Dipl.-Inform. Felix Jonathan Oppermann

Programming and Configuration of Wireless Sensor Networks

DOCTORAL THESIS

to achieve the university degree of

Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Inform. Dr. sc. ETH Kay Römer

Institute for Technical Informatics

Graz, December 2015

AFFIDAVIT

Date Signature

Acknowledgments

The work leading to this thesis put me on a journey much longer and far more varied than originally expected. Despite occasional hardship, this journey allowed me to significantly grow professionally as well as personally. In addition, it allowed me to get to know a number of great places and to meet a large number of truly great people. I am extremely thankful to each of them and it is not possible to adequately acknowledge everyone within the limited space.

My first thanks goes naturally to my current advisor, Prof. Kay Römer, who showed a lot of patience while guiding me towards the completion of this work. I am very glad to have an advisor with such a reputation and knowledge within the research area of wireless sensor networks. His knowledge was essential for me and I am grateful to be able to learn from his experience.

Equally, my gratitude goes to Prof. Oliver Theel, my previous advisor at Carl von Ossietzky University of Oldenburg. Without his trust in me, I would not have been able to embark on this journey. The experience and knowledge I gathered during my time in Oldenburg, later proved to be indispensable for me and for my work.

Special thanks also go to Prof. Koen Langendoen, who besides kindly agreeing to serve as an examiner of my thesis also provided invaluable feedback during the final writeup of this thesis. It is an honor to have an examiner with such reputation and experience in the field of wireless sensor networks.

I also want to thank the system and network administrators Oliver Bock, Engelbert Meissl, and Steffen Prehn for their constant and patient aid with diverse software and hardware issues. Similar thanks go to Peggy Baudach, Meike Burke, Elke Daniels, Anjes Kiencke, Silvia Reiter, Ines Schiebahn, and Ira Wempe for the equally competent handling of administrative matters.

The compilation of such a thesis usually requires the co-operation with a large number of people. Luckily, I had the pleasure to work on a number of projects and joint papers with a number of great researchers. In this regard, I want to especially thank Carlo Alberto Boano, James Brown, Fabio Casati, Florian Daniel, Guenadi Dantchev, Joakim Eriksson, Niclas Finne, Harald Fuchs, Andrea Gaglione, Kinga Lipskoch, Jens Kamenik, Chamath Keppitiyagama, Stamatis Karnouskos, Patricio Moreno Montero, Luca Mottola, Lars-Åke Nordén, Nina Oertel, Steffen Peter, Gian Pietro Picco, Antonio Quartulli, Utz Roedig, Kay Römer, Patrik Spiess, Oliver Theel, Stefano Tranquillini, Nicolas Tsiftes, Thiemo Voigt, Hjalmar Wennerström, Marco Antonio Zúñiga, and Marco Zimmerling. All of them provided valuable ideas and input to my research and without them a large share of this work would not have been possible.

I am also thankful to my excellent student assistant, Bengt-Ove Holländer, who provided very helpful support in the implementation of the macro-compiler and who served as an uncomplaining test subject in the evaluation of the system.

Another important source of support were my numerous colleagues at the three different universities I stayed at. In most of them I also found amazing friends with whom I also shared great moments in my spare time. Especially, Carlo Alberto Boano and Matteo Lasagni accompanied me on a large section of my journey and both were always an important source of support for me. Special thanks for their support, encouragement, and the nice time I shared with them also go to Marcel Carsten Baunach, Reinhard Berlach, Stefan Brunhorn, Dariush Forouher, Abhishek Dhama, Renata Martins Gomes, Andrea Höller, Johannes Iber, Christian Kreiner, Kinga Lipskoch, Georg Macher, Richard Mietz, Nils Müllner, Jens Oehlerking, Wolfgang Raschke, Tobias Rauter, Christian Renner, Andreas Daniel Sinnhofer, Grigore Stamatescu, Christian Storm, Cuong Truong, Timo Warns, and Patrick Weiß.

Of course, I am also thankful for all other dear friends that accompanied me on my way and who were always a great source of support and relief. At times, it was surely not an easy task for them.

Finally, I am highly indebted to my family for their love, support, and encouragement. Without the invaluable aid of my parents, I could not have dreamed of pursuing a PhD. I cannot express sufficient gratitude towards them.

Graz, December 12, 2015
Felix Jonathan Oppermann

The research leading to these results has received funding from the European Union 7th Framework Programme (FP7-ICT-2009-5, FP7-ICT-2011-8) under grant agreements n^o 258351 (make*Sense*) and n^o 317826 (RELYonIT). This work was supported by the German Research Foundation (DFG), grant GRK 1076/3 (TrustSoft). Part of this work was carried out at and employing resources of the Carl von Ossietzky University of Oldenburg and the University of Lübeck.

Abstract

Wireless sensor networks (WSNs) increasingly penetrate our everyday life and are already employed in a wide range of application areas, such as habitat monitoring, precision agriculture, home automation, and logistics. Recently, WSNs are also increasingly employed in safety-critical applications such as patient and structural health monitoring. These applications impose strict dependability requirements and failures can have severe consequences.

Due to an increasing number of real-world applications, the task of implementing and deploying WSNs is increasingly shifted to end-users. These end users, however, often do not have extensive expertise in the areas of embedded systems and wireless communication. To support a further spread of the technology it is important to empower domain experts to successfully realize WSN systems without the help of WSN experts. Current programming techniques for WSNs do not cope well with these challenges. Even today, the majority of WSN programs are still written in low-level programming languages, such as C, and in a highly node-centric mindset. Such a programming approach requires an in-depth understanding of the underlying technology and the employed hardware platform. A number of individual programming abstractions to solve individual programming challenges exist, but these are not well integrated and are difficult to combine. In addition, to meet strict dependability requirements in hostile environments, communication protocols need to be carefully tuned to the expected environmental changes. This can be a tedious task that requires significant expertise and a clear understanding of how the environment affects the hardware and the communication protocols.

To enable a widespread use of WSNs, one must ensure that the design and deployment process requires as little technical knowledge as possible and can be automated as much as possible. We identified and addressed two major challenges that significantly increase the effort needed to implement a WSN solution and that hinder application domain experts in successfully deploying a WSN on their own: (1) the lack of a high-level extensible macro-programming system for WSNs, and (2) the difficulty of dependably configuring WSN communication protocols. In this thesis we propose two frameworks that individually help to solve these challenges.

First, we introduce a novel macroprogramming framework that allows the seamless integration of existing and future programming abstractions. The underlying macro-programming language is based on Java and explicitly supports object-oriented programming. The use of a high-level language that hides many of the low-level issues of WSN programming greatly reduces the effort required to implement such systems.

An evaluation based on typical WSN applications demonstrates the feasibility of the approach and its suitability for resource-constrained devices, such as sensor nodes.

The second contribution is a framework to automate the configuration of WSN communication protocols based on user requirements and abstract models of the environment. This framework builds on stochastic optimization techniques to identify configurations that guarantee a specific performance without requiring significant knowledge from the user. In an evaluation, we demonstrate that this approach works well for a realistic case-study and that the performance of the employed tools is sufficient to be useful within a typical design process.

With this thesis, we are able to show that the burden and especially the required knowledge to successfully design and deploy a WSN can be significantly reduced. Support systems such as the ones presented in this thesis will make the technology more approachable and we believe that this will lead to an increased uptake in industry. Thereby, WSNs will turn into a useful tool for a large number of application areas.

Zusammenfassung

Drahtlose Sensornetze (WSN) spielen zunehmend eine bedeutende Rolle in unserem Alltag und werden für eine Vielzahl unterschiedlicher Anwendungen eingesetzt, beispielsweise bei der Überwachung von Biotopen, der teilflächenspezifischen Landwirtschaft, der Heim-Automatisierung und in der Logistik. Neuerdings werden WSN auch zunehmend für sicherheitskritische Anwendungen eingesetzt, wie der Überwachung von Patienten oder der strukturellen Integrität von Gebäuden. Diese Anwendungen stellen hohe Anforderungen an die Zuverlässigkeit, da Störungen schwerwiegende Folgen haben können.

Auf Grund der zunehmenden Anzahl von Anwendungen werden WSN zunehmend von Nutzern ohne tiefgreifende Kenntnisse im Bereich eingebetteter System und drahtloser Kommunikation umgesetzt. Um die Verbreitung der Technologie zu unterstützen, ist es wichtig, dass Anwendungsexperten entsprechende Systeme ohne die Hilfe von WSN-Experten umsetzten können. Aktuelle Entwicklungsmethoden für WSN sind hierzu weniger geeignet. Heutzutage wird die Mehrheit von WSN-Programme in der hardwarenahen Sprache C und auf eine knoten-zentrierte Weise geschrieben.

Folglich ist ein gutes Verständnis der zugrundeliegenden Hardware und Technologien nötig. Einige Programmierabstraktionen können bereits zur vereinfachten Lösung individueller Probleme eingesetzt werden, lassen sich jedoch schlecht in einer einzigen Anwendung kombinieren. Darüber hinaus müssen die verwendeten Kommunikationsprotokolle in der Regel auf die Eigenschaften der Einsatzumgebung abgestimmt werden, was eine mühsame Aufgabe darstellt. Diese Aufgabe kann nur mit einem umfangreichen Wissen bezüglich des Einflusses von Umgebungseigenschaften und dem Verhalten der Kommunikationsprotokolle bewältigt werden.

Um eine weitreichende Nutzung von WSN zu ermöglichen, muss der Entwicklungsprozess vereinfacht und so weit wie möglich automatisiert werden. Wir haben insbesondere zwei Herausforderungen identifiziert, die den nötigen Aufwand deutlich erhöhen und die einen erfolgreichen eigenständigen Einsatz durch Anwendungsexperten verhindern: (1) Das Fehlen eines erweiterbaren abstrakten Macro-Programmiersystems und (2) die Schwierigkeit der zuverlässigen Konfiguration von WSN-Kommunikationsprotokollen. In dieser Arbeit schlagen wir zwei Frameworks vor, die zur Lösung der beiden Probleme beitragen können.

Als erstes präsentieren wir ein neues Macro-Programmiersystem, das es erlaubt bestehende und zukünftige Programmierabstraktionen einzubinden. Die verwendete Sprache basiert auf Java und unterstützt die objektorientierte Programmierung. Die

Verwendung einer Hochsprache, die viele der technischen Details versteckt, verringert die Schwierigkeit der erfolgreichen Umsetzung von WSN-Systemen deutlich. In der Auswertung demonstrieren wie die Umsetzbarkeit des Ansatzes und seine Eignung für Systeme mit begrenzten Ressourcen anhand typischer WSN-Anwendungen.

Das zweite Framework unterstützt die automatisierte Konfiguration von WSN-Kommunikationsprotokollen, basierend auf den Anforderungen der Anwender und abstrakten Umgebungsmodellen. Es nutzt stochastische Optimierungsverfahren um Konfigurationen zu identifizieren, die eine bestimmte Leistungsfähigkeit garantieren ohne dass der Nutzer über ein umfangreiches Fachwissen verfügen muss. Wir zeigen, dass dieser Ansatz in vertretbarer Zeit nützliche Konfigurationen erzeugen kann und das er sich zur Anwendung in einem typischen Entwicklungsprozess eignet.

Mit dieser Arbeit können wir insgesamt zeigen, dass sich der Aufwand und das nötige Fachwissen bei der Entwicklung von WSN deutlich reduzieren lassen. Systeme wie die vorgestellten werden diese Technologien in Zukunft zugänglicher machen und damit zu einer zunehmenden kommerziellen Anwendung führen. WSN werden sich so als nützliche Werkzeuge in einer breiten Palette von Anwendungen erweisen.

Contents

List of Figures

List of Listings

List of Abbreviations

AST abstract syntax tree.

BPM business process model.

BPMN business process model and notation.

BSN body sensor network.

CCA clear channel assessment.

CPS cyber-physical system.

CPU central processing unit.

CSMA carrier sense multiple access.

DSL domain specific language.

EBNF extended Backus-Naur form.

GNU GNU's not Unix.

HVAC heating, ventilating, and air conditioning.

I/O input/output.

IDE integrated development environment.

IoT Internet of things.

IP Internet protocol.

ISM industrial, scientific and medical.

JNI Java native interface.

LED light-emitting diode.

MAC media access control.

MCU microcontroller.

MPL macroprogramming language.

OS operating system.

PC personal computer.

PDA personal digital assistant.

PRR packet reception ratio.

QoS quality of service.

RAM random access memory.

RFID radio-frequency identification.

SPL software product line.

SQL structured query language.

TMDA time division multiple access.

UAV unmanned aerial vehicle.

WSAN wireless sensor and actor network.

WSN wireless sensor network.

XML extensible markup language.

1 Introduction

Evolving from early research at the University of California, Berkeley at the beginning of the 21st century, wireless sensor networks (WSNs) have become an important research area with their own dedicated conferences and journals.

A typical WSN consists of a number of tiny devices equipped with a microcontroller, a low-power radio, and a number of sensors to perceive their surrounding environment. These devices form a multi-hop network that enables delivery of sensed data and cooperation among nodes. Individual nodes are usually battery powered to enable flexible placement without the need for wired infrastructure. The original vision of WSNs consisted of randomly dropping large quantities of these tiny and low-cost embedded devices over a large area in order to enable ad-hoc measurements [Warneke et al., 2001]. However, this vision was beyond the technological capabilities at the time and typical WSNs tend to still consist of a smaller number of approximately matchbox-sized devices, often called "motes". Nevertheless, their relatively small size enables placement close to the phenomenon of interest, enabling unprecedented spatial and temporal measurement resolution at rather low costs. Recently, such WSNs tend to be directly connected to the Internet or even use Internet technology within the network and form a part of the Internet of things (IoT). The IoT is often seen as the next major evolution of the Internet. According to Evans [2011] the number of devices connected to the Internet will reach 50 billion by 2020 and thus such devices will soon outnumber people as Internet users.

The possible increase in spatial and temporal resolution of remote measurements, combined with the minimal need of human intervention, led to a great success of the WSN vision, and paved the way for the adoption in a wide range of applications, ranging from environmental monitoring and precision agriculture to industrial automation and personalized health-care [Oppermann, Boano, and Römer, 2013]. Recently, WSNs have also been increasingly employed in safety-critical applications, such as systems to control traffic [Ganti et al., 2010; San Francisco Municipal Transportation Agency, 2011], to inspect the structural health of buildings [Kim et al., 2007] and to monitor patients [Boano, Oppermann, and Römer, 2013]. Such applications impose strict dependability requirements on communication performance, as failures can have severe consequences.

Due to an increasing number of real-world applications, the task of implementing and deploying WSNs is increasingly shifted to users without extensive expertise in the areas of embedded systems and wireless communication. Until today, most WSN deployments have been performed by experts with a strong scientific background.

Their main purpose is the demonstration of new technologies and the exploration of remaining limitations; providing a working solution for the original challenges of the application is often only a secondary concern. Consequently, most current deployments are extensively assisted by WSN researchers and experts. To support a further spread of the technology it is important to also empower application-domain experts to successfully realize WSN systems on their own. The unique properties of WSNs make improving the accessibility of the technology a challenging task. WSNs applications rely on efficient and reliable wireless communication and data processing – already a challenge in itself – while only employing cheap and resource-constrained hardware devices. In addition, these devices usually rely on non-rechargeable energy sources and consequently one needs to reduce energy consumption as much as possible to enable a sufficient system lifetime. Adequate solutions to these problems and successful deployments require significant expertise and experience [Buonadonna et al., 2005]. Even experts are not always able to successfully develop a WSN application without extensive experimentation [Langendoen, Baggio, and Visser, 2006]. While WSNs have received a high level of research attention and interest [Baronti et al., 2007], their adoption outside of the scientific community is still hindered by the difficulty of successfully designing a new WSN application.

Existing WSN programming techniques provide little support to the user. Even today, the majority of WSN programs are still written in low-level programming languages, such as C, and in a highly node-centric mindset. Communication is explicitly handled by passing messages between individual nodes. Such a programming approach requires an in-depth understanding of the underlying technology and the employed hardware platform. Existing platform-independent WSN operating systems, such as Contiki [Dunkels, Grönvall, and Voigt, 2004] and TinyOS [Levis et al., 2005] only provide a shallow abstraction from the peculiarities of the underlying hardware. A number of more high-level programming abstractions exist that simplify the implementation of communication and hide low-level details [Mottola and Picco, 2011; Sugihara and Gupta, 2008]. While these abstractions surely help to overcome individual programming challenges, they are usually not well integrated and difficult to combine within a single application. In addition, also a number of more holistic macroprogramming languages exist that include a fixed set of programming abstractions [Madden et al., 2005; Whitehouse, Zhao, and Liu, 2006]. These programming systems help to raise the abstraction level of WSN programming, but existing systems are usually monolithic; and newly developed or application-specific abstractions cannot be added easily. They lack the flexibility required in order to be adapted to the needs and requirements of a specific application. Well known examples, such as TinyDB [Madden et al., 2005], are only useful for a particular class of applications.

In addition, the implementation of the application is not the only challenge. The underlying system and communication protocols also need to be configured in a suitable way. Correctly tuning the parameters of the systems and protocols can be a tedious task by itself. Even for experts in the field, it is often difficult and labor intensive to find the right trade off that satisfies multiple requirements for the application at hand, for example, achieving both a high level of reliability and low energy consumption [Barrenetxea et al., 2008; Langendoen, Baggio, and Visser, 2006]. This challenge is further aggravated by the fact that WSNs are often deployed in hostile environments that significantly affect their performance. For instance, systems deployed outdoors are affected by temperature fluctuations and changing weather conditions. To cope with these challenges and to meet strict dependability requirements, including in hostile environments, the parameters of the communication protocols need to be carefully tuned in relation to the expected environmental changes. This is difficult, however, to attain, as every application has unique properties and requirements, and there is no one-size-fits-all solution [Römer and Mattern, 2004]. Consequently, current deployments often employ default parameters that lead to suboptimal and unpredictable performance.

1.1 Problem Statement

As of today, a more widespread deployment of WSN technology is hindered by the complexity associated to designing and deploying such systems. This complexity is difficult to overcome by the intended users, who are expected to be application-domain experts with little knowledge in the areas of wireless communication and embedded systems. To enable a more widespread use of WSNs, one must ensure that the design and deployment process requires as little specialized technical knowledge as possible and can be automated as much as possible.

In this thesis, we demonstrate that the accessibility of WSN technology can be significantly increased, without exceeding the capabilities of the underlying hardware. Within the WSN development process, we identified two major unsolved challenges that significantly increase the effort needed to implement a WSN solution and that hinder application domain experts in successfully deploying WSNs on their own.

The lack of a high-level extensible macro-programming system for WSNs. Despite 15 years of WSN research, in practice the programming of WSNs is still mostly done on a low level of abstraction and in a node-centric way. Each node is programmed individually with low-level C code. Communication between nodes is handled explicitly by the programmer, which requires an understanding of the peculiarities of radio communication and the hardware platform. In addition, building the network behavior from individual building blocks is tedious and error prone.

To ease the development of WSN applications and to make the technology more accessible for less-skilled users, we propose the use of a higher-level programming language that hides technical details. The language should also abstract from the employed hardware and details of wireless communication. To be useful, such a programming framework needs to support a wide range of different applications. In addition, as a body of more specific programming abstractions already exists, it should be possible to integrate existing and future programming abstractions. Consequently, the system needs to extendible. To be appealing for the intended users, the language should also be familiar to them and easy to use, for example by employing established concepts and syntactic elements from widely used programming languages. Nevertheless, such a programming framework needs to be designed in a way that makes it suitable for highly resource-constrained devices, like sensor nodes. Sensor nodes typically employ 8- or 16-bit microcontrollers and memory in the range of a few kilobytes. Consequently, the programming framework may not introduce a significant overhead and should not hamper the implementation of efficient algorithms.

The difficulty of configuring WSNs. A large number of different WSN applications already exist [Oppermann, Boano, and Römer, 2013], each with its own requirements. For example, some applications, like temperature monitoring, require only minimal bandwidth, others like the tracing of vibrations in structural health monitoring demand a high bandwidth [Xu et al., 2004]. The diverse set of application scenarios necessitates an individual configuration of employed protocols and components, to be able to successfully meet the specific requirements. In addition, WSNs tend to be deployed in very diverse environments that significantly affect the performance and reliability of the network. The configuration of the network needs to take these properties into account. The heterogeneity in the application scenarios and environments of WSN deployments effectively renders a one-size-fits-all solution impossible and it is to be expected that a need for different solutions to optimally address application-specific problems will also remain in the future. The use of default values often leads to significantly degraded performance and yields unpredictable results. Manual configuration typically relies mainly on intuition and experience. Suitable settings are often determined by trying out different configurations in the field – a time consuming task. Even for computer scientists with WSN experience, it is often difficult to make the right choices; for the intended end users with less-extensive experience this is close to impossible. This is a particular problem since a wrong decision taken in the design phase can severely affect the performance and reliability of the deployed WSN. Current solutions, even though deployed with the help of experts, are often poorly able or even unable to fulfill the intended purpose [Langendoen, Baggio, and Visser, 2006].

Therefore, there is a need for a simpler configuration of WSNs that does not overwhelm the intended users of the technology. The task of configuring the network should be largely automated. Still, the process needs to ensure that the WSN is configured such that the specific performance and dependability requirements of an application can be met. The configuration tool should also provide reliable predictions about the future performance of the network, to be able to ensure that the requirements will be met even under adverse environmental conditions. Nevertheless, the system needs to take the resource-constrained nature of WSNs into account. It may not introduce an excessive overhead, especially at run-time.

1.2 Methodology

The work presented in this thesis builds on a systematic literature research surveying applications of WSNs within the last 15 years. The careful analysis of existing WSN applications enabled us to identify important challenges for the adoption of WSN technology. In addition, we used this survey to identify typical performance metrics for WSNs and derive a catalog of application requirements that are meaningful for the average user. Based on this analysis and identified challenges our work focuses on macroprogramming concepts and automatic configuration for WSNs. Automatic configuration systems for WSNs particularly have seen comparatively little research and no established solutions exist. As a consequence our work is largely exploratory. The conceptual solutions are implemented in prototypical systems to assess their feasibility. The developed systems are evaluated based on real-world case studies to ensure their practicability and to identify remaining shortcomings. Most experiments were conducted in testbed environments to create a controlled but realistic environment [Boano et al., 2014]. Some experiments also employed the Cooja simulation environment [Eriksson et al., 2006] to be able to cover a larger design space.

An important goal of our work is to provide a foundation for future research. The software systems developed as part of this thesis can be employed as a basis and framework for further exploration of the design spaces of macroprogramming and automatic configuration solutions for WSNs. In the remainder of the thesis, we only explicitly consider WSNs, but most of the developed concepts and frameworks can be equally applied to the IoT and more generic cyber-physical system (CPS).

1.3 Organization of the Thesis

The remainder of the thesis is structured as follows. In Chapter 2 a bird's eye view of the general approach and overarching architecture behind the software systems developed as part of this work is provided. Chapter 3 reviews existing research and results in the areas of macroprogramming and automatic configuration. In addition, this chapter presents a brief overview of the development of WSN technology and applications. Chapter 4 introduces the macroprogramming framework developed within this thesis, while Chapter 5 presents our work on automatic configuration for WSNs. Finally, the thesis is concluded by Chapter 6. Here, we summarize the results and contributions of our work and take a look at possible future research. The scientific publications underlying this thesis are included in Chapter 7. This chapter also lists all remaining articles, papers, and technical reports published while working on this dissertation.

2 Approach and Architecture

In this chapter we give a high-level overview of our approach to make WSN programming and configuration more accessible to non-expert users. First, we present the envisioned development process for such systems. Here, we also highlight the different people and their expected roles within the process. Next, we provide a bird's eye view of the overarching system architecture. In this section, we also present how the individual components are expected to interact to support the previously introduced design process. Finally, we briefly sketch how the tools can be integrated in a model-based design process.

2.1 Envisioned Development Process

As of today, WSN systems are typically developed in a largely unstructured process. The development process does not usually clearly distinguish between implementation and configuration aspects. In many current projects, especially in a scientific environment, the users and the developers of the system are actually identical and consequently the responsibilities for individual tasks are not well defined. Tool support is limited and for important phases of the design process, such as requirement specification and configuration, no suitable tools are currently available.

As WSNs are increasingly used for important applications, a more structured development process is needed. In addition, the development of such systems will increasingly adopt tools to support all phases. We expect that in the future WSN system development will follow a process as depicted in Figure 2.1. In the current process model, we only consider software aspects and assume that the hardware is already deployed prior to the software deployment. Hardware deployment is already supported by a number of existing tools [Dyer et al., 2007; Kim and Cobb, 2012; Ringwald and Römer, 2007; Xiang et al., 2012]. In the following, the process model will serve as a basis for the description of the developed systems and tools.

At the start of the design process, the future user of the system defines functional and non-functional requirements for the WSN application under development. Based on these requirements, a software developer with domain knowledge implements the functionality of the system. In a next step, implementations of system components and communication protocols are selected and configured such that they enable the system to also meet the non-functional requirements of the user. The configured software system can finally be deployed at the pre-installed WSN. After the deploy-

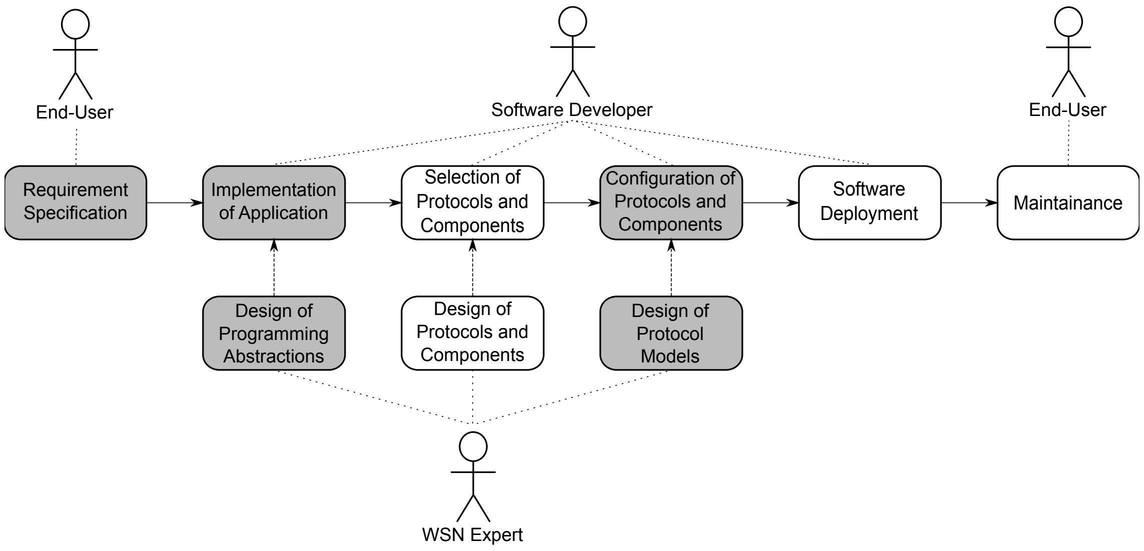
End-User
Software Developer
End-User
Requirement Specification
Implementation of Application
Selection of Protocols and Components
Configuration of Protocols and Components
Software Deployment
Maintainance
Design of Programming Abstractions
Design of Protocols and Components
Design of Protocol Models
WSN Expert

Figure 2.1: Wireless sensor network development process. Those steps of the development process that are covered by the systems and frameworks developed within this thesis are highlighted with a gray background.

ment, to ensure continued reliable operation, the system needs to be maintained throughout the entire lifetime of the system.

Three people are primarily involved with the process: the user, a software developer and a dedicated WSN expert. The ultimate goal of the system is to provide useful services to the end-user. Only the user is initially aware of the tasks the system is expected to fulfill and only he or she knows the required performance and reliability. As we cannot expect a regular user to have any experience with software development, the actual implementation work will usually be carried out by a hired software developer. In contrast to the current situation, this developer does not necessarily need to possess WSN-specific knowledge and skills, as we aim to also enable less specialized developers to successfully carry out the implementation and configuration of a WSN application. To enable cost-efficient operation, the continued maintenance following the deployment of the application should not require ongoing assistance by experts and basic maintenance tasks should be manageable by the users themselves. While the development of new applications should ideally not require expert knowledge, some of the involved tools depend on pre-made components. The development of these components still requires significant expertise. Consequently, we expect these components to be developed by experts that are also likely to be involved with the development of WSN hardware and operating systems. A growing repository of ready-made components should increasingly reduce the need for a direct involvement of these experts in WSN application development. In the next sections, we will look at the individual steps of the development process in more detail.

2.1.1 Requirement Specification

In the first phase of the design process, the requirements of the application are determined and recorded in a semi-formal specification. To be useful, the WSN needs to later fulfill the actual needs of the intended end-user and consequently it is inevitable to involve him or her with the requirement analysis. We need to empower the user to formulate expectations on non-functional aspects of the system, such as performance and reliability. In general, we do not expect the end user to be familiar with the employed technologies. Consequently, the requirements need to be formulated on an abstract level and in terms that are understandable and relevant to the intended users.

In Paper B, we developed a requirements catalog that can be employed to support users with formulating a complete and consistent specification of the non-functional properties of a WSN. This catalog can also be used to steer the requirement analysis and ensures that relevant information is available in the proceeding phases. The requirement catalog contains 29 individual dimensions that can be used to define different objectives and constraints for the design process. While this catalog also covers aspects like sensor coverage and security, in the remainder of the thesis we will focus on dependability properties and, in particular, on the dimensions of lifetime, packet reception rate, and latency.

2.1.2 Implementation

The second step of the development process is the actual implementation of the application based on the previously defined requirements. In contrast to the previous phase, we expect this step to be executed by a software engineer with some domain knowledge, but in contrast to the current situation, we do not expect this developer to possess extensive knowledge of embedded or wireless networked systems. By reducing the amount of required specialized knowledge, we largely increase the number of potential developers and increase their chance of successfully implementing an application that meets the user's expectations. For the implementation, the programmer can resort to pre-implemented programming abstractions, that solve typical programming challenges and hide the technological details of the underlying system.

As the development of these reusable programming abstractions requires extensive knowledge of networked embedded systems, we expect the development of novel programming abstractions to be conducted by experts. Especially innovative applications may thus initially still require the involvement of a WSN expert to create new application-specific abstractions, but over time an increasing number of ready-made abstractions can be expected to exist. The extensible macro-programming framework employed within this phase is described in more detail in Chapter 4 and Paper E.

2.1.3 Selection of Protocols and Components

After creating the actual application, the underlying system needs to be configured according to the deployment environment and application requirements. As the first step, one needs to select suitable components and protocols to provide the required basic functionality. In many cases, several components are available that provide similar functionality but differ in their performance and reliability. There is not usually a single solution that optimally satisfies the needs of all applications and environments. Consequently, one needs to select components with an optimal trade off for the given application.

In Chapter 3 we present a number of existing works that tackle the manual or automatic selection of suitable protocols and components. Like the programming abstractions, we expect the actual components to be implemented by experts. Common components are typically distributed by the respective sensor node operating system (OS) or by the hardware manufacturer.

2.1.4 Configuration

In addition to choosing suitable components, the performance and reliability of a WSN application is further affected by the configuration of the employed components. Typically sensor network protocols and components provide parameters to fine-tune their operation and to adapt them to the deployment environment and application requirements. A typical example is the timing parameters in media access control (MAC) protocols. As with the selection of protocols, there is usually not a one-size-fits-all setting that is suitable for all applications and environments. Optimal performance can usually only be reached if the parameters are carefully tuned to the given environment. This is typically a tedious and time-consuming task and requires significant knowledge of WSN technology. Consequently, current applications often employ default settings or rules of thumb which result in configurations that are usually far from optimal [Zimmerling et al., 2012]. To make the task more approachable for developers without specialized knowledge, it needs to be automated as much as possible.

As a solution, we propose an optimization-based automatic configuration process that is based on formal models of the employed protocols and the expected environment. As the creation of these models requires extensive knowledge of the inner workings of the components and the important properties of a typical environment, we again assume that these models are going to be provided by WSN experts. Ideally, the protocol models are provided by the developers of the respective protocols themselves. Our approach to automatic configuration is described in more detail in Chapter 5 and Paper F.

2.1.5 Software Deployment

The final step of the creation of a new WSN application is the actual deployment of the new software to the previously installed sensor nodes. While the hardware installation is also an important aspect with challenges of its own, within this thesis, we focus on the software side. Several approaches to reduce the effort required to deploy software to a potentially large number of nodes have previously been developed within the research community [Hui and Culler, 2004; Levis and Culler, 2004; Marrón et al., 2005; Mottola, Picco, and Amjad Sheikh, 2008; Wang, 2004]. With such systems, the deployment of the software can be handled by the respective developers. Most systems do not require manual treatment of the individual nodes, but employ over-the-air programming techniques.

2.1.6 Maintenance

After the initial deployment further maintenance is needed. Sensor nodes may require regular replacement of batteries and broken parts. In addition, it is necessary to monitor the environment for changes that might invalidate the assumptions that were made during the configuration phase. With unexpected changes in the environment, those assumptions might not hold anymore and consequently the performance of the system cannot be guaranteed. We expect that, due to automation, maintenance tasks can be largely handled by the actual end users of the system without involvement of experts. In case of significant environment changes or new application requirements, a developer needs to repeat parts of the development process to update the configuration of the software.

2.2 Architecture

In this section we give an overview of the envisioned system architecture supporting the previously introduced development process. The system architecture integrates the individual tools and components developed as part of this thesis. At this point, we only provide a high-level description of the system and more detail can be found in the respective papers and chapters.

Figure 2.2 shows a graphical representation of the architecture. The system primarily covers two aspects of the development process, the compilation of a macro-program describing the functionality of the application into deployable code and the configuration of underlying system components and communication protocols.

2.2.1 Compilation Tool Chain

The macroprogramming language (MPL) compiler is a central component of the architecture. It receives a *macro-program* written in a newly-developed MPL. As

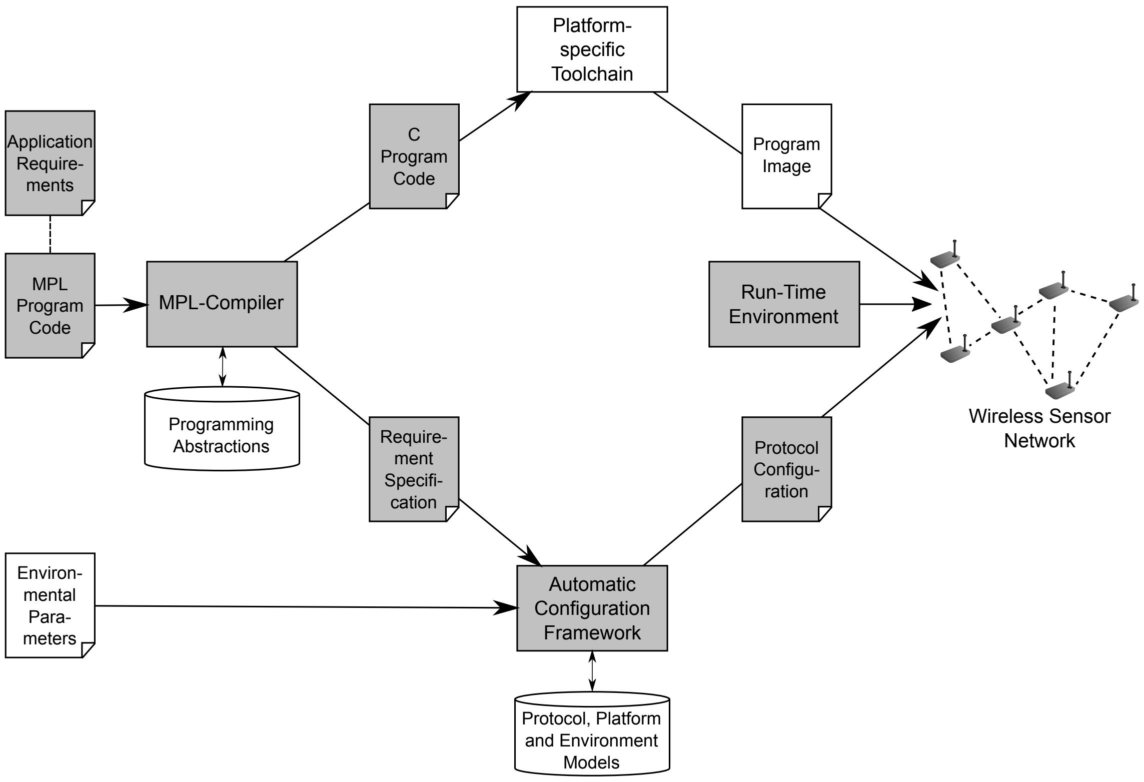
Platform-specific Toolchain
C Program Code
Program Image
Application Require-ments
MPL Program Code
MPL-Compiler
Run-Time Environment
Wireless Sensor Network
Programming Abstractions
Require-ment Specifi-cation
Protocol Configu-ration
Environ-mental Para-meters
Automatic Configuration Framework
Protocol, Platform and Environment Models

Figure 2.2: Overview of proposed system architecture. Components developed as part of this thesis are again highlighted with a gray background.

with other macroprogramming languages, no separate programs for individual nodes are required, but a single program covers the behavior of the network as a whole. Our MPL supports the straightforward integration of existing WSN programming abstractions via a dedicated extension mechanism. This mechanism also supports the use of existing abstraction-specific domain specific languages (DSLs) to enable extensive configuration. To provide a familiar programming experience and to be easy and comfortable to use for the intended users, the language explicitly supports object-oriented programming and employs syntax and semantics similar to the prevalent Java language. The use of an object-orientated language also provides an ideal framework to integrate the previously mentioned programming abstractions. Annotations allow performance and dependability requirements to be directly embedded within the program code. These requirements are later extracted by the macro-compiler and passed to the automatic configuration framework. At compilation time, the user program written in MPL is translated by the MPL compiler into *C code* targeted at the employed WSN platform, such as Contiki OS or TinyOS. Unlike most other compilers, the MPL compiler does not directly generate deployable machine code, but instead the generates C code that is passed to the existing *platform-specific tool-chain* in order to generate deployable *program images*. This separation of concerns allows a large number of different platforms to be easily supported and facilitates the integration of legacy code. Finally, the generated program images can be deployed on the nodes either manually or by employing some remote programming framework. The latter is not within the scope of this thesis, but a number of suitable systems exist [Marrón et al., 2005; Mottola, Picco, and Amjad Sheikh, 2008]. The complete macro-programming framework is described in more detail in Chapter 4 as well as in Papers C, D and E.

2.2.2 Configuration Framework

The second major component of the architecture is an automatic configuration framework for WSN components. The configuration tool takes a formal *requirements specification* as input that can be generated by the MPL compiler based on the performance annotations embedded within the MPL program[1]. The specification consists of a number of constraints on the network behavior and a single property to be either maximized or minimized. To increase the flexibility, the specification also supports probabilistic constraints that only need to hold at a specific point in time with a given probability.

Based on this specification and data about the environment, the configuration tool automatically determines an optimal configuration for the underlying communication protocols and system components. The tool employs mathematical optimization and formal models to accomplish this task. Three types of such models have been developed within the RELYonIT project [RELYonIT Consortium, 2015]: (1) *environment models*, that characterize the relevant properties of the environment; (2) *platform models*, that characterize how hardware components react to the specific environment factors and how their performance and reliability are affected; and (3) *protocol models*, that characterize how protocols cope with the effects captured by the previous models and with a specific parameter configuration. To be usable, these generic models need to be instantiated for a specific application. The environment models in particular require further input data, to represent a specific instance of the environment. These *environmental parameters* are collected at the future deployment site prior to the actual software deployment by model-specific data collection tools. Instead, for some models widely available data that can be collected manually, such as general climate data, is sufficient.

The optimization process of the configuration tool currently employs the stochastic optimization strategies simulated annealing and evolution strategies. These optimization strategies are also able to cope with non-convex search spaces and are usually relatively robust towards noisy data, but in contrast to deterministic strategies, they do not guarantee that the optimal solution is found. Nevertheless, the optimal

[1]In the current implementation, the specification is still written by hand and provided as a separate input.

solution is usually found with a high probability. In addition, it is typically sufficient to find a solution that is close enough to the true optimum. The final output of the configuration tool is a *protocol configuration* for each employed protocol. These configurations are later deployed alongside the application and the components can access their configuration data via a run-time environment described further below. The entire configuration framework is described in more detail in Chapter 5 and Paper F.

2.2.3 Run-Time Environment

Features of both previously described components of the architecture depend on the support of a *run-time environment*, which is deployed on the sensor nodes alongside the user application and the operating system. For the application, the run-time environment serves as an abstraction layer to isolate the generated code from the underlying WSN operating system. In addition, it provides support functions required for the implementation of object orientation and other language features, such as object serialization. In addition, the run-time environment is responsible for the management of the configuration parameters determined by the configuration framework. It provides a well-defined interface to enable the individual components to access their parameters.

While most applications will only employ a single set of requirements, some applications have a number of diverse modes of operation that get activated based on system state and environmental conditions. For example, a system to detect forest fires would be typically optimized for a long system lifetime during normal operation, but would switch to a low latency mode as soon as a fire is detected. To support such applications, the run-time environment provides an interface that enables the application program to select one of several modes, each with its own set of associated reliability and performance requirements. The components are automatically notified of resulting changes in their configuration. The functionality of the run-time environment is presented as part of the respective frameworks in Chapter 4 and Chapter 5 as well as in Paper E and Paper F.

2.2.4 Integration with Business Process Modeling

Within the `make`*Sense* project, we also explored how to provide an even more abstract method of implementing WSN applications by integrating the previously described tools into an intuitive model-based design process. In the `make`*Sense* approach the behavior of the WSN is defined as part of an overarching business process by employing an extended version of business process model and notation (BPMN). This also enables a seamless integration with other business processes executing on dedicated business process execution engines.

BPMN [Object Management Group, 2011] provides a graphical notation for modeling business processes, which may include responsibilities of systems and people. Modern variants of BPMN also specify a translation of the model to executable BPMN that can be processed on a generic business process execution engine.

Individual BPMN diagrams consist of a number of standardized elements. Of primary interest are the different flow elements that are used to define the control flow of the business process. *Activities*, depicted in the diagrams as boxes with rounded corners, represent any kind or work that is executed as part of the process. Two types of Activities are of particular interest. *Tasks* represent elemental activities that are either executed by a person or by a technical system. As such, Tasks are the primary building blocks of most BPMN diagrams. Several elementary Tasks can be combined to form a *Subprocess*, which can then be used in a diagram as a regular task. *Events* are used to represent incoming and outgoing triggers and messages, such as the start of the process or firing of timer events. In the diagram they are depicted by different types of circles. Of special interest are the *Start Event*, a simple circle with a thin border, that represents the entry point for the process and the *End Event*, a circle with a double border, that terminates the entire process.

Events and Activities are linked by *Sequence Flow Connections* that are depicted as solid arrows in the diagram. These Connections represent the control flow of the process. *Gateways* enable the implementation of decision points within the control flow. In the diagram Gateways are depicted as a diamond shape. Several Gateway types exist that represent different conditions, such as "and" or an "exclusive or." Additional Gateways enable the forking and merging of paths to implement parallel execution and synchronization. In addition to the primary control flow, it is also possible to represent data flow within a BPMN diagram in the form of *Message Flow*. Message Flow connections employ dashed arrows. It is possible to specify the content of the transmitted messages by attaching an *Artifact* to the message flow.

Finally, *Pools* and *Swim Lanes* allow activities to be assigned to specific actors. *Pools*, depicted as large labeled boxes containing the associated control flows, usually represent larger entities such as involved companies. Pools can be further divided into *Swim Lanes* to represent subordinated entities or different roles. Activities can be also grouped by *Group* elements that take the form of named dashed boxes with rounded corners in the diagram.

Our extended version of BPMN introduces a number of WSN-specific Tasks that allow the expression of typical WSN operations, such as sensing, actuation, and in-network data processing. These WSN operations can be used in special WSN Pools that contain the Activities to be executed within the WSN, while the remainder of the model is still executed on a conventional business process execution engine.

