we need to refine the detail and expressiveness of the applied models to improve the assessment of network properties, energy consumption, and security properties to provide a usable tool set for the integration of WSN applications.

But to summarize, with our design flow we stress the idea that in the future, actual end users will be able to set up WSN systems, facilitated by highly speciallized components available from expert designers. We evaluated our approch using examples from other groups which have been designed with a specific application domain in mind. The fact that our tool set produced equivalent combinations of components impressively demonstrates the applicability of our approach to different domains. We are convinced that our approach even not fully researched and implemented is a significant step towards tool supported WSN application design and implementation by end users.

Acknowledgments: The work has been supported by the European Commission through the ICT program under contract ICT-SEC-2009-5-258754 (TAMPRES).

References

[1] V. Cionca, T. Newe, and V. Dădârlat. A tool for the security configuration of sensor networks. *Journal of Physics: Conference Series*, 178, 2009.

[2] J.P. Fassino, J.B. Stefani, J.L. Lawall, and G. Muller. Think: A software framework for component-based operating system kernels. In *USENIX Annual Technical Conference, General Track*, pages 73–86, 2002.

[3] 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.

[4] R. Gummadi, N. Kothari, R. Govindan, and T. Millstein. Kairos: a macro-programming system for wireless sensor networks. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*. ACM, 2005.

[5] T.W. Hnat, T.I. Sookoor, P. Hooimeijer, W. Weimer, and K. Whitehouse. Macrolab: a vector-based macroprogramming framework for cyber-physical systems. In *6th ACM conference on Embedded network sensor systems*, 2008.

[6] P. Hosek, T. Pop, T. Bures, P. Hnetynka, and M. Malohlava. Comparison of Component Frameworks for Real-Time Embedded Systems. In *Component-Based Software Engineering: 13th International Symposium, CBSE 2010*, 2010.

[7] G. Jenson, J. Dietrich, and H.W. Guesgen. An Empirical Study of the Component Dependency Resolution Search Space. In *Component-Based Software Engineering: 13th International Symposium, CBSE*, 2010.

[8] K.K. Lau. Component certification and system prediction: Is there a role for formality. In *Proceedings of the 4th ICSE Workshop on Component-based Software Engineering*, 2001.

[9] F.J. Oppermann and S. Peter. Inferring technical constraints of a wireless sensor network application from end-user requirements. In *2010 Sixth International Conference on Mobile Ad-hoc and Sensor Networks*, pages 169–175. IEEE, 2010.

[10] N. Pham and M. Riguidel. Security assurance aggregation for it infrastructures. In *Second International Conference on Systems and Networks Communications*, 2007.

[11] P. Völgyesi, M. Maroti, S. Dora, E. Osses, and A. Ledeczi. Software composition and verification for sensor networks. *Science of Computer Programming*, 56(1-2):191–210, 2005.