Figure 2.3 shows an exemplary process modeled in the extended BPMN syntax. The modeled system is supposed to manage CO_2 levels in meeting rooms of a conference center by controlling ventilation. To save energy, the system has access to CO_2 sensors in the rooms and information about future room occupancy (i. e., room

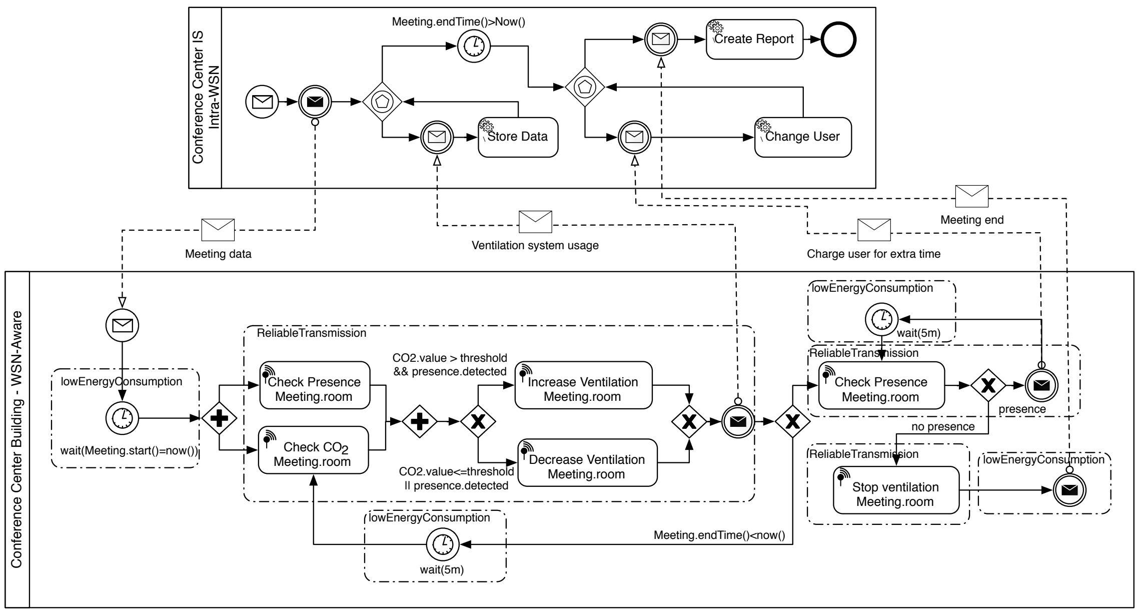
Conference Center IS
Intra-WSN
Meeting.endTime()>Now()
Create Report
Store Data
Change User
Meeting data
Ventilation system usage
Meeting end
Charge user for extra time
Conference Center Building - WSN-Aware
lowEnergyConsumption
wait(Meeting.start()=now())
ReliableTransmission
Check Presence Meeting.room
Check CO2 Meeting.room
CO2.value > threshold && presence.detected
CO2.value<=threshold || presence.detected
Increase Ventilation Meeting.room
Decrease Ventilation Meeting.room
wait(5m)
Meeting.endTime()<now()
presence
no presence
Stop ventilation Meeting.room

Figure 2.3: Exemplary model of a HVAC control system for a conference center employing the extended BPMN [Tranquillini et al., 2012].

booking calendar), which enables on demand ventilation. Rooms are not unnecessarily ventilated if CO_2 levels are still within an acceptable range or the room will not be used in the foreseeable future. The process is partially implemented within the WSN itself and a separate management system. Both sub-processes are realized as BPMN Pools. The Pool on the bottom, named “Conference Center Building,” implements the WSN functionality employing the extended syntax, while the Pool on the top, named “Conference Center IS,” implements the meeting management employing only standard BPMN elements.

During translation, the WSN-specific Tasks are mapped to the programming abstractions within MPL and functions provided for sensing, actuation, and data aggregation. WSN Tasks could also be assigned to a sub-set of the nodes, for example to only execute the Task on the nodes within a specific room of a building, but in the given example, this feature is not used. To enable the user to directly specify additional performance and reliability requirements, WSN-specific tasks can be supplemented by performance annotations. These annotations may differ for different states of the system. In the example, the BPMN Group notation of dashed rectangles is employed to attach these annotations to sets of tasks. These performance annotations are later passed to the automatic configuration framework and allow the generation of system configurations that are guaranteed to meet the expectations of the application. At run-time, the system can switch between these configurations and select the one that is appropriate for the current application state.

The required modifications of the system architecture are shown in Figure 2.4.

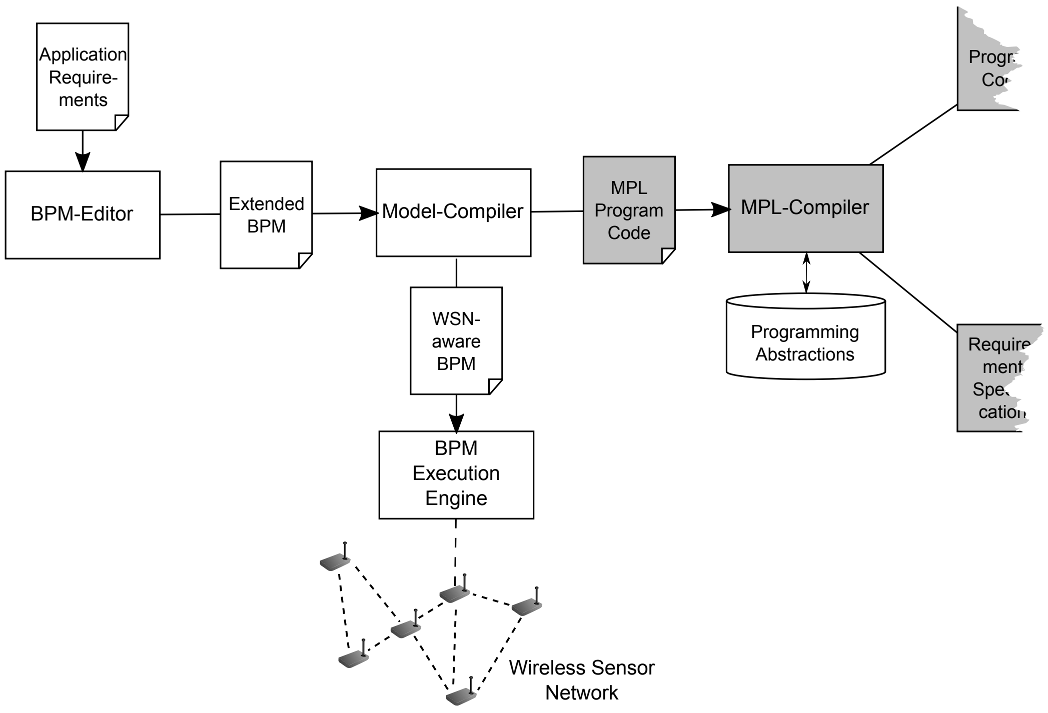

Figure 2.4: Extension of the architecture to enable model based design.

The remainder of the architecture remains largely unmodified and is not included in the diagram. In the extended development process, instead of manually writing a text-based program, the user employs an extended *BPMN editor* to create a business process model (BPM) in the extended BPMN introduced above. During compilation, this model is first split by the *model compiler* into an *intra-WSN part* that executes within the sensor network and a *WSN-aware part* that executes on a standard *business process execution* engine but which can communicate with the intra-WSN part via a dedicated interface.

While the WSN-aware part can be passed to a business process engine largely unmodified, the intra-WSN part is first translated into MPL code by the model-compiler. The generated MPL program is then further processed by the tool chain introduced in Section 2.2. In this scenario, MPL serves as an intermediate language and helps to reduce the semantic gap each individual compiler has to bridge. Performance annotations within the BPM are also translated to respective annotations within the MPL code and later passed to the automatic configuration framework.

A more thorough description of the extended system is given by Tranquillini et al. [2012]. The remainder of this thesis only covers the development process based on textual program input.

3 State of the Art

This chapter reviews the body of existing work on programming and configuration of WSNs. We provide a brief introduction to WSNs including the specific challenges and solutions. In addition, we briefly survey existing WSN applications and deployments.

3.1 Wireless Sensor Networks

The origins of WSN research can be traced to the smart dust vision in the late 1990's [Warneke et al., 2001]. Smart dust is based on truly tiny devices with an edge length of only a few millimeters that are equipped with sensors and that can be randomly spread over a larger area. These devices were envisioned to automatically form networks and collect data over a large area to monitor, for example, the distribution of chemicals or to track the movement of people and vehicles. Nevertheless, this vision was still ahead of its time and early prototypes of smart dust devices could not reach the reliability necessary for real-world deployments. As of today, WSNs consist of larger devices based on off-the-shelf components. The matchbox-sized "mote" was originally conceived as a research vehicle to support the development of novel communication protocols and paradigms for dense networks of autonomous sensing devices, but also turned out to be a viable basis for real-world WSN deployments. Until today, motes still dominate WSN research and applications.

3.1.1 Platforms

Typical WSNs consist of tens to hundreds of networked devices. Different node designs exist, but most modern designs employ a common basic architecture similar to the one depicted in Figure 3.1. The architecture is organized around a central microcontroller (MCU), consisting of an 8, 16, or sometimes 32 bit central processing unit (CPU) providing processing capabilities and integrated memory. The latter usually consists of random access memory (RAM) and persistent program memory. The input/output (I/O) pins and analog/digital converters of the MCU allow the attachment of different sensors and actuators. The selection of sensors ranges from simple temperature probes to more complex chemical sensors [Hayes et al., 2008] or even video cameras [Bagree et al., 2010]. To enable communication, sensor nodes are equipped with low-power radios, usually operating in the 2.4 GHz or

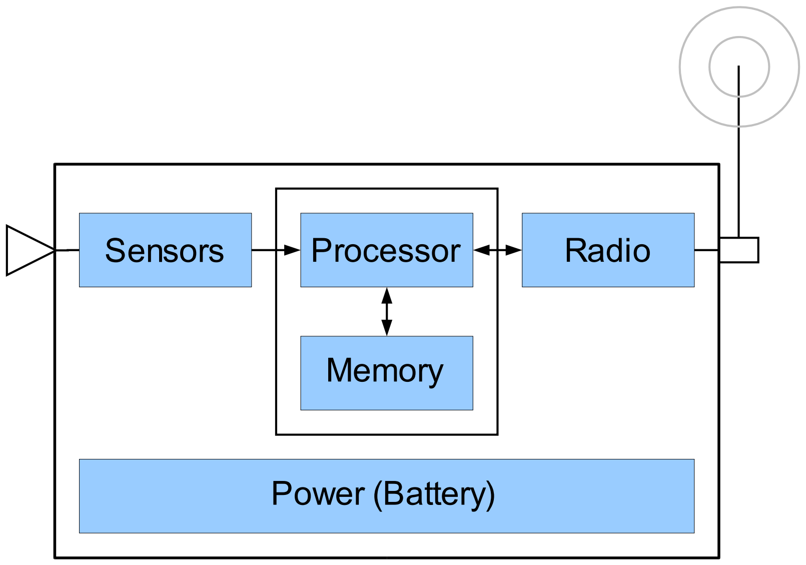

Figure 3.1: Schematic representation of the hardware architecture of a sensor node.

868/915 MHz license-free industrial, scientific and medical (ISM) frequency bands. The wireless communication capability combined with an autonomous power source, most often batteries, enables sensor nodes to be deployed independent of any wired infrastructure. Depending on the application requirements and the employed technology, nodes should usually operate for months to years without a need for battery replacement. A typical representative of such sensor nodes is the Telosb [Polastre, Szewczyk, and Culler, 2005], also known as Tmote Sky. This device was originally developed at the University of California in Berkeley, CA, USA, but the design was made freely available and consequently clones are offered by a number of manufacturers [Moteiv, 2006]. The device employs a Texas Instruments MSP430 MCU [Texas Instruments, 2006] and a CC2420 radio chip [Texas Instruments, 2013].

Due to the limited resources of WSN devices, classical OSs are not suitable for sensor nodes. Even systems targeting embedded devices, like embedded Linux, require significantly more processing power and memory than available on a typical sensor node. Instead, WSNs employ dedicated, lightweight OS that only require a few hundred bytes of memory [Levis et al., 2005]. In addition, these dedicated OSs also put a focus on energy efficiency to extend the lifetime of the battery-powered devices. Like many OSs targeted at embedded systems, most WSN OSs are event-driven. This allows tasks to be serviced in parallel without the need for multiple stacks, like in multi-threaded systems. The use of an event-driven execution model also simplifies power management decisions, as the processor can be put into sleep mode whenever there is no event handler running.

Two WSN OSs can be considered as de-facto standards within the WSN research community: TinyOS and Contiki. The development of TinyOS [Culler, 2006; Hill et al., 2000; Levis et al., 2005] is closely linked to the development of the first hardware devices for WSNs. A notable feature of TinyOS is the provision of a component model on top of C. The use of components enables the reuse of commonly needed functionality and facilitates structuring of larger software projects and also provides the basis for a comprehensive hardware abstraction layer. The component model is realized by employing the custom programming language nesC, which forms a superset of C. TinyOS applications are first translated by a nesC compiler into plain C code that is further processed by the usual tool chain of the platform.

Currently, Contiki [Dunkels, Grönvall, and Voigt, 2004; Dunkels et al., 2004] is the main contender of TinyOS. While the core system is also event-driven, Contiki also provides a thread-like user interface with the help of protothreads. Protothreads are stack-less, extremely lightweight threads that provide a blocking event handler and sequential code execution [Dunkels et al., 2006]. They enable a more intuitive programming model without introducing the overhead associated with regular threads. Both systems have been ported to a large number of different WSN hardware platforms.

3.1.2 Media Access Control

Due to the fundamental role of networking for WSNs, network protocols are an important area of WSN research. MAC and routing protocols have received particularly significant coverage by the research community. The MAC has to address a number of particular challenges in a WSN. As in all ad-hoc networks, MAC protocols are faced with a dynamic environment where nodes may be added or disappear and links between nodes are usually unstable due to interference. In addition, WSN MAC protocols also need to put special attention to energy efficiency. Typical WSN devices only possess a limited energy budget and one needs to extend the lifetime as much as possible. MAC protocols play an important role in the energy efficiency of WSN. The power consumption of the radio usually dominates the total power consumption of the node, so that, in order to conserve energy, the radio should be turned off as long as possible. Controlling the duty-cycle of the radio is an important additional responsibility of MAC protocols in WSNs.

Especially in the early years, a large number of diverse MAC protocols for WSNs were developed [Langendoen, 2008; Langendoen and Halkes, 2005]. Two predominant approaches exist, the use of time division multiple access (TMDA) with mostly fixed schedules [van Dam and Langendoen, 2003; El-Hoiydi and Decotignie, 2004; Rajendran, Obraczka, and Garcia-Luna-Aceves, 2003; Raman et al., 2010; Ye, Heidemann, and Estrin, 2002] and the use of the carrier sense multiple access (CSMA) variants low-power listening [Buettner et al., 2006; Dunkels, 2011; El-Hoiydi, 2002; Moss and Levis, 2008; Polastre, Hill, and Culler, 2004] and receiver-initiated low-power probing [Dutta et al., 2010]. Recently, multi-channel MAC protocols [Al

Nahas et al., 2014; Deligiannis et al., 2015; Duquennoy et al., 2015; Tang et al., 2011] have been receiving increasing interest.

None of the proposed protocols is ideal for all applications scenarios, and the MAC needs to be selected based on, among others, the expected environment and traffic volume. The performance of individual protocols also depends on a number of, often tunable, parameters. To ensure reliable operation according to the user requirements, these parameters need to be adapted to the properties of the environment. MAC protocol configuration is consequently a primary target of our configuration approach.

3.1.3 Routing

Similar to the MAC layer, routing in WSNs also demands specific solutions. While WSNs share many challenges with other ad-hoc networks, they also face some challenges of their own. Conventional routing protocols are typically optimized for a high throughput and low latency. In WSNs these properties are less important as the transmitted data volume is comparatively low and timeliness is often a secondary concern. Instead, due to the limited energy budget, WSN routing protocols need to be optimized for energy-efficient operation. In addition, typical low-power radios of WSN devices only enable short-range communication that does not cover the typical deployment area of a WSN. Consequently, WSN routing protocols need to be able to cope with multi-hop routing within dynamic and complex networks.

A large number of WSN-specific routing protocols exist [Akkaya and Younis, 2005; Pantazis, Nikolidakis, and Vergados, 2013; Reinhardt and Renner, 2015; Winter et al., 2012]. To conserve energy by reducing the amount of transmitted data, WSN routing is often combined with data aggregation. In this case, instead of just transmitting the data unmodified, the information from multiple nodes is aggregated whilst being transported through the network in order to remove unneeded duplicated information [Heinzelman, Chandrakasan, and Balakrishnan, 2002; Intanagonwiwat, Govindan, and Estrin, 2000; Landsiedel, Ferrari, and Zimmerling, 2013]. The properties of WSNs also enable the use of unconventional routing strategies based, for example, on flooding [Ferrari et al., 2011; Intanagonwiwat, Govindan, and Estrin, 2000; Levis et al., 2004; Pazurkiewicz, Gregorczyk, and Iwanicki, 2014] or geographic routing [Flury and Wattenhofer, 2008; Zhou et al., 2010].

Also, IP-based routing is increasingly adopted within WSN systems, enabling the vision of the IoT. A number of 6LoWPAN-based [Mulligan, 2007] low-footprint embedded Internet protocol (IP) stacks [Durvy et al., 2008; Hui and Culler, 2008; Ko et al., 2011] and IP-based routing protocols [Duquennoy, Landsiedel, and Voigt, 2013; Tsiftes, Eriksson, and Dunkels, 2010; Winter et al., 2012] exist.

Like for lower layer protocols, the careful selection and configuration of routing layer protocol parameters, such as the number of retransmissions and the duty cycle, can significantly affect the reliability and performance of a WSN application.

3.1.4 Applications[1]

Early prototypes of smart dust devices developed in the late 1990's were not robust enough for real deployments and key challenges such as energy storage could never be overcome. Consequently, no smart dust deployments outside of controlled lab environments exist. Early mote-based research focused on military applications. WSNs were primarily seen as a new tool for the reliable detection and tracking of intruders and enemy forces [Arora et al., 2004; Pister, 2001] at borders or frontiers. One of the earliest WSN deployments was conducted in March 2001 by researchers of University of California, Berkeley as part of the 29 Palms project [Pister, 2001]. The network consisted of five sensor nodes that were dropped by an unmanned aerial vehicle (UAV) to monitor a road for passing vehicles. Such early military scenarios employed quite complex systems that due to comparatively high data rates already required some in-network processing of the sensed data. This led to a high number of unsolved challenges and consequently these early deployments tended to stay small and to only operate for a few hours.

Almost simultaneously environmental monitoring emerged as a second major application area for WSNs. These environmental monitoring applications tended to be simpler than the contemporaneous military applications, but were usually designed for an extended lifetime. An early example is the first WSN deployment at Great Duck Island in the year 2002 [Kumagai, 2004; Mainwaring et al., 2002; Szewczyk et al., 2004]. This deployment is often seen as the first significant WSN application. In this application a WSN is employed to monitor nesting burrows of birds and to, more specifically, monitor temperature and humidity. The network was deployed on a remote island and the birds should not have been disturbed too often, which led to robustness and longevity becoming major design goals for the deployment. The network was able to generate useful data for the biologist involved and provided higher spatial and temporal resolution than possible with traditional measurement devices at a lower cost and an increased flexibility. Still, the original lifetime goals were usually not met and the network required more maintenance than intended. Over the following years a number of new application areas where explored that introduced new challenges. ZebraNet [Juang et al., 2002; Zhang et al., 2004] was one of the first mobile deployments. Here, a WSN was used to track the movements of zebras. The nodes were attached to the animals and formed a sporadically connected network. Sensed data was transmitted on a opportunistic basis. GlacsWeb [Martinez, Ong, and Hart, 2004] explored a new challenging environment, by deploying WSN nodes within a glacier. This environment posed significant challenges to radio communication and longevity.

Shortly after, the number of applications further increased, a trend also supported by the commercialization of the first WSN platforms, such as Mica2, Mica2Dot,

[1] The entire section is based on a section from a book chapter written by the author [Oppermann, Boano, and Römer, 2013].

and their later evolutions MICAz and TelosB, which became the de facto standard research platforms for WSNs [Polastre, Szewczyk, and Culler, 2005]. At the same time, the software infrastructure matured and the first dedicated WSN operating systems [Culler, 2006; Levis et al., 2005] and middleware systems [Madden et al., 2005] became available. An increasing number of more complex civilian scenarios, such as structural monitoring [Kim et al., 2007; Whang et al., 2004], cold chain management [Riem-Vis, 2004], precision agriculture [Burrell, Brooke, and Beckwith, 2004], emergency response [Lorincz et al., 2004], and health-care [Shnayder et al., 2005; Van Laerhoven et al., 2004], were put to the test. New applications also include the advent of body sensor networks (BSNs), which employ low-power non-invasive or invasive wireless biosensors to monitor, for example, the vital signs of patients [Yang, 2006]. The deployment close to the body introduces new challenges with radio communication, practicability and acceptance. Wireless sensor and actor networks (WSANs) further broadens the application space of WSNs [Akyildiz and Kasimoglu, 2004]. These networks contain not only sensors but also actuators that enable them to actively modify their environment. The inclusion of actuators led to a need for in-network processing and control processes, to enable localized control. Despite a large number of potential applications for such networks, they are still comparatively rare and only a few deployments exist. This might be partially caused by the increased complexity of developing such applications with today's technology. Increasingly, WSN technology was also combined with related technologies, such as mobile robots [Batalin, Sukhatme, and Hattig, 2004], radio-frequency identification (RFID) [Dyo et al., 2009; Dyo et al., 2010], cell phones, or smart cameras [Na, Kim, and Cha, 2009]. This also lead to the introduction of the IoT vision, in which devices are directly connected to the Internet and so provide world-wide access to the sensed data.

Recently, the number of economy-oriented scenarios increased and early real-world applications began to appear [Bijwaard et al., 2011; Ceriotti et al., 2011]. For example, in the SFpark project in San Francisco [San Francisco Municipal Transportation Agency, 2011] WSN and IoT technology is used to implement demand-responsive pricing and a live search for empty parking spots with the aim of steering demand and to reducing congestion in the streets. Nevertheless, such WSN applications outside the scientific community are still limited and most deployments remain prototypical in character. Commercial applications tend not to exploit the full potential of scientific innovations. For example, advanced multi-hop routing protocols are still rarely used in real-world applications.

3.2 Programming Support

WSN systems are complex and difficult to program. Consequently, a need for dedicated programming support has been recognized early within the research community [Römer, 2004]. WSNs are a special case of distributed systems and programming systems for WSNs share a number of challenges and properties with other distributed systems, such as high performance clusters and multi-core systems, for which a number of programming solutions have been developed [Diaz, Muñoz-Caro, and Niño, 2012]. Nevertheless, these solutions usually do not explicitly take energy consumption and unreliable links into account, which are fundamental aspects for WSN systems. In addition, many of these solutions are targeted at shared-memory systems. Consequently, existing solutions typically cannot be directly applied to the domain of WSN.

Initial work regarding WSN programmability was conducted as part of the WSN operating system interface, such as in the case of Active Messages [Culler et al., 2001] as part of nesC [Gay et al., 2003]. More complex systems, developed later, can be divided into two fundamental categories: macroprogramming systems that provide a holistic environment for WSN programming and individual programming abstractions that help to solve specific individual programming challenges.

3.2.1 Macroprogramming Systems

WSN applications were originally written by defining the operation of individual nodes and manually orchestrating their individual behavior to reach the desired goal of the system as a whole. The aim of macroprogramming is to leave this mind-set behind and instead view the network as a single system. With macroprogramming, the behavior of the entire system is defined by only one central program and the task of distributing the functionality to individual nodes is left to the compiler [Kothari et al., 2007]. Most modern WSN programming systems employ some form of macroprogramming.

As of today, a large number of systems that raise the abstraction level of WSN programming already exist. Early systems focus on pure sensing applications. For these systems, a natural abstraction is the representation of the whole system as a distributed database [Madden et al., 2005; Yao and Gehrke, 2002]. Access is provided by a query language similar to the structured query language (SQL). While it is possible to define in-network data aggregation by using special operators, the database view is not well suited for defining more complex processing or control processes as, for example, required by WSANs. More complex queries are possible with the Semantic Streams framework [Whitehouse, Zhao, and Liu, 2006], which employs a Prolog-based interference engine to allow users to pose queries over semantic interpretations of sensor data. In the proposed implementation, data processing is carried out on a more powerful server and not within the network and the execution of control processes is not addressed.

In contrast, programming systems like Pleiades [Kothari et al., 2007] provide extensions to nesC or plain C that enable a more network-centered view. Other systems extend high-level programming languages, such as Python, and provide a compiler to generate intermediate or source code suitable for existing WSN toolchains [Bocchino, Fedor, and Petracca, 2015; Gummadi, Gnawali, and Govindan, 2005; Tu et al., 2011]. An even higher level of abstraction can be provided by systems that employ custom languages [Chu et al., 2007; Hossain et al., 2011; Newton, Morrisett, and Welsh, 2007], but at the cost of requiring the user to learn a new language or, in the case of functional or logic based languages, even a new programming paradigm. Existing programming systems tend to be monolithic and do not provide a well-defined interface to integrate application-specific abstractions. In this regard, our MPL extends the state of the art by providing an extensible macroprogramming framework that supports in-network control logic and is based on Java [Oracle, 2015b], a widespread programming language many programmers are familiar with. Some other WSN systems also employ Java as a WSN programming language, but they usually also employ a more resource-demanding Java virtual machine [Aslam et al., 2010; Brouwers, Corke, and Langendoen, 2008; Lai et al., 2014; Shaylor, Simon, and Bush, 2003]. For example, the Squawk virtual machine uses 80 kB of program memory and consequently targets more powerful ARM-based embedded platforms [Simon et al., 2006]. Our approach differs in that it generates customized C code, which is in turn compiled into optimized machine code for the intended target platform, hence, reducing the introduced overhead. Extensibility is also not specifically considered by any of these virtual machine frameworks and the use of a virtual machine additionally complicates the integration of existing implementations of programming abstractions.

While it is also partially Java-based, our macroprogramming language clearly differs from the standard Java ME framework [Oracle, 2015b]. Java ME is a stripped down version of Java targeted at devices like mobile phones, personal digital assistants (PDAs), and set-top boxes. The devices posses several orders of magnitude more memory and significantly higher processing capabilities. The design of our macroprogramming framework shares some design decisions with the Java Card technology [Oracle, 2015a] that enables Java on smart cards. Smart cards are similar to sensor nodes in regard to processing and memory capacities, but their usage scenarios and modes of communication differ significantly. While sensor nodes actively participate within the communication with their neighbors, smart cards only reply to a direct request by special reader devices either via contact pads or a contact-less interface based on near-field communication. Neither Java Card nor Java ME provides any WSN-specific abstractions or extension points to integrate existing WSN abstractions that significantly increase the utility of our MPL.

3.2.2 Programming Abstractions

While macroprogramming systems provide holistic solutions to WSN programming, programming abstractions provide solutions to specific programming challenges. As demonstrated by extensive surveys conducted by Sugihara and Gupta [2008] and by

Mottola and Picco [2011] a significant number of these programming abstractions used to solve individual WSN programming challenges already exist.

A typical example of such programming abstractions is Logical Neighborhoods [Mottola and Picco, 2006], a system that allows the definition of groups of nodes based on their state. Such groups are defined by using a custom declarative language. Once defined, the programmer can interact with the group in a way that is similar to sending broadcast messages to physical neighbors, but instead of being sent to the nodes within radio range, messages are forwarded to all nodes that match a given group definition. Abstract Regions [Welsh and Mainland, 2004] and Hood [Whitehouse et al., 2004] provide similar abstractions, but are focused on spatial locality. Hood is even limited to only selecting a subset of the physical neighbors. Another example is the DICE system [Gună, Mottola, and Picco, 2014] that allows one to define and monitor network-wide invariants which can be, for example, deployed in intrusion detection applications. The global invariants are expressed by predicates over the state of multiple sensor nodes and are constantly evaluated by a distributed monitoring system. Generic Role Assignment [Frank and Römer, 2005] provides a generic method to assign roles to specific nodes within the network based on user-defined rules. The system can be, for example, used to implement clustering or data aggregation schemes. The Wiselib library [Baumgartner et al., 2010] assembles a wide variety of basic algorithms that are useful in a large number of typical WSN applications. Finally, more specialized systems exist that provide highly-specialized abstractions for specific tasks, such as the tracking of moving objects [Abdelzaher et al., 2004; Luo et al., 2006].

Nevertheless, on their own, such programming abstractions are typically not sufficient to adequately support the design of WSN applications. Each abstraction only focuses on a single task and more complex applications would require the integration of multiple programming abstractions. This is complicated by the fact that interaction between abstractions is typically not considered and consequently, integration of several abstractions is difficult. To overcome this challenge, we propose an integrated macroprogramming framework. A fundamental idea of our macroprogramming framework is to leverage this pool of existing solutions by providing a straight-forward path to integrating existing programming abstractions.

3.3 Design and Configuration Support

In contrast to WSN programming, WSN configuration has received comparatively little attention. Support systems tend to focus on aspects such as deployment support [Dyer et al., 2007; Ringwald and Römer, 2007], node placement [Kim and Cobb, 2012; Xiang et al., 2012], debugging [Beutel et al., 2009b; Ramanathan et al., 2005; Sasnauskas et al., 2010; Sommer and Kusy, 2013], or remote programming [Hui and Culler, 2004; Levis and Culler, 2004; Marrón et al., 2005; Mottola, Picco, and

Amjad Sheikh, 2008; Wang, 2004]. The focus of this dissertation are systems to support the user with assembling and configuring a software stack to support the needs of his or her application. For WSNs this primarily affects the selection and configuration of protocols on the different layers of the network stack.

3.3.1 Component Selection

The automatic selection of components is, comparatively, infrequently considered for WSNs. Nevertheless, existing applications demonstrate a continued need for different protocols with application-dependent properties.

One of the few systems exploring automatic component selection for WSN is the ConfigKit system developed by Peter, Piotrowski, and Langendörfer [2008] [Langendoerfer et al., 2007; Peter, 2011]. ConfigKit allows the automatic selection of suitable software modules for a mission-specific WSN application. Currently, ConfigKit has a strong focus on security aspects, but also considers some WSN-specific aspects like network lifetime. The component selection process of ConfigKit is based on three inputs: a hardware description, a module repository, and an application description. The hardware description defines the parameters and capabilities of the hardware platform to be deployed. For example, it defines the processor type, the operating system, the available memory, available battery life time and a quality classification of the sensor readings. The module repository is a collection of the available software modules. Each module consists of the provided interfaces, the module dependencies, the expected memory footprint, and a security rating. Based on a dependency graph, the tool determines all possible module combinations that satisfy the explicit and implicit module dependencies. Out of these configurations, those that also satisfy the security and energy consumption requirements of the application are selected as viable solutions. Currently, this process does not employ any optimization technique, but employs an exhaustive search. According to the authors, this approach works well enough in practice, but it might lead to issues if the number of available software modules is increased [Peter, Piotrowski, and Langendörfer, 2008]. The existing prototype is focused on security aspects and does not consider most performance aspects. Nevertheless, the general approach could likely be extended to also cover performance. In comparison to our own work, the goals of ConfigKit are largely orthogonal and both works could be combined to cover the automatic selection and configuration of WSN protocols. In this case, ConfigKit could be used to select communication protocols and other software modules based on the user's requirements and our configuration framework could be employed to adjust the configuration of the selected modules to the expected environment and performance requirements.

Component selection was explored in the RELYonIT project [Oppermann et al., 2015b]. The proposed approach provides a semi-automatic structured process to select the most suitable protocol for a specific layer based on previously conducted experiments. In RELYonIT, this approach was only evaluated for the selection of WSN routing protocols, but the authors are confident that the approach can, for example, also be applied to MAC protocol selection. To select a suitable protocol, the user first identifies parameters for the deployment environment and the application. In the examined examples, this includes parameters like temperature, radio interference, message load, and network density. Second, the user needs to specify the relative importance of different performance metrics, such as throughput, transmission reliability, and lifetime. Finally, this information is used to select the most suitable protocol based on a weighted average method. This selection process employs a pre-compiled table with protocol evaluations. This table is generated by systematically evaluating the available protocols under different parameters within a testbed environment. To restrict the size of the table and the number of experiments to a reasonable number, only sensible and promising parameter combinations are evaluated. The selection of sensible parameter combinations is based on expert knowledge and experience. In a number of case studies, this approach enabled the selection of suitable protocols without requiring any experience on the user side. Like ConfigKit, this approach is orthogonal to our own work and could be used to preselect protocols. The selected protocols are further adapted to the properties and requirements of the application by means of automatic configuration.

Sha et al. [2013] propose a MAC layer framework that switches between different MAC protocol implementations at run-time based on quality of service (QoS) requirements and the current environment. Delaying the protocol selection until run-time enables more flexibility and enables adaption to unpredicted environmental conditions. Nevertheless, the dynamic decision prevents performance guarantees and provisioning for different implementations increases the overhead.

Component or protocol selection for WSNs shares a number of properties with the more generic approach of software product lines (SPLs) [Clements and Northrop, 2006], where a generic software system is adapted to an application-specific task by selecting relevant features from a set of predefined options. Even though recently the automatic selection of components based on user requirements is increasingly considered [Guo et al., 2011; Harman et al., 2014; Rosenmüller and Siegmund, 2010], the focus is still on the manual configuration of derived software products. SPLs do not take the specific properties of WSN and the influence of environmental factors into account.

3.3.2 Configuration

A strategy to adapt WSN protocols to the requirements of specific applications that is still widespread is the creation of novel protocols that are exactly tailored to meet the application-specific requirements [Iyer, Woehrle, and Langendoen, 2011]. While this approach provides the widest level of adaptability, a high level of effort is required. Due to the complexity of this approach, it is only suited for experts. Less experienced developers need to resort to the use of the default protocols offered by the employed platform, even though these might not deliver an optimal performance [Beutel et al., 2009a; Langendoen, Baggio, and Visser, 2006]. Parameterization of existing protocols is often a difficult task that requires an in-depth understanding of the implementation of the protocol and the effect of environmental influences.

One approach used to make WSN protocol stacks easier to manage is the creation of overarching modular frameworks [Buonadonna et al., 2005; Klues, Xing, and Lu, 2010; Klues et al., 2007] that can also incorporate some degree of self-management. While these frameworks already reduce the effort needed, they usually still require the user to make fundamental configuration decisions that can significantly affect the performance of the system. In addition, none of these systems takes the impact of environmental changes into account.

Existing approaches for fully-automated protocol configuration usually utilize some form of self-adaption. For example, a number of MAC protocols employ adaptive frequency hopping to counter interference [Tang et al., 2011; Voigt and Österlind, 2008; Yoon et al., 2010]. Principally, such adaption mechanisms can only react to changes and therefore only provide a best-effort performance. In addition, these adaptation mechanisms are typically implicitly integrated into protocols [Boano et al., 2014; Meier et al., 2010] and may even conflict across protocol layers.

Even though the challenge has been identified early on within the WSN research community [Bakshi, Ou, and Prasanna, 2002], only a few systems exist that support users in the complex task of finding optimal configuration parameters for a given environment prior to the actual deployment. The few existing approaches tend to rely on simulation [Bakshi, Ou, and Prasanna, 2002; Simon et al., 2003; Strübe et al., 2014]. With existing simulation environments, this is usually a time-consuming task as the high-fidelity models require significant processing power and consequently only allow a limited speed-up for larger networks. The long computation times significantly limit the number of configurations that can be evaluated and easily lead to overlooking superior solutions. While our approach shares the same basic strategy, we can significantly reduce the run-time of the model evaluation by using more abstract formal models. This allows the evaluation of a larger number of possible configurations and thus increases the likelihood of finding an optimal configuration.

Only a very small number of works apply formal models and mathematical optimization to WSN protocol configuration. Maróti et al. [2003] use an automaton-based process to synthesize and configure an application-specific middleware. Kogekar et al. [2004] employ constraint-guided software reconfiguration for WSN and demonstrated the approach using simulation results for a simple one-dimensional tracking problem. Another example is the pTunes system [Zimmerling, 2015; Zimmerling et al., 2012]. PTunes employs a formal protocol model and constraint programming to find the optimal MAC protocol configuration settings for a specific network topology and radio environment. PTunes' goals are very similar to our approach, but, at least in its current form, pTunes is limited to MAC protocol configuration, while our approach targets protocols at different levels of the network stack. More importantly, pTunes does not explicitly model any environmental effects and only considers internal interference. Instead, pTunes is intended to work online and constantly reconfigure the network, which allows the configuration to be constantly adapted to a changing environment, but significantly limits the available run-time for optimization. Our approach of pre-deployment configuration can use more sophisticated models that require a higher run-time, but lead to more precise and dependable results.

Automatic configuration is already frequently used in other domains such as embedded [Jóźwiak, Nedjah, and Figueroa, 2010] and real-time systems [Chatterjee et al., 2014; Rajkumar et al., 1997], generic networking [Movahedi et al., 2012] and software engineering [Aleti et al., 2013]. Automatic configuration of protocols is also seen as an essential service for the management of large IP-based networks and self-configuration techniques are widely applied [Zhang et al., 2013] and even led to standardized solutions [Guttman, 2001]. Pre-deployment configuration techniques are also frequently used to configure networks [Barik and Divakaran, 2013; Shu, Wu, and Yun, 2013; Sun et al., 2015; Yun et al., 2012]. None of these approaches takes the specifics of WSN and the influence of environmental factors into account. In the networking domain, existing solutions tend to rely on self-configuration at run-time and as a consequence these approaches can only provide a best-effort performance and cannot give performance guarantees.

Similar techniques are also employed for design space exploration as part of the development process of embedded systems [Künzli, Thiele, and Zitzler, 2005]. While employing a similar approach as to the one presented in this thesis, these systems do not aim at automatically selecting a near-optimal solution. Instead, a number of alternative solutions is presented to the user. In contrast to our approach, this requires a sufficient knowledge on the part of the developer to make informed decisions.

3.3.3 Component and Protocol Models

Our approach to protocol configuration relies on accurate, but lightweight models of the environment, the hardware platform, and the employed protocols. A number of protocol models have been developed in the community, mostly focusing on the MAC layer. The presented models also cover some aspects of the radio environment. Polastre, Hill, and Culler [2004] derived an analytical model of node lifetime for the newly developed B-MAC protocol that enables predictions of the performance. Ye, Silva, and Heidemann [2006] employ a model of energy consumption to determine optimal configuration parameters for SCP. Buettner et al. [2006] created a model of the energy consumption and per-hop latency of the X-MAC protocol. A model of the same protocol was later refined and significantly extended by Zimmerling et al. [2012]. The latter model is also suitable for protocol configuration, as demonstrated by the authors [Zimmerling et al., 2012]. An early approach to capture the energy of the entire software stack employs an automation based formalism to model a TinyOS-based WSN [Coleri, Ergen, and Koo, 2002]. Jung et al. [2007] and Kellner et al. [2008] also employ abstract models to predict the overall energy consumption and expected lifetime of a WSN. A survey by Schmid and Wattenhofer [2006] presents a number of additional simple models of different WSN aspects and protocol components, which could be employed as parts of more complex protocol models and hardware platforms.

Within the RELYonIT project, our project partners also developed analytic and Monte-Carlo-simulation-based models of different protocols that are already specifically tailored towards use in an optimization process [Brown et al., 2014; King et al., 2015; Oppermann et al., 2015a; Zúñiga et al., 2014].

Similar performance and QoS models are already successfully employed in a number of diverse application areas. Different types of performance and QoS models are, for example, employed to predict and analyze the performance and reliability of web services and similar software systems [Balsamo et al., 2004; Becker, Koziolek, and Reussner, 2009; Brunnert et al., 2013].

3.4 Conclusions

Within the last years, WSNs have established themselves as a promising technology and we have witnessed an increasing number of real-world applications. Many of the early challenges have seen satisfying solutions. Nevertheless, the large number of diverse protocols and software systems increase the complexity of WSN design and development. Due to the wide spectrum of WSN applications, standardization can only partially remedy this situation. In addition, WSNs are at the intersection of wireless networking and embedded systems and existing solutions are often not applicable as they do not take into account the specific properties and their requirements. The difficulty of WSN design has been recognized within the scientific community and a large number of approaches exist.

Nevertheless, these existing systems are often too specific to gain extensive popularity and real-world use. The integration of different systems as proposed in this thesis, has rarely been considered by the community.

4 Macroprogramming

As of today, the implementation of WSN or IoT applications is a difficult task that requires significant expertize in a number of relevant fields. The low abstraction level of current programming languages and environments forces the user to concern himself with the details of radio communication and the resource-constrained nature of typical sensor nodes. A number of programming abstractions already exist that aim to reduce the burden of WSN programming. Currently, most of these abstractions only target individual aspects and it is usually difficult to combine several of these abstractions to build a more complex application. A survey by Mottola and Picco [2011] indicated that most WSN programming abstractions fit into a taxonomy with a limited number of classes.

The goal is to develop an abstract macroprogramming language that provides a high-level view of the system and allows existing and novel WSN programming abstractions to be seamlessly integrated. This task is additionally complicated by the fact that many of these programming abstractions employ their own domain-specific configuration languages and the resulting need to provide a mechanism to integrate existing DSLs within the macroprogramming language. On the other side, one also needs to keep the resource-constrained nature of WSNs in mind. The language needs to support the generation of highly efficient code that is suitable for typical WSN devices.

Our approach consists of an object-oriented language largely inspired by the well-known Java language [Gosling et al., 2005]. To optimally support the seamless integration of existing programming abstractions, the language provides an explicit extension mechanism. Special keywords allow code fragments written in these DSLs to be directly embedded within the macro-programming language program. The DSL code is not compiled by the main compiler itself, but the translation of these fragments is handled by compile-time components of the respective programming abstractions. To be well suited for low-resource environments, the language implements, among others, a more flexible memory model than Java and features automatic allocation. The latter reduces the amount of code that needs to be deployed on individual nodes by removing features unnecessary for a specific node.

In the following, we will first introduce a meta model for WSN programming abstractions that has been jointly developed within the `make`*Sense* project. In Section 4.2, we then introduce the features of the novel macroprogramming language itself and explain individual design decisions. Next, in Section 4.3 we introduce a compiler framework to realize the conceptualized language and finally conclude the

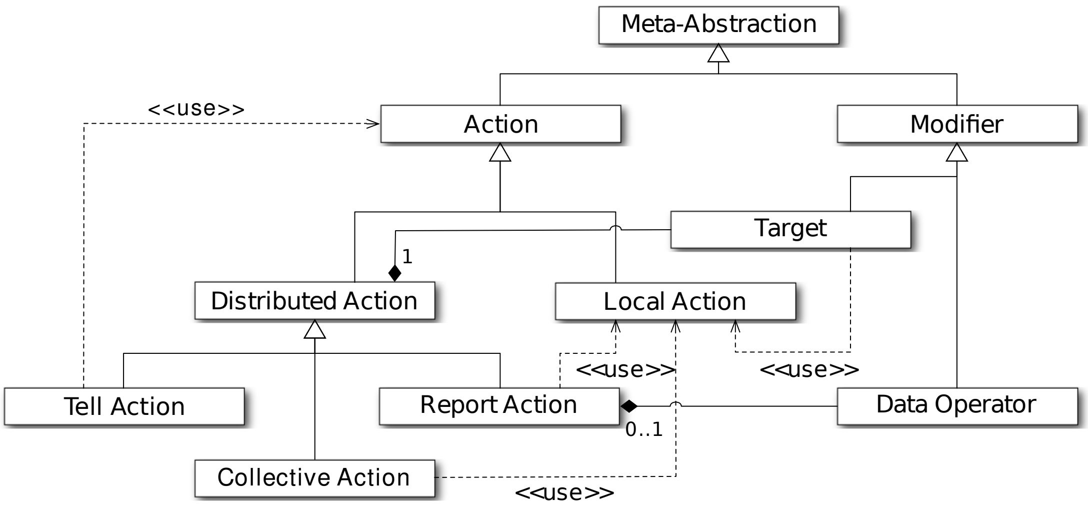

Figure 4.1: The `make`*Sense* meta model for programming abstractions [Casati et al., 2012].

chapter with an evaluation of the applicability and performance of the language implementation in Section 4.4.

The entire chapter is based on and employs material from parts of a number of deliverables and previous publications written by the author of this thesis [Casati et al., 2012; Daniel et al., 2013; Eriksson et al., 2012a; Eriksson et al., 2012b; Eriksson et al., 2013; Gaglione et al., 2013; Mottola et al., 2011a; Mottola et al., 2011b; Mottola et al., 2011c; Oppermann et al., 2014b]. The results presented within this chapter are also covered by Paper C, Paper D, and Paper E.

4.1 The `make`*Sense* Meta-Abstraction Framework

The macroprogramming language is based on a conceptual framework of WSN abstraction jointly developed within the `make`*Sense* project [`make`*Sense* Consortium, 2014]. A large number of WSN programming abstractions exist [Mottola and Picco, 2011], that provide a higher-level of programming and support WSN programmers with their task. Most of these abstractions support a single specific aspect of WSN programming. The framework developed within `make`*Sense* provides a means to categorize programming abstractions based on their intended use. Due to its extensibility, existing and future WSN programming abstractions can be easily integrated within this framework.

A central concept of the framework is the so called "meta-abstractions." The framework organizes programming abstractions primarily based on supported tasks and exported interfaces. Such groups of similar abstractions can be seen as an abstraction of abstractions or in different terms as "meta-abstractions." The different types of abstractions in turn form a hierarchy of increasingly specialized meta-abstractions. In the following, we describe this hierarchy in more detail. It is also shown in Figure 4.1.

Starting from the top, abstractions can be first divided into two major groups: Actions and Modifiers. Actions represent tasks a node or a group of nodes can execute. This can be a simple task, like reading a sensor on a single node, or more sophisticated tasks, like collecting an aggregated temperature value from a larger set of nodes. In contrast, Modifiers do not represent tasks themselves, but allow one to further specify how a task is executed, for example by selecting the set of nodes that actually participates in a distributed task or by specifying how the collected values are aggregated.

Actions can be further subdivided into two major groups, Local and Distributed Actions. Local Actions execute locally on a single node without employing communication between nodes. They provide basic functions, such as reading a sensor value or controlling an actuator. Nevertheless, their use is not limited to the interaction with sensor and actuators. Local Actions can, for example, also be employed to implement data storage in flash memory or to provide sophisticated data processing operations. As such, Local Actions also form the interface to the underlying hardware and operating system. In the context of the macroprogramming language, we also support user-defined Local Actions in order to enable the creation of scripts to be executed on a specific node.

While Local Actions are always bound to a single node, Distributed Actions enable communication and cooperation among nodes. Distributed Actions allow one to request action from other nodes. In the `make`*Sense* framework this is the only means of communication, as explicit message transfer is intentionally not supported.

Distributed Actions can be further classified based on the supported communication pattern. Tell Actions implement one-to-many communication and can, for example, be used to distribute commands to a set of nodes. On each of the receiving nodes, a specified action is triggered. This may be either a Local Action, e.g., steering an actuator, or another Distributed Action. The latter allows the creation of cascades of Tell Actions. Report Actions implement the inverse case of many-to-one communication and, for example, allow the collection of data from a set of nodes. In contrast to Tell Actions, they may only execute Local Actions on the remote nodes. Finally, Collective Actions provide a flexible interface for more complex communication patterns, usually in the form of peer-to-peer communication. Such communication patterns are for example used to implement global assertions [Gună, Mottola, and Picco, 2014] or role assignment [Frank and Römer, 2005].

Modifiers allow one to further customize the exact behavior of Actions. Aspects such as the specification of affected nodes are separated from the implementation of the Action and can be replaced with different concepts. This way, Modifiers provide a separation of concerns and enable increased modularity. Currently, we distinguish between Target Modifiers and Data Operators. Target Modifiers allow one to select which nodes are involved in a Distributed Action. This allows the selection of a subset of the deployed nodes based on some abstract property, for example to select all nodes that are within a specific room or all nodes with a low battery level. Finally, Data Operators are employed to modify data during its transmission through the network. This can be simple data filtering, but could also be used to implement different aggregation schemes. The concept is similar to Query Processing Operators in TinyDB [Madden et al., 2005].

In the MPL implementation, meta-abstractions are used as extension points that can be instantiated with the implementation of a specific abstraction. Each of these instances implements a specific protocol or function. For example, the Target Modifier can be instantiated by Logical Neighborhoods [Mottola and Picco, 2006]. As applications may wish to employ a number of different protocols in parallel, it is possible to instantiate the same meta-abstraction with different abstractions within a single program.

4.2 Macroprogramming Language

The MPL provides a high-level approach to WSN programming and allows to interconnect the previously introduced abstractions to create more complex applications. It primarily provides the glue between the constructs presented in Section 4.1, but also directly enables the user to define algorithms for data processing and similar tasks. It is our goal to keep this language as simple and familiar to the intended users as possible. An average developer should be able to build useful WSN applications with the MPL without an in-depth understanding of WSN technology and without extensive training. Based on these requirements, we chose the widespread Java programming language [Gosling et al., 2005] as a model for the MPL. Like Java, MPL is an imperative, object-oriented language with static typing. Its syntax and semantics follow the Java model to a large extent.

Nevertheless, MPL deviates from the Java model to better support the specific features of WSNs and to make the language more suitable for resource-constrained platforms. A major difference in comparison to common Java implementations is the fact that MPL does not employ a virtual machine. Instead, the program is translated into C code, that is in turn translated to machine code for the employed hardware platform with the existing tool chain. As a consequence, object-oriented features are implemented in detail differently than they are typically realized in Java. The mapping of object oriented features is more similar to the approach employed for the C++ programming language [Stroustrup, 1989]. For example, dynamic dispatch is realized with the help of virtual method tables instead of relying on the more flexible but also more memory-demanding hash-table-based approach typically found in Java implementations. Due to resource constraints, MPL also lacks garbage collection. Instead, it provides and extends the memory model such that it enables the user to largely rely on static allocation of memory. The extended memory model is described more verbosely in Section 4.2.2.

To better support WSN programming, the language also adds a number of features to the Java basis. Most notable are features to ease the seamless integration of existing and new WSN programming abstractions. The language provides predefined extension points that allow the incorporation of those abstractions that best suit the task at hand. This mechanism is described in more detail in Section 4.2.1. In addition, the language provides an extended interface to the underlying hardware, that enables easy access to sensors and actuators. The following sections introduce the central concepts and design decisions of MPL.

4.2.1 Embedding of Meta-Abstractions

An integral purpose of MPL is to provide an implementation of the meta-abstraction framework introduced in Section 4.1. Consequently, the language needs to provide means to seamlessly integrate programming abstraction within an MPL program. In MPL, programming abstractions are represented by three components:

1. an MPL class,
2. a run-time component, and
3. optionally, a compiler plug-in.

The MPL class provides an interface to interact with the programming abstractions, but usually does not itself implement any of the functionality. Instead, the implementation of its methods is relegated to the run-time component of the abstraction. This relegation employs a mechanism of MPL that allows methods to be marked as `native` and to provide a C implementation of these methods. In contrast to the MPL program, these run-time components are expected to be written by WSN experts, that are familiar with writing low-level C code and that can benefit from the additional flexibility. As a side effect, this approach also eases the reuse of existing legacy components.

A number of programming abstractions employ their own domain-specific languages. To enable the reuse of these languages, MPL provides a mechanism to seamlessly embed domain-specific languages. This mechanism is similar to the approach used to embed SQL statements in standard programming languages, like Java. In contrast to prepared statements, embedded code fragments are not represented by regular strings, but employ the special data type `code`. The use of a dedicated data type eases type checking and optimization. Variables of this type can only be instantiated by a constant expression, consisting of domain-specific language statements that is enclosed by the delimiters `{:` and `:}`. As the compiler is not able to correctly interpret the domain-specific language fragments, their handling is

Listing 4.1: Memory allocation with `static` and `auto` [Oppermann et al., 2014b].

```
Object a = static Object();
```

referred to compiler plug-ins provided by the respective programming abstraction. Also in contrast to prepared statements that are typically interpreted at run-time, the embedded code is fully translated by the plug-ins at compile time. The generated code is later deployed along-side the application and the run-time component of the programming abstraction. A more detailed description of the plug-in interface and the functionality of the compiler plug-ins will be given in Section 4.3.

4.2.2 Memory Management

Modern programming languages rely often exclusively on dynamic memory allocation. In particular, if this memory model is coupled with automatic garbage collection, it can help to significantly reduce the burden introduced by memory allocation and can remove a number of possible pitfalls. On the other side, dynamic memory allocation introduces a significant overhead and reduces the efficiency of memory management. This makes the dynamic memory allocation and in particular garbage collection less suited for small embedded devices with limited memory, such as sensor nodes.

To make MPL more suitable for such devices, its memory model encourages the use of statically allocated memory. If absolutely needed, memory can still be dynamically allocated on the heap. In this case, like in the C++ programming language, it is the responsibility of the programmer to free memory that is not needed any more. Garbage collection is intentionally not available. Nevertheless, due to additional modes of memory allocation, dynamic allocation is less frequently needed. Despite of the different memory model, MPL retains the reference semantics from Java and, in contrast to C++, objects are only accessed via references. No variables of object type exist and memory allocation for objects has to be done separately from variable declarations.

To implement this behavior, the `new` operator inherited from the Java language is supplemented by two additional operators for object creation: `static` and `auto`. Both operators are used in a similar manner as the `new` operator, as can be seen in Listing 4.1. Objects allocated with the `static` operator are part of the program image and are available for the entire execution of the program. These objects live until the program terminates and the memory cannot be reclaimed. If the same `static` operator is executed repeatedly, for example in a loop, then the object allocated already is reinitialized at each call. The `auto` operator allows objects to be allocated on the stack. These objects are destroyed if the current block is left. The allocated memory is automatically reclaimed in this case. Similar to `static`,

Listing 4.2: Use of the auto operator within the initializer of a member variable [Oppermann et al., 2014b].

```
class Rectangle {
  Point topLeft       = auto Point();
  Point bottomRight   = auto Point();
}

...

Rectangle rec_local  = auto Rectangle();
Rectangle rec_global = static Rectangle();
```

repeated calls of the same `auto` operator reinitialize objects created by previous iterations. In combination with the member variables of classes, the `auto` operator is also used with slightly different semantics. If a member variable is immediately initialized with an object created by the `auto` operator, the memory required by the object is embedded in the representation of the encapsulating object at compile time. The required memory is thus automatically reclaimed, when the encapsulating object is destroyed. The example shown in Listing 4.2 demonstrates this use of the `auto` operator. In this example two instances of the class `Point` are allocated with the `auto` operator and references of the newly allocated objects are employed to initialize the member variables `topLeft` and `bottomRight` of the class `Rectangle`. When the class `Rectangle` is later instantiated by either using the `auto` or `static` operator as shown in lines 8 and 9, the memory required for the two `Point` instances is also automatically allocated.

A major advantage of static memory allocation is its predictable run-time behavior. The exact amount of memory required can be determined at compile-time and memory allocation cannot lead to run-time errors. Nevertheless, some constructs, like dynamic container structures cannot be implemented solely with static memory allocation. To still enable the use of such dynamic data structures, MPL also provides the `new` operator known from the Java programming language. Objects created with the `new` operator are allocated on the heap. As MPL does not provide automatic garbage collection, the programmer needs to take care of reclaiming the memory when it is no longer needed. To enable deletion of objects, MPL provides the `delete` operator, that destroys an object previously created with the `new` operator and reclaims the memory. As frequent allocation and deallocation of objects may lead to issues like heap fragmentation, the use of `new` and `delete` is only intended as a last resort and the static allocation methods should be preferred whenever possible.

4.2.3 Multithreading

Most WSN programming languages do not explicitly support the parallel execution of code or employ an event-based programming model. Existing multithreading concepts in WSN programming languages are often severely limited. For example, Contiki protothreads [Dunkels et al., 2006] provide a basic multithreading abstraction, but for instance threaded code cannot be reentrant.

To support an increasing need for parallel execution in WSN code and to provide an intuitive programming interface, MPL explicitly supports multithreading code. The support of a full-blown multithreading interface significantly increases the memory overhead, as each thread requires its own stack. In addition, as the stack size is not known in advance, one usually needs to over-provision stack memory. To be able to still limit the memory demand, the maximal number of concurrent threads in MPL is defined at compile time and an attempt to create more threads beyond this limit fails by generating an error signal. Coupled with a highly optimized implementation, this limitation allows the required overhead to be made more predictable and to enable the use of full multithreading on typical WSN platforms, as demonstrated in Section 4.4.

The interface to the multithreading function of MPL is closely modeled after the Java thread interface. The current implementation is based on the Contiki multithreading library, which provides a platform-independent interface for stack switching. Implementations of this interface are available for all major hardware platforms supported by Contiki. The actual thread scheduling is handled by a custom scheduler implemented on top of this library. This scheduler executes within a single protothread-based Contiki process and schedules runnable MPL threads in a round robin fashion. More sophisticated scheduling algorithms than round robin could be easily integrated, if required by future applications. The approach could also be ported to other operating systems by employing similar support functions provided by TinyOS and other WSN operating systems.

4.2.4 Object Serialization

The distributed nature of WSNs, frequently requires the transmission of data between nodes. In object-oriented languages, the data is usually organized as objects and consequently some mechanism to transfer objects between nodes is required. In addition, Distributed Actions allow other Actions to be executed on remote nodes. Instances of these remotely executed Actions may contain attributes, that need to be transfered to the remote node before execution. In MPL, both requirements are supported by the same built-in object serialization mechanism.

The object-serialization facility allows the state of an object to be written to a standardized flat representation as a byte array. This byte array can later be used to recreate an exact copy of the serialized object on the same or a different

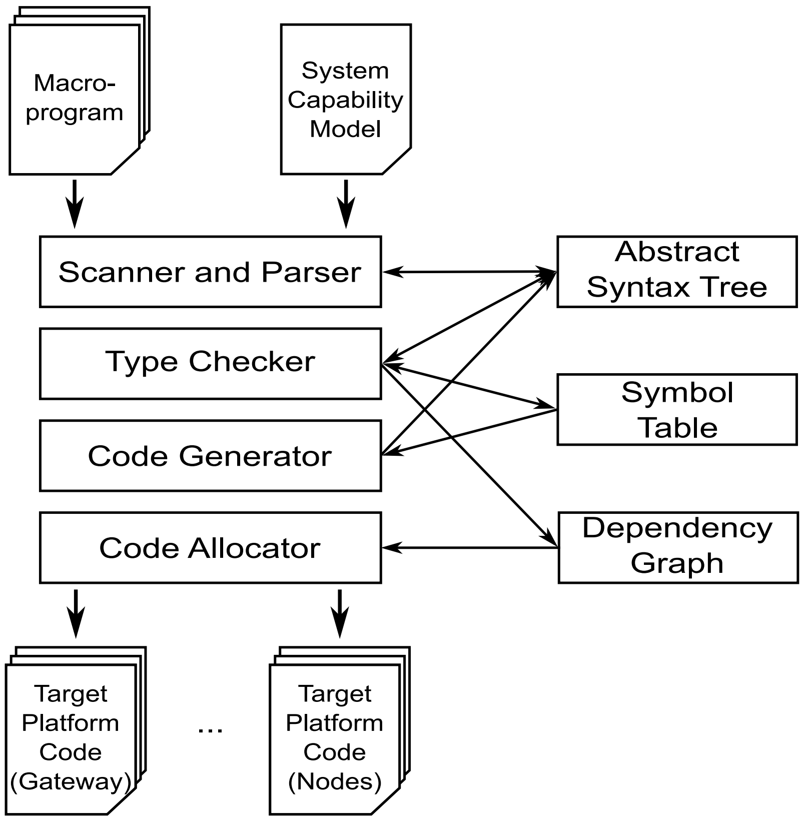

Figure 4.2: Overview of the architecture of the macroprogramming language compiler [Oppermann et al., 2014b].

node. Object serialization in MPL employs a language interface similar to the one employed by Java. This interface consists primarily of the special marker interface `Serializable` to distinguish serializable objects and a build-in class to provide the actual serialization and derserialization functionality.

4.3 Implementation

The language is implemented by a prototypical compiler that translates an MPL program into C code for the nodes of a WSN. Figure 4.2 gives a high-level overview of the architecture with its central components and artifacts. Arrows denote the data flow between these elements. At the start of the compilation process, the macrocompiler receives a macroprogram written in MPL as input. The MPL program is supplemented by an abstract description of the hardware and network properties in the form of a system capabilities model. The information provided by the system capability model can be used to aid optimization and the allocation of specific functions to individual nodes in the network. In addition to these inputs, the macro-compiler has access to a repository of components implementing the available programming abstractions.

The compiler is implemented as a multi-pass compiler and the compilation process consists of four distinct phases: parsing, type checking, code generation, and code allocation. All phases access a shared abstract syntax tree (AST) and symbol table. Information sharing between the individual phases is largely limited to these shared data structures. Both the *scanner* and the *parser* are automatically generated from an extended Backus-Naur form (EBNF) representation of the language grammar with the help of the ANTLR parser generator [Parr, 2007; Parr and Fisher, 2011]. The parser builds an initial AST to represent the syntax of the program. The information in the AST is further complemented by the following compilation phases. The *Type checking* phase is split into two passes to support forward declarations for classes, interfaces and methods. Both passes of the type checker are implemented employing the visitor pattern [Gamma et al., 1995]. In addition to checking the consistent use of types, the type checker builds a dependency graph for all classes and interfaces, which is later used to support the allocation of code and objects to the different node classes (e.g., “gateway” and “node”). The implementation of the type checker incorporates ideas from the Espresso compiler [Doerig, 1998], such as the use of a distance matrix for type conversion. During *code generation* the complete macroprogram including all imported classes and interfaces is translated to C source code. In a second step, the generated code is further compiled into executable binary code by the existing compiler of the target platform. Currently this second step is initiated manually, but could be handled by a separate utility that manages the full compilation process in the future. In a typical application, not all functionality is actually required on all nodes. For example, if only a subset of the nodes is connected to an actuator, only the nodes with access to one will execute the corresponding control code. If the same program image is deployed to all nodes, this wastes memory by also deploying functionality for which it is known that it will never be needed. To reduce memory overhead, in the final *code allocation* pass, the MPL compiler generates individual code images for a predefined set of node classes. Each code image only includes the implementation of classes that can be potentially used on nodes of the corresponding class. In the current implementation, the macro-compiler only distinguishes between two node classes, a special gateway node and the remaining sensor nodes. In the future this classification could be extended with additional predefined and custom node classes.

A central feature of the MPL language is the possibility to extend the language with WSN-specific programming abstractions. The methods of abstraction classes are typically not implemented directly with MPL code, but instead refer to an external implementation provided by the run-time environment. To indicate that the method is implemented by external C code, it is marked as `native` within the class definition. The interface between MPL and native C code is largely similar to the one employed by the Java native interface (JNI) [Liang, 1999]. It provides a mapping between class and method names as well as native and foreign data types. Programming abstractions that make use of custom DSL code, in addition need to provide a compiler plug-in. During translation of the MPL program, this plug-in handles the parsing, type checking, and translation of the abstraction-specific code fragments. A dedicated plug-in interface allows these plug-ins to interface with the MPL compiler at compile-time. The plug-ins are only loosely coupled with the MPL compiler and each plug-in is essentially a small independent compiler with its own parser and code generator. This approach enables a large degree of freedom in the design of individual abstraction-specific DSLs.

4.4 Evaluation

To evaluate the feasibility and usefulness of the approach, we compare an MPL-based implementation of three typical WSN applications with a functionally equivalent implementation employing hand-written C code targeted at the Contiki operating system. For each of these applications, the performance of both implementations is compared in terms of typical software metrics. In the following we first briefly describe the goals and properties of each application scenario.

4.4.1 Application Scenarios

To evaluate the performance of the MPL-based approach, we selected the following set of WSN applications with WSN-typical properties.

Blink to Radio

The first scenario consists of a typical example application with one-to-many communication. A command is send from the gateway node to all nodes within the network. Upon receiving this command, the receiving nodes toggle their light-emitting diode (LED). Similar communication patterns are often used to distribute configuration settings within a network or by WSANs to control actuators.

The MPL implementation of the application uses the Logical Neighborhoods and Tell programming abstractions to realize the previously described behavior. The Logical Neighborhoods abstraction [Mottola and Picco, 2006] provides a custom declarative language to define groups of nodes based on state. Once defined, the programmer can interact with these groups in a similar way to sending broadcast messages to physical neighbors, but instead of being sent to the nodes within radio range, messages are forwarded to all nodes that match a given group definition. The Contiki-based implementation uses the Trickle protocol from the Rime stack [Dunkels, Österlind, and He, 2007] to accomplish the same behavior.

Collect

The second scenario employs many-to-one communication to collect temperature and light intensity readings from all nodes within the network. The data is transmitted to the gateway node for further processing. Data collection is a typical WSN task and most WSN or IoT systems employ similar communication patterns.

The MPL implementation of the application uses the logical neighborhoods and report programming abstractions. Processing of the collected readings is done by an averaging Data Operator. The Contiki-based implementation employs the collect protocol from the Rime stack.

HVAC application

The final scenario consists of a more complex application, based on a demonstrator application used within the `make`*Sense* project. The application controls a heating, ventilating, and air conditioning (HVAC) system to ensure that the CO_2 level is kept within comfortable limits while reducing the amount of ventilation in order to save energy. As in the actual demonstrator, the nodes are spread over two rooms and ventilation is controlled individually for each room.

The MPL implementation employs Tell abstractions and Logical Neighborhoods. Processing and control are not implemented centrally at the gateway, but are offloaded to one selected node per room. Currently, leader election is rather simplistic and one of the available actuator nodes within each room is chosen based on an attribute set at deployment time. The hand-written Contiki code is again based on the Rime protocol stack and attempts to closely mimic the behavior of the MPL application. Data processing is also offloaded to nodes within the network, but in contrast to the MPL application, the selection of these nodes is fixed as part of the application program. In this regard, the hand-written application is significantly less flexible than the MPL variant, as any change of the network organization would require modifications of the program code.

4.4.2 Performance

For the evaluation, both variants of all three applications were implemented by a computer science student with little prior knowledge of WSN programming and compiled for the Tmote Sky with the default Contiki tool chain. The resulting application programs are executed within a simulated network consisting of five to six nodes, employing the event-based simulation framework Contiki Cooja [Eriksson et al., 2006], while the gateway application is executed in a non-simulated personal computer (PC) environment and communicates with the simulated network via a serial socket and a simulated interface node. The use of simulation already allows the elimination of most external influences and thus generates repeatable experi-

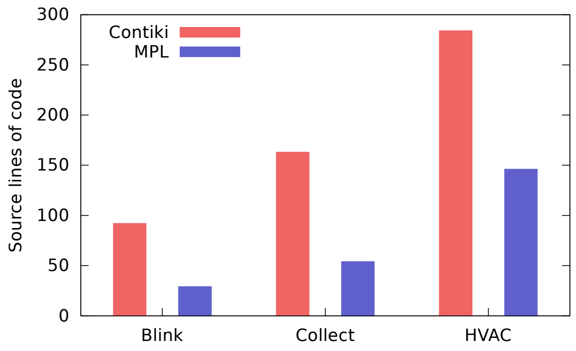

Figure 4.3: User-written source lines of code per application [Oppermann et al., 2014b].

mental conditions. To reduce the remaining effect of randomness in the network emulation of the simulator and due to the non-simulated execution of the gateway application, each experiment is repeated ten times. During the experiment, each application is executed until a scenario-specific termination condition is met. The same termination condition is used for the MPL-based and the manually implemented application. For each implementation, we investigate the following software metrics:

Source lines of code The number of lines of code is used as a course-grained measure of the programming effort. Even though this is a very imprecise measure, it can still provide some insight into the relative complexity of the different implementations. For the line count, all source files are considered that are typically written manually by the application programmer. Comments and blank lines are excluded, as they do not affect the complexity of the actual code.

Code size The size of the code image is especially relevant for memory-constrained devices such as sensor nodes. Sensor nodes such as the Tmote Sky nodes have a program memory of only 48 kB [Moteiv, 2006], which significantly limits the maximal size of programs. The code size is measured with the help of the `size` program from the GNU Binary Utilities.

Static memory consumption Similar to program memory, the amount of available RAM is also severely limited. We distinguish between the amount of statically allocated memory that can be determined prior to execution and the dynamically allocated memory captured by the next metric. Our measure of static memory allocation also includes the memory reserved for stack space. To measure the static memory allocation, we also employ the `size` program from the GNU Binary Utilities.

Dynamic memory consumption To estimate the use of additional dynamically allocated memory, we also determine the maximal encountered heap alloca-

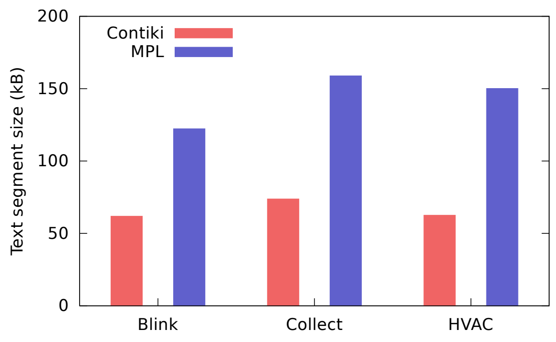

Figure 4.4: Code size of the compiled gateway programs [Oppermann et al., 2014b].

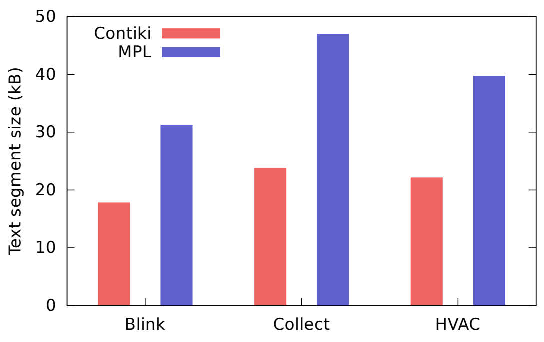

Figure 4.5: Code size of the compiled node programs [Oppermann et al., 2014b].

tion at run time. Heap allocation is measured by linking the applications to a modified `malloc` implementation that tracks heap usage. Effects such as memory fragmentation are not accounted for and the memory demand might be slightly higher in reality.

Computational effort To coarsely estimate the effect on performance and energy consumption, we count the CPU clock cycles spent while running the program for a predefined time interval. The count of the spent CPU clock cycles is provided by mspsim the cycle-accurate WSN node simulator employed by Cooja.

Communication overhead Wireless communication typically has the most significant impact on the energy consumption of wireless sensor nodes. To estimate the relative overhead in terms of communication, we determine the total number and size of the messages transmitted within the network while running the program for a fixed time interval. Both values are derived from radio message traces automatically recored by the Cooja simulation environment.

In the following we present the results of the conducted experiments in terms of the above metrics and discuss the implications of these results. For all three application scenarios the MPL program requires a significantly lower number of lines of code to implement the same functionality than the hand-written versions as can be seen in Figure 4.3. This indicates that less effort is required to write a functionally equivalent program with MPL. The interpretation that MPL reduces the programming effort is further strengthened by the results of a usability study conducted within the context of the `make`*Sense* project [Eriksson et al., 2012b].

Still, these benefits come with a cost. The use of MPL increases the overhead. As can be seen in Figures 4.4 and 4.5 the text size of the code images is higher for the MPL-based implementations. On resource-constrained devices, such as wireless

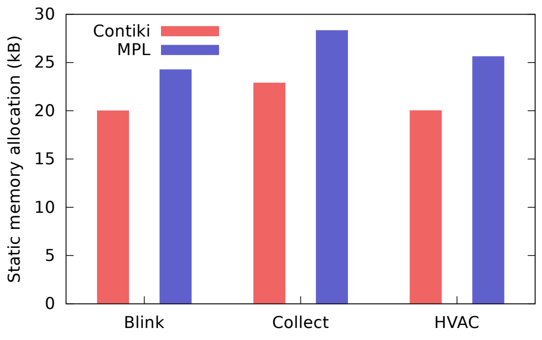

Figure 4.6: Static memory allocation of the gateway programs [Oppermann et al., 2014b].

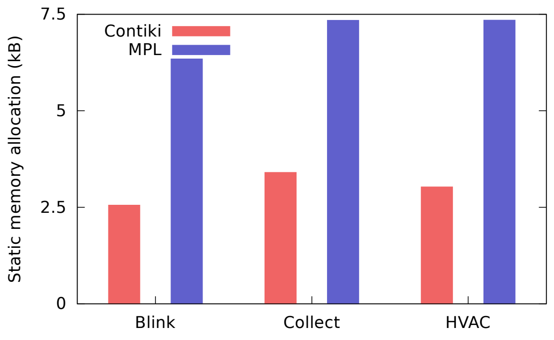

Figure 4.7: Static memory allocation of the node programs [Oppermann et al., 2014b].

sensor nodes, this is a serious drawback. Nevertheless, the memory demand is still well within typical platform limits. In addition, a significant share of the memory requirement is caused by the fixed-size run-time environment, so that the increase in memory demand for larger applications can be assumed to be moderate. The MPL implementation also provides additional flexibility that is not thoroughly used by the evaluated applications. A manual implementation of these features would also increase the code size of the hand-written applications and would additionally increase the programming effort even more. The use of libraries with a larger memory footprint also leads to a higher memory consumption of the Collect application even though the size of its program code is actually smaller. This effect is more distinct for the MPL-based application.

Also the memory consumption at run-time is higher for the MPL code as can be seen in Figures 4.6 and 4.7. The overhead is particularly significant on the nodes. On the gateway, which is usually a more powerful machine, the slight relative increase in memory demand is less problematic. The overhead in terms of memory demand is mainly caused by the data structures required for the support of object orientation. We assume that the relative overhead will also decrease for more complex applications. In addition, the current implementation of object-oriented features leaves potential for optimization.

Dynamic memory allocation is only used by the MPL code, but the maximal amount of dynamically allocated memory is comparatively low. As can be seen in Figure 4.8, none of the applications allocated significantly more than 420 bytes at a time on any node. Consequently, for the evaluated applications, the overhead is negligible.

In contrast to memory demand, the computational overhead introduced by the use of MPL is very low as can be seen in Figure 4.9. Surprisingly, for two of the scenarios, the MPL program requires even fewer CPU cycles for the same task. This demonstrates that features like virtual method dispatch do not introduce significant

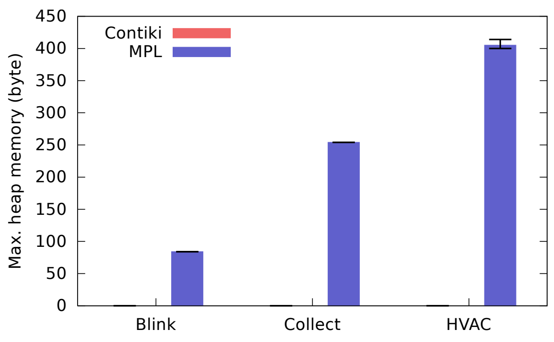

Figure 4.8: Maximal dynamic memory allocation on the nodes [Eriksson et al., 2013].

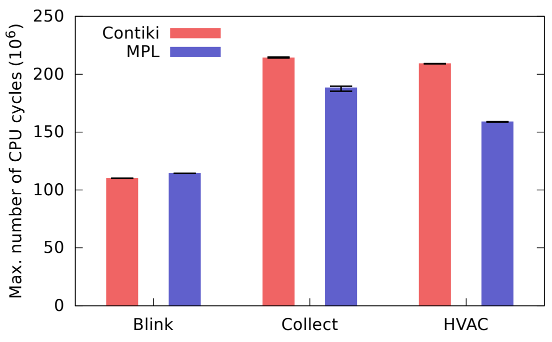

Figure 4.9: Maximal number of spend CPU cycles on the nodes [Oppermann et al., 2014b].

overhead and the high-level abstractions help the programmer to implement efficient algorithms.

Similarly, the number of transmitted radio messages also stays largely the same as can be seen in Figure 4.10. Still, the average size of the messages increases for the MPL-based solution as shown in Figure 4.11. The significant overhead is mainly introduced by the extensive use and naïve implementation of object serialization. As can also be seen in Figure 4.11, the relative overhead is higher for those applications that transmit little actual data. For the collect application, which transmits a significant amount of data, the relative overhead is comparatively low. The current implementation of object serialization leaves significant potential for further optimization. The transmission of pure signaling messages without associated user data could particularly be improved, as these currently carry a large amount of dispensable data.

The macroprogramming language versions of the applications indeed require less effort as was demonstrated by a reduction of source lines of code by 50% and more in comparison to hand-written C code. This indicates that less effort is required to write a functionally equivalent program with MPL and the programmer's burden is reduced. The interpretation, that MPL reduces the programming effort is further strengthened by the results of a usability study conducted within the context of the `make`*Sense* project [Eriksson et al., 2012b].

Nevertheless, the reduction in programming effort comes with a significant cost. The binary code size and memory usage increase significantly for the macroprogramming-language-based applications. Nevertheless, the absolute numbers are still well within the limits of the employed platform and the overhead is largely due to a higher flexibility and additional features of the framework. Consequently, we expect the effect to be far less significant for larger, more complex applications.

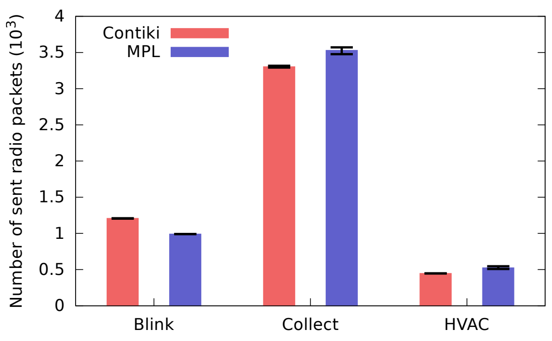

Figure 4.10: Total number of transmitted radio messages [Oppermann et al., 2014b].

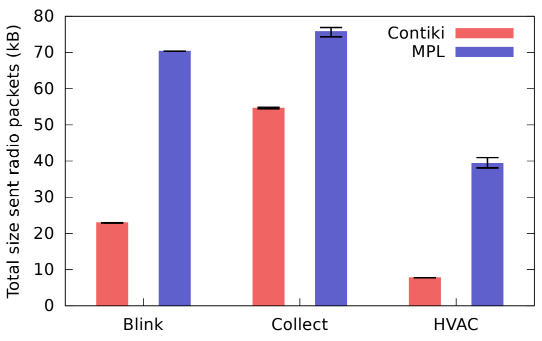

Figure 4.11: Total size of transmitted radio messages [Oppermann et al., 2014b].

In contrast to memory consumption, the execution efficiency measured in employed CPU cycles and transfered messages, is the same or even slightly superior for the macroprogramming language applications. This demonstrates that it is indeed possible to create efficient programs with an abstract high-level language. The observed slight increase in performance is most likely due to the macroprogramming language solutions benefiting from the expertise of the original developers of the employed programming abstractions.

In conclusion, these results demonstrate the benefits and the feasibility of the approach. The observed overhead is still low enough to make the approach suitable for devices with low resources, such as wireless sensor nodes.

5 Automatic Configuration

While a novel macro-programming language with a high level of abstraction already significantly reduces the complexity of successfully creating a WSN application, this is not sufficient to enable non-expert users to successfully implement such applications on their own. To reach a suitable performance and dependability, the WSN communication stack needs to be carefully tuned to the properties of the application scenario at hand. An optimal configuration strongly depends on the properties of the environment and the user's requirements. To also reduce the effort of configuring a WSN system and to make the technology truly approachable for non-experts, we propose an automatic configuration framework to supplement the previously introduced macroprogramming framework.

As the selection of an optimal configuration depends on the user's requirements, we also need to provide a way to support the user with concisely formulating his requirements. To assist this task, we developed a catalog of relevant properties that are meaningful to the intended users and that are suitable as a basis for an automatic parameterization approach [Oppermann and Peter, 2010].

The actual configuration framework relies on mathematical optimization to determine a near-optimal protocol configuration for a specific application based on a user-generated requirement specification and models of the employed protocols, hardware platform, and environment parameters. The generated protocol configuration is later deployed on the sensor nodes alongside the user's application code and the implementation of the employed WSN protocols. A runtime support component provides a convenient interface for protocols to access their configuration settings and enables switching between different configurations based on the current state of the system.

In the following, we first describe the underlying modeling framework developed within the RELYonIT project. Section 5.2 introduces the optimization strategies employed by the framework. In Section 5.3 we take a closer look at the underlying mathematical concepts of the optimization process. Section 5.4 introduces the system architecture and highlights important aspects of its software implementation. Finally, in Section 5.4 we evaluate the applicability and performance of the system.

The entire chapter is based on and employs material from parts of a number of deliverables and previous publications written by the author of this thesis [Mottola et al., 2013; Oppermann et al., 2015a; Oppermann et al., 2015b]. The results presented within this chapter are also covered by Paper B and Paper F.

5.1 Environment, Platform, and Protocol Models

To enable the automatic selection of protocol configuration parameters, it is necessary to be able to reliably predict the performance to be expected with a defined parameter set. Previous approaches often relied on simulation. With sophisticated simulation environments, it is possible to yield a reliable prediction of the expected behavior with a high fidelity, but the simulation of complex systems tends to require significant resources and is very time consuming. The extensive run-time required to obtain reliable results, limits the number of possible configurations that can realistically be evaluated and thus reduces the chances of actually identifying an optimal solution. To overcome these limitations, we opted to employ more-abstract analytic models that only capture the most relevant properties. Such models provide less fidelity, but the use of abstract mathematical models enables the evaluation of a significantly higher number of possible solutions and combined with established optimization strategies, enables a reliable identification of optimal configuration settings.

For the configuration framework, we employ a conceptual modeling framework that has been developed within the RELYonIT project. The framework consists of three categories of interacting models:

1. *Environment models.* WSNs are often deployed in challenging environments and thus the properties of the environment have a profound effect on the performance of a WSN. Environment models provide a systematic approach to capture the characteristics of the deployment environment. These models consist of an abstract representation of those environmental parameters that directly or indirectly affect the performance of the system. The complexity of the representation can vary between models. A more complex environment model could for example capture the exact temporal distribution of temperature values while a simpler environment model could limit itself to reflecting expected minimal and maximal temperatures within a specific time interval.

   For each specific deployment environment, an individual instance of the relevant environment model needs to be derived. Such an instance captures concrete values for the key parameters of the model and is such adapted to a specific location. Depending on the model, these location-specific values can be either derived from readily available data, such as climate records, or they are determined by running a data collection application prior to the actual application deployment. As the required data conversion and data collection applications tend to be model-specific, we assume that they will be co-developed with the respective models.

   In this thesis, we focus primarily on two environment aspects, temperature and radio interference. Nevertheless, the developed concepts can be easily extended to other environmental factors.

2. *Platform models.* The performance of a WSN is not only affected by the environment, but the capabilities and properties of the employed hardware platform also play an important role. In addition, different brands and types of sensor nodes react differently to the same environmental conditions. Both aspects are captured by platform models. A platform model would, for example, capture the relationship between the on-board temperature of sender and receiver nodes, and the attenuation of the received signal strength for the employed hardware platform [Boano et al., 2013].

   Within this thesis, we focus on models of the Maxfore MTM-CM5000-MSP node platform and the CC2420 radio chip.

3. *Protocol models.* Finally, protocol models allow one to predict the performance of a WSN protocol under certain environmental conditions and with a predefined hardware configuration. Typically, the entire performance of a protocol cannot be measured by just a single metric. For example, in the case of a routing protocol one would at least be interested in the performance in terms of latency, throughput, and energy consumption. Consequently, protocol models usually provide performance measures in a number of different metrics. The protocol models rely on the previously introduced models to factor in the influence of the environment and hardware platform. In addition, protocol models expose all relevant configuration parameters of the underling protocol implementation, to enable the evaluation of different configurations.

5.2 Optimization Strategies

The actual configuration process is based on mathematical optimization. The current prototype implements the two stochastic optimization strategies: simulated annealing and evolution strategies.

In contrast to deterministic strategies, stochastic optimization strategies are not guaranteed to find the optimal solution, but are usually more robust to noisy data and require less fine tuning for a specific problem instance. Suitable stochastic strategies still possess a very high probability of convergence and are usually at least able to find a near-optimal solution within a reasonable time. Due to their inherent randomness, stochastic strategies are able to escape local optima and to approach a global optimum even in a non-convex search space.

In earlier versions of the prototype, we also explored the use of different deterministic optimization strategies, but these proved unsuitable for our framework, as they require a significant amount of fine-tuning for each problem instance. Deterministic strategies have the advantage of repeatably generating the same results if the input is left unchanged and are guaranteed to reliably find an actual minimum or maximum, even though not necessarily a global one.

Algorithm 5.1: Basic algorithm of Simulated Annealing.

```
$c \leftarrow$ random_configuration()
$e \leftarrow$ evaluate($c$)
for $i \leftarrow 0$ to $i_{\max}$ do
    $t \leftarrow$ temperature($i$)
    $c' \leftarrow$ neighbor($c$)
    $e' \leftarrow$ evaluate($c'$)
    if $p(e, e', t) \geq$ random$(0, 1)$ then
        $c \leftarrow c'$
        $e \leftarrow e'$
    end if
end for
return $c$
```

In addition to the true optimization strategies, we also implemented an exhaustive search as an alternative. While, due to its extensive run-time, exhaustive search is not suitable for the configuration of larger systems, it is still useful for debugging models and as an evaluation baseline.

In the following, we briefly describe the most important characteristics of the employed optimization strategies.

5.2.1 Simulated Annealing

Simulated annealing is a probabilistic meta-heuristic originally developed by [Kirkpatrick, Gelatt Jr., and Vecchi, 1983]. For a large number of problems it is more efficient than exhaustive search, but as a stochastic strategy, cannot guarantee that a true optimum is found. An advantage over deterministic strategies is the ability of simulated annealing to escape local optima and to find a global optimum also in non-convex search spaces.

The idea of the simulated annealing strategy is taken from metallurgy, where slow cooling of metals is employed to improve the crystalline structure of the material. The strategy emulates the reduction of energy in the material during the cooling process. At the initial high temperatures, the material structure can still change significantly. At later stages, when the material gets colder, only increasingly small changes are possible and the state of the material converges to a final configuration.

To emulate this behavior, simulated annealing employs a basic algorithm similar to the one shown as pseudo code in Algorithm 5.1. The algorithm starts with a randomly initialized configuration and determines the initial energy e for this configuration by applying the problem-specific *evaluate* function. The energy represents a problem-dependent cost metric, while lower energy levels denote superior configurations. After the initialization, the following steps are repeated until a predefined number of iterations $i_{\max}$ is reached. In each iteration, the current temperature is adjusted according to the temperature schedule implemented by the *temperature* function. The exact schedule is typically adapted to the problem at hand, but it needs to ensure that the temperature is monotonically decreasing. After adjusting the temperature, a new configuration c' is generated by a problem-specific *neighbor* function. This functions randomly selects a configuration that is close to the current configuration and has values that deviate only slightly. The new configuration is evaluated employing the *evaluate* function and the resulting energy measure e' is compared with the energy e of the current configuration. Based on the comparison, a probabilistic decision is made, if to keep the old configuration or to continue with the new one. This decision is based on the function p for which we assume the following definition [Kirkpatrick, Gelatt Jr., and Vecchi, 1983]:

$$p(e, e', t) \mapsto \begin{cases} 1 & e' < e \\ \exp(-(e' - e)/t) & \text{otherwise}\,. \end{cases} \tag{5.1}$$

If the new configuration has superior properties and consequently a lower energy, it is selected and the process continues with this configuration. Nevertheless, inferior configurations still have a slight chance of being chosen, which enables the algorithm to escape local optima. The likelihood of selecting an inferior solution for continuation decreases with temperature. Consequently, the algorithm behaves similar to a random walk at very high temperatures, while at lower temperatures its behavior is more similar to a steepest descent and the process quickly converges towards an optimal solution. After employing the maximal number of iterations, the algorithm returns the current configuration as final solution.

A common optimization of the algorithm is to not just store the current configuration, but also to save the best configuration seen so far. In the end, the best configuration seen is reported instead of the final configuration. This way, superior configurations are never lost and the likelihood of finding an optimal solution is increased.

5.2.2 Evolution Strategies

Like simulated annealing, evolution strategies [Rechenberg, 1973] are a probabilistic meta-heuristic. As evolution strategies also belong to the probabilistic class, they are also only able to find an optimal solution with a high probability. A major difference in comparison to simulated annealing is the use of a population instead of just a single configuration, which allows the algorithm to explore multiple areas of the search space in parallel. Consequently, a lower number of iteration is needed to find a good solution, but each individual iteration is computationally more demanding. The use of a population also makes the extension to multi-objective optimization comparatively straightforward. Another particular advantage of evolution strategies is its low number of tunable parameters. Instead, evolutionary strategies rely on self-optimization, by adapting important parameters as part of the optimization process. This allows the strategy to automatically adjust itself to different problem classes. The advantages of evolution strategies are partially offset by the significantly higher complexity that increases the implementation effort and the risk of errors.

The concept of evolution strategies is inspired by evolutionary process in nature. In comparison to other evolution-inspired strategies, such as genetic algorithms, evolution strategies employ a higher-level of abstraction and, for example, do not explicitly model genes.

A large number of variants of the basic algorithm exists, but they all follow a similar pattern. The process starts with a randomly generated population. Within each iteration, first, a predefined number of new configurations is created, each based on a cross-over of two configurations of the previous generation. Next, a mutation operator is applied to the newly created configurations. A typical implementation of the mutation operator modifies each of the individual values of the configuration by a normally-distributed random value. The parameters of this distribution are stored alongside the configuration values and are also adapted by the algorithm, which enables the algorithm to self-adapt to the problem at hand. Finally, a fixed-size sub-set of the configurations is semi-randomly selected to form the population of the next iteration. This selection is biased by the relative quality of the available solutions. Superior configurations are more likely to be selected, but still inferior solutions can be selected with a low probability. The occasional selection of inferior solutions enables the algorithm to escape local optima. The process is repeated until a predefined number of iterations has been executed or the quality of the best solution reaches a predefined threshold.

For our implementation we employed an optimized and extended variant of the basic strategy developed by Reehuis and Bäck [Reehuis and Bäck, 2010]. This variant supports configurations consisting of integer, floating point, nominal, and Boolean values.

5.3 Mathematical Model

The configuration framework employs mathematical optimization to generate an optimal parameter configuration for one or more WSN protocols based on a user-defined dependability specification.

The goal and constraints of a requirement specification define a typical constrained optimization problem. The formulation of this problem is based on a number of metrics $m_i(\mathbf{c})$ that allow the evaluation of the quality of a specific configuration $\mathbf{c}$ by assessing the performance of the configuration in regard to a specific aspect. Upon evaluation of a configuration, the employed protocol model provides a single score value for each supported metric. To incorporate relevant properties of the deployment environment and the employed hardware platform, the protocol model can in turn resort to environmental and platform models. We currently only support the single objective case where only a single metric is optimized, while the user may specify an arbitrary number of additional constraints. A possible extension to multi-objective problems has not been studied within the scope of this thesis. To increase the flexibility, we support probabilistic constraints that only need to hold with a given probability. For example, a user could specify that the packet reception rate should stay above 0.8 with a probability of 0.9 and above 0.5 with a probability of 1. Probabilistic constraints grant the users additional flexibility in defining their constraints, which is especially important for wireless systems, where a continuous fulfillment of strict constraints can rarely be guaranteed with absolute certainty. The optimization problem defined by a requirement specification document essentially matches the following form:

$$\begin{array}{ll} \text{Minimize} & m_0(\mathbf{c}) \\ \text{Subject to} & m_1(\mathbf{c}) \leq t_1 \text{with probability } p_1 \\ & m_2(\mathbf{c}) \leq t_2 \text{with probability } p_2 \,. \end{array} \tag{5.2}$$

The goal of the exemplary specification from Equation 5.2 is to find a protocol configuration $\mathbf{c}$ that optimizes a single metric m_0. In addition, this specification defines two constraints that need to be fulfilled by ensuring that $m_1(\mathbf{c})$ and $m_2(\mathbf{c})$ are below the respective thresholds t_1 and t_2 with a probability of at least p_1 or p_2. Note that m_0, m_1, and m_2 may all refer to the same metric.

In the remainder of the section, to simplify the presentation and without loss of generality, we assume that the metrics (m_0, m_1, m_2) and the thresholds (t_1, t_2) are normalized to the $[0, 1]$ range and that smaller values denote superior properties. In addition, only minimization and less-or-equal constraints are considered, as other goals and constraints can be easily converted into this form.

The employed optimization strategies do not readily support an optimization problem of the previously introduced form. To be usable as input for these optimization strategies, it needs to be transformed into a more suitable representation. Specifically, we need to overcome two challenges.

First, most optimization techniques do not directly support constraints but work only with a single optimization function. To be usable for these techniques, the constraints need to be integrated with the objective function. To accomplish this, we employ an approach similar to penalty functions often used in stochastic optimization [Smith and Coit, 1997]. This approach ensures that invalid configurations are unlikely to be selected as a final solution, but still allows the optimization algorithm to traverse infeasible regions of the search space and explore potentially superior regions. In the unconstrained case, the cost $f(\mathbf{c})$ of a specific configuration $\mathbf{c}$ is determined exclusively by the goal function $m_0(\mathbf{c})$. To integrate constraints, a measure of violation is computed for each constraint and is added to the total cost of the current configuration. To ensure that invalid solutions are unlikely to be selected, while still allowing the optimization algorithm to traverse infeasible regions, the influence of the constraint violation is given slightly more weight than the cost of the goal. If we assume a weight of k, the total cost for a given configuration $\mathbf{c}$ and r constraints can now be calculated as given by

$$f(\mathbf{c}) = \frac{f_{\text{goal}}(\mathbf{c}) + k\sum_{j=1}^{r} f_{\text{cons},i}(\mathbf{c})}{1+kr}\,. \tag{5.3}$$

The goal's cost depends only on the value determined by the respective metric, so the function can be defined according to

$$f_{\text{goal}}(\mathbf{c}) = m_0(\mathbf{c})\,. \tag{5.4}$$

The calculation of the degree of violation for each constraint is calculated according to the function

$$f_{\text{cons},i}(\mathbf{c}) = \max\{(t_i - m_i(\mathbf{c}))\,,0\}\,. \tag{5.5}$$

To allow the use of probabilistic constraints, we also need to convert these into a form usable with established optimization techniques. In contrast to regular constraints, the newly introduced probabilistic constraints only need to be fulfilled with the specified probability. To support this feature, we exploit the fact that most environment parameters exhibit some kind of periodic behavior. This enables us to sample the environment at different points in time and identify time frames with distinct environment conditions. Each time frame leads to a distinct environmental model instance. If we assume that communication events are more or less evenly distributed over time, we can associate a probability q_j with each interval based on its relative length. This probability indicates how likely it is that a specific communication event falls within a specific interval and is such affected by the properties of this interval. Instead of using only a single function $m_i(\mathbf{c})$ per metric, we now employ a set of functions $m_{i,j}(\mathbf{c})$. Each of these function corresponds to one of the n intervals and outputs a score value based on the environmental conditions within this interval. Consequently, each of these functions uses a different instance of the environmental model that represents the distinct interval. For the support of probabilistic constraints, we need to adapt the definition of the functions $f_{\text{goal}}(\mathbf{c})$ and $f_{\text{cons},i}(\mathbf{c})$ in Equation 5.3.

To take the use of multiple cost functions for each metric into account, the cost of the goal $f_{\text{goal}}(\mathbf{c})$ is redefined as the average of the cost for each individual interval, which leads to

$$f_{\text{goal}}(\mathbf{c}) = \sum_{j=1}^{n} q_j m_{0,j}(\mathbf{c})\,. \tag{5.6}$$

The updated calculation of the cost of the individual constraints takes the interval's

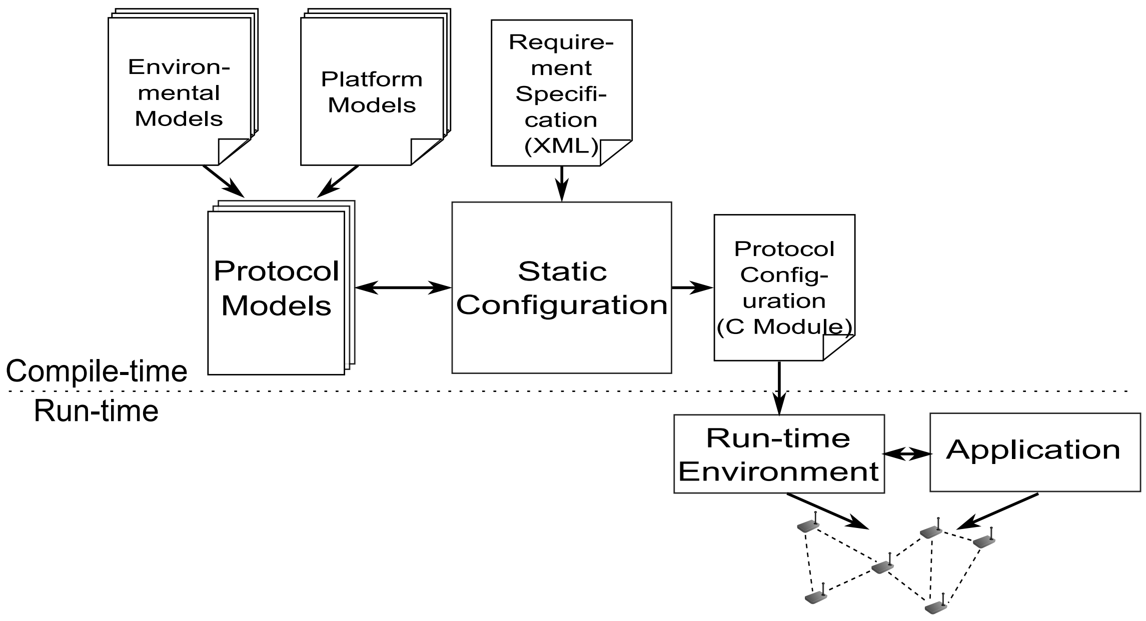

Figure 5.1: Architecture of the parameterization framework [Oppermann et al., 2015a].

probabilities into account by employing the definition of

$$f_{\text{cons},i}(\mathbf{c}) = \max\left\{\left(p_i - \sum_{j=1}^{n} \tau(m_{i,j}(\mathbf{c}), t_i, q_j)\right), 0\right\}, \tag{5.7}$$

where function $\tau(v, t, q)$ implements a filter to only consider constraints that are violated under the current environmental conditions by using the definition

$$\tau(v, t, q) = \begin{cases} q, & v > t \\ 0, & \text{otherwise}. \end{cases} \tag{5.8}$$

The resulting transformed definition of $f(c)$ can finally be used as the goal function of an unconstrained optimization problem that is readily supported by the employed optimization strategies.

5.4 Implementation

To evaluate the approach, we implemented it within a prototypical system. The framework consists of three major components: (1) the actual configuration tool, (2) a number of protocol model implementations, and (3) a run-time environment. As can be seen in Figure 5.1, the framework is organized around the central configuration tool, which implements the automatic parameter selection process. This

Listing 5.2: Exemplary XML-encoded requirement specification.

```
<?xml version="1.0" encoding="UTF-8"?>
<dependabilityReq xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:noNamespaceSchemaLocation="schema.xsd">
  <name>Sample Requirement TempMAC</name>
  <protocol_class>tempmac</protocol_class>
  <objective>CCA</objective>
  <criteria>MAX</criteria>
  <constraints>
    <constraint>
      <metric>PRR</metric>
      <threshold>0.85</threshold>
      <probability>1.0</probability>
    </constraint>
    <constraint>
      <metric>PRR</metric>
      <threshold>0.95</threshold>
      <probability>0.9</probability>
    </constraint>
  </constraints>
</dependabilityReq>
```

tool selects a suitable parameter set to ensure the dependable performance of WSN protocols.

The tool receives a specification of the dependability requirements encoded in extensible markup language (XML) as shown in Listing 5.2 as input. The specification consists of a single optimization goal (e.g., minimization of energy expenditure), and optionally an arbitrary number of constraints. It essentially defines an optimization problem such as introduced in Section 5.3. While most applications will only employ a single set of requirements, some applications have a number of diverse modes of operation that are activated based on system state and environmental conditions. These operation modes often have very different requirements in terms of network performance and reliability. For example, a system to detect forest fires would be typically optimized for a long system lifetime during normal operation. As soon as a forest fire is detected, lifetime is not a primary concern anymore, but instead the system should operate in a mode that ensures a very fast dissemination of detailed data about the current state of the fire. To also adequately handle such applications with several modes of operation, each with its own set of requirements, the configuration tool can be fed with multiple independent requirement specifications. Each of these specifications is assigned to one of the possible application states. At run-time, the application can switch between these modes and select different sets of active requirements to adapt to the current situation. If multiple requirement specifications are given, each one is handled separately during the optimization process and an individual configuration is generated for each one.

Based on these inputs the static configuration tool employs the mathematical optimization techniques introduced in Section 5.2 to determine a near-optimal configuration for the employed protocols. The configuration tool is implemented as a standalone Python [van Rossum and de Boer, 1991] application.

To predict the behavior of protocols with a specific configuration and in a specific environment, the configuration tool relies on protocol models introduced in Section 5.1 that evaluate the protocol performance. These models are highly protocol dependent and we expect that the model implementations will typically be provided by the respective protocol developers. To enable the use of externally defined and developed model implementations, the configuration tool provides a plug-in interface. The plug-in interface makes it comparatively easy to add new models for different protocols. Model implementations are expected to also employ the Python programming language, but interfacing to implementations in different programming languages is possible via Python language bindings.

The run-time environment provides the protocol implementations access to the generated configuration file at execution time. As of today, most implementations either employ hard-coded values that are set at compile time or employ their own configuration mechanism. Instead, our automatic configuration framework relies on a centralized configuration. Configuration parameters for all deployed protocol implementations are stored at a single location and access to the settings is provided by the dedicated run-time environment. The run-time environment is platform dependent and needs to be reimplemented for different WSN operating systems. The current prototype contains a run-time environment suitable for Contiki. The actual configuration settings are stored within a C module that is compiled separately and linked with the run-time environment and the actual application code. The module contains a hierarchic representation of the configuration as nested C structures. The use of a compiled configuration largely simplifies access by the run-time environment and leads to a very low memory overhead. The use of a centralized configuration also simplifies the implementation of different performance states. For applications that support multiple modes of operation, the run-time environment manages a separate configuration for each mode. Based on the current application state the application can select the most suitable one. To enable the transition between different performance states, the run-time environment provides a dedicated interface. After a performance state transition, all affected protocol implementations are automatically triggered to update their operation based on the new configuration settings.

5.5 Evaluation

To evaluate the feasibility of the approach, the framework is employed to determine an optimal configuration for an exemplary façade monitoring application based on a previously recorded temperature profile of a real deployment. To further evaluate the performance of the framework, different possible configurations are evaluated on a WSN testbed emulating the temperature profile of the real environment and compared with the actual performance of the configuration proposed by the configuration framework.

In the following, we first describe the application scenario and the employed protocol model. Subsequently, we introduce the setup for the two experiments and discuss their results.

5.5.1 Application Scenario

The evaluation of the automatic configuration framework employs a façade monitoring application as a case study. A more detailed description of the case study can be found in Paper F. The application was originally developed within the context of the RELYonIT project [RELYonIT Consortium, 2015] and deployed on the façade of one building of DEMOPARK, an experimentation façility for building isolation operated by ACCIONA in the vicinity of Madrid, Spain. Its buildings are used by ACCIONA to study the efficiency of different building isolation methods and materials. These studies require precise and reliable temperature measurements at different locations on the building façade. To enable a meaningful analysis of the measurements, only a very low loss of data packets can be tolerated, since gaps in the transmitted data might lead to false conclusions. Achieving reliable data transmission is further complicated by the environment at the deployment site. By being deployed on the outside of a building façade, the nodes are exposed to different climate effects, such as temperature changes and sun radiation. This leads to significant temperature variations within the node's enclosures over the course of a day. These temperature differences significantly affect the operation of the employed radio chips, which in turn affects the effectiveness of clear channel assessment (CCA) typically used by MAC layer protocols. This can compromise the ability of nodes to avoid packet collisions or prevent successful wake-up from low power modes [Boano, Römer, and Tsiftes, 2014]. In the Madrid deployment, ContikiMAC [Dunkels, 2011] is used as MAC protocol. ContikiMAC is a low-power MAC protocol that employs CSMA-based duty cycling with a power efficient wake-up mechanism to reduce the power consumption by keeping the transmitter off as much as possible. To detect ongoing transmissions and as a part of the duty cycling mechanism, ContikiMAC uses CCA with a fixed threshold. The CCA threshold is usually left at the default setting of the hardware platform and hence does not take temperature changes into account which may lead to a significantly degraded performance [Boano, Römer, and Tsiftes, 2014]. An inaccurate selection of the CCA threshold can, at high temperatures, prevent receiving nodes from correctly detecting transmissions and wake-up from low-power mode, which in turn leads to broken links and a reduced reliability of the network. A too low threshold setting may lead to energy wastage due to an increased number of false wake-ups caused by radio noise.

To enable the use of the automatic configuration framework to determine a more suitable threshold, a protocol model of ContikiMAC was developed within RELYonIT [Oppermann et al., 2015a]. This model captures how the performance of the MAC protocol, measured in terms of packet reception ratio (PRR), is affected by a change in environmental temperature. The protocol model in turn relies on two additional models: a platform model characterizing the signal strength attenuation of the radio and an environmental model characterizing the expected variations of on-board temperature during operation.

The environmental model is comparatively simple and just captures the maximum temperature. To enable the use of probabilistic constraints, the day is divided into six intervals of equal length and an individual model instance is created for each interval. For the experiments, the six models were instantiated by collecting temperature traces over a time frame of several days in October 2014.

The platform model reflects the characteristics of the Maxfor MTM-CM5000-MSP sensor nodes employed in the deployment. More specifically, it captures the properties of the TI CC2420 radio these nodes are equipped with. The model is based on earlier work by Boano et al. [2013] that characterizes the effect of temperature on the efficiency of the radio. The model defines three constants α, β, and γ that denote the respective effect on transmission power, received power, and the noise floor in dB/K based on the temperature difference compared to a fixed reference temperature [Boano et al., 2013]. These parameters need to be instantiated individually for each sensor node type. In this case study, we employed empirically determined settings for these parameters.

Combined, the three models allow the predict of the expected PRR in a specific climate and with a given setting of the CCA threshold ζ. For a more thorough description of the model developed by Boano and Zúñiga the reader is also referred to Paper F.

The probability of successful transmission of packets over a single link in the network can be estimated by analyzing the effective signal strength at the receiver. In the ContikiMAC protocol, nodes periodically wake up from low-power mode and check if the channel is busy by comparing the signal strength to the CCA threshold. If a possible transmission is detected, the node stays awake to receive the message. Otherwise, the node returns to low-power mode until the next wake-up interval. Transmissions are detected by comparing the current signal strength s_r with a fixed threshold ζ. If $s_\text{r} \geq \zeta$, a possible transmission is detected else if $s_\text{r} < \zeta$ it is assumed that no transmission is ongoing. The same mechanism is also used at the sender side to prevent interfering with ongoing transmissions. If a transmission is already in progress, the sender delays its own transmission until the channel is free.

The PRR is affected by the sensitivity threshold of the radio. If the perceived signal strength of the receiver is below this threshold, the message cannot be correctly decoded and the packet is lost. Actually, one can distinguish three regions, a connected region, where signal strength is above the threshold $s_{0.99}$ and nearly all packets are received; a disconnected region, where the signal strength below $s_{0.01}$ and almost no packets are received; and finally a transitional region, where the reception probability follows a sigmoid curve defined by

$$p = (1 - f(P_\mathrm{r} - P_\mathrm{r}))^b \tag{5.9}$$

with P_r being the received signal strength in dBm, P_n the sensitivity threshold of the radio in dBm, and b the number of bits in the packet.

Temperature changes influence the effective perceived signal strength at the receiver. If the temperature at the receiver T_r or the temperature at the sender T_s rises, the signal strength is attenuated. According to the model the change of the signal strength can be characterized by

$$s_r^{'} = (s_r - \alpha\Delta T_\mathrm{t} - \beta\Delta T_\mathrm{r})\,. \tag{5.10}$$

Also, similar the bounds between the different regions are shifted according to

$$s'_{0.99} = s_{0.99} - \Delta T_\mathrm{r}\gamma \tag{5.11}$$

and

$$s'_{0.01} = s_{0.01} - \Delta T_\mathrm{t}\gamma\,. \tag{5.12}$$

This allows one to estimate the expected worst-case package reception rate PRR' for a given link in the network and a known temperature range by

$$PRR' = \begin{cases} 1, & \text{if } \max\{\zeta, s'_{0.99}\} < s'_r \\ p, & \text{if } s'_{0.01} \leq \zeta < s'_r \leq s'_{0.99} \\ 0, & \text{otherwise}\,. \end{cases} \tag{5.13}$$

The ContikiMAC model has been realized as a configuration framework plug-in implementing the interface defined by the framework. The implementation exposes one configuration parameter, ζ, and provides a number of metrics that can be used to define goals or constraints. The primary metric provided by the protocol model is the expected worst-case PRR of a link averaged over all links in the network. Environmental and platform models are currently integrated within the implementation of the protocol model and have not been realized as separate modules.

5.5.2 Applicability

In this section, we demonstrate the applicability and usefulness of the automatic configuration framework, by employing it to generate an optimized configuration for the aforementioned case-study. This evaluation consists of two parts. In the first part we use the framework to configure only a single link between two nodes which eases the analysis and presentation of the results. In a second step, we demonstrate that the same principles can be also successfully transfered to an entire network.

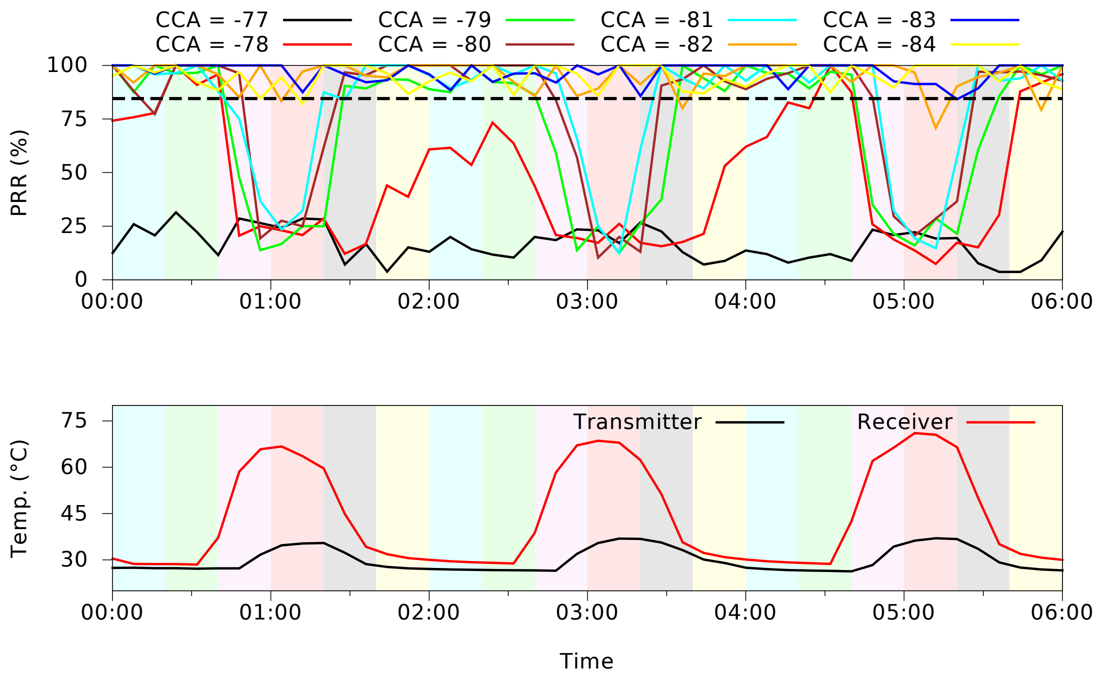

Figure 5.2: Impact of CCA threshold on PRR for a single link [Oppermann et al., 2015a]. Background colors indicate different time intervals. The 85% PRR constraint of the employed requirement specification is represented by a dashed line. The bottom plot shows the temperature profiles of the sender and receiver node.

The evaluation employs TempLab [Boano et al., 2014], a temperature-controlled testbed, to emulate the environment of a real-world outdoor façade within a controlled laboratory environment. The emulation of temperature effects is based on traces previously recorded at DEMOPARK. Consequently, the sensor nodes in the testbed experience the same on-board temperature profile as in the real-world scenario. To enable a large number of experiments within reasonable time, the experiments are run with a time-lapse factor of 12. Hence, a trace recorded over the course of one day is replayed within only two hours. As the temperature changes are still comparatively slow, the time-lapse is unlikely to affect the results of the experiments.

The parameters of the platform model are configured based on the results of earlier experiments by Boano et al. [2013], which leads to settings of $\alpha = \beta = 0.078\text{dB/K}$, and $\gamma = 0.037\text{dB/K}$. The width of the transitional region $\tau = 5\,\text{dB}$ and the packet size $b = 35$ bytes are fixed for all conducted experiments. Not to neglect the support of disparate time intervals, the day was uniformly divided into six intervals of four hours in length. All experiments were conducted with the optimization process employing the simulated annealing strategy.

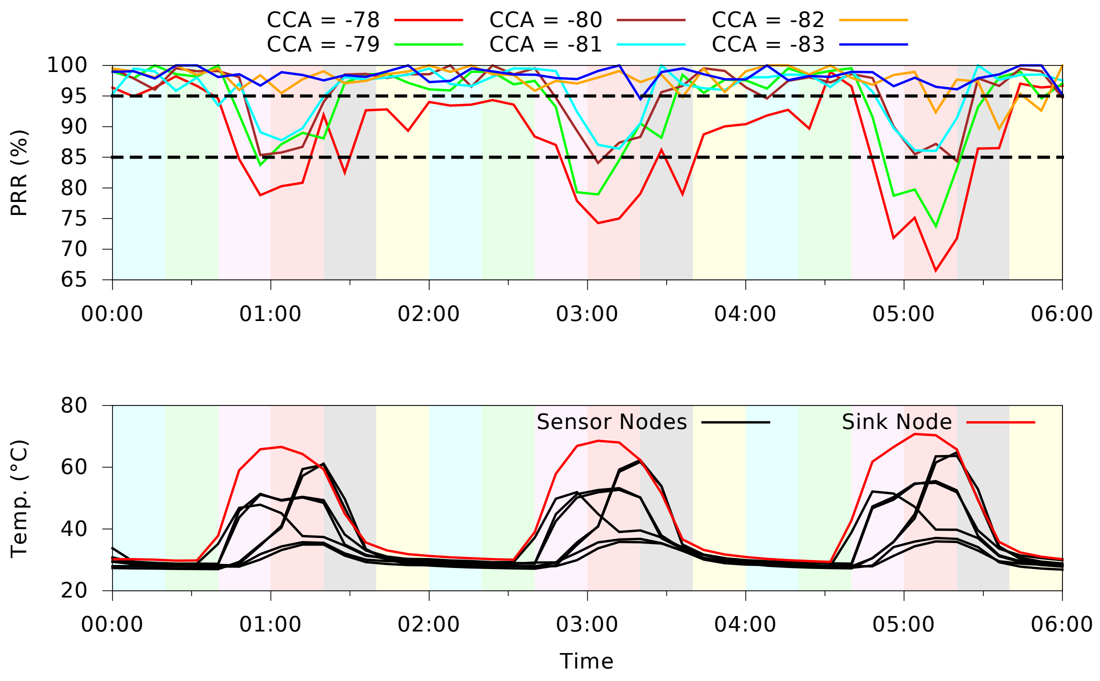

Figure 5.3: Impact of CCA threshold on PRR on a network of nodes [Oppermann et al., 2015a]. Background colors indicate different time intervals. The 85% and 95% PRR constraints of the employed requirement specification are represented by dashed lines. The bottom plot shows the different temperature profiles of the nodes in the network.

Single Link. The first set of experiments only considers the simpler case of configuring only a single link. The goal in this scenario is to ensure dependable communication despite the predicted environmental influences. More precisely, the goal is to find the maximal possible CCA threshold while still ensuring that a PRR of at least 0.85 is maintained at all times. Based on temperature traces recored at the DEMOPARK deployment, the configuration framework determines an optimal CCA threshold of −83 dBm for the specified requirements.

To verify the correctness of the result, we run a simple application that repeatedly sends a packet via the link within the TempLab testbed using a number of different CCA threshold settings. Figure 5.2 shows that with the suggested threshold of −83 dBm, the PRR actually stays above 0.85 for the whole duration of the experiment. For any higher CCA thresholds the PRR requirement is violated in at least one of the intervals. For example, for a CCA threshold of −82 dBm, the PRR drops to just 0.75 at time 5:10. For higher CCA thresholds the effect is even more severe with PRR values of less than 0.25 during intervals with high temperature. As a site note, at −77 dBm, the default value of the CC2420 radio [Texas Instruments, 2013], the link is almost disconnected throughout the entire experiment, only reaching a maximal PRR of slightly more than 0.25.

In a second experiment, to explore the effect of probabilistic constraints, we employ the same PRR constraint of 0.85 but only require it to be met with a probability of 0.6. For this scenario, the framework determines an optimal CCA threshold of −81 dBm. As is visible in Fig. 5.2, with a threshold of −81 dBm the PRR actually stays above 0.85 in at least four out of the six intervals per day, which fulfills our requirement of sustaining a PRR of at least 0.85 in at least 60 % of the cases. We can also see that a CCA threshold of −80 dBm would violate the constraint by additionally dropping below 0.85 in the first interval. Hence, the suggested threshold of −81 dBm can actually be assumed to be the most energy conserving configuration that is able to meet the specified constraint. If the application can tolerate periods with low packet reception, the use of a lower CCA threshold allows energy conservation in comparison to the settings required for the stricter reliability constraints as a lower CCA threshold reduces the number of false wake-ups due to radio noise.

Network. To facilitate analysis and presentation, the previous experiments only considered a single link. In realistic scenarios, the framework will typically be used to configure protocols based on the behavior of an entire network. Typically, it is not feasible to configure different links individually. Consequently, we will now evaluate the framework in a more realistic scenario that replicates the topology of the deployment that was used to monitor the façade of a DEMOPARK building. The network consists of seven sensor nodes that are connected to a single gateway node in a star topology. During the experiment, all nodes are exposed to individual temperature profiles recorded on different locations on the building façade. The network is configured based on a requirement specification that leads to the following abstract representation as an optimization problem:

$$\begin{array}{ll} \text{Maximize} & \mathrm{CCA}([\zeta]) \\ \text{Subject to} & \mathrm{PRR}([\zeta]) \geq 0.85 \text{ with probability } 1.00 \\ & \mathrm{PRR}([\zeta]) \geq 0.95 \text{ with probability } 0.9\,. \end{array} \tag{5.14}$$

For this scenario, the framework suggests an optimal CCA threshold of −83 dBm. As in the single link scenarios, we compare this suggestion with other possible CCA threshold settings. Based on the results reported in Figure 5.3, it can be seen that for the suggested CCA threshold, the average PRR permanently stays above 0.85, even if the nodes are exposed to heat. For a CCA threshold of −80 dBm or higher, this constraint is clearly violated, as the PRR drops below 0.85 within several intervals. While the first constraint is already met with a threshold of −82 dBm, the PRR drops below 0.95 for at least three out of the 18 intervals, which indicates that the second constraint cannot be met by this configuration. The suggested CCA threshold of −83 dBm actually provides the best performance for the given scenario. These results demonstrate that our framework is actually capable of generating

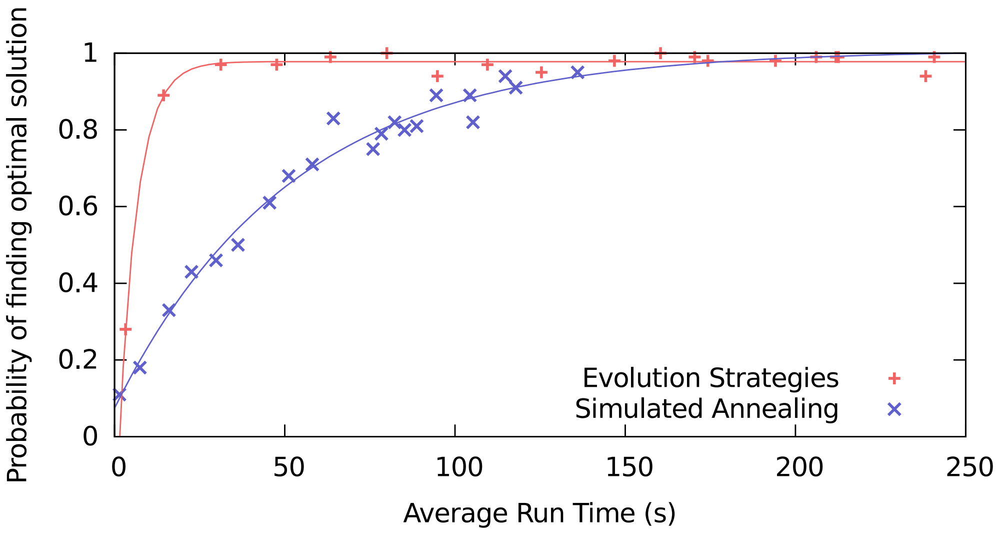

Figure 5.4: Probability of finding the optimal solution within a given time [Oppermann et al., 2015a].

useful configurations for realistic deployment scenarios and is also able to handle more complex requirements of typical users.

5.5.3 Performance

To be truly useful, the framework does not only need to reliably generate correct results, but these results need to be available within a reasonable time. Consequently, we also need to evaluate the basic performance of the system by analyzing the trade-off between speed and precision. In addition, as the system supports different optimization strategies, we briefly compare the relative performance of the different optimization algorithms. Both evaluations employ the same model and similar settings as in the previous section. To establish a ground truth value for the evaluation of the different optimization strategies, we employ an exhaustive search to identify the true optimum. With the previously introduced settings, a globally optimal solution can be found with a CCA threshold of -84 dBm and a cost value $f = 0.14344$.

To evaluate the time/quality trade-off, we execute the configuration framework with both implemented optimization strategies and with different time budgets. Limiting the time budget is indirectly accomplished by artificially limiting the maximal number of iterations the algorithm may spend before the result is collected. To reduce the effects of uncontrolled random events, each algorithm was executed 100 times for each examined time budget. All experiments were conducted on a Lenovo ThinkPad T430s notebook equipped with an Intel Core i7-3520M dual core CPU

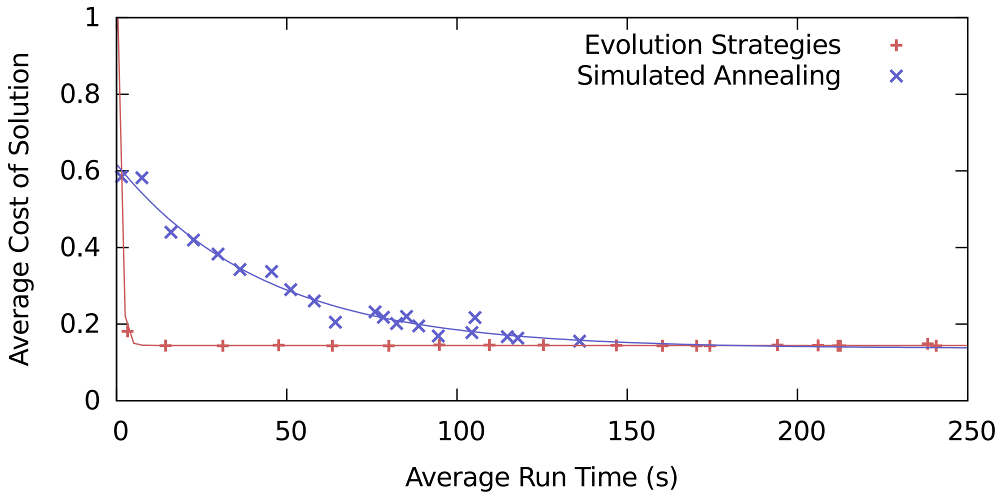

Figure 5.5: Time/quality trade-off for different optimization strategies [Oppermann et al., 2015a].

running at 2.90 GHz and 8 GB of RAM.

The resulting time/quality trade-off is plotted in Figure 5.4. To better illustrate the observable trend, the exponential function $x \mapsto a * e^{b*x} + c$ has been fitted to the raw data. As it can be easily seen, longer run-times naturally lead to superior results. While for shorter run-times the true optimum is rarely found, with a run-time of more than 120 s both algorithms are reliably able to find the optimal solution. With evolutionary strategies the optimum is typically already identified after a run-time of only 20 s.

In addition, the non-optimal solutions returned after a shorter run-time, do not necessarily represent bad solutions in practice. Most returned solutions, even after a run-time of only 150 seconds, are already close to the true optimum, as can be observed in Figure 5.5. Their cost value is only slightly above that of the optimal solution. For many applications, such near-optimal solutions are already good enough and the actual difference in performance is negligible in reality.

The results demonstrate that the performance of the automatic configuration framework is good enough to be useful within a typical WSN development process. The reliability of the results is high, despite the use of stochastic optimization strategies and the involved randomness. For the specific scenario, evolutionary strategies exhibit a slightly better performance, but this observation does not necessarily transfer to other scenarios. Due to a small number of nodes and tunable parameters, an exhaustive search is still a feasible approach in this scenario, but its performance degrades quickly if the number of nodes or configuration parameters is increased.

6 Conclusions

During the last few years, the use of WSN technology has largely increased. As pointed out by a systematic survey [Oppermann, Boano, and Römer, 2013], a large number of applications with diverse properties exists. Newer applications are often safety-critical and require a high level of performance and dependability. Still, most applications are developed in a scientific context and with the help of highly-skilled WSN experts. To enable the success of WSNs in the real world, the use of the technology needs to be more approachable for domain experts without a strong background in WSN technology. With this thesis, we propose an integrated approach to overcome these two challenges and to pave the way for a more widespread adoption of WSNs.

In the following, we first summarize the contributions of this thesis and secondly discuss remaining limitations of the proposed solutions. We also take an outlook on potential future research to overcome these limitations and to increase the applicability of the proposed concepts.

6.1 Scientific Contributions

We set out to reduce the barriers to successful employment of WSNs in real-world applications. In this regard, we identified two major obstacles that need to be overcome to enable a more widespread adoption of the technology:

1. the lack of a WSN-specific macroprogramming system that enables the seamless integration of existing and future application-specific WSN programming abstractions, and

2. the difficulty of configuring WSN protocol stacks which often leads to inadequate performance and dependability, especially in hostile environments.

To rectify the current situation and enable a more widespread application of WSNs, we propose an integrated WSN development framework. The contribution of this framework is twofold. First, we proposed a novel extensible macroprogramming language targeted at resource-constrained devices that allows the integration of existing programming abstractions and that significantly reduces the effort of WSN programming. Second, to also reduce the required expertise and effort in correctly configuring the underlying protocol stack, we propose an automatic configuration framework. This framework derives optimal protocol configuration values for a specific application that enable a predictable performance and dependability. The applied optimization process relies on models of the environment, the hardware platform, and the employed protocols.

The macroprogramming language demonstrates the applicability of employing a high-level object-oriented macroprogramming approach for the implementation of WSN applications. The language's syntax and semantics are partially modeled after the well-known Java language. An especially noteworthy feature of the language is its extensibility. The language enables the seamless integration of existing and future WSN programming abstractions, by employing a meta-abstraction concept developed within the `make`*Sense* project. The language has been implemented by a prototypical compiler and we assess the feasibility of the approach by evaluating the performance in a number of typical WSN application scenarios. The evaluation especially focuses on the use of a WSN control system in an energy-efficient distributed HVAC system. This use-case scenario introduces a number of additional challenges by requiring in-network control processes and the close integration with external management systems. With the implementation and evaluation, we could quantify the overhead introduced by the proposed approach and could assert its appropriateness for low-power, resources-constrained devices. The evaluation showed that the overhead is manageable in typical WSN applications. Even the use of Java syntax and semantics and the explicit support for object-oriented programming do not generate excessive overhead if carefully implemented with the resource-constrained nature of WSN nodes in mind. In addition, the evaluation revealed the usefulness of the complete macroprogramming framework and its capability to reduce the required coding effort. Finally, the successful implementation validates the applicability of the `make`*Sense* meta-abstractions as a conceptual framework for the integration of programming abstractions. Due to its flexibility, the compilation framework itself can easily serve as a platform for future research on macroprogramming and code allocation concepts.

The second pillar of our approach is the concept for an automatic configuration of WSNs based on predicted environmental conditions and the application's requirements. As part of this thesis this concept has been implemented as a prototypical configuration framework employing environmental, platform, and protocol models developed within the RELYonIT project. The resulting framework has been successfully evaluated within a building façade monitoring case study. In this case study, a WSN is used to monitor the heat distribution on a building façade in order to enable the study of the building's heat dissipation. The implementation and evaluation of the approach demonstrates the feasibility of automatic model-based WSN protocol configuration. The automatic system can identify configuration settings satisfying the user's performance and dependability requirements even in hostile environments. Yet, the process requires less WSN-specific expert knowledge and is significantly less time consuming than manual tuning of protocol parameters. The configuration process is further assisted by a newly developed requirement catalog that supports the requirement analysis and enables less technology-savvy users to provide suitable input for an automatic parameterization process as proposed above.

Combined, the programming and configuration frameworks significantly reduce the effort and required knowledge for successfully designing and deploying a WSN/IoT application and represent a significant step towards a broader adoption of the technology in real-world scenarios.

6.2 Remaining Issues and Future Work

Despite the positive results of the evaluation, current limitations open up an avenue for future work. A major limitation of the current implementation is its lack of integration. Both frameworks have been implemented as prototypical systems, but those systems are not yet integrated. Workable concepts for such an integration have been proposed in Chapter 2, but are not currently realized. In the future, we intend to implement such an integrated system, in order to truly deliver the vision of an integrated WSN development framework that enables less technology-savvy users to successfully deploy WSNs.

Both individual concepts and implementations also have limitations of their own. While the meta-abstraction framework provides well-defined interfaces for the integration of programming abstractions within the macroprogramming language, it does not define any interfaces for the direct interaction between different programming abstractions. For example, currently, no standardized interface exists to enable a Tell Action to access the properties of an associated Target Modifier. To resolve this issue, the framework needs to be extended with a more precise definition of the building blocks of programming abstraction and well-defined interfaces between these building blocks. The current implementation of code allocation is also limited in some regards. Currently, it operates entirely at class level, but a more fine-grained approach that extends the allocation decisions to individual methods and attributes would provide increased flexibility and more significant savings in terms of memory. In addition, it would be useful to make the allocation algorithm work with a larger number of node classes and to explicitly take data-flow of the application into account. A more sophisticated data-flow analysis would allow the compiler to improve the allocation of individual program elements to nodes within the network. Another drawback identified during the evaluation is the lack of an integrated debugging framework. Identifying and analyzing errors in the generated C code proved difficult and reintroduced the need for an understanding of the underlying systems. To make the system more useful in practice, we also intend to improve debugging support. The user experience could be further increased by providing an adapted editor or an integrated development environment (IDE) for the microprogramming language. The overall performance of the system can be further improved to make it even more suitable for resource-constrained devices typically found in WSNs. Memory consumption of the generated code, in particular, still leaves significant room for improvement. It could be, for example, improved by a more sophisticated strategy for code allocation. Finally, to be truly useful, the number of available programming abstractions also needs to be increased. Currently, only a limited number of programming abstractions is integrated with the macroprogramming framework.

The configuration framework is currently primarily restricted by its limitation to single protocol configuration. To be more useful, the framework should support the configuration of an entire protocol stack. In addition, for many scenarios, alternative protocols and protocol implementations are available, so that the system would significantly benefit from enabling the automatic selection of suitable protocols. The latter goal could be reached by integrating existing approaches for protocol selection such as ConfigKit [Peter, Piotrowski, and Langendörfer, 2008]. Both extensions would likely also necessitate a further improvement of the actual optimization process, especially if more complex models are to be used. In the future, we hope to see the implementation of additional protocol models and the use of the framework in additional case studies. This will allow further assessment of the performance of the system and to identify means to increase the flexibility of the framework.

Finally, both individual frameworks as well as an integrated system could benefit from further evaluation in a large-scale deployment. The evaluation of the usability of the programming framework, in particular, could be supplemented by a comprehensive user study that more precisely quantifies the actual reduction in development effort. Initial work in this direction has been conducted by one of the project partners from `make`*Sense*, but still awaits publication.

We hope that the developed concepts and software systems can serve as a viable basis for future research and will one day help to enable the implementation and configuration of typical WSN and IoT applications without requiring extensive technical expertise in the area.

7 Publications

This Thesis is based on the following peer-reviewed workshop and conference papers co-authored by the author of this thesis:

A. Felix Jonathan Oppermann (2009). "Towards an End-User-Requirements-Driven Design and Deployment of Wireless Sensor Networks." In: *Adjunct Proceedings of the Work in Progress Session 17th EUROMICRO International Conference on Parallel, Distributed, and Network-based Processing (PDP).* (Weimar, Germany). Ed. by Erwin Grosspietsch and Konrad Klöckner. SEA-Publications of the Institute for Systems Engineering and Automation SEA-SR-21. J. Kepler University Linz. ISBN: 978-3-902457-21-9

B. Felix Jonathan Oppermann and Steffen Peter (2010). "Inferring Technical Constraints of a Wireless Sensor Network Application from End-User Requirements." In: *Proceedings of the 6th International Conference on Mobile Ad-hoc and Sensor Networks (MSN).* (Hangzhou, China). Piscataway, NJ, USA: IEEE, pp. 169–175. ISBN: 978-1-4244-9456-9. DOI: 10.1109/MSN.2010.32

C. Fabio Casati, Florian Daniel, Guenadi Dantchev, Joakim Eriksson, Niclas Finne, Stamatis Karnouskos, Patricio Moreno Montero, Luca Mottola, Felix Jonathan Oppermann, Gian Pietro Picco, Antonio Quartulli, Kay Römer, Patrik Spiess, Stefano Tranquillini, and Thiemo Voigt (2012a). "Towards Business Processes Orchestrating the Physical Enterprise with Wireless Sensor Networks." In: *Proceedings of the 34th International Conference on Software Engineering (ICSE).* (Zurich, Switzerland). Piscataway, NJ, USA: IEEE, pp. 1357–1360. ISBN: 978-1-4673-1066-6. DOI: 10.1109/ICSE.2012.6227080

D. Florian Daniel, Joakim Eriksson, Niclas Finne, Harald Fuchs, Andrea Gaglione, Stamatis Karnouskos, Patricio Moreno Montero, Luca Mottola, Felix Jonathan Oppermann, Gian Pietro Picco, Kay Römer, Patrik Spieß, Stefano Tranquillini, and Thiemo Voigt (2013). "makeSense: Real-world Business Processes through Wireless Sensor Networks." In: *Proceedings of 4th International Workshop on Networks of Cooperating Objects for Smart Cities.* (Philadelphia, PA, USA), pp. 58–72

E. Felix Jonathan Oppermann, Kay Römer, Luca Mottola, Gian Pietro Picco, and Andrea Gaglione (2014b). "Design and Compilation of an Object-Oriented

Macroprogramming Language for Wireless Sensor Networks." In: *Workshop Proceedings of the 39th IEEE Conference on Local Computer Networks (LCN).* (Edmonton, AB, Canada). Piscataway, NJ, USA: IEEE, pp. 574–582. ISBN: 978-1-4799-3782-0. DOI: `10.1109/LCNW.2014.6927705`

F. Felix Jonathan Oppermann, Carlo Alberto Boano, Marco Antonio Zúñiga, and Kay Römer (2015a). "Automatic Protocol Configuration for Dependable Internet of Things Applications." In: *Workshop Proceedings of the 40th IEEE Conference on Local Computer Networks (LCN).* (Clearwater, FL, USA). Piscataway, NJ, USA: IEEE. ISBN: 978-1-4673-6772-1

Related book chapters, journal articles, and peer-reviewed papers, poster, and demos co-authored by the author of this thesis and presented at international conferences not included in this thesis:

1. Kinga Kiss Iakab, Felix Jonathan Oppermann, Oliver Theel, and Jens Kamenik (2010). "Exploiting Semantic Quorum-Based Data Replication in Wireless Sensor Networks." In: *Proceedings of the 9. GI/ITG KuVS Fachgespräch Sensornetze.* (Würzburg, Germany). Julius-Maximilians-Universität Würzburg, pp. 55–58

2. Fabio Casati, Florian Daniel, Guenadi Dantchev, Joakim Eriksson, Niclas Finne, Stamatis Karnouskos, Patricio Moreno Montero, Luca Mottola, Felix Jonathan Oppermann, Gian Pietro Picco, Antonio Quartulli, Kay Römer, Patrik Spiess, Stefano Tranquillini, and Thiemo Voigt (2012b). "Towards Business Processes Orchestrating the Physical Enterprise with Wireless Sensor Networks." In: *Adjunct Proceedings of the 9th European Conference on Wireless Sensor Networks (EWSN).* (Trento, Italy)

3. Stefano Tranquillini, Patrik Spieß, Florian Daniel, Stamatis Karnouskos, Fabio Casati, Nina Oertel, Luca Mottola, Felix Jonathan Oppermann, Gian Pietro Picco, Kay Römer, and Thiemo Voigt (2012). "Process-Based Design and Integration of Wireless Sensor Network Applications." In: *Proceedings of the 10th International Conference on Business Process Management (BPM).* (Tallinn, Estonia). Berlin, Germany: Springer, pp. 134–149. ISBN: 978-3-642-32884-8. DOI: `10.1007/978-3-642-32885-5_10`

4. Felix Jonathan Oppermann, Carlo Alberto Boano, and Kay Römer (2013). "A Decade of Wireless Sensing Applications: Survey and Taxonomy." In: *The Art of Wireless Sensor Networks.* Ed. by Habib M. Ammari. Vol. 1. Berlin, Germany: Springer, pp. 11–50. ISBN: 978-3-642-40008-7. DOI: `10.1007/978-3-642-40009-4_2`

5. Carlo Alberto Boano, Felix Jonathan Oppermann, and Kay Römer (2013). "The Use of Body Sensor Networks in Clinical Settings and Medical Research." In: *Sensor Networks for Sustainable Development*. Ed. by Mohammad Ilyas, Sami S. Al-Wakeel, Mohammed M. Alwakeel, and El-Hadi M. Aggoune. Boca Raton, Florida: CRC Press, pp. 215–252. ISBN: 978-1466582064

6. Carlo Alberto Boano, Hjalmar Wennerström, Marco Antonio Zúñiga, James Brown, Chamath Keppitiyagama, Felix Jonathan Oppermann, Utz Roedig, Lars-Åke Nordén, Thiemo Voigt, and Kay Römer (2013b). "Hot Packets: A Systematic Evaluation of the Effect of Temperature on Low Power Wireless Transceivers." In: *Proceedings of the 5th Extreme Conference on Communication (ExtremeCom)*. (Thórsmörk, Iceland). New York, NY, USA: ACM, pp. 7–12. ISBN: 978-1-4503-2171-6

7. Felix Jonathan Oppermann, Carlo Alberto Boano, Marco Zimmerling, and Kay Römer (2014a). "Automatic Configuration of Controlled Interference Experiments in Sensornet Testbeds." In: *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems (SenSys)*. (Memphis, Tennessee, USA). New York, NY, USA: ACM, pp. 342–343. ISBN: 978-1-4503-3143-2. DOI: `10.1145/2668332.2668355`

Related technical reports and project deliverables co-authored by the author of this thesis:

1. Adam Dunkels, Joakim Eriksson, Luca Mottola, Thiemo Voigt, Felix Jonathan Oppermann, Kay Römer, Fabio Casati, Florian Daniel, Gian Pietro Picco, Stefano Soi, Stefano Tranquillini, Paolo Valleri, Stamatis Karnouskos, Patrik Spieß, and Patricio Moreno Montero (2010a). *D-1.1 – Application and Programming Survey*. Tech. rep. `make`*Sense*: Easy Programming of Integrated Wireless Sensor Networks

2. Adam Dunkels, Luca Mottola, Thiemo Voigt, Stamatis Karnouskos, Nina Oertel, Patrik Spiess, Gian Pietro Picco, Felix Jonathan Oppermann, and Kay Römer (2010b). *D-7.1 –* `make`*Sense Goal Definitions*. Tech. rep. `make`*Sense*: Easy Programming of Integrated Wireless Sensor Networks

3. Luca Mottola, Thiemo Voigt, Felix Jonathan Oppermann, Kay Römer, Fabio Casati, Gian Pietro Picco, Stefano Tranquillini, Paolo Valleri, Stamatis Karnouskos, Nina Oertel, Patrik Spieß, and Patricio Moreno Montero (2011a). *D-1.2 – Requirement Specification*. Tech. rep. `make`*Sense*: Easy Programming of Integrated Wireless Sensor Networks

4. Luca Mottola, Joakim Eriksson, Niclas Finne, Thiemo Voigt, Felix Jonathan Oppermann, Kay Römer, Fabio Casati, Florian Daniel, Gian Pietro Picco, Stefano Tranquillini, Paolo Valleri, Stamatis Karnouskos, Nina Oertel, Patrik Spieß, and Patricio Moreno Montero (2011b). *D-1.3 – Initial System Architecture.* Tech. rep. make*Sense*: Easy Programming of Integrated Wireless Sensor Networks

5. Luca Mottola, Gian Pietro Picco, Paolo Valleri, Felix Jonathan Oppermann, and Kay Römer (2011c). *D-3.1 – The* make*Sense Programming Model.* Tech. rep. make*Sense*: Easy Programming of Integrated Wireless Sensor Networks

6. Luca Mottola, Paolo Pettinato, Gian Pietro Picco, Antonio Quartulli, Felix Jonathan Oppermann, and Kay Römer (2011d). *D-3.2 – Language Core Implementation.* Tech. rep. make*Sense*: Easy Programming of Integrated Wireless Sensor Networks

7. Joakim Eriksson, Nicla Finne, Luca Mottola, Thiemo Voigt, Paolo Pettinato, Fabio Casati, Florian Daniel, Andrea Gaglione, Davide Molteni, Gian Pietro Picco, Antonio Quartulli, Stefano Tranquillini, Felix Jonathan Oppermann, Kay Römer, Guenadi Dantchev, Stamatis Karnouskos, Patrik Spieß, and Patricio Moreno Montero (2012a). *D-5.1 – Initial Integrated System.* Tech. rep. make*Sense*: Easy Programming of Integrated Wireless Sensor Networks

8. Joakim Eriksson, Niclas Finne, Luca Mottola, Thiemo Voigt, Fabio Casati, Florian Daniel, Andrea Gaglione, Davide Molteni, Gian Pietro Picco, Antonio Quartulli, Stefano Tranquillini, Felix Jonathan Oppermann, Kay Römer, Harald Fuchs, Nina Oertel, Patrik Spiess, and Patricio Moreno Montero (2012b). *D-5.2 – Report from Application Implementations and Evaluation.* Tech. rep. make*Sense*: Easy Programming of Integrated Wireless Sensor Networks

9. Martin Bor, Koen Langendoen, Carlo Alberto Boano, Felix Jonathan Oppermann, Kay Römer, Alejandro Veiga Rico, Patricio Moreno Montero, Rafael Socorro Hernández, Ignasi Vilajosana, Mischa Dohler, Màrius Montón, Utz Roedig, Andreas Mauthe, Geoff Coulson, Thiemo Voigt, Luca Mottola, Zhitao He, and Nicolas Tsiftes (2013). *D-4.1 – Report on Use Case Definition and Requirements.* Tech. rep. RELYonIT: Research by Experimentation for Dependability on the Internet of Things

10. Joakim Eriksson, Niclas Finne, Luca Mottola, Thiemo Voigt, Fabio Casati, Florian Daniel, Andrea Gaglione, Davide Molteni, Gian Pietro Picco, Antonio Quartulli, Stefano Tranquillini, Felix Jonathan Oppermann, Kay Römer, Harald Fuchs, Nina Oertel, Patrik Spieß, and Patricio Moreno Montero (2013a). *D-1.4 – Refined System Architecture.* Tech. rep. make*Sense*: Easy Programming of Integrated Wireless Sensor Networks

11. Andrea Gaglione, Davide Molteni, Gian Pietro Picco, Felix Jonathan Oppermann, and Kay Römer (2013). *D-3.4/D-3.5 – Final* `make`*Sense Programming Model and Final Language Core Implementation.* Tech. rep. `make`*Sense*: Easy Programming of Integrated Wireless Sensor Networks

12. Stamatis Karnouskos, Patrik Spieß, Patricio Moreno Montero, Luca Mottola, Thiemo Voigt, Gian Pietro Picco, Felix Jonathan Oppermann, and Kay Römer (2013). *D-6.5 – Final Workshop.* Tech. rep. `make`*Sense*: Easy Programming of Integrated Wireless Sensor Networks

13. Joakim Eriksson, Niclas Finne, Luca Mottola, Thiemo Voigt, Fabio Casati, Florian Daniel, Andrea Gaglione, Davide Molteni, Gian Pietro Picco, Stefano Tranquillini, Felix Jonathan Oppermann, Kay Römer, Harald Fuchs, Nina Oertel, Patrik Spiess, and Patricio Moreno Montero (2013c). *D-5.3 – System Integration.* Tech. rep. `make`*Sense*: Easy Programming of Integrated Wireless Sensor Networks

14. Joakim Eriksson, Niclas Finne, Luca Mottola, Thiemo Voigt, Fabio Casati, Florian Daniel, Andrea Gaglione, Davide Molteni, Gian Pietro Picco, Stefano Tranquillini, Bengt-Ove Holländer, Felix Jonathan Oppermann, Kay Römer, Sebastian Doeweling, Nina Oertel, Florian Probst, Patrik Spieß, and Patricio Moreno Montero (2013b). *D-3.6 & D-5.4 – Final application implementations and evaluation of system & Final evaluation of the programming model.* Tech. rep. `make`*Sense*: Easy Programming of Integrated Wireless Sensor Networks

15. Marco Antonio Zúñiga, Carlo Alberto Boano, James Brown, Chamath Keppitiyagama, Felix Jonathan Oppermann, Paul Alcock, Nicolas Tsiftes, Utz Roedig, Kay Römer, Thiemo Voigt, and Koen Langendoen (2013). *D-1.1 – Report on Environmental and Platform Models.* Tech. rep. RELYonIT: Research by Experimentation for Dependability on the Internet of Things

16. Luca Mottola, Thiemo Voigt, Felix Jonathan Oppermann, Kay Römer, and Koen Langendoen (2013). *D-3.1 – Report on Specification Language.* Tech. rep. RELYonIT: Research by Experimentation for Dependability on the Internet of Things

17. Carlo Alberto Boano, Felix Jonathan Oppermann, Kay Römer, James Brown, Utz Roedig, Chamath Keppitiyagama, and Thiemo Voigt (2013a). *D-4.2 – Prototype of Testbeds with Realistic Environment Effects.* Tech. rep. RELYonIT: Research by Experimentation for Dependability on the Internet of Things

18. Nicolas Tsiftes, Thiemo Voigt, Faisal Aslam, Ioannis Protonotoarios, Marco Antonio Zúñiga, Koen Langendoen, Carlo Alberto Boano, Felix Jonathan Op-

permann, Kay Römer, Marcel Baunach, Utz Roedig, Patricio Moreno Montero, Rafael Socorro Hernández Màrius Montón, and José Carlo Pacho (2014). *D-4.3 – First Integrated Prototype and Experiment.* Tech. rep. RELYonIT: Research by Experimentation for Dependability on the Internet of Things

19. Marco Antonio Zúñiga, Ioannis Protonotoarios, Si Li, Koen Langendoen, Carlo Alberto Boano, Felix Jonathan Oppermann, Kay Römer, James Brown, Utz Roedig, Luca Mottola, and Thiemo Voigt (2014). *D-2.2 & D-2.3 – Report on Protocol Models & Validation and Verification.* Tech. rep. RELYonIT: Research by Experimentation for Dependability on the Internet of Things

20. James Brown, John Vidler, Ibrahim Ethem Bagci, Utz Roedig, Carlo Alberto Boano, Felix Jonathan Oppermann, Marcel Baunach, Kay Römer, Marco Antonio Zúñiga, Faisal Aslam, and Koen Langendoen (2014). *D-1.3 – Report on Runtime Assurance.* Tech. rep. RELYonIT: Research by Experimentation for Dependability on the Internet of Things

21. Felix Jonathan Oppermann, Carlo Alberto Boano, Marcel Baunach, Kay Römer, Faisal Aslam, Marco Zúñiga, Ioannis Protonotarios, Koen Langendoen, Niclas Finne, Nicolas Tsiftes, and Thiemo Voigt (2015b). *D-3.2 – Report on Protocol Selection, Parameterization, and Runtime Adaptation.* Tech. rep. RELYonIT: Research by Experimentation for Dependability on the Internet of Things

22. Nicolas Tsiftes, Niclas Finne, Zhitao He, Thiemo Voigt, Faisal Aslam, Ioannis Protonotoarios, Marco Antonio Zúñiga, Koen Langendoen, Carlo Alberto Boano, Felix Jonathan Oppermann, Kay Römer, Marcel Carsten Baunach, James Brown, Utz Roedig, Ibrahim Ethem Bagci, John Vidler, Alejandro Veiga, Rafael Socorro Hernández, Màrius Montón, and José Carlos Pacho (2015). *D-4.4 – Final Integrated Prototype and Experiment.* Tech. rep. RELYonIT: Research by Experimentation for Dependability on the Internet of Things

7.1 Paper A: Towards an End-User-Requirement-Driven Design and Deployment of Wireless Sensor Networks

Felix Jonathan Oppermann (2009). “Towards an End-User-Requirements-Driven Design and Deployment of Wireless Sensor Networks.” In: *Adjunct Proceedings of the Work in Progress Session 17th EUROMICRO International Conference on Parallel, Distributed, and Network-based Processing (PDP).* (Weimar, Germany). Ed. by Erwin Grosspietsch and Konrad Klöckner. SEA-Publications of the Institute for Systems Engineering and Automation SEA-SR-21. J. Kepler University Linz. ISBN: 978-3-902457-21-9.

7.1.1 Summary

This paper describes a way of making WSN technology more approachable and increasing its uptake outside of the scientific community.

Wireless sensor networks receive high research attention but despite this large interest, the adoption of WSNs outside of the scientific community is still very limited. To allow a widespread use of WSNs, it is necessary to enable technically less skilled personnel to successfully deploy a WSN to solve their specific problems. While these end users can be assumed to be experts in their application domain, like in biology or in geology, they cannot be expected to be especially trained in computer science. One must ensure that the design and deployment process requires as little technical knowledge as possible and can be automated as much as possible. The proposed approach aims to reduce the burden of creating a successful WSN application and thus make the technology more accessible for non-experts. We believe that an implementation of this approach will pave the way for this technology finally being a success “in the field.”

7.1.2 Contributions

I am the sole author of this paper. It incorporates some ideas from Oliver Theel, who was my PhD thesis advisor at that time. The paper was presented by me at the work in progress session at the 17th EUROMICRO International Conference on Parallel, Distributed and Network-based Processing (PDP 2009) in Weimar, Germany.

Towards an End-User-Requirements-Driven Design and Deployment of Wireless Sensor Networks [*]

Felix J. Oppermann
Department of Computer Science
Carl von Ossietzky University Oldenburg, Germany
felix.oppermann@informatik.uni-oldenburg.de

1. Introduction

A wireless sensor network (WSN) is a network potentially composed of a large number of small and relatively cheap sensor nodes. Each node contains sensors to gather information about its environment. In addition it is equipped with a microcontroller that allows preprocessing of data. These nodes communicate with one another via low power radio interfaces and form ad-hoc networks to relay the data. A WSN allows cheap and dependable monitoring of a large area. Sensor nodes generally have limited power supplies, so that energy consumption must be reduced to extend the overall lifetime of the network as much as possible. The unique properties of WSNs demand for specific solutions to many problems and often disallow the use of traditional approaches. Currently a successful deployment requires expertise and experience [2]. Wireless sensor networks receive high research attention but despite this large interest [1], the adoption of WSNs outside of the scientific community is still very limited.

To allow a widespread use of WSNs, it is necessary to enable technically less skilled personnel to successfully deploy a WSN to solve their specific problems. While these end-users can be assumed to be experts in their application domain, like in biology or in geology, they cannot be expected to be especially trained in computer science. One must ensure that the design and deployment process requires as little technical knowledge as possible and can be automated as much as possible. We believe that these characteristics will pave the way for this technology finally being a success "in the field."

2. Problem Statement

Depending on the intended application, a WSN must fulfill very specific requirements, which in turn demand for rather diverse solutions to different subproblems like routing or data dissemination. For example, some applications like temperature monitoring only require very little bandwidth, others like the tracing of vibrations in structural health monitoring demand for a high bandwidth [8]. The heterogeneity in the application scenarios and WSN deployments effectively renders a one-size-fits-all solution impossible. It is expected that a need for different solutions to optimally address application-specific problems will remain. Even for an experienced computer scientist, it is difficult to make the right choices; for the intended end-user it is almost impossible. This is a particular problem since a wrong decision taken in the design phase can severely affect the performance and reliability of the WSN. Current solutions, even though deployed with the help of computer scientists, are often poorly able or even unable to fulfill the intended purpose [6].

3. Related Work

There are several approaches that try to attack the problem of easing the design and deployment of WSNs, but at a different — and we believe — inappropriate level: Most of them aim at making the actual programming of a WSN easier [2, 3, 7].

For example, the Sensor Network Application Construction Kit (SNACK) [4] developed by Greenstein et. al. automates the generation of deployable code based on an abstract combination of components. The SNACK framework is intended to provide a repository of components that provide solutions for common problems found in WSNs. The system allows the easy creation of a working application based on a selection of preprogrammed components. The selection of the algorithms and parts that best solve the given problem is still left to the user.

With TASK, Buonadonna et. al. [2] developed a toolkit to allow users to deploy a WSN for habitat monitoring. This toolkit supports the user in all design and deployment

[*]Supported by the German Research Foundation (DFG), grant GRK 1076/1 (TrustSoft)

framework only considers the specific application class of habitat monitoring. In addition the framework does only support TinyDB as middleware, which presents the WSN as a virtual SQL-database. The TASK framework has a modular design to allow the exchange of low level components to better support a specific deployment, but the important selection of suitable components is left to the user. In most cases the user is supposed to stick with a predefined selection that suits most scenarios in the given application area.

4. Our Approach

In order to make the deployment of WSNs easier and more reliable for unacquainted end-users, a methodology is required to synthesize a possible structure for an applicable WSN based solely on the requirements and constraints of the intended task. Such a framework eases the decisions to be made during the design of a sensor network and allows the less acquainted end-users to successfully deploy a WSN tailored for solving a specific problem. The aim is a system that allows the end-user to automatically generate a design for a WSN based on a limited amount of functional and non-functional requirements and constraints. These requirements are given in a language familiar to the end-users and without a need for deep technical knowledge. For example, a biologist might require that a WSN generates a temperature reading every five minutes at measuring points approximately ten meters apart. In addition to the basic functional parameter the user has specific expectations on non-functional requirements of the system, like the intended system lifetime and reliability. To allow the reuse of already available hardware components, it is important that the selection procedure can be further constrained. For example, the user might want to limit the selection to a given node hardware he or she already owns. The ultimate goal is to offer the end-user a tool to automatically generate a mission specific selection out of the available hardware and software components at “the push of a button.” The design can then, for example, be used as the groundwork of an automatic software generation using SNACK. Both processes can be integrated in a broader deployment support infrastructure. This would allow to support the user during the whole design and development process. The barriers for deploying a WSN would be reduced without sacrificing the necessary flexibility of tailoring the WSN to the specifics of the intended application.

Several problems must be solved before this vision can be realized. Until now, no consensus on the architecture of WSNs has been reached [5]. It is quite unclear how a WSN system can cleverly be structured. Building a modular architecture is thus still considered difficult [7]. As a consequence, a first step must be the identification of ity. Besides, it is necessary to determine what kind of basic requirements the possible applications of WSNs pose. Until now these requirements are not well understood [7]. Finally, to be able to quantify the fulfillment of non-functional aspects like system lifetime and quality of service there is a need for applicable models for energy consumption, availability and security of WSN algorithms. These models must be detailed enough to allow a solid comparison of the available options but must be simple enough to allow effective algorithm selection.

In a final step the practicability of the approach must be demonstrated by the application to different well known scenarios. This allows the comparison between generated design and existing manually generated solutions.

References

[1] P. Baronti, P. Pillai, V. W. C. Chook, S. Chessa, A. Gotta, and Y. F. Hu. Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards. *Computer Communications*, 30(7):1655–1695, May 2007.

[2] P. Buonadonna, D. Gay, J. M. Hellerstein, W. Hong, and S. Madden. TASK: Sensor network in a box. In *Proc. of the 2nd European Workshop on Sensor Networks (EWSN 2005)*, pages 133–144, Istanbul, Turkey, 2005.

[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. *ACM SIGPLAN Notices*, 38(5):1–11, 2003.

[4] B. Greenstein, E. Kohler, and D. Estrin. A sensor network application construction kit (SNACK). In *Proc. of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*, pages 69–80, New York, NY, USA, 2004. ACM.

[5] K. Henricksen and R. Robinson. A survey of middleware for sensor networks: state-of-the-art and future directions. In *Proc. of the International Workshop on Middleware for Sensor Networks (MidSens '06)*, pages 60–65, New York, NY, USA, 2006. ACM.

[6] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *Proc. of the 14th Int. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 2006)*, pages 1–8, Apr. 2006.

[7] A. Tavakoli, P. Dutta, J. Jeong, S. Kim, J. Ortiz, D. Culler, P. Levis, and S. Shenker. A modular sensornet architecture: past, present, and future directions. *ACM SIGBED Review*, 4(3):49–54, 2007.

[8] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A wireless sensor network for structural monitoring. In *Proc. of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*, pages 13–24, New York, NY, USA, 2004. ACM.

7.2 Paper B: Inferring Technical Constraints of a Wireless Sensor Network Application from End-User Requirements

Felix Jonathan Oppermann and Steffen Peter (2010). “Inferring Technical Constraints of a Wireless Sensor Network Application from End-User Requirements.” In: *Proceedings of the 6th International Conference on Mobile Ad-hoc and Sensor Networks (MSN).* (Hangzhou, China). Piscataway, NJ, USA: IEEE, pp. 169–175. ISBN: 978-1-4244-9456-9. DOI: `10.1109/MSN.2010.32`.

URL: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5714494`

7.2.1 Summary

This paper describes a two-step process to infer specific technical constraints and parameters needed for a reliable mission-specific design of WSNs. As the first step, we propose a new requirement catalog helping end users to formulate a complete and consistent specification of WSN mission requirements. As the second step, we introduce a methodology to deduce fine grained technical specifications from the general requirements. The proposed automatic graph-based requirement expansion approach translates the content of the catalog and additional requirements to specific technical terms, which provide the basis for an application-specific WSN design. A real-world use case – a new WSN application in the area of critical infrastructure protection – demonstrates the applicability of the presented approach.

7.2.2 Contributions

I am the main author of this paper and developed most of the ideas described in Sections I to IV. Section V was primarily written by Steffen Peter, while Section VI represents joint work. The paper was presented by Stefen Peter at the 6th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN 2010) in Hangzhou, China.

Inferring Technical Constraints of a Wireless Sensor Network Application from End-User Requirements

Felix Jonathan Oppermann*
Department of Computer Science
Carl von Ossietzky University of Oldenburg
26111 Oldenburg, Germany
Email: oppermann@informatik.uni-oldenburg.de

Steffen Peter
IHP GmbH
Im Technologiepark 25
15236 Frankfurt(Oder), Germany
Email: peter@ihp-microelectronics.com

***Abstract*—This paper describes a two-step process to infer specific technical constraints and parameters needed for a reliable mission-specific design of wireless sensor networks (WSN). As the first step, we propose a new requirement catalog helping end-users to formulate a complete and consistent specification of WSN mission requirements. As the second step, we introduce a methodology to deduce fine grained technical specifications from the general requirements. The proposed automatic graph-based requirement expansion approach translates the content of the catalog and additional requirements to specific technical terms, which provide the basis for an application-specific WSN design. A real-world use case – a new WSN application in the area of critical infrastructure protection – demonstrates the applicability of the presented approach.**

***Keywords*-Requirements/Specifications; Design Tools and Techniques; Sensor Networks;**

I. Introduction

As of today, advantages in microelectronics permit the equipment of individual sensors with limited computing capabilities and radio interfaces. It is envisioned to apply such wireless sensor networks (WSN) in a wide range of different scenarios, including application areas like habitat and structure monitoring, catastrophe management, and home automation. The diversity of the application range and the specific properties of WSNs make their design difficult and especially challenging for the intended end-users, like biologists, geologists, and engineers. To allow a widespread use of WSNs, it is necessary to enable technically less skilled people to successfully deploy a WSN. While these end-users can be assumed to be experts in their application domain, for example in biology or in geology, they cannot be expected to be especially trained in computer science in general or WSN technology in particular. One must ensure that the design process requires as little technical knowledge as possible and can be automated as far as possible.

Often, a large number of alternative solutions is available and it is difficult to select a good one. Naturally such a selection needs a clear definition of what is actually required. Thus, the definition of requirements is one key issue during the development of WSNs. Only with the precise information of what should be achieved it is possible to perform precise and goal oriented engineering. To the best of our knowledge currently no accepted and reliable way of formally specifying WSN applications and their parameters is known. A major problem is the diversity of application domains. Another problem is that domain experts are usually not familiar with the terms used in WSN engineering. Thus, a translation of rather high level, fuzzy, domain-specific requirements to measurable metrics that can be used within the WSN development process is required.

This paper proposes a two step approach to infer the required precise technical parameters from the general end-user requirements. After the description of the WSN design flow in Section II, the requirement definition process is introduced. The first step of the requirement process applies a novel requirement catalog helping end-users to formulate complete and consistent specifications of WSN applications. This requirement catalog is described in Section III. Section IV describes the second step, a graph-based requirement expansion process which outputs a fine granular technical specification. A real-world use case, a new WSN application in the area of critical infrastructure protection, demonstrates the applicability of the proposed approach in Section V. Finally, the paper is concluded by a summary and brief outlook in Section VI.

II. Design Process

It is our vision that in the future the design of WSN application is not the expensive and extremely time consuming development task it is today, but a rather straightforward process. Ultimately, it should even be possible for end-users to execute this process on their own.

Two distinct approaches have been proposed towards a simplified design and setup of WSNs. The first approach is based on a standard middleware that is deployed on all nodes. It is configured according to the requirements of the application. TASK [1] is a prominent examples for such configurable middlewares in sensor networks. The approach is promising and appears to be feasible for application

* Also affiliated with the University of Lübeck, Germany

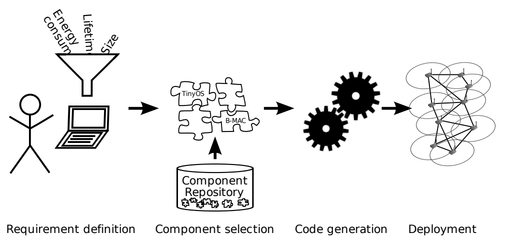

Figure 1. Stages of our envisioned partly automated WSN design process

scenarios that are not too diverse and if sufficient memory and processing resources are available. The second approach includes a (semi-)automatic composition process. Examples are SNACK [2] and ConfigKIT [3]. Based on the application requirements, exactly the components (i.e., hardware and software modules) that promise to deliver the required task optimally are assembled. Hence, it promises to be highly efficient with regard to memory and computational costs. A major disadvantage of such a composition process is its technical complexity: how to find modules that can be composed and how to predict how such a composition behaves?

In this paper, we focus on the latter approach, since we are convinced that coping with stringent memory and computation constraints – a key benefit of the composition process – will stay a key factor in WSN engineering. In particular, we pick up the development process introduced by ConfigKIT [3]. ConfigKIT allows the automatic selection of suitable components from a repository based on abstract requirements. In contrast to the broader scope of the approach presented in this paper, the focus of ConfigKIT is primarily on security aspects.

Figure 1 shows the general flow of a composition driven design process. Based on the requirements defined by the user, components and software modules are selected that promise to satisfy the user's requirements. The selection and assessment process employs a repository containing models of components and their properties. Based on the resulting configuration, the actual source code is generated and compiled. Finally, the sensor nodes are equipped with the resulting code images and are deployed at the application site.

It is apparent that the basic requirements given by the user define and start the process that finally concludes in the deployment of the sensor network. They define the constraints, the environment, the functional and qualitative properties that have to be met. Thus, they influence each following step in the development process. The requirement definition step can only be done in close co-operation with the future users of the WSN. To allow such a co-operation, it is necessary to agree on a common language for this phase. On the one hand, this language needs to be understandable for the users and on the other hand it needs to allow a complete specification of all important aspects of the mission.

From our experience, in most cases such a requirement definition – even if it is correct, consistent, and complete – cannot be used directly to start the composition process. Rather, it is necessary to translate the user requirements into a technical specification, containing terms such as "topology", "data rate" or "message integrity". In ConfigKIT, the responsibility for this translation was mainly shifted to the user side. It is expected that the user is able to understand and to define the technical terms. Thus, this tool is clearly focused on developers and not on less experienced end-users. In order to support end-users directly, a more general requirement formulation step has to be added before defining the actual technical WSN terms. We presume that end-users are able to formulate abstract application requirements sufficiently and correctly. However, since we do not assume they can use technical WSN terms correctly, a deduction step is required to infer the technical terms from the requirements given by the user.

This process poses two general questions. First, how can a user enter the application requirements so that they are understandable to him and the WSN engineer, and also usable in a formal framework? And second, how to deduce the technical terms from the (partially fuzzy) end-user requirements? To tackle the first question, we propose a catalog of possible requirements that can structure the requirement definition process and allows a more formal specification of the application. In addition, it may help to ask the right questions when communicating with the future users. A detailed description of the catalog can be found in the next section. In a second step, as described in Section IV, the resulting requirement definition is translated to a more detailed technical specification.

III. Requirement Catalog

In this section we present a catalog of WSN requirement dimensions that is intended to structure the requirement analysis for WSNs. Even though different in aim and scope, this catalog has some similarities with previously proposed WSN classification schemes. In 2002, Tilak, Abu-Ghazaleh and Heinzelman [4] defined an early taxonomy for WSNs. This taxonomy allows a classification of WSNs according to different communication functions, data delivery models, and network dynamics. Römer and Mattern [5] define a list of properties that allow the characterization of WSN applications. Their design space contains 12 major dimensions, some of which contain several sub-dimensions. Similar to the taxonomy defined by Tilak et al., their focus is on communication and topology aspects. This classification scheme was later refined by Rocha and Gonçalves [6]. The result is a simplified classification scheme with only seven major dimensions. All of these classification schemes have in common that they do not only consider requirements and constraints, but also more technical properties of an actual deployment, like network size, network topology, and heterogeneity. These properties are useful for classifying existing WSN solutions, but are unsuited for structuring the requirement analysis.

The requirement dimensions of the proposed catalog can be assigned to five categories. A complete overview is given in Table I. Each dimension is either specified by possible instances (italics) or a short definition. In the following sections, we examine the dimensionsin more detail.

A. Mission

This category groups the functional requirements of the application, which define the goal of the WSN deployment. A central factor is the selection of suitable *sensors*, which determine what can be detected and monitored by the sensor network. Furthermore, this already defines several properties of the WSN. In addition to a selection of sensors we also specify how often the sensors need to be read. The *sampling interval*, combined with the properties of the sensor, defines how much data is generated over time or, in case of event detection, how often the node needs to evaluate whether an event has been detected. The sensory range of typical sensors usually found in WSNs is often rather restricted. As a consequence, full *coverage* of a given area requires a large number of nodes, but many application scenarios require only partial coverage of the area and thus a lower number of sensor nodes. If the exact *number of measurement* points is known to the user, it is helpful to allow him to directly specify this number. A sensible interpretation of the data gathered by a WSN often requires a *spatial* and *temporal correlation* of the individual measurements to generate a coherent picture of the situation. In particular, some scientific applications, like the monitoring of earthquake shock waves, can require very precise information of the time and location at which an event is detected [7]. The demanded temporal and spatial accuracy determines the need for time synchronization and localization and the degree of precision these algorithms have to provide. The *mode of operation* defines how the sensed data is to be retrieved by the user. Scenarios range from *event detection*, where the user is only alerted if some predefined event is detected, to the *monitoring* of a given area up to more *interactive* systems replying to the user's queries on demand. The mode of operation implies how much intelligence is required inside the WSN. The degree of *mobility* in the network has a strong influence on routing and media access. The choice of localization methods is also influenced by the degree of mobility.

B. Operation Environment and Deployment

Besides functional requirements of the mission, the design of WSNs is also largely affected by the properties of the operation environment. The most important environmental aspects are probably the *size* and *dimensionality* of the deployment space. The size of current WSN deployments is quite diverse and ranges from few nodes in a single room to thousands of nodes spread over several square kilometers. Combined with the desired coverage and the network dynamics, the size determines the number of nodes that are needed for the deployment. Often overlooked is the dimensionality of the deployment space, although it has a strong impact on routing algorithms and positioning systems, as many common algorithms only work well in a two-dimensional network. We consider the space to be two-dimensional if one or two dimensions dominate in size and no nodes need to be placed above each other. The exact *environmental conditions* also largely influence the design. As a starting point – analogously to Römer and Mattern [5] – we only differentiate between outdoor, indoor, and mixed environments. A fully automated design process, especially if also considering hardware design, might require a more detailed distinction. As an exception, we included an additional more detailed description of the constraints for radio communication as these have a strong impact on the design of a WSN. WSNs are usually envisioned to operate autonomously in remote locations, but in many scenarios, this assumption does not hold. In case it is known that some infrastructure is available or the WSN is well accessible, it makes sense to exploit this. How the WSN is going to be deployed at the final operation site can also significantly affect the design of the sensor network. A careful manual placement of all nodes puts less demand on self-organizing qualities of the routing method than randomly dropping the nodes from an unmanned aerial vehicle (UAV). In some scenarios it is possible or desired to modify the network during its use and, for example, to add further nodes in order to replace failed ones or extend the monitored area.

C. Performance and Dependability

To be a useful tool, a WSN needs to live up to performance and dependability expectations. While early environmental monitoring applications seldom had challenging performance requirements, this changed with recent safety critical applications, like forest fire detection. As sensor nodes are usually powered by non-replenishable energy sources and WSNs are intended for extended operation periods, lifetime is probably the most important non-functional property of a WSN. There are several definitions of WSN lifetime. In this context we define *lifetime* as the amount of time the network can operate until too many nodes fail and other requirements are irrevocably violated. Expected lifetime may range from hours to several years. A high lifetime goal puts strict limitations on other properties of the WSN. In some scenarios, energy harvesting could significantly increase lifetime, but in current applications it is rarely used. In general, it is expected that a WSN is always operational during its lifetime, even in the presence of failures. For a

Table I
REQUIREMENT CATALOG

Category	**Requirement Dimensions**	***Instances* / Definitions**
Mission	sensors	list of sensors
	sampling interval	minimal interval to sample the sensors
	coverage	*points of interest, sparse, dense, redundant*
	number of measurement points	number of required sensors (if known)
	accuracy of spatial correlation	maximal position error for measurements or events
	accuracy of temporal correlation	maximal error of measurement/event timestamps
	mode of operation	*event detection, monitoring, tracking, interactive*
	mobility of sensors	*static, partly mobile, mobile*
	mobility of observer	*static, mobile*
Operation Environment and Deployment	dimensionality	*two-dimensional, three-dimensional*
	size	maximal dimensions of the deployment space
	environment conditions	*indoors, mixed, outdoors*
	radio interference level	*none, low, medium, high*
	radio regulations	country/region of deployment
	available infrastructure	list of infrastructure systems (e.g., GPS, power grid)
	mode of deployment	*manual, random*
	time-frame of deployment	*one-time, continuous*
	accessibility	*inaccessible, limited accessibility, accessible*
Performance and Dependability	lifetime	minimal time until the WSN permanently fails
	availability	percentage of lifetime the network is operational
	channel dependability	percentage of reported events out of all locally detected events
	response time	maximal time between event detection and report
Security	eavesdropping resistance	*none, low, medium, high*
	tampering resistance	*none, low, medium, high*
	denial of service resistance	*none, low, medium, high*
	access control	*none, monitored, authorized, restricted*
	stealthiness	*none, limited*
Development Costs	monetary costs	maximal overall costs
	development effort	maximal man-hours

number of scenarios, it is possible to trade availability for higher lifetime or reduced costs. We define *availability* as the percentage of the lifetime the network is operational and can respond to queries. In addition the user's expectations on the *dependability* and *response time* of the communication need to be defined. There is usually a conflict between certain dependability and latency constraints and lifetime goals. A low latency, for example, reduces lifetime as it prevents long sleep periods for the nodes.

D. Security

For early WSN applications, security was not a concern. Typically, WSNs were deployed for habitat monitoring or similar scientific applications. Data secrecy is usually no concern in such applications and tampering is unlikely especially if the WSN is deployed in a remote location. Other WSN applications pose demanding requirements in terms of security. For example in medical applications privacy of the sensed data needs to be protected and obviously military or security applications demand for reliable operation even in the presence of attacks. We differentiate four orthogonal security dimensions. If the data generated by a WSN is of privacy critical nature, it is necessary to prevent easy *eavesdropping*. Eavesdropping can be countered by using protected communication channels, either by using physically secured channels or by applying cryptographic means. If the soundness of the reported data is important and it is likely that an attacker tries to manipulate the communication, additional protection is necessary. *Tampering* can be countered by employing authentication. Finally, an attacker could also try to completely interrupt the operation of the network. For all the above dimensions, the required level of protection depends on the capabilities of a likely attacker. We distinguish four security levels by applying an attacker classification, similar to the classification scheme proposed by Abraham et al. [8]: no protection, individuals accidentally detecting and playing around with an unprotected network, small groups with limited resources and knowledge, and large organized and experienced groups that carry out planed attacks. The required level of protection also depends on what kind of access control is already enforced in the operation environment. If the nodes are physically well protected, only attacks on the radio channel are likely. The classification of *access control* measures is based on a similar scheme by Weingart et al. [9]. The last security dimension describes the *stealthiness* of the WSN. Especially in military and security applications, it can be required to conceal the presence of the network.

E. Development Costs

Besides technical aspects, available funding and development capacities can play an important role for design decisions. Especially for large scale networks, it is important to limit *monetary costs* of individual nodes and limited

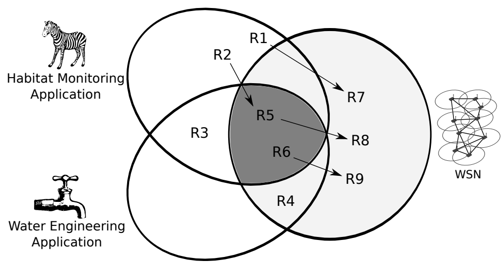

Figure 2. Requirement design spaces: Each ellipse represents the space of requirement types that are understood by corresponding domain experts. It is the goal to infer requirements understood by WSN engineers (R4 to R9) from domain-specific requirements (R1 to R3). Key is a requirement catalog (dark gray area, contains R5, R6) understood by all experts.

development resources might, for example, disallow the excessive use of custom built components.

IV. Requirement Expansion

It is the goal that the requirement catalog introduced in the previous section covers the space of possible requirement combinations as broadly as possible. Nevertheless, the space will never be complete in a way that it can describe all possible application requirements. This is mainly caused by the notion that the catalog space only covers requirements that are well understood by experts of all domains. Naturally, there remain requirements that do not fit in such a general catalog. Additionally, new requirements can occur for new application areas. In Figure 2 the potential catalog area is the shaded shared space in the middle, that only overlaps a fraction of the likely requirements. Each ellipse in Figure 2 represents the space of requirement types that are understood by corresponding domain experts, for example habitat monitoring designers or waterworks operators, on the left and the domain of WSNs on the right. Overlapping areas represent shared knowledge space. Basically, there are two means to cope with the problem of undefined requirements. Either the catalog is extended by new dimensions, or additional specifications beside the actual catalog are allowed. Extending the catalog to less general categories and properties is not appealing. It would render the catalog cluttered and thus less usable, and contradict the initial notion that the catalog is a shared space for all domain experts. Therefore, it is inevitable that requirements exist outside of this catalog on both sides. On the one side, the end-user has requests that are not cataloged and are not initially understood by WSN engineers. For instance in ZebraNet [10] the nodes are deployed to zebras. A fact from which a biologist can easily imply other requirements based on the behavior of zebras as herd animals, while that knowledge is not common for computer scientists. On the other side, WSN engineers eventually need technical definitions that are beyond what end-users have to know, for example the need for and parameters of congestion protocols in the transport layer.

Figure 2 illustrates the three different types of possible deductions:

- inferring requirements into the catalog (R2 $\rightarrow$ R5),
- inferring requirements from the catalog to core technical definitions (R6 $\rightarrow$ R9),
- inferring requirements alongside the catalog (R1 $\rightarrow$ R7).

The first type has already been tackled by the catalog itself, since its task is to raise the questions that support users entering correct requirements. The latter two inferring types can be resolved by a forwarding process, we call "requirement expansion."

The requirement expansion process works on a flexible graph structure containing all known requirement types. Requirement types are types that can (but do not need to) be set for applications under development. The 29 dimensions in the requirement catalog already define 29 different requirement types. The basic idea of the graph is that once certain requirement types are defined, it implicitly also defines other deducible requirement types. For example, from the fact that zebras are herd animals we can deduce that many similar nodes are in range, so that the density of the network is rather high, which can be inferred to multi-hop, short distance communication requirements.

We apply this forwarding methodology for deducing requirement types of high abstraction to technical terms, but also within the technical requirement space. The latter is motivated by the fact that the same requirement can be expressed in different ways. For example, the sampling frequency can be expressed as sampling period. It is our goal that all possible expressions of a requirement are set, especially with regard to an automatic composition process.

The underlying graph structure G can be expressed as directed graph $G = (R, T)$, while R is the set of requirements types, and T are the edges describing the translation of derivable requirements. Each element R_i out of R is a tuple $R_i = (D, V)$, D is the name or description of the requirement dimension; and V is the description of the value space. The value space can define a numeric range or a nominal scale, as needed for most of the requirements of the catalog. The translation set T is the set of triples $T_j = (R_{\text{from}}, R_{\text{to}}, f)$, which describe the mapping f from requirement R_{from} to R_{to}. The mapping function f can be an arbitrary function. In most cases basic math operations and conditional expressions are sufficient.

Without changing this general semantics, in practice we found it valuable to include the translations in the description of individual requirements R_i. By this, individual requirements contain information on how they relate to other requirements. To realize this, we followed an optional push/pull methodology. *Push* means that a requirement translates its properties to neighbored requirements. As an example, the translation from qualitative requirement metrics to absolute values is usually a push from the qualitative properties. This allows the existence of more than one qualitative metric in the system to describe the same property. In the opposite direction, a requirement type can *pull* parameters of other requirements to define its own settings. For example, a data throughput requirement can pull values of packet size and measurement interval to infer its setting.

Adding the push and pull translation R_i can be refined by the tuple $R_i = (D, V, T_{\text{push}}, T_{\text{pull}})$. The set T_{push} is a subset of T, where $R_{\text{from}} = R_i$. The set T_{pull} is a subset of T, where $R_{\text{to}} = R_i$. This allows to build G solely on the information stored in the requirements.

Since as a result of this structure translations are part of the elements, new requirement types can be easily added without interfering with existing structures. For instance, if a domain engineer wants to add a requirement without using the catalog (like R1 in Figure 2) , it is sufficient to describe D and V of the new type, and add push translations to describe how the new requirement affects already existing requirement types. In the habitat monitoring example, R1 could be the species of monitored animals, and the translations describe how the behavior of the animals affects network properties. In contrast, novel technical terms typically apply pull translations. For example a new metric for expressing energy consumption would describe how it can be deduced from existing types, that is how it is connected to the existing knowledge. The modular approach of describing requirements allows adding requirement types in a flexible way. It is even possible to bundle requirement types in packages that can be added to the core database (catalog and WSN types) based on the application domain under development.

V. Case Study: water pipe surveillance

In the following, we demonstrate the applicability of the proposed methodology by using it to conduct the requirement analysis for a recent WSN application. The scenario is part of the WSAN4CIP project [11]. The *Frankfurter Wasser- und Abwassergesellschaft mbH* (FWA) demonstrator concerns the fail-safe and secure data transmission for monitoring the operation of water mains at Frankfurt (Oder), Germany, and thus to protect that type of critical infrastructure. It is mandatory that the system can provide a similar degree of reliability and security as wired monitoring systems, which are currently used for the task. By reducing the necessary infrastructure, a WSN should allow to reduce costs and provide greater flexibility.

The pipeline connects the waterworks, where the drinking water is collected, and an elevated tank by two parallel underground water pipes with a total length of 17.5 km. Part of the infrastructure are five stations along the pipes that are equipped with flow rate and pressure measuring devices permitting optimal operating and monitoring of the pipe system. The gathered data is displayed at the central process management system for supervisory control. The state of the system is measured every 30 seconds. The projected WSN consists of five measurement delivering nodes that are placed along the pipe at a distance of up to 5 km. The nodes will be located at the existing monitoring stations that provide information on water pressure and flow rate as analog data. Additional nodes are placed between the measuring nodes to relay the network traffic.

In the manual process, as first step, the WSN engineers, probably in dialogue with the end-user, has to derive the technical specifications. If, in contrast, the proposed requirement definition catalog is applied, the clearly structured initial requirements would look similar to Table II. The table lists the determined requirement values for the FWA demonstration scenario. These 29 requirements are the output of the user definition phase. Following the translation flow as described in the previous section we could derive many technical requirements. For example, we directly derive the properties of the sensors. These derived properties (floating point values as data format and a measurement time of less then one second) are new constraints for the further development process. Thus the sensor types *pushed* the new requirements as introduced in the previous section. The expanded requirement space directly feeds the selection process as introduced in Section II.

While the catalog and the requirement expansion in this example could demonstrate the general suitability, also several problems were discovered: (1) The translation from the catalog to the technical terms still needs human interaction. New translation functions had to be added or refined. We expect the issue to be resolved with the help of more experience and data we get from other applications. (2) Some properties needed additional information beside the catalog. (3) Push translations can lead to ambiguous definitions of requirements. The current implementation ("take what's defined first") is not always satisfying. A solution for the issue is either human interaction or a precedence order. (4) The number of resulting requirements at the end of the deduction process becomes extremely large, as all deducible requirement types are inferred without additional reasoning.

VI. Conclusions

In this paper, we proposed a novel requirement definition process for wireless sensor networks to bridge the semantic gap between application requirements as they can be expressed by end-users and technical terms and constraints, as they are needed for the WSN design process. As a first step, a new requirement catalog assists the requirement analysis for WSN applications. This catalog is specifically designed with the end-user in mind. It allows the easy and complete specification of different WSN missions. Since a more detailed technical specification is required for the WSN design, we proposed, as a second step, a requirement expansion methodology. The demonstration of the

Table II
REQUIREMENTS FOR THE FWA DEMONSTRATOR

Category	Requirement Dimensions	Value
Mission	sensors	flow rate, pressure
	sampling interval	30 s
	coverage	points of interest
	number of measurement points	5
	accuracy of spatial correlation	n/a (exact positions are known)
	accuracy of temporal correlation	60 s
	mode of operation	monitoring, event detection (optional)
	mobility of sensors	static
	mobility of observer	static
Operation Environment and Deployment	dimensionality	two-dimensional
	size	17 500 m × 10 m
	environment conditions	mixed
	radio interference level	high
	radio regulations	Germany/Europe
	available infrastructure	power grid (sensor nodes, only)
	mode of deployment	manual
	time-frame of deployment	one-time
	accessibility	limited accessibility
Performance and Dependability	lifetime	3 months
	availability	98 %
	channel dependability	98 %
	response time	2 s
Security	eavesdropping resistance	low
	tampering resistance	low
	denial of service resistance	low
	access control	restricted, none
	stealthiness	none
Development Costs	monetary costs	20 000 €
	development effort	unknown

requirement definition process, employed in the design of a new WSN application in the area of critical infrastructure monitoring, showed concepts and benefits of the process, but also exposed potential room for further improvement.

Thus, in the future we will optimize the given catalog and extend the set of requirement expansions based on practical experience and feedback from the community.

ACKNOWLEDGMENT

This work is supported by the German Research Foundation (DFG), grant GRK 1076/3 (TrustSoft), and the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n^o 225186 (WSAN4CIP).

REFERENCES

[1] P. Buonadonna, D. Gay, J. M. Hellerstein, W. Hong, and S. Madden, "TASK: Sensor network in a box," in *Proc. of the 2nd European Workshop on Sensor Networks (EWSN)*, 2005.

[2] B. Greenstein, E. Kohler, and D. Estrin, "A sensor network application construction kit (SNACK)," in *Proc. of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.

[3] S. Peter, K. Piotrowski, and P. Langendörfer, "In-network-aggregation as case study for a support tool reducing the complexity of designing secure wireless sensor networks," in *Proc. of the Third IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp)*, 2008.

[4] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman, "A taxonomy of wireless micro-sensor network models," *SIGMOBILE Mobile Computing and Communication Review*, vol. 6, 2002.

[5] K. Römer and F. Mattern, "The design space of wireless sensor networks," *IEEE Wireless Communications*, vol. 11, 2004.

[6] V. Rocha and G. Gonçalves, "Sensing the world: Challenges on WSNs," in *Proc. of the IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, May 2008.

[7] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, "Fidelity and yield in a volcano monitoring sensor network," in *Proc. of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[8] D. G. Abraham, G. M. Dolan, G. P. Double, and J. V. Stevens, "Transaction security system," *IBM Systems Journal*, vol. 30, no. 2, 1991.

[9] S. H. Weingart, S. R. White, W. C. Arnold, and G. P. Double, "An evaluation system for the physical security of computing systems," in *Proc. of the Sixth Annual Computer Security Applications Conference*, Dec. 1990.

[10] P. Zhang, C. Sadler, S. Lyon, and M. Martonosi, "Hardware design experiences in ZebraNet," in *Proc. of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.

[11] WSAN4CIP project, "Deliverable D1.1," 2009. [Online]. Available: http://www.wsan4cip.eu/fileadmin/public_documents/Deliverables/WSAN4CIP_D1.1_Def.of.parameters.pdf

7.3 Paper C: Towards Business Processes Orchestrating the Physical Enterprise with Wireless Sensor Networks

Fabio Casati, Florian Daniel, Guenadi Dantchev, Joakim Eriksson, Niclas Finne, Stamatis Karnouskos, Patricio Moreno Montero, Luca Mottola, Felix Jonathan Oppermann, Gian Pietro Picco, Antonio Quartulli, Kay Römer, Patrik Spiess, Stefano Tranquillini, and Thiemo Voigt (2012a). "Towards Business Processes Orchestrating the Physical Enterprise with Wireless Sensor Networks." In: *Proceedings of the 34th International Conference on Software Engineering (ICSE)*. (Zurich, Switzerland). Piscataway, NJ, USA: IEEE, pp. 1357–1360. ISBN: 978-1-4673-1066-6. DOI: 10.1109/ICSE.2012.6227080.

URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6227080

7.3.1 Summary

This paper describes the overarching WSN application development process developed within the make*Sense* project. Currently, the adoption of WSNs within industry is hampered by two main factors. First, there is a lack of integration of WSNs with business process modeling languages and back-ends. Second, programming WSNs is still challenging as WSN development is typically still based on low-level C code and a node-centric programming model. We propose a unified programming framework and a tool chain that generates code ready for deployment on WSN nodes from an abstract business process specification in BPMN syntax.

7.3.2 Contributions

Besides contributing ideas and solutions to most parts of the paper, I am a co-author of Section III-C where I provided a description of the core language and the compilation process.

Towards Business Processes Orchestrating the Physical Enterprise with Wireless Sensor Networks

Fabio Casati‡, Florian Daniel‡, Guenadi Dantchev†, Joakim Eriksson*, Niclas Finne*, Stamatis Karnouskos†, Patricio Moreno Montero**, Luca Mottola*, Felix Jonathan Oppermann+, Gian Pietro Picco‡, Antonio Quartulli‡, Kay Römer+, Patrik Spiess†, Stefano Tranquillini‡, Thiemo Voigt*
***Acciona Infraestructuras S.A. (Spain), †SAP AG (Germany), *Swedish Institute of Computer Science,*
+University of Lübeck (Germany), ‡University of Trento (Italy)

***Abstract*—The industrial adoption of wireless sensor networks (WSNs) is hampered by two main factors. First, there is a lack of integration of WSNs with business process modeling languages and back-ends. Second, programming WSNs is still challenging as it is mainly performed at the operating system level. To this end, we provide `make`*Sense*: a unified programming framework and a compilation chain that, from high-level business process specifications, generates code ready for deployment on WSN nodes.**

I. Introduction

Wireless Sensor Networks (WSNs) are small, untethered computing devices equipped with sensors and actuators. WSNs can be easily deployed and are able to self-organize to achieve application goals. Research has made significant progress in solving WSN-specific challenges such as energy-efficient communication. Industry, however, is reluctant to adopt WSNs. We believe this is due to two unsolved issues, integration and unification, schematically shown in Figure 1.

Integration refers to the need for strong cooperation of business back-ends with WSNs. Current approaches typically consider the WSN as a stand-alone system. As such, the integration between the WSN and the back-end infrastructure of business processes is left to application developers. Unfortunately, such an integration requires considerable effort and significant expertise spanning from traditional information systems down to low-level system details of WSN devices. Moreover, these two sets of technologies satisfy very different goals, making the integration even harder. This paper presents a holistic approach where application developers "think" at the high abstraction level of business processes, but the constructs they use are effectively implemented in the challenging reality of WSNs.

Unification refers to the need for a single, comprehensive programming framework. It is notoriously difficult to realize WSN applications. They are often developed atop the operating system, forcing the programmer away from the application logic and into low-level details. Many programming abstractions exist [1], but are hard to use since they typically focus on one specific problem. To drastically simplify WSN programming, particularly for business scenarios, we need a broader approach enabling developers to use several abstractions at once. In this paper, we present a unified comprehensive programming framework into which existing WSN programming abstractions can blend smoothly.

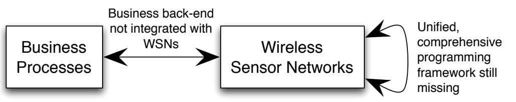
Business back-end not integrated with WSNs
Business Processes
Wireless Sensor Networks
Unified, comprehensive programming framework still missing

Figure 1. Open problems for using WSNs in business processes.

II. Application Scenarios

A paradigmatic example of our target scenarios is ventilation in buildings. Fans are commonly operated at a fixed rate, independent of room occupation, resulting in unnecessary ventilation of unoccupied rooms and over-ventilation of sparsely occupied ones, ultimately wasting energy. A smarter strategy may consider room occupation, resulting in sustainable building management. Consider an office environment, in which employees book meeting rooms on the Web through a back-end process notifying the expected participants. Room ventilation is minimal when no meeting is scheduled. Sensors and actuators driven by the business process increase ventilation before the meeting and until either human presence is detected or CO_2 levels are above a certain threshold.

Realizing this system requires a tight integration between the business process and the network of sensors and actuators dispersed in the environment, as the application logic needs to extend to the latter. Moreover, implementing the processing for adaptive ventilation complicates application development, as it departs from the traditional data collection—most common in WSN applications—to encompass possibly distributed control loops. Similar requirements are shared by numerous application domains such as predictive maintenance aboard cargo vessels.

III. Approach

Our design revolves around three fundamental goals:

- `make`*Sense* must *seamlessly integrate* with existing business process technology, providing an adoption path that complements, instead of disrupts, existing methodologies and technologies with WSN ones.

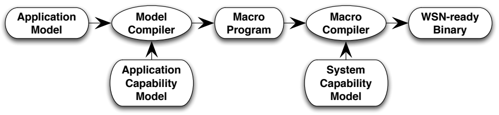

Figure 2. Compiling business process models into WSN-executable code.

- make*Sense* must be *modular* and *extensible*. As we aim for our system to be useful across several real-world applications, extensibility is key to ensure that the programming abstractions and their implementation can be easily adapted to the specificity of the target domain as well as to unforeseen needs.
- for extensibility not to be detrimental to performance, make*Sense* must *self-optimize* w.r.t. high-level performance goals. This is necessary to support long-lasting business processes subject to the randomness of the real world and rapidly changing requirements.

A. Architecture Overview

Our architecture is based on the separation of concerns provided by a distinction in layers of functionality: *i)* an *application* layer concerned with business processes and their modeling; *ii)* a *macroprogramming* layer concerned with the distributed execution of activities within the WSN; *iii)* a *run-time* layer concerned with the low-level aspects supporting the above and enabling self-optimization. The term "macro-programming" [1] refers to approaches that, unlike node-centric ones, allow specifying the behavior of multiple WSN nodes at once.

A model-driven approach connects the three layers (Figure 2). The application model represents a holistic, network-agnostic view of the entire business process, i.e., including the WSN and the process back-end. It includes performance requirements (e.g., a certain level of reliability, or a minimum lifetime). Details are further described in Section III-B.

Two compilation steps link the layers above. The model compiler takes as input the application model and an application capability model. The latter is a coarse-grained description of the WSN, providing information such as the type of sensors/actuators available and their operations. The model compiler translates these descriptions into a program written in a macro-programming language, described in Section III-C, serving as an intermediate language closer to the reality of WSN systems, yet high-level enough to be potentially used directly by a developer.

The macro compiler takes as input the macro-program generated by the model compiler and a system capability model. The latter provides finer-grained information on the deployment environment (e.g., how many sensors of a given type are deployed at a location). The macro-compiler generates executable code that relies only on the basic functionality provided by the run-time support available on the target nodes. By leveraging the system capability model, the macro compiler can generate different code for different nodes, based on their application role. The executable code contains the mechanisms enabling self-optimization, described in Section III-D.

B. Business Process Modeling

When integrating WSNs with business processes, most research projects and productive set-ups merely add a service facade to the WSN and orchestrate its services centrally. If middleware is deployed, that is done either purely in a central system [2] or with additional local components close to the WSN or on its gateway [3]. In make*Sense*, we use a more radical approach. As our goal is to decrease the effort of programming WSN applications, tools for process modeling are used to create the application top-most level. A process modeler models hybrid processes, of which one part is executed conventionally in a central execution engine, while another part is executed directly by the WSN.

We use and extend the *Business Process Modeling Notation (BPMN)*. By introducing new attributes, the modeler can specify a new *intra-WSN participant*, containing the logic executed by the WSN. As the latter is resource-constrained, we allow only a subset of BPMN elements. Furthermore, we introduced a new *WSN activity* type. This can be used only within the *intra-WSN participant* and is (except for the message activity) the only allowed activity type there. The WSN activity is backed by a meta-model, described in the next section. As WSNs are inherently distributed systems, we also introduced a *Target* attribute for lanes and activities within the *intra-WSN participant*, that allows specifying where the respective logic should be executed, based on labels that are relevant at the modeling layer. Finally, we added performance annotations, expressing that the WSN should optimize its operation for a specific goal (e.g., system lifetime or reliability) within a certain subsets of activities.

To assist the process modeler in creating correct, executable models, we use a set of meta-models that describe the WSN in terms of the logical functionality it provides, along with the way it is embedded into the physical set-up (e.g., which sensing or actuation is supported at which logical location). Instances of these meta-models can be created either manually or through dynamic service discovery.

At run-time, the BPMN process is executed in a distributed fashion. For message exchange between the intra-WSN participant and the other participants, the run-time uses a lightweight protocol, reducing encoded message size by using message structure information on both sides. Communication endpoints caring for serialization and deserialization of messages and for process instance correlation are generated automatically as part of the compilation process.

*C. The make*Sense* Macroprogramming Language*

Our intent is not to propose another macro-programming language. Rather, it is to provide a framework where the abstractions contributing to the language are decoupled,

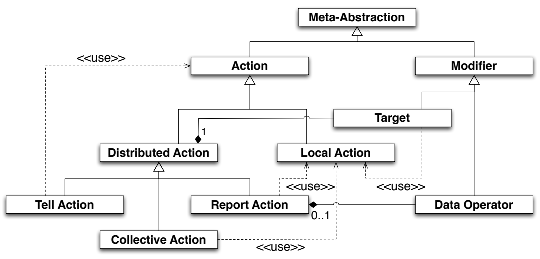

Figure 3. A model for the meta-abstractions of the make*Sense* macro-programming language.

leverage on existing implementations, and can be changed or extended easily to suit specific application needs.

This goal influenced the entire language design. To properly identify the units of functionality, reuse, and extensions we defined the notion of *meta-abstraction*, implemented through different "concrete" abstractions, as described later. Abstractions provide the key concepts enabling interaction with the WSN. However, their composition can be achieved by using common control flow statements, provided by a core language that serves as the "glue" among macro-programming abstractions. The core language, in our case a stripped-down version of Java we tailored for WSNs, is also the *trait d'union* between the macroprogramming abstractions and the BPMN business process model.

Figure 3 shows a UML meta-model for the meta-abstractions provided by the macroprogramming language. It focuses on the notion of *action*, a task executed by one or more WSN nodes. Actions are separated into *local*, whose effect is limited to the node where the action is invoked (e.g., acquiring a reading from the on-board temperature sensor), and *distributed*, whose effect instead spans multiple nodes.

Distributed actions are further divided into *tell*, *report*, and *collective* actions. The former two represent the one-to-many and many-to-one interaction patterns commonly used in WSNs to enable communication between the node (the "one") issuing the action and a set of nodes (the "many") where the latter is executed. A tell action enables a node to request the execution of a set of actions on other nodes, e.g., to issue actuation commands or to trigger reconfiguration of system parameters such as the sampling rate. A report action enables a node to gather data from other nodes. Event-based abstractions and periodic, continuous queries both fall in this category. Data acquisition occurring on each target node is specified by a local action given as input to the report action. The output of the local action is returned to the report one. Collective actions, in contrast to tell and report ones, do *not* focus on a special node where the action starts or ends. They enable a global network behavior and are executed cooperatively by the entire WSN through many-to-many communication. An example are distributed assertions [4], where programmers specify a (global) property monitored collectively by the WSN nodes.

Distributed actions may optionally have *modifiers* associated with them, "customizing" their behavior. We defined two modifiers, target and data operator. In our scenarios the nodes possibly differ along several dimensions, both physical and logical. For example, ventilation in Section II requires both CO_2 and presence sensors. Programmers must be able to map actions to the set of nodes of interest. A *target* identifies a set of nodes satisfying application constraints, and gives the ability to apply a distributed action to the nodes in this set. Instead, a report action may have a *data operator*, specifying processing performed on the results after gathering and before they are returned to the caller, e.g., to filter or aggregate the data.

To create an instance of a meta-abstraction, a class implementing its interface must be defined in the core language. As abstraction implementations typically closely interact with the operating system, methods of abstraction classes are implemented in C using a native code interface provided by the core language. Some abstractions require extensive configuration, for example, a target needs to define a set of nodes based on their properties [5]. To simplify such configuration, the core language supports the concept of *embedded languages*, code snippets formulated in the declarative configuration language provided by an abstraction. These are efficiently compiled by appropriate compiler plugins, instead of being interpreted at runtime.

D. Run-Time System

Besides providing a foundation for the distributed protocols in support of the macro-programming language, the run-time system offers self-optimization functionality to adapt the system behavior to changing requirements based on developer-provided high-level performance goals. For example, in the scenario of Section II, the high data reliability required to accurately monitor the persons' presence will correspond to different protocol settings compared to situations with no ongoing meetings, when energy preservation is the major performance concern.

To achieve this functionality, we gather run-time information from the WSN (e.g., network topology and protocol performance) and feed these to a reinforcement learning algorithm that uses simulations to explores the space of possible protocol configurations. At the end of each simulation round, the learning process evaluates the performance obtained with a given protocol setting w.r.t. the application's performance goals. Based on this, we derive self-optimization policies that specify which protocol parameters provide better performance as a function of the current application performance goal. We distribute the policies back to the deployed network where nodes will apply them upon recognizing changes in the current performance goal.

This approach sharply differentiates from existing solutions. Rather than requiring detailed modeling of the individual protocols, we treat the entire application as a

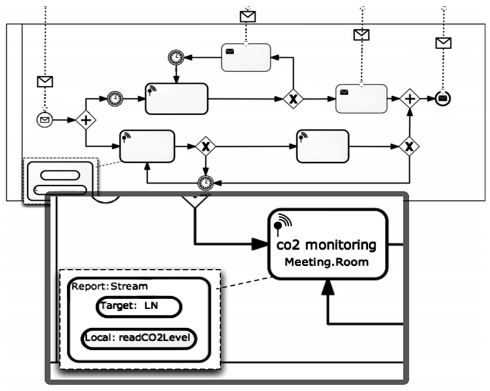

Figure 4. BPMN diagram for a fragment of the ventilation scenario.

black-box. This may lead to sub-optimal solutions, but also enjoys greater flexibility as it lets users add programming abstractions to the framework along with their supporting protocols and have the latter "implicitly" optimized.

IV. Current Status

We implemented the extended BPMN meta model in *Signavio Core Components*, an open source, browser-based BPMN editor. Our prototype implementation is focused on the model-to-macrocode transformation. Future work includes extending a BPMN runtime with the lightweight messaging protocol in JSON. The macro-compiler prototype is implemented as a multi-pass compiler employing the ANTLR parser generator and the StringTemplate engine. Currently, the compiler is primarily optimized for maintainability and extensibility. We also implemented concrete abstractions for *report* and *tell* actions, and for the *target* modifier. The former is a variation of a standard WSN data collection protocol, whereas the others rely on Logical Neighborhoods [5]. The self-optimization functionality is a separate stand-alone prototype written in Java that we are currently integrating into the make*Sense* run-time.

V. Case Study

Figure 4 depicts a fragment of the business process model for ventilation we briefly outlined in Section II. The whole process is modeled with two participants, the WSN-aware participant on top and the intra-WSN participant (modeled in more detail) that is converted into an application by generating macrocode. The zoomed part of the process shows a WSN activity that sets up and executes a periodic reading of CO_2 sensors in a certain room. By graphically combining abstractions—here a *target* specifying the room and a *local action* to read the sensor are used with a *report* action to collect sensor data—along with meta-information of the current WSN setup, the model becomes rich enough to be transformed into macrocode.

The corresponding code in Figure 5 describes the instructions to define a *target* including all CO_2 sensors and to

```
...
code nhoodTemplateS = {:
  neighborhood template CO2Sensors()
    f.getFunction() = "sensor" and t.getType() = "co2"
  create neighborhood co2Sensors from CO2Sensors () :};
Target co2Sensors = lnew LN(sensorNeighborhoodDef);

Report co2Stream = lnew Stream();
co2stream.setTarget(co2Sensors);
co2Stream.setParameter("period", 5 * 60);
co2Stream.execute();
...
```

Figure 5. Macro-programming language fragment for Figure 4.

collect periodic data from them using an instance of *report* action implemented with `Stream`. The abstraction-specific code inside the `code` variable is the Logical Neighborhood [5] custom language. This is used to create an instance of *target*, referring to local actions to retrieve the function and type of node to possibly include in the target. The *target* is given as parameter to a `setTarget` method invoked on an instance of *report*. The remaining method invocations are used to set parameters for the functioning of the `Stream` instance, e.g., its reporting period.

The BPMN model also contains performance annotations. Based on this and monitoring data, the self-optimization functionality tunes the protocols' parameters, e.g., by going into a very low power mode when no meeting is scheduled and no presence of people has been detected.

VI. Conclusion

We presented early results of the make*Sense* project, which tackles the unification of existing WSN programming abstractions and the integration of WSNs with business process models and back-ends. These issues are hampering industrial WSN adoption, thus, we believe that make*Sense* will foster adoption of WSNs in industry applications.

Acknowledgments

This work is supported by the European Commission through the make*Sense* and CONET projects.

References

[1] L. Mottola and G. Picco, "Programming Wireless Sensor Networks: Fundamental Concepts and State of the Art," *ACM Computing Surveys*, vol. 43, no. 3, 2011.

[2] C. Decker, T. Riedel, M. Beigl, L. de Souza, P. Spiess, J. Muller, and S. Haller, "Collaborative business items," in *IET Int. Conf. on Intelligent Environments*, 2007.

[3] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-based Internet of Things: Discovery, query, selection, and on-demand provisioning of Web services," *IEEE Trans. on Service Computing*, vol. 3, no. 3, 2010.

[4] K. Römer and J. Ma, "PDA: Passive distributed assertions for sensor networks," in *Proc. of the Int. Conf. on Information Processing in Sensor Networks (IPSN)*, 2009.

[5] L. Mottola and G. Picco, "Logical Neighborhoods: A Programming Abstraction for Wireless Sensor Networks," in *Proc. of the Int. Conf. on Distributed Computing in Sensor Systems (DCOSS)*, 2006.

7.4 Paper D: makeSense: Real-world Business Processes Through Wireless Sensor Networks

Florian Daniel, Joakim Eriksson, Niclas Finne, Harald Fuchs, Andrea Gaglione, Stamatis Karnouskos, Patricio Moreno Montero, Luca Mottola, Felix Jonathan Oppermann, Gian Pietro Picco, Kay Römer, Patrik Spieß, Stefano Tranquillini, and Thiemo Voigt (2013). "makeSense: Real-world Business Processes through Wireless Sensor Networks." In: *Proceedings of 4th International Workshop on Networks of Cooperating Objects for Smart Cities.* (Philadelphia, PA, USA), pp. 58–72.

URL: `http://ceur-ws.org/Vol-1002/`

7.4.1 Summary

WSNs have been a promising technology for quite some time, but their success is largely limited to scientific applications, such as environmental monitoring. To increase their adoption within industrial applications, it is necessary to integrate WSNs with business process modeling and existing back-ends. In addition, the abstraction level of WSN programming needs to be raised significantly. The `make`*Sense* project developed a unifying WSN programming framework that generates deployment-ready WSN software from high-level BPMN business process models. This paper describes the complete WSN application design and maintenance process developed within the `make`*Sense* project including the required software artifacts and presents its adoption to an exemplary use-case scenario.

7.4.2 Contributions

Besides contributing individual ideas and solutions to most components of the described system, I am a co-author of Section 4 and the the sole author of Section 5 of this paper. The paper was presented by Thiemo Voigt, Kay Römer, and Luca Mottola at the 4th International Workshop on Networks of Cooperating Objects for Smart Cities, in Philadelphia, PA, USA.

makeSense: Real-world Business Processes through Wireless Sensor Networks

Florian Daniel[3], Joakim Eriksson[1], Niclas Finne[1], Harald Fuchs[4], Andrea Gaglione[3], Stamatis Karnouskos[4], Patricio Moreno Montero[5], Luca Mottola[1], Nina Oertel[4], Felix Jonathan Oppermann[2], Gian Pietro Picco[3], Kay Römer[2], Patrik Spieß[4], Stefano Tranquillini[3], and Thiemo Voigt[1]

[1] SICS Swedish ICT, Kista, Sweden
[2] University of Lübeck, Lübeck, Germany
[3] University of Trento, Italy
[4] SAP AG, Germany
[5] Acciona Infraestructuras S.A., Spain

Abstract Wireless sensor networks (WSNs) have been a promising technology for quite some time. Their success stories are, however, restricted to environmental monitoring. In the industrial domain, their adoption has been hampered by two main factors. First, there is a lack of integration of WSNs with business process modeling languages and back-ends. Second, programming WSNs is still challenging as it is mainly performed at the operating system level. To this end, we provide the make*Sense* framework, a unified programming framework and a compilation chain that, from high-level business process specifications, generates code ready for deployment on WSN nodes. In this paper, we present the make*Sense* framework and the application scenario for our final deployment.

1 Introduction

Wireless sensor networks (WSN) are small, untethered computing devices equipped with embedded sensors and actuators. WSNs can be deployed much more easily than traditional wired sensors, and are able to coordinate and self-organize so that some high-level application goal is achieved. Many of the early sensor network deployments involved only sensors and realized environmental monitoring applications, that reported aggregated data to a base station [1]. While sensor networks have been successful in this domain, in other domains their adoption has been rather limited.

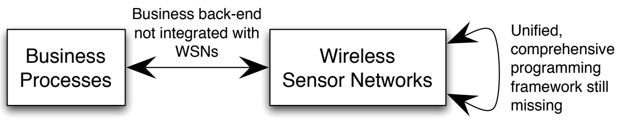
Business back-end not integrated with WSNs
Business Processes
Wireless Sensor Networks
Unified, comprehensive programming framework still missing

Figure 1. Open problems for using WSNs in business processes.

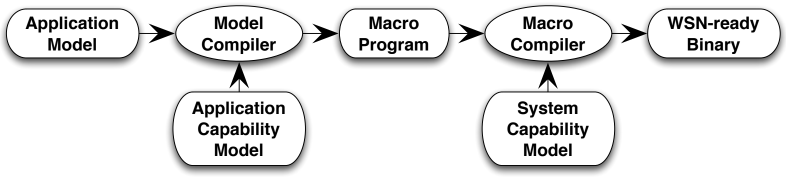

Figure 2. Compiling business process models into WSN-executable code.

As shown in Figure 1, we see two limiting factors that enable widespread adoption of sensor networks: *(i)* there is a lack of *integration* of WSNs with business process modeling languages and back-ends; *(ii)* programming WSNs is still challenging as it is mainly performed at the operating system level since there is no *unifying comprehensive programming* framework.

In the make*Sense* project [2], we tackle these two issues. We tackle the problem of integration by providing a holistic approach where application developers "think" at the high abstraction level of business processes, but the constructs they use are effectively implemented in the challenging reality of WSNs. Concretely, we let the application developers specify the application in a WSN-specific extension for Business Process Modeling Notation (BPMN). A model compiler transforms the extended BPMN models into traditional and WSN-specific code which allows to distribute process execution over both a WSN and a standard business process engine.

To simplify WSN programming, many programming abstractions have been developed [3], but they are hard to use since they typically focus on one specific problem. To drastically simplify WSN programming, particularly for business scenarios, we provide a broader approach that enables developers to use several abstractions at once. Towards this end, we present a unified comprehensive programming framework into which existing WSN programming abstractions can blend smoothly. These abstractions are "glued" together using a core language, a stripped-down version of Java tailored for WSNs. This macro-programming language is also the target language of the model compiler mentioned above. It can, however, also be used directly by WSN programmers. A macro compiler takes the macro-programming code as input and compiles it down to plain Contiki code that can be executed on WSN nodes or on the gateway between the sensor network and the business process engines.

Effectively, this leads to two compilation steps as shown in Figure 2. The model compiler takes as input the application model (in extended BPMN) and an application capability model. The latter is a coarse-grained description of the WSN, providing information such as the type of sensors/actuators available and their operations. The macro compiler takes as input the macro-program generated by the model compiler and a system capability model. The latter provides finer-grained information on the deployment environment (e.g., how many sensors of a given type are deployed at a location). The macro-compiler generates executable code that relies only on the basic functionality provided by the run-time support available on the target nodes. By leveraging the system

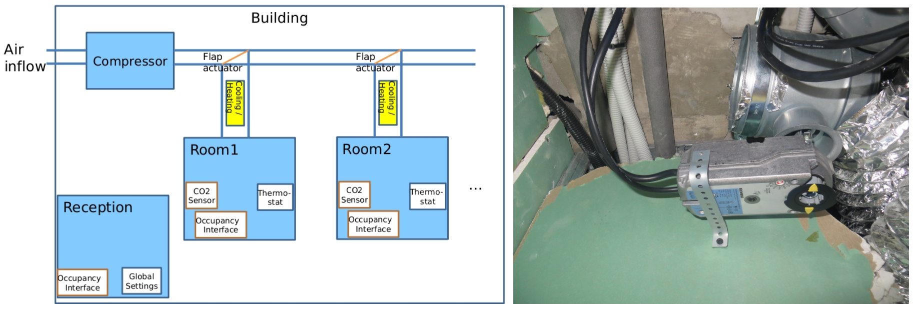

Figure 3. Deployment scenario for make*Sense* final deployment. An overview of the scenario (left part), and the actuator with the flap (right part).

capability model, the macro compiler can generate different code for different nodes, based on their application role.

As described in Section 6, the executable code runs atop a dedicated run-time layer, which provides access to low-level functionality such as MAC protocols and sensor devices. The run-time system also contains mechanisms enabling self-optimization of the network functionality, also described in Section 6.

The paper proceeds as follows. In the next section, we briefly present our deployment scenario. In Section 3 we discuss make*Sense* application modelling. The subsequent sections present the make*Sense* macro-programming language and the macro-compiler. We give an overview on the make*Sense* run-time system in Section 6 and the conclude with some final remarks.

2 Deployment

The make*Sense* project's deployment is in a student residence in Cadiz, Spain. As shown in Figure 3 we implement a room ventilation scenario, where the actuator (right part of the same figure) opens the flap in the student's bathroom if the measurement of the CO_2 sensor is above a configurable threshold. The external business process is managed by a room reservation system. We use an open source reservation system to manage room reservations that interacts with the sensor network through an occupancy interface. Hence, the sensor network can save energy by not ventilating rooms when they are vacant.

3 Application Modeling

For the integration of WSNs with business processes, we do not just add a service facade to the WSN or deploy middleware components on the gateway as others have done [4]. Instead, we want to enable a process modeler to model processes that are partially executed directly by the WSN itself and partially by traditional business process execution engines. Towards this end, we use and extend the *Business Process Modeling Notation (BPMN)*. We introduce new attributes that allow the modeler to specify a new *intra-WSN participant* that contains the logic executed by the WSN. Since the latter is resource-constrained we allow only a subset of BPMN elements. Moreover, we introduce a special *WSN activity* type to be used within the *intra-WSN participant*. The WSN activity is (except for the message activity) the only allowed activity type there.

The WSN activity is backed by a meta-model that we describe in the next section. As WSNs are inherently distributed systems, we also introduce a *Target* attribute for lanes and activities within the *intra-WSN participant*, that allows specifying where the respective logic should be executed, based on labels that are relevant at the modeling layer. Finally, we add performance annotations, expressing that the WSN should optimize its operation for a specific goal (e.g., system lifetime or reliability) within a certain subsets of activities. This is used for the self-optimization in the run-time system as described in Section 6.

To assist the process modeler in creating correct, executable models, we use a set of meta-models that describe the WSN in terms of the logical functionality it provides, along with the way it is embedded into the physical set-up (e.g., which sensing or actuation is supported at which logical location). Instances of these meta-models can be created either manually or through dynamic service discovery.

At run-time, the BPMN process is executed in a distributed fashion. To execute the intra-WSN process in the WSN, it is entirely transformed into macro-code, compiled into C, and distributed by the run-time as described in the next sections. For message exchange between the intra-WSN process and the other process, the run-time uses a lightweight protocol, reducing encoded message size by using message structure information on both sides. The compilation step automatically generates process communication endpoints that handle serialization and deserialization of messages and implement process instance correlation.

4 Macro-programming Language

To bridge the gap between business processes and WSNs we defined a high level intermediate macro-programming language where the abstractions contributing to the language are decoupled, leverage on existing implementation, and can be changed or extended easily to suit specific application needs.

The make*Sense* macro-programming language is based on a core set of *meta-abstractions* which define the fundamental building blocks of the language as units of functionality, reuse, and extensions. They are implemented through different “concrete” abstractions and provide the key concepts enabling interaction with the WSN. The language serves as the “glue” among abstractions, whose composition can be achieved by using common control flow statements. The core language, in our case a stripped-down version of Java we tailored for WSNs, is also the *trait d’union* between the macro-programming abstractions and the BPMN business process model.

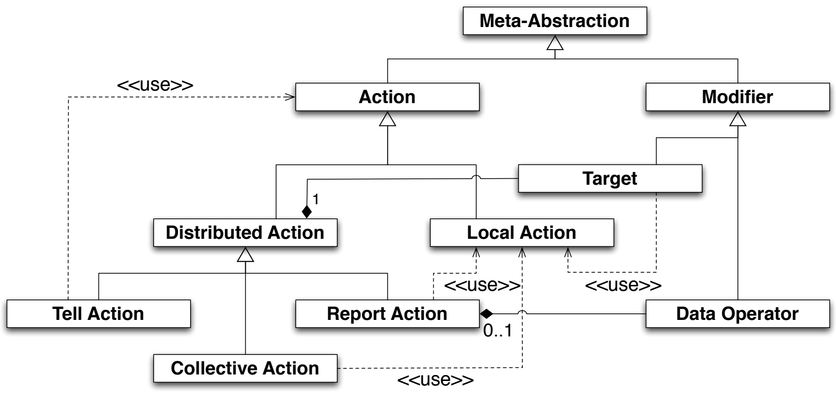

Figure 4. A model for the meta-abstractions of the make*Sense* macro-programming language.

Figure 4 shows a UML meta-model for the meta-abstractions provided by the macro-programming language. It focuses on the notion of *action*, a task executed by one or more WSN nodes. Actions are separated into *local*, whose effect is limited to the node where the action is invoked (e.g., acquiring a reading from the on-board temperature sensor), and *distributed*, whose effect instead spans multiple nodes.

Distributed actions may run on several nodes in parallel and are further divided into *tell*, *report*, and *collective* actions. The former two represent the one-to-many and many-to-one interaction patterns commonly used in WSNs to enable communication between the node (the “one”) issuing the action and a set of nodes (the “many”) where the latter is executed. A tell action enables a node to request the execution of a set of actions on other nodes, e.g., to issue actuation commands or to trigger reconfiguration of system parameters such as the sampling rate. A report action enables a node to gather data from other nodes. Event-based abstractions and periodic, continuous queries both fall in this category. Data acquisition occurring on each target node is specified by a local action given as input to the report action. The output of the local action is returned to the report one. Collective actions, in contrast to tell and report ones, do *not* focus on a special node where the action starts or ends. They enable a global network behavior and are executed cooperatively by the entire WSN through many-to-many communication. An example are distributed assertions [5], where programmers specify a (global) property monitored collectively by the WSN nodes.

The behavior of distributed actions can be customized by a *modifier*. We defined two modifiers, *target* and *data operator*. In our envisioned scenarios the nodes possibly differ along several dimensions, both physical and logical. For example, the ventilation scenario of our deployment in Section 2 requires both CO_2 sensors and flap actuators to be installed in two different rooms. Programmers must be able to map actions to the set of nodes of interest. A target identifies a set of nodes satisfying application constraints, and gives the ability to apply a distributed action to the nodes in this set. Instead, a report action may have a *data operator*, specifying processing performed on the results after gathering and before they are returned to the caller, e.g., to filter or aggregate the data.

General concepts and operations defined by meta-abstractions are implemented by concrete abstractions, which may then provide different levels of expressiveness and run-time guarantees. To create an instance of a meta-abstraction, a class implementing its interface must be defined in the core language. As abstraction implementations typically closely interact with the operating system, methods of abstraction classes are implemented in C using a native code interface provided by the core language. Some abstractions require extensive configuration, for example, a target needs to define a set of nodes based on their properties [6]. To simplify such configuration, the core language supports the concept of *embedded languages*, code snippets formulated in the declarative configuration language provided by an abstraction. These are efficiently compiled by appropriate compiler plugins, instead of being interpreted at runtime.

The `make`*`Sense`* macro-programming core language provides a framework to integrate the previously described abstractions. In the `make`*`Sense`* framework it mainly serves as an intermediate language for the translation of BPMN models to platform code, but it is also suitable for direct use by programmers. The core language features a Java-like syntax and full support for object-oriented programming. In addition, to make the programmer's task easier, we decided to provide full multi-threading with a Java-like interface based on the Contiki `mt` library [7]. Nevertheless, as we are targeting very resource-constrained microcontrollers, the language needs to be simpler than standard Java. Consequently, some language features had to be removed. For example, the `make`*`Sense`* macro-programming language does not provide garbage collection, but relies on manual memory management. To reduce the resulting burden on the programmer, the language also provides specific constructs to allocate automatic or static objects, for which the memory management is handled by the compiler. In contrast to Java we do not employ a virtual machine approach, but the program is translated to target code that can be directly run on the target platform. The resulting code is predeployed on all nodes, so that it is not necessary to migrate code fragments at run-time.

Abstractions are represented in the language as ordinary classes with a predefined interface. Some abstractions require extensive configuration, for example in order to specify the set of nodes that form a target. To facilitate such configurations the macro-programming language features an extension mechanism that allows to embed abstraction-specific languages in the macro-programming code. This mechanism relies on specific compiler plug-ins as described in Sec. 5. Listing 1.1 demonstrates the use of embedded code to specify a logical neighborhood [6] to limit the scope of a stream action to this set of nodes. In lines 1 to 6 the logical neighborhood is defined by an abstraction-specific code fragment and the definition is assigned to a code-type variable neighborhoodDef. This variable is used in line 8 to associate the neighborhood definition with a new instance of the logical neighborhood abstraction. Note the use of the newly introduced

Listing 1.1. Use of embedded code in the MPL core language

```
 1  code neighborhoodDef = {:
 2    neighborhood co2Sensors() {
 3      ACM.getFunction() == "sensor"
 4          and ACM.getType() == "co2"
 5    }
 6  :};
 7
 8  Target co2Sensors = lnew LN(neighborhoodDef);
 9
10  Report co2Stream = lnew Stream();
11
12  co2stream.setTarget(co2Sensors);
13  co2Stream.setAction(lnew ReadCO2Level());
14  co2Stream.setDataOperator(lnew MedianOperator());
15  co2Stream.setParameter("period", 5 * 60);
16
17  co2Stream.execute();
18
19  co2Stream.waitResult();
20
21  Object result = co2Stream.getResult();
```

`lnew` operator to create an automatic object instance for which memory management is handled by the compiler. In line 12, the neighborhood is assigned as target scope to a newly created stream action. In the following lines, additional parameters are set and finally the action is executed in line 17. After execution of the action, the program needs to wait until a result and can be fetched.

Another significant feature of the make*Sense* macroprogramming language is the provision of a generic object serialization interface. This feature is primarily used by the different abstractions in order to transfer object state between the involved nodes. The object serialization facility is similar to the one provided by Java and allows to write the state of an object to a standardized flat representation. This representation is, for example, suitable to be send over the network and can later be used to recreate an exact copy of the serialized object on the same or a different node. To be applicable for serialization, a class needs to implement the predefined interface `Serializable`. The serialization and deserialization functionality is automatically generated by the macro-compiler, but can be customized by overriding specific methods.

5 Macro-compiler

The make*Sense* macro-compiler is responsible for the translation of the macroprogramming language program to Contiki-based C code. The generated C code can than be compiled with the existing Contiki tool chain and can be finally deployed on the nodes.

The basic architecture of the compiler follows the established reference architecture. As shown in Fig. 5, the compilation process consists of four major phases: scanning and parsing, semantic analysis, target code generation, and code partitioning. To support different platforms, like Contiki and TinyOS, it is

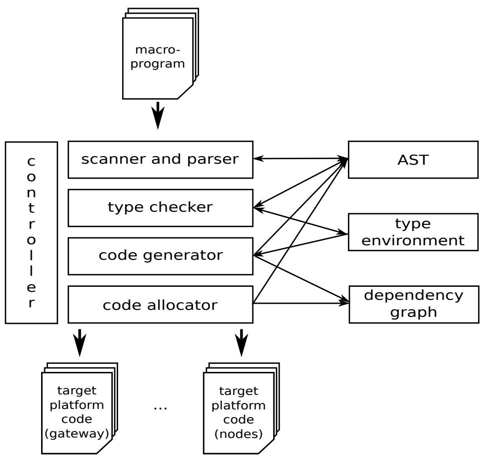

Figure 5. Architecture of the make*Sense* macro-compiler

possible to replace the generation back end, but the currently implementation only supports Contiki. In the non-standard final code partitioning phase, the compiler determines which translated classes need to be deployed at a specific node class based on a previously established dependency graph and a data flow analysis for the program. The goal of this phase is to remove unneeded code from specific program images. To reduce the size of the deployed program image, the single macro-program specifying the behavior of the whole network is partitioned into node-specific program parts. Each segment only contains those classes that are potentially executed on the nodes belonging to the respective class. For example, it is not necessary to provision program code for actuator control on pure sensor nodes. In the current implementation, we only differentiate between regular nodes and a dedicated gateway, but this concept can be easily extended to a larger number of node classes.

To enable the embedded code introduced in Sec. 4 the macro-compiler exhibits a plug-in interface that allows to integrate small sub-compilers for the abstraction-specific languages. Each of these plug-ins is responsible for parsing, type checking, and translation of the respective code fragments. As shown in Fig. 6, the plug-ins are automatically invoked by the main compiler, if it encounters an embedded code fragment in the macroprogramming code. A return channel allows the plug-ins to inform the compiler about references to macroprogramming language constructs encountered in the embedded code fragments. Like the macro-compiler, the plug-ins are implemented in Java.

6 Run-time System

Figure 7 shows the high-level architecture of the make*Sense* run-time system. The business process execution engine connects to the sensor network through a dedicated gateway we design. Application performance requirements are specified in

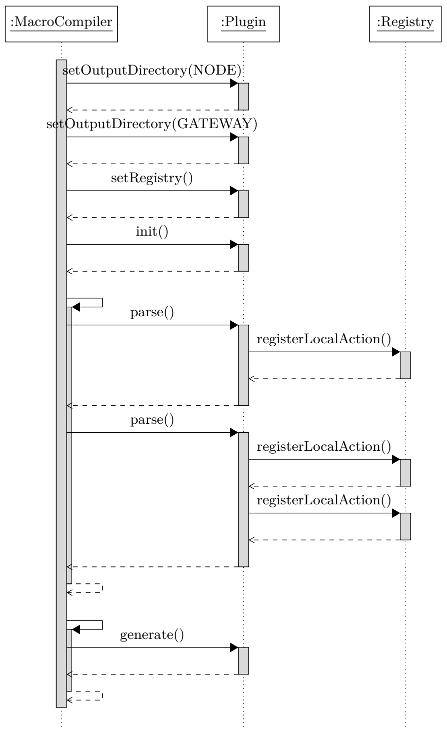

Figure 6. Typical communication sequence of `make`*Sense* macro-compiler plug-in.

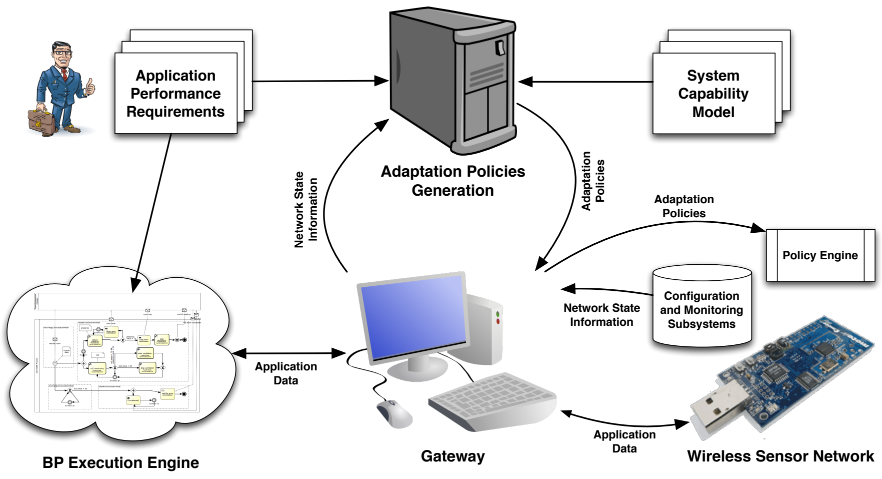
Application Performance Requirements
System Capability Model
Adaptation Policies Generation
Network State Information
Adaptation Policies
Adaptation Policies
Policy Engine
Configuration and Monitoring Subsystems
Network State Information
Application Data
Application Data
BP Execution Engine
Gateway
Wireless Sensor Network

Figure 7. `make`*Sense* run-time architecture.

the extended business processes. These are taken as input by a dedicated optimization engine that generates *self-optimization* policies that allows the network to dynamically tune its behavior. The latter task is carried out based on information from the system capability model and network state information from the deployed network. On the sensor nodes we deploy a dedicated configuration and monitoring subsystem that oversees the application execution inside the sensor network and executes the adaptation policies depending on the observed state.

While the make*Sense* gateway is implemented with mainstream technology as it is intended to run on a standard machine, the key functionality of the make*Sense* run-time system lies within the configuration and monitoring subsystem aboard the sensor nodes and in the generation of self-optimization policies. We describe these mechanisms next.

6.1 Monitoring and Configuration

The key design principle of the configuration and monitoring subsystem is to separate protocol logic from configuration [8]. This way, parameters in all parts of the system can be configured through a separate configuration component based on the settings that the self-optimization policies dictate. This makes it simple to handle changes in the objectives of the application, e.g., when the application demands a new objective such as high throughput instead of low energy consumption. Furthermore, we aim at keeping a layered design to make it possible to exchange layers, for example, when a new MAC layer should be used. While researchers have argued that cross-layering is required in wireless sensor networks to achieve high performance, we showed that we can both rely on a layered system and achieve high throughput [8].

In designing the configuration and monitoring functionality, we wish to lessen the burden on developers of configuration policies due to gathering and processing the data input to the self-optimization mechanism. To this end, we opt for a unified tuple space-like API spanning both read and write operations on the local blackboard, and distributed operations to share the configuration and monitoring information across 1-hop neighboring devices [9]. We also aim at a design that has clearer boundaries and hence requires little re-engineering work when new Contiki releases are available. Therefore, we use wrappers between Contiki components, e.g., the MAC protocol, and our configuration run-time.

As shown in Figure 8, the configuration and monitoring subsystem includes a central black-board for storage of configuration parameters, system state, and statistics. The other modules access the blackboard storage via tuple space-like APIs [9] that operate on the relevant data. These APIs can both operate on local data and on the blackboard of the one-hop neighbors. make*Sense* modules handle their configuration directly via the blackboard while non-make*Sense* modules, such as Contiki components, are wrapped so that relevant configuration and state can be stored in the blackboard. The monitoring modules are responsible for acquiring information on performance and resource consumption, storing it in the blackboard to make it available to upper layers.

The configuration policy and policy engine are responsible for setting the performance-related parameters. They also provide the interface to the optimizer

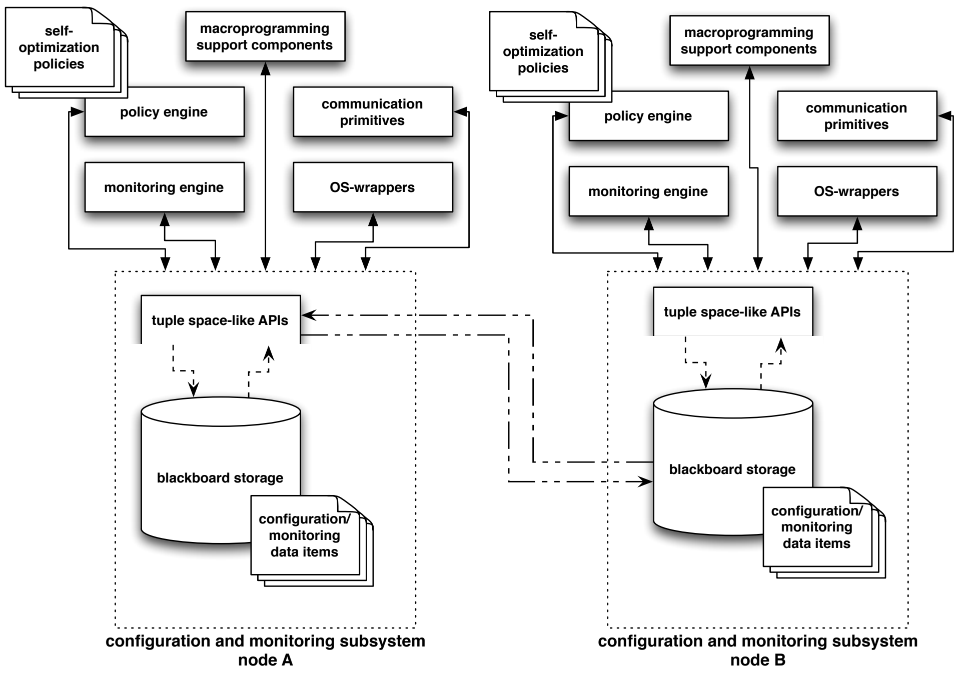
self-optimization policies
macroprogramming support components
policy engine
communication primitives
monitoring engine
OS-wrappers
tuple space-like APIs
blackboard storage
configuration/ monitoring data items
configuration and monitoring subsystem node A
self-optimization policies
macroprogramming support components
policy engine
communication primitives
monitoring engine
OS-wrappers
tuple space-like APIs
blackboard storage
configuration/ monitoring data items
configuration and monitoring subsystem node B

Figure 8. Overview of the configuration and monitoring subsystem

that runs outside the network and is in charge of optimizing the performance, as described next. The policy engine enforces these policies by setting appropriate parameters in the blackboard that determine the corresponding modules' behavior and performance. As any initial configuration is likely to be sub-optimal, the optimizer will dynamically update the configuration. Dynamic updates might also be required when the radio environment changes. For example, when it becomes more difficult to deliver packets due to interference, the optimizer might decide to increase the maximum number of retransmissions.

6.2 Self-optimization

In several real-world deployments the application and operating system code are finely-tuned to achieve a certain performance goal [10]. Most often, this is based on the developers' intimate knowledge of the internal sensor networks mechanisms and a deep understanding of the application requirements. The deployed code is also entirely in the hands of the same developers, who are free to modify and tune the implementations depending on the performance goals.

In general, the approach above is not possible in make*Sense*. Two main reasons concur to this: *i)* the executable code is generated from high-level application models, and the mapping from the latter to low-level Contiki C is not trivial; and *ii)* the programming framework is open to external developers, who may contribute new concrete abstractions along with their supporting run-time. Furthermore, make*Sense* allows application developers to specify performance objectives that can change at the run-time. This is necessary to support long-lasting business processes subject to real world interactions and rapidly changing requirements. Therefore, the make*Sense* run-time must be able to *self-optimize* towards the stated performance objectives.

We define *self-optimization* as the property of a system to automatically find near-optimal system configurations whenever application objectives, system parameters, or environmental conditions change. To enable self-optimization, we gather run-time information from the deployed sensor network, e.g., network topology and protocol performance, and feed these to a reinforcement learning algorithm that explores the space of possible configurations using simulations. At the end of each simulation round, the learning process evaluates the performance obtained with a given setting w.r.t. the application's performance goals. Based on this, we derive self-optimization policies that specify which parameters provide better performance as a function of the current application and environment state, including the performance goal. We distribute the policies back to the deployed network where nodes will apply them whenever needed.

This approach sharply differentiates from existing solutions. Rather than requiring detailed modeling of the individual protocols, as done for example with great effort for MAC protocols [11], we treat the entire application as a black-box. This may lead to sub-optimal solutions, but also enjoys greater flexibility as it lets users add programming abstractions to the framework along with their supporting protocols and have the latter "implicitly" optimized.

We describe next the key aspects of the self-optimization functionality

Off-line learning. A typical make*Sense* application will have several modes of operation, along with different performance objectives. The same application can, for example, have energy efficiency as the major objective for most of the time, but in some emergency situations switch to latency. This means that there is no static system configuration that is optimal at all times. The configuration needs to be adapted as soon as the objective changes.

The approach taken for adapting configurations is to have a simulation framework that can simulate the set-up of a specific make*Sense* application. The simulation is fed with the applications network topology, the sensor network nodes firmwares being used, and network state information from the deployed system. The simulation is then run together with learning mechanisms that tune the configuration while simulating the application scenario. During the simulation, the learning mechanism will evaluate the performance given the application objectives. We choose to use a reinforcement learning based approach for the learning mechanism. As shown later, initial results demonstrate that this is a promising approach.

State monitoring. The nodes need to monitor their internal state to adapt their configuration. This state is an important part of the input to the configuration policies and includes, as a function of the needed policy, information such as density, network congestion, and energy levels. The decision on what is needed is partly set by what information is relevant to the performance objectives. Monitoring the local state is needed to allow the performance goals to change over time because the nodes only adapt their configuration based on what they know in their local state. To change the performance objectives during run-time, the relevant parameters need to be updated in the nodes local state.

The selection of what should be included in the nodes' local states is important. Including too many parameters increases the time required to learn configuration policies and also increases energy consumption for parameters requiring active monitoring. Including too few parameters, on the other hand, makes it difficult to find reliable configuration policies because they might not have enough information.

Learning. In its simplest form, a policy is a mapping between a state and a set of actions that should be performed when the application is in this state. An action in this case can be a value to update in the blackboard that triggers a reconfiguration.

We are using a reinforcement learning based approach for the process of learning policies. The learning is performed during simulation using a plug-in for the Cooja simulator. A utility function based on the performance objectives provides the reinforcement learning with the needed rewards to implement the learning process. The specific learning mechanism that we use is First Visit Monte Carlo Policy Iteration [12]. We use the Cooja simulator as it allows to to accurately emulate sensor nodes such as TMote Sky and Wismote. This makes it possible to reuse the firmwares that are executed on the real sensor network in the simulator, making the simulation behavior as realistic as possible.

To automate the learning process in Cooja, we design and implement a new extension for the simulator that is able to run multiple simulation rounds of the same scenario. This extension uses the same simulation configuration files as Cooja and after the regular simulation it restarts the scenario at fixed time intervals. Before resetting the scenario the learning process takes place.

Initial results. We run experiments to assess the ability of the self-optimization framework to dynamically identify policies that improve the resulting system performance. We consider as example the following performance objective, formulated as a linear combination of desired reliability, goodput, and energy consumption:

$$utility = \frac{received}{sent} * 25.0 + received * 100.0 - 0.07 * energy \qquad (1)$$

Figure 9 reports a screenshot of the reinforcement learning simulation framework while optimizing for the performance objective above. In the figure, stream denotes the goodput in received packets per learning period. Throughout different simulation runs, the learning algorithms understands that a way to maximize the value of (1) is to favor packet transmissions (denoted as stream in the figure) even though they lead to slightly higher energy consumption. This is a

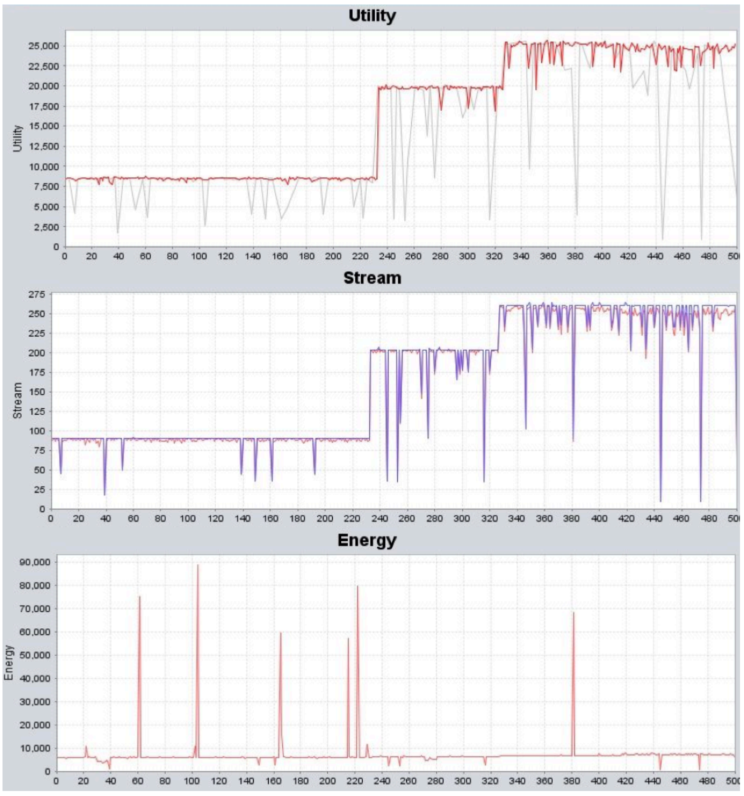

Figure 9. The utility improves significantly over time (top); the learning algorithm detects that it is better to send many messages, improving goodput (middle), even if the energy cost slightly increases (bottom).

direct result of the objective formulation, which poses the largest weight on the goodput.

7 Conclusions

In this paper, we have presented the `make`*Sense* approach for generating sensor networking code from business process models. Our approach integrates business processes with sensor networks in a novel way. Through a compilation chain an application models specified in slightly extended BPMN is transformed to both code that runs in the sensor network and code that is executed by traditional business process engines. We have also presented our final application scenario we are currently deploying in a student residence in Spain.

Acknowledgments. The work leading to these results has received funding from the European Union Seventh Framework Programme (FP7-ICT-2009-5) under grant agreement n° 258351 (`make`*Sense*).

References

1. A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *First ACM Workshop on Wireless Sensor Networks and Applications (WSNA 2002)*, Atlanta, GA, USA, September 2002.
2. F. Casati and F. Daniel and G. Dantchev and and J. Eriksson and N. Finne and S. Karnouskos and P. Moreno Montera and L. Mottola and F. Oppermann and G.P. Picco and A. Quartulliz and K. Römer and P. Spiess and S. Tranquilliniz, and T. Voigt. Towards business processes orchestrating the physical enterprise with wireless sensor networks. In *NIER track, 34th International Conference on Software Engineering (ICSE)*, pages 1357–1360, Zurich, Switzerland, June 2012.
3. L. Mottola and G.P. Picco. Programming Wireless Sensor Networks: Fundamental Concepts and State of the Art. *ACM Computing Surveys*, 43(3), 2011.
4. D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio. Interacting with the SOA-based Internet of Things: Discovery, query, selection, and on-demand provisioning of Web services. *IEEE Trans. on Service Computing*, 3(3), 2010.
5. Kay Römer and Junyan Ma. PDA: Passive distributed assertions for sensor networks. In *Proc. of the Int. Conf. on Information Processing in Sensor Networks (IPSN)*, 2009.
6. L. Mottola and G. Picco. Logical Neighborhoods: A Programming Abstraction for Wireless Sensor Networks. In *Proc. of the Int. Conf. on Distributed Computing in Sensor Systems (DCOSS)*, 2006.
7. A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proc. of the Workshop on Embedded Networked Sensor Systems (Emnets)*, 2004.
8. N. Finne, J. Eriksson, N. Tsiftes, A. Dunkels, and T. Voigt. Improving sensornet performance by separating system configuration from system logic. In *Proceedings of the Sixth European Conference on Wireless Sensor Networks (EWSN2010)*, Coimbra, Portugal, 2010.
9. P. Costa, L. Mottola, A. L. Murphy, and G. P. Picco. Programming Wireless Sensor Networks with the TeenyLIME Middleware. In *Proc. of the* 8^{th} *ACM/USENIX Int. Middleware Conf.*, 2007.
10. M. Ceriotti, L. Mottola, G. P. Picco, A. Murphy, S. Guna, M. Corra, M. Pozzi, D. Zonta, and P. Zanon. Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment. In *Proceedings of the International Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*, pages 277–288, Washington, DC, USA, 2009. IEEE Computer Society.
11. M. Zimmerling, Federico Ferrari, Luca Mottola, Thiemo Voigt, and Lothar Thiele. pTunes: runtime parameter adaptation for low-power MAC protocols. In *ACM/IEEE Int. Conference on Information Processing in Sensor Networks (IPSN)*, 2012.
12. R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction.* The MIT press, 1998.

7.5 Paper E: Design and Compilation of an Object-Oriented Macroprogramming Language for Wireless Sensor Networks

Felix Jonathan Oppermann, Kay Römer, Luca Mottola, Gian Pietro Picco, and Andrea Gaglione (2014b). "Design and Compilation of an Object-Oriented Macro-programming Language for Wireless Sensor Networks." In: *Workshop Proceedings of the 39th IEEE Conference on Local Computer Networks (LCN).* (Edmonton, AB, Canada). Piscataway, NJ, USA: IEEE, pp. 574–582. ISBN: 978-1-4799-3782-0. DOI: 10.1109/LCNW.2014.6927705.

URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6927705

The paper was nominated for the best paper award of the SenseApp workshop.

7.5.1 Summary

This paper introduces a novel macroprogramming framework for WSNs. The framework is based on the previously developed conceptual framework that classifies programming abstractions into few groups and allows the composition of multiple abstractions into a complete application. Based on this framework we developed an object-oriented macroprogramming language modeled after the common Java programming language. Compared to Java, the language has been modified to allow efficient execution on resource-constrained microcontrollers and to support the above conceptual framework. The structure of a compiler that translates this language into C code is also described.

7.5.2 Contributions

I am the main author of the paper, implemented the described software system and carried out the described experiments. The meta-abstraction concept was developed as shared work within discussions among the co-authors. My work also incorporates feedback and ideas from Kay Römer. I presented the work at the 9th IEEE International Workshop on Practical Issues in Building Sensor Network Applications (IEEE SenseApp 2014) in conjunction with the 39th IEEE Conference on Local Computer Networks (LCN 2014) in Edmonton, Canada.

Design and Compilation of an Object-Oriented Macroprogramming Language for Wireless Sensor Networks

Felix Jonathan Oppermann*, Kay Römer*, Luca Mottola†‡, Gian Pietro Picco§, Andrea Gaglione¶
*Graz University of Technology, Austria
Email: {oppermann,roemer}@tugraz.at
†Politecnico di Milano, Italy
‡SICS Swedish ICT
§University of Trento, Italy
¶Imperial College London, UK

***Abstract*—Wireless sensor network (WSN) programming is still largely performed by experts in a node-centric way using low-level languages such as C. Although numerous higher-level abstractions exist, each simplifying a specific aspect of distributed programming, real applications often require to combine multiple abstractions into a single program. Using current programming frameworks, this represents a difficult task. In previous work, we therefore defined a conceptual framework that facilitates abstraction composition by defining sound compositional rules among few fundamental abstraction categories. The framework is extensible: programmers can add new abstractions within the boundaries determined by the compositional rules.**

In this paper we describe the design of a language—called MPL—that instantiates this conceptual framework. To support the extensible nature of the framework, the language is object-oriented, which allows programmers to add new abstractions by inheriting from existing classes that implement predefined interfaces. We modeled the syntax after Java, to make it more palatable to inexperienced embedded programmers. Compared to Java, we modified the language to enable efficient execution on WSN devices. We designed and implemented a compiler that translates MPL language into executable C code, which spares the overhead of a virtual machine. By comparing MPL implementations against functionally-equivalent Contiki/C implementations of several benchmark applications, we determined that the performance overhead of MPL is limited, and yet the programming task is simplified.

***Index Terms*—Compilers, Java, object-oriented languages, wireless sensor networks.**

I. Introduction

During the last years, the use of wireless sensor networks (WSNs) significantly increased [14] while WSNs also started to make their way into commercial and industrial real-world applications. Nevertheless, a more widespread WSN adoption is still hampered by the unavailability of easy-to-use development tools. As of today, most WSN applications are still implemented in low-level C code and their design requires in-depth knowledge of the specifics of embedded systems and low-power wireless communication. Consequently, WSN programming is usually carried out by WSN experts. To gain more widespread use, WSN development needs to be more accessible to domain experts and programmers without a strong WSN background.

This requires a move from the still prevalent node-centric programming model towards a more holistic view of the network that hides low-level details. To this end, a growing number of macroprogramming abstractions have been designed that simplify programming of a specific distributed computing aspect (such as assigning roles to nodes [7] or defining a subset of nodes to communicate with [11]) by offering a domain-specific language. However, integrating multiple of these abstractions into a single program is still difficult. There also exist a number of macroprogramming languages that include a fixed set of abstractions, but new abstractions cannot be added easily.

In the make*Sense* [3] project we have therefore analyzed existing WSN abstractions, classified them based on few fundamental dimensions, and developed a conceptual framework that allows the composition of arbitrary abstractions according to predefined sound rules. The framework is also extensible in that abstractions that were not known by the time the framework was designed can be added later.

In this paper, we describe how this conceptual framework can be instantiated by means of a concrete language and we also describe a compiler to translate that language into efficient executable code. To sustain the extensibility of the conceptual framework, the language is object oriented and inspired by Java and thereby easy to use for programmers familiar with Java or C++. The language differs from Java to support efficient compilation to resource-constrained WSN devices and to support the extensibility of the above conceptual framework. The compiler offers a plug-in interface to support addition of new abstractions and automatically distributes application functionality among the gateway and the sensor nodes.

The increased abstraction level and the use of powerful programming abstractions enables a reduction of user-written code by more then 50% in comparison to an implementation in low-level code for a set of typical applications.

The remainder of the paper is organized as follows. Section II briefly reviews the current state of the art. Section III summarizes the make*Sense* abstraction framework. In Section IV we derive a set of requirements for a language implementing this framework. Section V introduces design decisions and implementation details that enable us to implement a language that is capable of meeting these requirements. Section VI introduces the architecture of the underlying compiler framework and provides a closer look to the plug-in interfaces that enable the required extensibility. Finally, the performance and overhead of the approach for a typical set of applications is evaluated and discussed in comparison with more conservative C-based implementations in Section VII.

II. Related Work

A multitude of different systems aims to raise the abstraction level of WSN programming by providing high-level macroprogramming frameworks. These systems either represent the WSN as a distributed database [10], or provide more sophisticated frameworks on-top of C [9] or with custom high-level languages [4], [13]. Database-like interfaces are usually limited to data collection applications, while more complex frameworks usually require the use of an unfamiliar language. Systems of both categories are usually monolithic and do not provide a well-defined interface to integrate application-specific abstractions. In this regard, MPL extends the state of the art by providing an extensible macroprogramming framework that supports in-network control logic and is based on Java, a widespread programming language.

Nevertheless, our macroprogramming language also differs from the standard Java ME [15] framework by targeting even more resource constrained devices. In addition, Java ME does not provide WSN-specific extension points to integrate existing concepts that abstract from typical WSN challenges such as communication and distributed data processing. In comparison to standard Java, WSN-specific extensions significantly increase the utility of our macroprogramming language. Some low-resource Java virtual machines also exist that are targeted at low-power embedded and networked systems [1], [2], [17], but they still require a comparatively significant amount of resources. For example, the Squawk virtual machine uses 80 kB of program memory and consequently targets more powerful ARM-based embedded platforms [18]. Our approach differs in that it generates customized C code which is in turn compiled into optimized machine code for the intended target platform hence reducing the introduced overhead.

III. The make*Sense* framework

This section describes the conceptual WSN abstraction framework developed in the make*Sense* project [3]. Previous research has demonstrated that a large number of WSN programming abstractions exist that solve common challenges of WSN programming [12]. WSN programming abstractions tackle issues such as node addressing, definition of communication patterns, and distributed data processing. A typical example of such programming abstractions is Logical Neighborhoods [11], a system that allows to define groups of nodes based on their state and to communicate with them in a way similar to sending broadcast messages to physical neighbors. Programming abstractions, like Logical Neighborhoods, already simplified WSN programming, but they were difficult to combine with other abstractions in a single program.

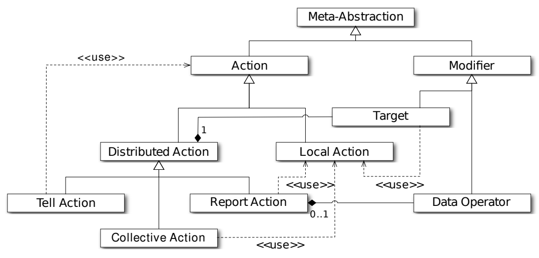

Fig. 1. The make*Sense* meta model for programming abstractions [3].

The make*Sense* project improved this situation by providing a unifying framework in which existing and future WSN programming abstractions can be easily integrated. To allow such extensibility, the make*Sense* framework employs the concept of meta-abstractions. WSN programming abstractions can be grouped into classes that aim to solve similar issues and that usually expose very similar interfaces. We call such a group of similar abstractions a "meta-abstraction", i.e., an abstraction of abstractions. Based on this insight, a hierarchy of typical WSN programming abstractions was developed in the make*Sense* project [3]. This hierarchy forms the basis of the make*Sense* framework as displayed in Fig. 1.

Two major types of *Meta-Abstraction* can be distinguished. *Actions* represent anything a node or a set of nodes can execute. This can be simple commands, like reading a sensor value, or more complex operations such as requesting aggregated values from a group of nodes. *Modifiers* allow a more precise specification of the behavior of Actions. For example, a Modifier could be used to specify which nodes should be part of the group that provides the aggregated value.

The Action class is further divided into two subclasses of actions. *Local Actions* are executed locally on a single node to implement basic operations, e.g., reading a sensor. In the make*Sense* framework, Local Actions also define the interface to the node hardware. Hardware operations, like reading sensor values or storing data on flash, are exposed to the user as Local Actions. In addition to predefined Local Actions, the user may also define custom Local Actions within the framework.

Distributed Actions can be used to request some action from multiple remote nodes. Distributed Actions typically define some form of communication between the nodes. The make*Sense* meta-abstraction hierarchy distinguishes between three basic communication patterns. The *Tell* abstraction is used for one-to-many communication. It allows to execute other Actions on a set of remote nodes. Dually, the *Report* abstraction is used for many-to-one communication, e.g., to request sensor values from a set of nodes. On the remote node, the requested data is extracted by employing a Local Action that has been associated with the Report. Finally, *Collective Actions* provide a default interface for abstractions that implement peer-to-peer coordination patterns, such as global assertions [5] or role assignment [7].

Programmers can customize an action's behavior by using Modifiers, e.g., to select the set of nodes that participate in a Report. This separation of concerns enables a more flexible interaction among abstractions. The `make`*Sense* framework provides two subtypes of modifiers. *Target* modifiers are used to implement the aforementioned selection of nodes, to limit the scope of operation of any Distributed Action. *Data Operators* can be used with Report actions to define additional data processing to be applied to the collected data, such as aggregation.

The different meta-abstractions define extension points that can be instantiated by concrete implementations of these abstraction types. For example, the Target meta-abstraction could be instantiated by Logical Neighborhoods or another group-defining programming abstraction. Several instances of the same abstraction may exist at a time and an application may employ different abstractions instantiating the same meta-abstraction concurrently.

IV. Language Requirements

To implement actual applications, several abstractions need to be combined and augmented with application-specific functionality. This requires a programming language that allows to define the interaction between individual programming abstractions and to implement algorithms for custom data processing. A number of fundamental requirements for such a language can be defined:

(I) The language needs to be suitable to reflect the previously described meta-abstraction hierarchy supporting also later addition of instances of the abstraction classes.
(II) As some programming abstractions employ sophisticated custom languages for their configuration, the language requires a convenient mechanism to integrate such domain-specific languages.
(III) Individual nodes may need to handle several remote actions at once. Consequently, the language needs to support concurrent execution of tasks.
(IV) The language needs to be expressive enough to implement moderately complex algorithms, it needs to support at least basic mathematical and logical operators, conditionals, and basic looping constructs.
(V) The programming language should be familiar and easy to use for a large number of programmers. It should be especially appealing for typical domain experts.
(VI) Last but not least, the language needs to be adequate to generate efficient code suitable for considerably resource-constrained devices, such as wireless sensor nodes.

V. Detailed Language Design

Based on these requirements and the `make`*Sense* framework, we designed MPL, a high-level macroprogramming language for WSNs. As determined by Requirements IV and V, a goal for this programming language is to provide an expressive programming environment that is familiar to a large set of programmers. This made Java [8] a natural choice as basis for the design of MPL as it is well-established and widely used language with appropriate features. Building upon Java's object-oriented model also provides a stepping stone to implement the `make`*Sense* meta-abstraction hierarchy in accordance with Requirement I. Each meta-abstraction maps to an interface and abstractions can be added by implementing a meta-abstraction interface. The language is purely object-oriented with the exception of a limited set of primitive data types, including integers, chars, and Booleans.

To make the language suitable for resource-constrained devices as demanded by Requirement VI, some limitations compared to standard Java were necessary. The most significant difference is the absence of a virtual machine. Programs are instead translated into C code targeted at the Contiki platform that can be further processed by the established tool chain to generate deployable binary images.

Support for object-orientation in MPL had to be implemented in C in a standard-compliant way relying on structures and function pointers. The approach taken is conceptually similar to the approach taken by C++ [19]. Most notably, dynamic dispatch is implemented with the help of virtual method tables instead of relying on the more flexible but also more memory-demanding hash-table-based approach typically found in Java implementations.

In the following, we highlight other important design decisions that enabled us to meet the aforementioned requirements in an efficient manner.

A. Memory Model

Most object-oriented languages primarily employ dynamic memory allocation for objects and even though it is convenient for the programmer—especially if supported by garbage collection—it is ill suited for memory-constrained devices and reduces the efficiency of the program.

Therefore, to meet Requirement VI, MPL encourages the use of static memory allocation. In contrast to Java, it is not only possible to allocate new objects on the heap, but instead the language also provides additional operations to support different allocation schemes. Like in Java and in contrast to C++, objects are always accessed via references. To support alternative allocation strategies, we introduce two new allocation operators: `static` and `auto`, which return a reference to a static global object or an object allocated on the stack.

An object created with `static` is available for the complete run-time of the application. For example, the allocation of the rectangle `rec_global` in Listing 1 employs this feature. Static objects are represented by global variables in

```
class Rectangle {
  Point topLeft      = auto Point();
  Point bottomRight  = auto Point();
}

...

Rectangle rec_local   = auto Rectangle();
Rectangle rec_global  = static Rectangle();
```

Listing 1. Exemplary use of the extended memory model of MPL.

the generated C code. Their memory is statically allocated as part of the program image.

The operator `auto` allows to allocate objects on the stack. These objects are automatically deleted if the current block enclosed by { and } is left. If the newly allocated object has only been assigned to variables that are declared in the same block, then it behaves essentially like an automatic variable of a primitive type. Automatic objects are represented by local variables in the generated C code. The compiler also takes care of inserting appropriate statements in the C code to finalize objects before they are deleted.

The operator `auto` can also be used in a second role within the initializer expression of member variables. If used in an initializer, `auto` ties the lifetime of the newly created object to that of the host object. The class `Rectangle` in Listing 1 demonstrates this use of the `auto` operator. The memory required for the two `Point` instances is automatically allocated if a new instance of the `Rectangle` class is created. This type of automatic allocation is implemented by directly embedding the representation of the dependent object in the C representation of the host object. Consequently, the required storage space becomes part of the memory demand of the host.

To provide programmers with additional flexibility and to allow the creation of dynamic data structures, MPL also supports dynamic memory allocation and the standard `new` operator. Nevertheless, garbage collection is not supported, as this would add a significant run-time overhead. Dynamically allocated objects need to be destroyed explicitly by using a newly introduced `delete` operator. Dynamic memory allocation currently relies solely on the `malloc` implementation of the underlying target platform without additional optimization.

B. Multithreading

For the intended usage scenarios of MPL it is often necessary to execute several tasks in parallel. To support this in a user-friendly way, multithreading functionality is needed for the language as also indicated by Requirement III. The multithreading capabilities of MPL can be accessed via an interface that is closely modeled after the Java thread interface.

The current implementation is based on the Contiki multithreading library that provides a platform independent interface to switch the current stack. This interface is implemented for all major hardware platforms supported by Contiki. Based on this library, we implemented a custom thread scheduler running as a concurrent Contiki process that schedules runnable threads in a round robin fashion.

```
code neighborhoodDef = {:
  neighborhood HighTemperature() {
    System.getRole() == "sensor"
        and System.getType() == "temperature"
        and System.getTemperature() > x
  }
:};

Target highTemperature = auto LN(neighborhoodDef);
highTemperature.bindFloat("x", 30.0);

Report temperatureStream = auto Stream();
temperatureStream.setTarget(highTemperature);
temperatureStream.setAction(auto ReadTemperature());
temperatureStream.setDataOperator(auto MedianOperator());

temperatureStream.execute();
temperatureStream.waitResult();
Object result = temperatureStream.getResult();
```

Listing 2. Use of embedded code in MPL.

A major drawback of full-blown multithreading is the comparatively high memory overhead as each thread has its own stack that needs to be constantly kept in memory. To restrict the memory demand, the maximal number of concurrent threads is limited. If the maximum is reached, attempts to create further threads fail and an error is signaled to the user program. In the extreme case of an application that does not itself require multithreading, the whole application can be executed in a single thread. A separate thread running in parallel to the operating system is still required to support blocking operations without interfering with the operating system functions.

C. Embedding of Meta-abstractions

The core goal of the language design was the provision of an implementation of the make*Sense* framework. As a consequence and in line with Requirements I and II, the language needs to be able to cleanly expose the abstraction-based extensibility features of the make*Sense* framework.

To this end, the implementation of a programming abstraction within the language consists conceptually of three distinct components: (1) an MPL class, (2) a run-time module, and (3) (optionally) an MPL compiler plug-in. The MPL class serves as a means for MPL code to interface with the abstraction. Abstraction classes slightly differ from regular MPL classes in that their methods are typically not implemented by MPL code. Instead, each abstraction provides an additional run-time module that implements the required functionality in low-level C code. This increases efficiency and allows one to reuse existing implementations. In addition, the implementation of programming abstractions will typically be conducted by WSN experts that are already familiar with C programming on embedded devices. Finally, as indicated by Requirement II, programming abstractions may employ their own domain-specific language. To support a domain-specific language, the abstraction also needs to provide a compiler plug-in to translate the domain-specific code into code executable on the target platform. These plug-ins employ a compiler interface that is described in more detail in Section VI-B.

To support the use of domain-specific languages in a consistent way, MPL provides a dedicated syntax for *embedded code*. The mechanism for embedding code shares some similarities with prepared statements for embedding SQL code in programming languages. In contrast to prepared statements, embedded code fragments are not represented by strings, but by the special data type `code` to facilitate type checking and special handling by the compiler. Variables of this type can be instantiated only by a constant expression, consisting of a code fragment in an abstraction-specific language that is enclosed by the delimiters `{:` and `:}`. Also, unlike prepared statements, embedded code is not interpreted at run-time but compiled by abstraction-specific compiler plug-ins. As the embedded code is handled by plug-ins, it does not necessarily need to follow the syntax and semantics of MPL. To enable passing of values between embedded code and the MPL program, a binding mechanism is provided. Calls to a `bind` method of the affected abstraction allow one to bind variables in the embedded code to MPL variables.

To illustrate these aspects we employ Logical Neighborhoods [11] and a Stream abstraction as an example. A wrapper has been implemented in the make*Sense* project, that integrates an existing implementation of Logical Neighborhoods in the make*Sense* framework [3] by instantiating the Target meta-abstraction. Here, Logical Neighborhoods is used in combination with the Stream abstraction that implements the Report meta-abstraction. The Stream abstraction has been created from scratch within the make*Sense* project [3].

The Logical Neighborhoods abstraction employs a custom declarative language to define membership of nodes in a specific neighborhood, used as domain-specific language within MPL. This use of the functionality can be seen in lines 1–7 of Listing 2. In this example, a logical neighborhood `highTemperature` is defined that contains all nodes equipped with a temperature sensor and that read a high temperature value. The threshold value `x` is left as a parameter in the embedded code, and is later bound to the actual value in the main MPL program by means of an appropriate `bind` call in line 10.

The definition of the logical neighborhood relies on a set of node attributes (“role”, “type”, and “temperature”) provided by the run-time environment. To allow simple access to such node attributes and operations in embedded code, these are exposed as static methods of a predefined `System` class. In this example, the `System` methods `getType` and `getTemperature` are used in the embedded code to determine the value of the corresponding attributes associated to the target node (i.e., the type of attached sensor and the sensor value). In line 9 of the example, a new instance of Logical Neighborhood is created, which is configured by the embedded code. This Logical Neighborhood instance is associated to a new Stream instance in line 11 to define the set of nodes from which the data should be streamed towards the sink. After also specifying a Local Action to read a sensor value and a Data Operator to aggregate the individual readings, the stream is started in line 17. Finally, the program waits for a result and stores the returned result in the `result` variable, as soon as available.

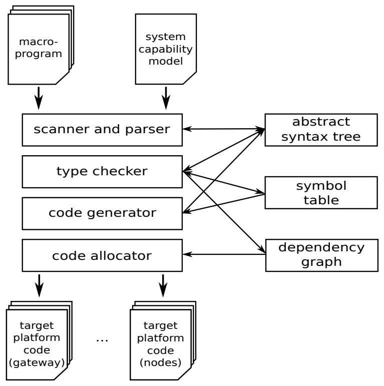

Fig. 2. Overview of the architecture of the MPL compiler. Arrows denote the data flow between these elements.

D. Interface to the Underlying Platform

MPL programs typically need to access functions of the underling hardware and software platforms. In MPL, such platform functions are also exposed through the previously introduced abstraction interface.

All communication-related tasks are handled by distributed actions. Direct access to functions to manually send or receive individual messages is intentionally not provided by the default implementation of MPL to shield the programmer from low-level details of communication. Nevertheless, such facilities could be provided by dedicated abstractions, if needed.

Node-local functions are exposed as Local Actions. This typically includes tasks like reading of sensor values and control of actuators, but the same facility could be also used to implement local data storage or data processing algorithms. We expect such platform-related functions to be implemented by WSN experts using C code with direct access to the underlying operating system. To support such an approach, methods in MPL can be declared as *native*, in which case the actual implementation of the method is provided by an external C implementation. Access to MPL language features is possible via a predefined C interface. Amongst other things, this interface provides the means to access, manipulate, and create MPL-defined objects within user-provided C code.

A number of Local Actions for commonly used functions are predefined by the language. As seen before, a subset of these functions is also available as static methods of an automatically generated `System` class to enable access from embedded code.

VI. MPL Compiler

Fig. 2 gives a high-level overview of the MPL compiler architecture[1]. Primary input to the MPL compiler is a macro-

[1]The compiler and supplementary software are available as part of the make*Sense* tutorial at http://project-makesense.eu/tutorial/makeSense-tutorial.zip

program written in MPL, possibly consisting of several source files. The macroprogram is supplemented by information about the system's capabilities, such as the hardware features and on-board sensors of deployed nodes. This information is used by the MPL compiler to aid optimization and the allocation of functionality to the different nodes. In addition to these inputs, the MPL compiler has access to a repository of components implementing the macroprogramming abstractions and run-time functionality. As output, the MPL compiler generates platform-specific source code for each node type of the target WSN, e. g., *gateway* and *regular nodes*, which is translated into a deployable binary image by the regular platform tool-chain. Our current prototype generates C-code for the Contiki operating system [6]. It is intended that later versions of the compiler will be extended with further code generators for different platforms.

A. Code Generation and Allocation

The compiler is implemented as a multi-pass compiler in Java. The compilation process consists of four distinct phases: parsing, type checking, code generation, and code allocation. All phases access the abstract syntax tree (AST) of the program and a shared symbol table. The implementations of parsing and type checking largely follow established approaches for compiler design. The code generation phase only differs from typical compilers in that it does not directly generate machine code. In the final code allocation phase, the compiler maps the compiled classes and interfaces to the available node types. A WSN may contain nodes with different capabilities that serve different purposes in the network. Not all of these nodes require the full functionality of the MPL program and part of the program is only ever executed on the nodes of a specific type. The code allocation phase allows one to remove unnecessary classes from the final code images and thus to reduce memory demands. In contrast to possible local code optimizations by the downstream C compiler, the allocation algorithm of the MPL compiler can take the entire control- and data-flow of the complete application, including remote invocations of abstractions, into account.

The current compiler prototype only distinguishes between a more powerful gateway node and the regular sensor nodes, but more complex allocation schemes are conceivable. The allocation procedure is based on the dependency graph generated by the type checker. For the gateway, the allocation process starts at the `main` method. This method is the central entry point of the MPL program and is always executed on the gateway machine. Starting from the `main` method, all classes used by this method are recursively collected. Only the compiled code of these classes is deployed on the gateway. For the nodes, the process starts at the set of actions that are defined in the MPL program. The compiler determines for each Action, if it is used by a Remote Action (e. g., Tell or Report). Only those Actions can be executed on a remote device and thus on one of the nodes in the WSN. For each of those Actions, all required classes are recursively collected. With the current prototype, this set of classes is deployed on all nodes. In the future, we will also take the capabilities of the nodes into account and build independent sets for each node type. In a second step, all Actions requiring capabilities that are unavailable at a specific node type are removed from the respective set.

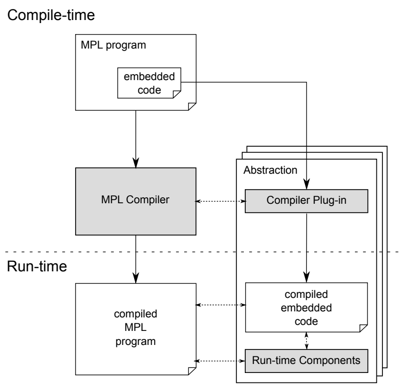

Fig. 3. Interaction of the MPL compiler and the compiler plug-ins. Continuous arrows represent data flow. Dashed arrows represent communication.

B. Plug-in Interfaces

Some programming abstractions employ embedded code to enable extensive configuration, as described in Section V-C. The abstraction-specific code cannot be handled by the MPL compiler itself, instead these abstractions need to provide compiler plug-ins to analyze and translate the abstraction-specific code. The MPL compiler plug-in interface allows these plug-ins to communicate with the MPL compiler at compile-time. Each compiler plug-in is essentially an independent little compiler with its own parser and code generator. Due to the fundamentally different nature of the object-oriented MPL code and the typically declarative embedded code, plug-ins cannot reuse the parsing and code generation functionality of the MPL compiler. Fig. 3 gives an overview of the interaction between the MPL compiler and the compiler plug-ins. Embedded code fragments are extracted by the MPL compiler and passed to the compiler plug-in provided by the abstraction that uses the code fragment. The compiler plug-in translates the domain-specific code into C code. The resulting C code is compiled with the native tool-chain and linked with the compiled binary image of the MPL program. The generated code can communicate with the abstraction-specific run-time component.

In some cases, it is necessary that the plug-ins generate different code for devices with different roles in the network. For example, the plug-in might differentiate between regular nodes and the network gateway. The gateway code might, for example, need to perform additional bookkeeping that is not required on the regular nodes. To support this, the plug-in may generate different C files for each of the available node types. The code is only linked with the correct binary image, in this case. Like the MPL compiler itself, plug-ins are implemented in Java.

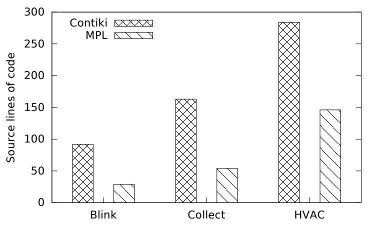

Fig. 4. User-written source lines of code per application.

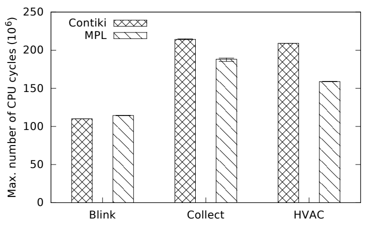

Fig. 5. Maximal number of CPU cycles spent on the nodes.

VII. Evaluation

To evaluate the performance of MPL we selected a small set of WSN applications and implemented each of them as a macroprogram and as functionally-equivalent Contiki/C programs. We then compared the performance of both implementations based on a set of typical software performance metrics for WSNs.

A. Settings and Metrics

We selected the following application scenarios for the evaluation:

1) *Blink to Radio.* This application scenario represents one-to-many communication. Such communication patterns are often used to distribute commands or configuration settings to a number of nodes. During run-time, the gateway process regularly sends a command to all sensor nodes requesting them to toggle their LED.
2) *Collect.* This application scenario consists of a simple data collection application. Data collection is a typical task for WSNs and as such is an important component of many real world applications. In the evaluated application, temperature and light sensor readings of all nodes are periodically sent to the gateway. At the gateway, the readings of all nodes are averaged and the result is reported to the user as a command line output.
3) *HVAC application.* The last application, originally developed in the make*Sense* project, implements a simple ventilation control system that regulates ventilation based on CO_2 readings. This scenario represents a simple real-world WSN application. Our setup consists of sensor and actuator nodes deployed in two rooms. The control process for each room is offloaded to one of the nodes located in the respective room.

All applications were implemented by an average programmer without a strong background in WSN programming. Before implementing the applications, he was provided with an introductory tutorial for the respective technology. During the experiments, the MPL and the Contiki/C programs was executed in identical simulated environments with five to six nodes and a gateway[2]. The WSN was simulated in real-time with Cooja, while the gateway code was executed in parallel in a 64-Bit Linux environment. The latter communicated with the simulated network via a serial socket and a simulated interface node [3]. Each application was executed until a scenario-specific termination condition was met. The same termination condition was used for the MPL and the Contiki/C implementation.

For each of the programs we investigated the following metrics:

1) The number of *source lines of code* is employed as an approximation of the programming effort for each application. We are aware that source lines of code are a very imprecise metric, especially if comparing different languages. Nevertheless, it can provide an initial idea of the relative complexities of the evaluated applications if consistent code formatting is used.
2) *Code size* is an important metric for WSN applications, as sensor nodes typically only possess limited program memory.
3) *Memory consumption* also needs to be kept low as random access memory is also a severely limited resource on sensor nodes. We need to distinguish the space required by statically allocated objects and the amount of heap space employed by dynamic memory allocation. Especially the use of dynamically allocated memory should be reduced, as dynamic memory allocation requires significant over-provisioning of memory resources. We measure heap allocation by linking the applications to a modified `malloc` implementation that tracks heap usage. Effects like fragmentation are not taken into account.
4) To compare the *computational overhead*, we count the CPU clock cycles spent on a typical execution of an application in a controlled environment.
5) The *communication overhead* is assessed by recording the number and size of radio messages sent during application execution. Ins WSN, energy consumption is often dominated by wireless communication, so that this metric also provides some insight on energy consumption.

B. Results

Fig. 4 demonstrates that the plain Contiki/C applications require the user to write more than twice as much code to ac-

[2]Please note, that even though the simulated scenarios employed only a small number of nodes, this does not invalidate the obtained results. The values for most of the employed metrics are not affected by the number of nodes.

[3]The node programs were compiled for the Contiki Tmote Sky target with an MSP-enabled version of the GNU C compiler (version 4.7.0 20120322, mspgcc patch set 20120911). The gateway code was compiled with the GNU C compiler (version 4.6.3).

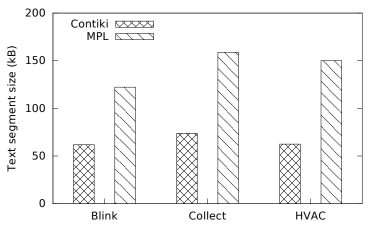

Fig. 6. Code size of the compiled gateway programs.

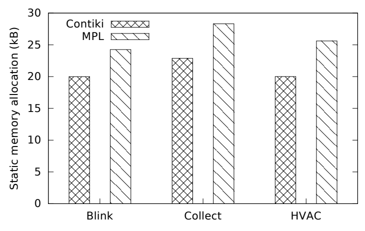

Fig. 8. Static memory allocation of the gateway programs.

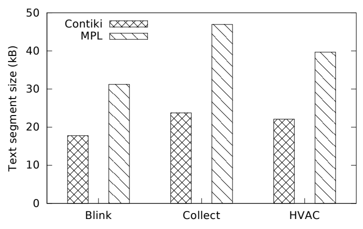

Fig. 7. Code size of the compiled node programs.

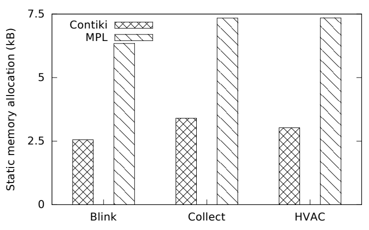

Fig. 9. Static memory allocation of the node programs.

complish the same task. In addition, further low-level technical details are exposed in the Contiki/C program. This indicates that the use of a high-level macroprogramming language like MPL can reduce the effort needed for the implementation of typical WSN applications. The programmer needs to write less code to achieve the same result.

Figs. 6 and 7 present the respective text size of the compiled MPL and Contiki/C implementations of the evaluated applications. It can be seen in Fig. 7 that the program image of the MPL programs is significantly larger for the evaluated applications. A cause for the increase in program size is the run-time environment required for some of the advanced features of MPL, like the support for multithreading. It should be noted that the overhead is largely constant (i.e., it does not grow with program size) and the relative overhead should be lower for larger, more complex applications. The program memory demand of the applications is still well within the limits of typical WSN platforms.

As presented in Figs. 8 and 9, the MPL-based code also makes use of more statically allocated memory. The overhead in static memory consumption is mainly caused by the additional data structures required for object-orientation support in MPL. As the relative overhead decreases with application size, we expect the relative overhead in the evaluated scenarios to represent a worst case scenario. More complex applications should exhibit a less significant relative overhead. Further optimization of the object representations can likely also significantly reduce the static memory demand of MPL-based programs.

While the Contiki/C-based programs do not make use of dynamic memory allocation, some features of the MPL programming model rely on dynamically allocated memory, e.g., to handle concurrent execution of Actions. This introduces a slight additional overhead, but the demand of dynamically allocated memory is comparatively little. None of the applications allocated more than 420 bytes at a time on any node in our test scenarios.

The computational overhead introduced by the use of MPL turns out to be very low in the experimental scenarios, as shown in Fig. 5. In the collect and the HVAC scenario, the MPL-based code actually employs less CPU cycles on the most active node of each network than the respective Contiki/C-based code. This demonstrates that features like virtual method dispatch do not introduce a significant overhead in terms of execution speed and energy consumption in typical applications.

As expected, the choice of a higher-level language does not significantly affect the total number of transmitted radio messages, as shown Fig. 10. Nevertheless, as shown in Fig. 11, the total amount of transmitted data is significantly higher for the MPL-based applications. This is mainly caused by the fact that the MPL code transmits complete objects that need to be serialized. An implementation providing a similar level of flexibility and features as the MPL code would be far more complex and consequently even more difficult to implement and maintain. At the same time its resource consumption would probably be much closer to the MPL-based implementation. Consequently, we conclude that the overhead for supporting a high-level macroprogramming language with object orientation is still reasonable for resource-constrained devices, like WSN nodes.

VIII. Conclusion

In this paper, we introduce the design and implementation of a Java-like macroprogramming language for the make*Sense* framework. A preliminary evaluation demonstrated that it is

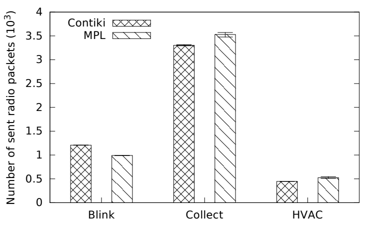

Fig. 10. Number of transmitted radio messages.

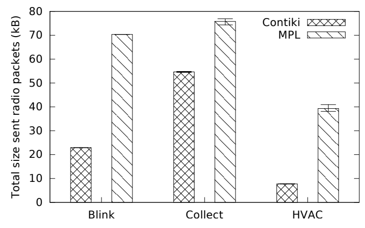

Fig. 11. Total size of transmitted radio messages.

possible to implement a high-level object-oriented macroprogramming language with a reasonable overhead.

Despite the positive results of the evaluation, current limitations open up an avenue for future work. The performance of the system can be further improved to make it even more suitable for the resource-constrained devices typically found in WSNs. Especially, memory consumption of the generated code still leaves significant room for improvements. Memory consumption could be, for example, improved by a more sophisticated strategy for code allocation. In the current implementation, code allocation operates at class level. Instead it would be possible to extend these decisions to individual methods and attributes. In addition, it would be useful to make the allocation algorithm work with a larger number of node types and to take the actual program behavior into account. To make the system more useful in practice, we also intend to improve debugging support. Finally, compatibility and dependencies among abstractions and between abstractions and the underlying protocols are not satisfactorily handled by the framework. The selection of suitable abstractions still requires manual intervention and some degree of expertise. Ideally, this selection would be largely automatic based on an abstract set of user-defined requirements. We currently explore possible solutions in the RELYonIT project [16].

Acknowledgment

Special thanks go to Niclas Finne for his support of the implementation of the MPL language run-time and compiler and to Bengt-Ove Holländer for his significant assistance with the implementation of the example programs used in the evaluation. The research leading to these results has received funding from the European Union 7th Framework Programme (FP7-ICT-2009-5, FP7-ICT-2011-8) under grant agreements n° 258351 (make*Sense*) and n° 317826 (RELYonIT).

References

[1] F. Aslam, L. Fennell, C. Schindelhauer, P. Thiemann, G. Ernst, E. Haussmann, S. Rührup, and Z. A. Uzmi, "Optimized Java binary and virtual machine for tiny motes," in *Proc. of the 6th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2010, pp. 15–30.

[2] N. Brouwers, P. Corke, and K. Langendoen, "Darjeeling, a Java compatible virtual machine for microcontrollers," in *Proc. of the ACM/IFIP/USENIX Middleware '08 Conference Companion*, 2008, pp. 18–23.

[3] F. Casati, F. Daniel, G. Dantchev, J. Eriksson, N. Finne, S. Karnouskos, P. Montera, L. Mottola, F. Oppermann, G. Picco, A. Quartulli, K. Romer, P. Spiess, S. Tranquillini, and T. Voigt, "Towards business processes orchestrating the physical enterprise with wireless sensor networks," in *Proc. of the 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 1357–1360.

[4] D. Chu, L. Popa, A. Tavakoli, J. M. Hellerstein, P. Levis, S. Shenker, and I. Stoica, "The design and implementation of a declarative sensor network system," in *Proc. of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2007, pp. 175–188.

[5] Ş. Gună, L. Mottola, and G. Picco, "DICE: Monitoring global invariants with wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 10, no. 4, pp. 54:1–54:34, Jun. 2014.

[6] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki – a lightweight and flexible operating system for tiny networked sensors," in *Proc. of the First IEEE Workshop on Embedded Networked Sensors (EmNets)*, Nov. 2004, pp. 455–462.

[7] C. Frank and K. Römer, "Algorithms for generic role assignment in wireless sensor networks," in *Proc. of the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2005, pp. 230–242.

[8] J. Gosling, B. Joy, G. Steele, and G. Bracha, *The Java^{TM} Language Specification*, 3rd ed. Addison-Wesley Professional, 2005.

[9] N. Kothari, R. Gummadi, T. Millstein, and R. Govindan, "Reliable and efficient programming abstractions for wireless sensor networks," in *Proc. of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, Jun. 2007, pp. 200–210.

[10] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: An acquisitional query processing system for sensor networks," *ACM Transactions on Database Systems*, vol. 30, no. 1, pp. 122–173, Mar. 2005.

[11] L. Mottola and G. Picco, "Programming wireless sensor networks with logical neighborhoods," in *Proc. of the 1st International Conference on Integrated Internet Ad Hoc and Sensor Networks (InterSense)*, May 2006, p. 8.

[12] L. Mottola and G. P. Picco, "Programming wireless sensor networks: Fundamental concepts and state of the art," *ACM Computing Surveys*, vol. 43, no. 3, pp. 19:1–19:51, Apr. 2011.

[13] R. Newton, G. Morrisett, and M. Welsh, "The Regiment macroprogramming system," in *Proc. of the 6th International ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN)*, 2007, pp. 489–498.

[14] F. J. Oppermann, C. A. Boano, and K. Römer, "A decade of wireless sensing applications: Survey and taxonomy," in *The Art of Wireless Sensor Networks*, ser. Signals and Communication Technology, H. M. Ammari, Ed. Berlin, Germany: Springer, 2014, pp. 11–50.

[15] Oracle, "Java ME and Java Card technology." [Online]. Available: http://www.oracle.com/technetwork/java/javame/index.html

[16] RELYonIT Consortium, "RELYonIT: Research by experimentation for dependability on the internet of things," 2014. [Online]. Available: http://relyonit.eu

[17] N. Shaylor, D. Simon, and W. Bush, "A Java virtual machine architecture for very small devices," in *Proc. of the 2003 ACM SIGPLAN Conference on Language, Compiler, and Tool Support for Embedded Systems (LCTES)*, 2003, pp. 34–41.

[18] D. Simon, C. Cifuentes, D. Cleal, J. Daniels, and D. White, "Java^{TM} on the bare metal of wireless sensor devices: The Squawk Java virtual machine," in *Proc. of the 2nd International Conference on Virtual Execution Environments (VEE)*, 2006, pp. 78–88.

[19] B. Stroustrup, "Multiple inheritance for C++," *Computing Systems*, vol. 2, no. 4, pp. 367–395, 1989.

7.6 Paper F: Automatic Protocol Configuration for Dependable Internet of Things Applications

Felix Jonathan Oppermann, Carlo Alberto Boano, Marco Antonio Zúñiga, and Kay Römer (2015a). "Automatic Protocol Configuration for Dependable Internet of Things Applications." In: *Workshop Proceedings of the 40th IEEE Conference on Local Computer Networks (LCN).* (Clearwater, FL, USA). Piscataway, NJ, USA: IEEE. ISBN: 978-1-4673-6772-1.

The paper was nominated for the best paper award of the SenseApp workshop.

7.6.1 Summary

In this paper, we propose a novel framework to automate the parametrization of IoT communication protocols. By automating the challenging task of selecting suitable parameters for a specific application and environment, it enables the configuration of WSN or IoT systems without a need for extensive knowledge on the user side. The framework uses models of the environment as well as of the employed hardware and protocols to predict the effects of environmental changes on network performance and to automatically select a configuration that meets user-specified dependability requirements. We also demonstrate how to use this framework to configure a state-of-the-art MAC protocol for an IoT application deployed in a challenging outdoor environment. An evaluation demonstrates the usefulness of the approach and its performance.

7.6.2 Contributions

I am the main author of the paper, implemented the described software system and carried out the majority of the described experiments. Sections II and V were written by Carlo Alberto Boano based on work by him and Marco Antonio Zúñiga. Carlo Alberto Boano also played a fundamental role in the evaluation of the software system as described in Section V. I presented the work at the 10th IEEE International Workshop on Practical Issues in Building Sensor Network Applications (IEEE SenseApp 2015) in conjunction with the 40th IEEE Conference on Local Computer Networks (LCN 2015) in Clearwater, Florida, USA.

Automatic Protocol Configuration for Dependable Internet of Things Applications

Felix Jonathan Oppermann*, Carlo Alberto Boano*, Marco Antonio Zúñiga†, and Kay Römer*
*Institute for Technical Informatics, Graz University of Technology, Austria
†Embedded System Group, TU Delft, The Netherlands
Email: {oppermann,cboano,roemer}@tugraz.at, m.zuniga@tudelft.nl

***Abstract*—To meet strict dependability requirements in hostile and highly-varying environments, IoT communication protocols need to be carefully tuned in relation to the expected environmental changes. However, this is difficult to attain, as every application has unique properties and requirements. Tuning communication protocols correctly requires indeed significant expertise as well as a clear understanding on how hardware and software components are affected by environmental changes.**

In this paper, we propose a novel framework to automate the parametrization of IoT communication protocols. The framework uses models of the environment as well as the employed hardware and protocols to predict the effects of environmental changes on network performance and to automatically select a configuration that meets user-specified dependability requirements.

We demonstrate how to use this framework to configure a state-of-the-art MAC protocol for an IoT application deployed in a challenging outdoor environment and evaluate its accuracy in predicting how environmental changes affect network performance. We further evaluate the performance with different optimization strategies and show that the average run-time necessary to find a solution is sufficiently low to enable the use of our system in a typical IoT design process.

***Index Terms*—Dependability, Environmental Impact, Communication Protocols, Optimization, Internet of Things, Wireless Sensor Networks.**

I. Introduction

An increasing number of Internet of Things (IoT) and wireless sensor network (WSN) systems has been installed in real-world settings during the last years [1]. These systems are becoming an integral part of our daily lives, as they are used in application areas such as civil infrastructure monitoring, home automation, smart cities, smart grid, and smart healthcare. Several of these application domains are safety-critical, and rely on the dependable and predictable operation of sensors and actuators that are wirelessly networked. For example, systems employed to monitor patients, to control traffic, and to inspect the structural health of buildings impose *strict dependability requirements* on communication performance, as their failure can have severe consequences.

Fulfilling these dependability requirements can be very difficult, as WSN and IoT systems are often deployed in hostile environments that significantly affect their performance. For instance, systems deployed outdoors are affected by temperature fluctuations and changing weather conditions [2], [3]. To cope with these challenges and meet strict dependability requirements also in hostile environments, *communication protocols need to be carefully tuned* in relation to the expected environmental changes [4]. This is, however, difficult to attain, as every application has unique properties and requirements, and there is no “one-size-fits-all” solution. Tuning communication protocols correctly can indeed be a tedious task that requires significant expertise, as well as a clear understanding of how the environment affects the hardware in use and the different communication protocols [5], [6]. Furthermore, even for experts in the field, it may be difficult to find the right trade-off that satisfies multiple requirements for the application at hand (e.g., achieving both a high reliability and a low energy consumption). Therefore, there is a need for a simpler configuration of IoT communication protocols that does not overwhelm the intended users of the technology.

In this paper, we propose a novel framework to support the deployment of IoT and WSN applications by *automating the configuration of communication protocols* such that specific dependability requirements can be met. The framework uses models of the environment as well as of the employed IoT hardware and communication protocols to predict the effect of environmental changes on network performance. Given a set of user requirements, these models are used in combination with mathematical optimization techniques to select a protocol configuration that provides the required performance.

We demonstrate how our framework can be used to find a suitable protocol configuration that meets user-defined dependability requirements using a building façade monitoring application as a case study. We show that our framework can help to predict and avoid the adverse effects of temperature variations on communication performance found in this type of application and evaluate its accuracy and performance. We further *evaluate the performance* of different optimization strategies within the framework and show that the average run-time necessary to find a solution is sufficiently low to enable the use of our system in a typical IoT design process.

The contributions of this paper are three-fold:

1) We present the architecture of a new framework automating the configuration of IoT communication protocols;
2) We demonstrate how to apply this framework to configure a state-of-the-art MAC protocol for an IoT application deployed in a challenging outdoor environment;
3) We evaluate the accuracy of the framework in predicting the behavior of the environment and its effect on network performance.

The remainder of the paper is structured as follows. The next section introduces an exemplary application that will serve as running example and as basis for the evaluation of our system. We illustrate our approach in Sect. III and detail on the architecture and implementation of our framework in Sect. IV. Thereafter, in Sect. V, we demonstrate how to use the framework to find a suitable configuration that meets user-defined dependability requirements. In Sect. VI we evaluate the overall performance of the framework. After describing related work in Sect. VII, we conclude our paper in Sect. VIII.

II. Case Study: Reliable Monitoring of Building Façades

A large number of IoT and WSN systems are deployed *outdoors* and are expected to operate dependably over extended periods of time, e.g., wildfire detection systems in forests [7] or wireless networks monitoring structural damage in civil infrastructures and buildings [8], [9]. Unfortunately, the performance of wireless systems deployed outdoors is typically affected by time-varying environmental conditions such as meteorological changes and variations in humidity [2], [3], which makes it difficult to satisfy strict dependability requirements. For example, large temperature variations can have a severe impact on network performance, as they reduce the efficiency of radio transceivers [4].

We have experienced these problems in the context of the RELYonIT project [10], during a pilot deployment of a WSN on the different façades of a building in Madrid, Spain. The purpose of our outdoor deployment is to promptly detect structural damage as well as to measure the energy efficiency of the construction by analyzing how the employed insulating materials reduce heat transfer. This type of application requires *a continuous reliable collection* of sensor data, such as temperature, humidity, and vibration. On the one hand, achieving a high packet delivery rate across the network is necessary to have a complete picture of the integrity of the building and to avoid severe issues, such as the detachment of loose parts from the building façade. On the other hand, to draw conclusions about the effectiveness of a constructing material or HVAC system, engineers rely on tiny changes in the measured variables, and any gap in the collected data may lead to false conclusions.

The major obstacle towards a reliable data collection is that nodes deployed on the building façades often experience high temperature fluctuations, especially if they are placed inside IR-transparent enclosures exposed to direct sun radiation. During our pilot deployment we have indeed observed daily temperature variations as high as 50 °C.

These large temperature variations can have a severe impact on the operation of CSMA protocols, because they can reduce the effectiveness of clear channel assessment (CCA) methods and compromise the ability of a node to avoid collisions and to successfully wake-up from low-power mode [11]. Indeed, at high temperature the efficiency of low-power radio transceivers may reduce significantly: as a result, the signal strength between two wireless sensor nodes A and B decreases when the on-board temperature of one of the two nodes (or of both nodes) increases [4].

This problem is exacerbated by the fact that most state-of-the-art CSMA MAC protocols *rely on default system settings*, e.g., on the default CCA threshold of the employed radio device, hence neglecting the impact of the specific environmental properties of the target deployment site. As we show in Sect. V, protocols should instead be carefully parametrized in relation to the network configuration and to the properties of the environment. For example, an inaccurate selection of the CCA threshold may lead to a situation in which receiver nodes constantly remain in low-power mode at high temperatures, causing the disruption of links and a drastic reduction in network performance that may violate the dependability requirements of the application [11].

However, configuring protocols correctly is a very complex task. For example, to select the optimal CCA threshold for a MAC protocol employed outdoors, engineers need to consider the expected temperature variations at the deployment site, their impact on the hardware platform in use and on protocol operations, as well as the overall implications on a network level. This is not only time-consuming, but also non-trivial, given that the parameter value needs to be computed in relation to the specific application requirements defined by the user and to the actual placement of nodes.

To correctly predict the effects of environmental changes on network performance and select an optimal configuration of the system, it is hence highly desirable to employ a tool that automates the parametrization of communication protocols so that user-specified dependability requirements can be met. We describe next our attempt to build such an automatic framework by first illustrating the approach we have followed.

III. Approach

In order to fine-tune communication protocols such that user-specified requirements can be met, our framework needs a set of *user requirements* and a number of formal *models* to make reliable predictions about the system and its surrounding environment. The framework uses then mathematical *optimization techniques* to find a (near-)optimal configuration that meets the desired performance.

We distinguish between three categories of interacting *models*: environmental models, platform models, and protocol models.

1) *Environment models.* These models capture relevant aspects of the environment and provide an abstract representation thereof. Individual model instances are created for each specific environment by setting key parameters. The latter are determined by running a data collection application prior to the actual deployment. In our façade monitoring application example, we need to employ a model capturing the evolution of temperature in the target deployment site. Such a model can consist of the expected minimal and maximal temperatures for specific time intervals of the day (e.g., dawn, morning, noon, afternoon, evening, and night). To instantiate the model,

temperature is monitored over an extended time period prior to the deployment to compute minimal and maximal temperatures for each time interval.

2) *Platform models.* Different brands and types of sensor nodes react differently to specific environmental conditions. This relationship is captured by platform models. The latter provide a mapping of environmental parameters to variables that are relevant for the operation of WSN software. In our case study, temperature affects the operation of low-power radio transceivers. Consequently, our platform model needs to capture the relationship between the on-board temperature of sender and receiver nodes and the attenuation of the received signal strength for the employed hardware platform [4].
3) *Protocol models.* These models characterize the operation of a protocol under certain environmental conditions and with a predefined hardware configuration, and are hence built upon environmental and platform models. To be able to assess the effects of performance changes, these models need to expose all the relevant parameters of the protocol that may suffer from environmental impact. In our façade monitoring application case study, we use a network of nodes running ContikiMAC [12], a CSMA-based MAC protocol that is vulnerable to the impact of temperature variations on clear channel assessment. We therefore need to derive a model for ContikiMAC that estimates how different CCA settings affect the expected packet reception rate (PRR) as a function of the expected temperature variations and hardware platform employed.

These models alone already allow predictions of the performance based on a specific configuration. To automatically identify a (near-)optimal configuration that meets the user's requirements based on these predictions we employ *mathematical optimization*. Optimization strategies provide a way to systematically evaluate configurations such that a (near-)optimal solution can be found in a relatively short time.

The configuration process is steered by *user-defined dependability requirements*, as an optimal selection of a parameter strongly depends on the actual application needs. The definition of application requirements is complicated by the fact that some applications support different states of operation, often with significantly different requirements. For example, a system to detect wildfires in forests would be typically optimized for a long system lifetime during normal operation. However, as soon as a forest fire is detected, lifetime is not a primary concern anymore, and the primary goal becomes instead the fast dissemination of the fire-front direction. Consequently, it is necessary to allow the user to define multiple performance states and their associated sets of requirements. For each state, an individual configuration is created, and at run-time the application can select the configuration that best meets its current state.

IV. Parameter Selection Framework

We now describe the parameter selection framework implementing the automatic protocol configuration approach introduced in Sect. III. After giving a coarse-grained description of the software architecture of the parametrization component, we then detail on the optimization techniques employed by the framework and illustrate the actual software implementation.

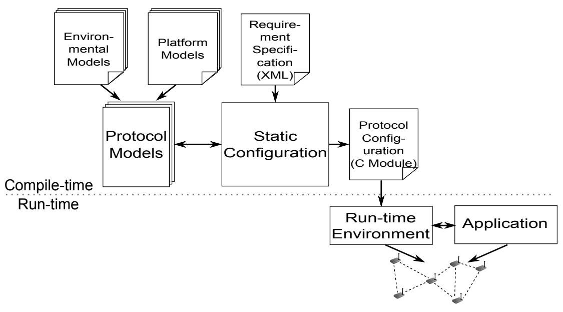

Fig. 1. Architecture of the parametrization framework.

A. Architecture

A bird's eye view of the architecture of the parametrization framework is presented in Fig. 1. The system architecture is organized around the *static configuration* component for protocol parametrization, which coordinates the automatic parameter selection process. It receives a user-provided *specification* of the dependability requirements as input.

The specification consists of a number of constraints on properties of the network behavior and a single property that should be either maximized or minimized (e.g., maximization of network lifetime). To increase the flexibility, our framework also supports probabilistic constraints that only need to hold at a specific point in time with a given probability. It is also possible to define more than one constraint for the same metric, typically with different probabilities. For example, a user could specify that the packet reception rate should stay above 0.8 with a probability of 0.9 and above 0.5 with a probability of 1. For applications that support different operation modes, the requirements for each mode are specified individually and, for each mode, an individual protocol configuration is derived.

Based on these inputs a near-optimal configuration for the respective protocol is generated by the static configuration tool, employing mathematical optimization techniques. If the requirement specification defines different modes of operation, the process is executed individually for the requirements of each mode.

The final output of the parametrization tool is a *protocol configuration* for each employed protocol. These configurations are static and do not change at run-time. Nevertheless, it is possible to switch between configurations associated to the different performance modes. In the following sections we will look at individual aspects of the framework in more detail.

B. Underlying Mathematical Model

The static configuration component employs mathematical optimization to generate an optimal parameter configuration.

We employ stochastic optimization strategies that are not able to give absolute guarantees, but are usually more robust to noisy data and require less fine-tuning for a specific problem instance. Due to their inherent randomness, stochastic strategies tend indeed to be able to escape local minima and to approach a global optimum even in a non-convex search space.

Each dependability specification essentially defines a constrained optimization problem. The formulation is based on a number of metrics $m_i(\mathbf{c})$ that allow to evaluate a specific configuration $\mathbf{c}$ based on a protocol model. A single protocol model may support more than one metric, each representing a specific relevant performance measure of the underlying protocol implementation. During the evaluation, the protocol model resorts to a suitable environment and platform model to incorporate the exact properties of the the environment and platform at hand. In the current system, we only consider the single objective case where a single metric is optimized. In addition to this optimization goal, the user may specify a number of constraints that further determine the properties of the desired solution. This results in optimization problems of the following form[1]:

$$\begin{array}{ll} \text{Minimize} & m_0(\mathbf{c}) \\ \text{Subject to} & m_1(\mathbf{c}) \leq t_1 \text{with prob. } p_1 \\ & m_2(\mathbf{c}) \leq t_2 \text{with prob. } p_2 \end{array} \tag{1}$$

The goal is to find a set of protocol configuration parameters $\mathbf{c}$ that optimize a single metric m_0. In addition, a variable number of constraints need to be fulfilled by ensuring that $m_1(\mathbf{c})$ and $m_2(\mathbf{c})$ are below the respective thresholds t_1 and t_2 with a probability of at least p_1 or p_2. Note that m_0, m_1, and m_2 may refer to the same metric.

To be suitable for automatic optimization, this optimization problem needs to be transformed into a suitable goal function, as most optimization strategies cannot immediately handle probabilistic constraints.

First, the constraints are integrated with the optimization function in an approach resembling penalty functions [13]. In the unconstrained case, the cost $f(\mathbf{c})$ of a specific configuration $\mathbf{c}$ is determined only by the goal function $m_0(\mathbf{c})$. To integrate constraints, a measure of violation is computed for each constraint and is added to the total cost of the current configuration. To ensure that invalid solutions are unlikely to be selected, while still allowing the optimization algorithm to traverse infeasible regions of the search space in order to reach more promising regions, the influence of the constraint violation is given more weight. If we assume a weight of k, the total cost for a given configuration $\mathbf{c}$ and r constraints can now be calculated as:

$$f(\mathbf{c}) = \frac{f_{\text{goal}}(\mathbf{c}) + k \sum_{j=1}^{r} f_{\text{cons},i}(\mathbf{c})}{1 + kr} \tag{2}$$

[1]To simplify the presentation and without loss of generality, we assume that the metrics (m_0, m_1, m_2) and the thresholds (t_1, t_2) are normalized to the $[0, 1]$ range and that smaller values denote superior properties. In addition, only minimization and less-or-equal constraints are considered, as other goals and constraints can be easily converted into this form.

where $f_{\text{goal}}(\mathbf{c}) = m_0(\mathbf{c})$ depends only on the evaluation of the goal and $f_{\text{cons},i}(\mathbf{c}) = \max\{(t_i - m_i(\mathbf{c})), 0\}$ is determined by the degree of violation for the i^{th} constraint.

Second, the non-standard feature of probabilistic constraints that is not supported by the employed optimization techniques needs to be handled. To support this, we exploit the fact that most environmental parameters exhibit a periodic behavior, e.g., a day and night cycle. This allows us to divide the period into a number of intervals with individual environmental properties. If we assume that communication events are evenly distributed over time, we can associate a probability q_j with each interval based on its relative length. This indicates how likely it is that a communication event is affected by the properties of this specific interval. Instead of a single function $m_i(\mathbf{c})$ per metric, we now employ a set of functions $m_{i,j}(\mathbf{c})$, each corresponding to one of the n intervals. Each of these functions uses a different instance of the environmental model that represents the distinct interval. To support probabilistic constraints, we now need to adapt the definition of the functions $f_{\text{goal}}(\mathbf{c})$ and $f_{\text{cons},i}(\mathbf{c})$ in Eq. 2. The cost of the goal $f_{\text{goal}}(\mathbf{c})$ is simply defined as the average of the cost for each individual interval:

$$f_{\text{goal}}(\mathbf{c}) = \sum_{j=1}^{n} q_j m_{0,j}(\mathbf{c}) \tag{3}$$

The calculation of the cost of the individual constraints now takes the probabilities in account by employing the definition:

$$f_{\text{cons},i}(\mathbf{c}) = \max\left\{\left(p_i - \sum_{j=1}^{n} \tau(m_{i,j}(\mathbf{c}), t_i, q_j)\right), 0\right\} \tag{4}$$

where n is the number of intervals and the function

$$\tau(v, t, q) = \begin{cases} q, & v > t \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

implements the aforementioned check for violation of the constraint. The resulting definition of $f(\mathbf{c})$ can now be employed as goal function of an unconstrained optimization problem that is well supported by the employed optimization techniques.

C. Implementation

The protocol parametrization tool is implemented as a standalone Python application. Its primary input is a user-provided requirement specification employing a custom XML-based specification language. This file contains an encoded specification of the user's requirements and defines an optimization problem as detailed in Sect. IV-B. If the application supports different states of operation, an independent requirements specification document is provided for eachstate.

In addition to the specification, the parametrization component has access to a collection of protocol model implementations. Available protocol model implementations are located via a search path and are automatically loaded by the framework as needed. To enable dynamic loading and the easy addition of additional models without the need to modify the framework itself, protocol models employ a plug-in interface.

This interface provides methods for initialization and model evaluation. The latter is used during the optimization process to evaluate the quality of individual protocol configurations. In addition, the interface allows to query which metrics are supported by the model and can be used in a related requirement specification. Most models only cover a subset of the available performance and reliability metrics. When evaluating a configuration, the protocol model implementations make use of platform and environmental model instances. For simpler models, their implementation is usually directly integrated with the implementation of the protocol models. For more complex platform and environmental models, they are implemented as separate modules which are accessed via method calls. Both environmental and platform models depend on application-specific empirical data that is usually loaded from a file at initialization. This interface is defined by an abstract `Model` class.

The protocol parameterization component can utilize different optimization strategies to solve the optimization problem, allowing the user to choose a strategy that is most appropriate for the specification and models at hand. The current prototype implements two stochastic optimization strategies. Stochastic optimization strategies cannot guarantee that an optimal solution is found in each run, but they are usually more robust to noisy data and require less fine tuning for a specific problem instance. Suitable stochastic optimization strategies still possess a high probability of convergence and are usually able to find a near-optimal solution. Due to their inherent randomness, stochastic strategies tend to be able to escape local minima and to approach a global optimum even in a non-convex search space. The current prototype supports simulated annealing and evolutions strategies. Both strategies use custom implementations that support configurations with integer, floating point, nominal, and Boolean values. The implementation of evolution strategies builds on ideas from Reehuis and Bäck [14].

The final output of the configuration tool is a protocol configuration encoded in a C source file. It contains individual configuration values for the configurable parameters of the involved protocols. The C representation is later compiled and linked with a run-time environment to enable the correct configuration of the communication protocols during operation. To give the protocol implementations access to the derived configuration values, a unified configuration interface is provided by a run-time environment, which enables them to receive their configuration parameters at initialization. In addition, to support different performance states for the application, the static optimization tool generates an individual parameter configuration for each performance state. The run-time environment provides methods for the user-application to switch between different performance states and to notify protocols of a state change.

V. Application of the Framework

We now demonstrate the applicability of our framework in a typical IoT application using the façade monitoring application introduced in Sect. II as a case study. In this scenario, nodes deployed on the building façades may experience high fluctuations of their on-board temperature. As we have discussed previously, these variations can have a severe impact on the operation of low-power CSMA MAC protocols such as ContikiMAC [12] due to the inefficiency of clear channel assessment at high temperatures. Indeed, the signal strength between two nodes A and B decreases in a linear fashion when the on-board temperature of one of the two nodes (or of both nodes) increases [4]. When low-power CSMA protocols perform an inexpensive clear channel assessment (CCA) check to determine if a node should remain awake to receive a packet or whether it should return to sleep mode, they essentially compare the current received signal strength with a CCA threshold ζ. The latter is typically chosen at compile-time and often set to the default value of the employed radio device (e.g., -77 dBm for the off-the-shelf CC2420 radio transceiver). When temperature increases, the received signal strength of a node may decrease to a point in which it becomes lower than ζ. When this happens, the receiver node remains constantly in low-power mode, causing disruption of the link [11]. Still, ζ should not be set to an arbitrarily low value, as this may lead to an increased number of false wake-ups due to interference and noise in the surroundings that would significantly increase the energy expenditure. To avoid the issues, we need to properly configure ζ such that any potential increase in the on-board temperature of the deployed nodes will not lead to a loss of connectivity, i.e., we need to find a suitable configuration of ζ such that the network can sustain the desired PRR despite temperature changes while still minimizing energy expenditure. Towards this goal, we need to (i) derive the necessary models, (ii) select the application requirements, and (iii) integrate them into the framework.

A. Deriving the Models

Our framework requires three different types of models: environmental, platform, and protocol models. The *protocol model* will describe how the operations of the employed MAC protocol, ContikiMAC, are affected by temperature changes. More specifically the protocol model captures the effect on the performance of ContikiMAC in terms of PRR. This model will build upon a *platform model* characterizing the signal strength attenuation of the radio transceiver embedded in the employed sensor nodes, as well as upon an *environmental model* capturing the possible variations of on-board temperatures in the specific deployment environment.

a) Environmental model: Our façade monitoring application is deployed in the city of Madrid, Spain, and we hence need to capture how its Mediterranean climate can affect the on-board temperature of sensor nodes. We devise an environmental model based on the maximum on-board temperatures recorded on the sensor nodes at specific times of the day. In particular, we sub-divide each day into six intervals of equal length, and run a specific data collection application prior deployment that records the on-board temperature variations over several days [10]. The model thus provides the maximal temperature as a function of the time of the day.

b) Platform model: In our deployment we employ MTM-CM5000-MSP motes embedding a CC2420 radio transceiver. In earlier research, we have shown the effects of temperature on the efficiency of this transceiver, and derived a linear model characterizing the decrease in signal strength as a function of temperature [4]. Denoting PL as the path loss between a transmitter-receiver pair, P_{t} as the transmission power, $P_{\mathrm{r}} = P_{\mathrm{t}} - PL$ as the received power, and P_{n} as the noise floor at the receiver, this model describes the temperature effect on the SNR as follows:

$$\begin{aligned} SNR &= (P_{\mathrm{t}} - \alpha\Delta T_{\mathrm{t}}) - (PL + \beta\Delta T_{\mathrm{r}}) \\ &\quad -(P_{\mathrm{n}} - \gamma\Delta T_{\mathrm{r}} + 10\log_{10}(1 + \tfrac{\Delta T_{\mathrm{r}}}{T_{\mathrm{r}}})) \\ &= (P_{\mathrm{r}} - \alpha\Delta T_{\mathrm{t}} - \beta\Delta T_{\mathrm{r}}) \\ &\quad -(P_{\mathrm{n}} - \gamma\Delta T_{\mathrm{r}} + 10\log_{10}(1 + \tfrac{\Delta T_{\mathrm{r}}}{T_{\mathrm{r}}})) \end{aligned} \tag{6}$$

where the constants α, β, and γ with units dB/K denote respectively the effect on transmitted power, received power, and on the noise floor. These values are obtained by regression over data obtained from testbed experiments. The values T_{t} and T_{r} represent the reference temperature in Kelvin of transmitter and receiver; whereas ΔT_{t} and ΔT_{r} capture the difference of current temperature in Kelvin with respect to T_{t} and T_{r} [4].

c) Protocol model: The reception of a packet in CSMA-based protocols such as ContikiMAC can be estimated by analyzing how the signal strength with which the packet is received relates to the transitional phase of the radio response and to the selected CCA threshold.

Each node periodically wakes up from low-power mode and checks for incoming packets by verifying if the received signal strength s_{r} is above a fixed CCA threshold ζ. The PRR is hence firstly influenced by the relationship between s_{r} and ζ: if $s_{\mathrm{r}} \geq \zeta$, the node infers that an ongoing transmission is present and remains awake to receive the packet; if $s_{\mathrm{r}} < \zeta$, the node believes that there is no ongoing transmission and returns to sleep mode without receiving the packet.

Furthermore, PRR is also affected by the transitional phase of the radio response. When the signal strength of the received packet is too close to the sensitivity threshold of the employed radio, it becomes unlikely to successfully demodulate the packet. Early WSN research has shown that the decrease of PRR in the transitional region follows a sigmoid curve [15], in which the probability p of receiving a packet is

$$p = (1 - f(P_{\mathrm{r}} - P_{\mathrm{n}}))^b \tag{7}$$

with P_{r} being the received signal strength in dBm, P_{n} the sensitivity threshold of the radio in dBm, and b the number of bits in the packet.

Denoting $s_{0.99}$ as the signal strength that leads to a delivery rate of 0.99, and $s_{0.01}$ as the corresponding signal strength for a delivery rate of 0.01, we can define three reception regions, as shown in Fig. 2 in the black sigmoid curve: a connected region, where the received signal strength is above $s_{0.99}$; a disconnected region, where the signal strength is below $s_{0.01}$, and a transitional region of length τ dB, where the delivery rate drops monotonically between 1 and 0.

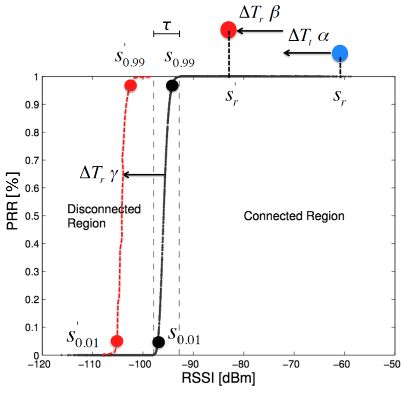

Fig. 2. The impact of temperature on the operation of a MAC protocol.

Because of the dependency between signal strength and temperature, according to Eq. 6, a variation in the on-board temperature at the receiver or at the transmitter will cause the receiver to measure a signal strength

$$s'_{\mathrm{r}} = (s_{\mathrm{r}} - \alpha\Delta T_{\mathrm{t}} - \beta\Delta T_{\mathrm{r}}) \tag{8}$$

i.e., an increase (decrease) in T_{t} and/or T_{r} will attenuate (strengthen) s_{r} into s'_{r}. Fig. 2 shows an example in which the received signal strength s_{r} decreases (i.e., is shifted to the left) due to an increase of temperature in both transmitter ($\alpha\Delta T_{\mathrm{t}}$ component) and receiver ($\beta\Delta T_{\mathrm{r}}$ component). To predict if a change in the on-board temperature at the transmitter and/or receiver node will affect packet reception, we need to verify if $s'_{\mathrm{r}} < \zeta$. If this is the case, no packet will be received, as the node will return to sleep mode after having assumed no ongoing transmission.

Similarly, if the on-board temperature of the receiver changes, also the position of the sigmoid curve may change. For example, an increase in temperature would lower the noise floor (see Eq. 6), shifting this curve towards left (red sigmoid curve). To predict how $s_{0.01}$ and $s_{0.99}$ would change in relation to temperature variation we use Eq. 6 to derive $s'_{0.99} = s_{0.99} - \Delta T_{\mathrm{r}}\gamma$ and $s'_{0.01} = s_{0.01} - \Delta T_{\mathrm{t}}\gamma$.

We can hence estimate the packet delivery rate PRR' given a specific ζ value for each link i in the network as:

$$PRR' = \begin{cases} 1, & \text{if } \max\{\zeta, s'_{0.99}\} < s'_{\mathrm{r}} \\ p, & \text{if } s'_{0.01} \leq \zeta < s'_{\mathrm{r}} \leq s'_{0.99} \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

This allows us to estimate the worst case delivery rate given a specific temperature variation/range.

B. Integration with the Framework

The aforementioned model has been realized as a plug-in implementing the interface defined by the framework. The

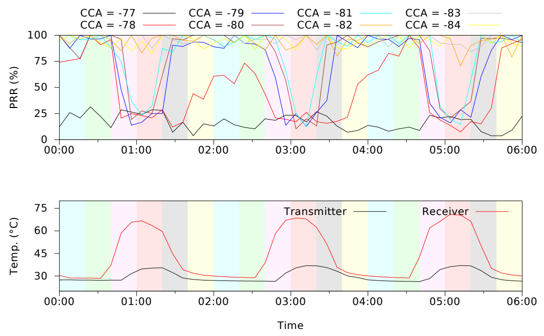

Fig. 3. Impact of CCA threshold on PRR for a single link. Background colors indicate different time intervals.

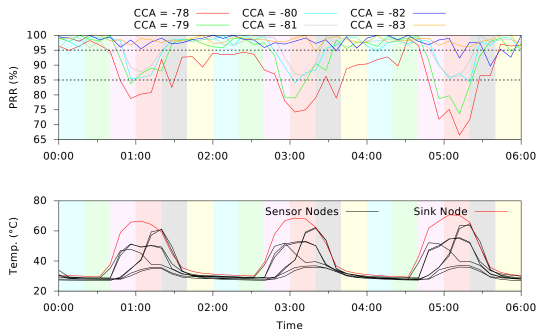

Fig. 4. Impact of CCA threshold on PRR on a network of nodes. The bottom plot shows the different temperature profiles of the nodes in the network.

implementation exposes one configuration parameter, ζ, and provides a number of metrics that can be used to define goals or constraints. The primary metric is the expected worst case PRR of a link averaged over all links in the network.

C. Selecting the Dependability Requirements

As discussed in Sect. II, the target application needs a highly reliable data collection. Therefore, the MAC layer needs to ensure a high PRR even under adverse environmental conditions. At the same time, it is also important to ensure that the system will have a long lifetime in order to keep maintenance at a manageable level. Hence, we need to find a CCA threshold setting that still ensures that the user requirements are met but leads to as little energy wastage as possible. Consequently, the CCA should be maximized while still not violating the constraints, as lower CCA values lead to a higher number of false wake-ups.

An industry partner running the deployment specified, that to enable useful insights about a building's energy-efficiency, at least 85% of the collected measurements should be successfully delivered to the sink at all times and with a probability of 0.9 at least 95 % of the packets should arrive at their destination.

Based on these requirements and the implemented models it is now possible to determine a protocol configuration that is able to support the expected performance. These requirements lead to the following optimization problem:

$$\begin{aligned} &\text{Maximize} && \text{CCA}([\zeta]) \\ &\text{Subject to} && \text{PPR}([\zeta]) \geq 0.85 \text{ with probability } 1.00 \\ & && \text{PPR}([\zeta]) \geq 0.95 \text{ with probability } 0.9 \end{aligned} \tag{10}$$

VI. Evaluation

In this section we evaluate the performance of the proposed framework.

A. Suitability of the Framework

To evaluate the applicability of the framework, we employ it to generate a suitable configuration for the case-study introduced in Sect. II. The goal is to find an optimal CCA threshold value that conserves energy while still fulfilling the user's dependability requirements. To ease the analysis and presentation of the results, we first consider a single link between two nodes. In a second step, we demonstrate that the same principle correctly works on a network of nodes.

We use TempLab [16], a temperature-controlled testbed, to recreate a network scenario similar to the one found on real-world outdoor façades. In particular, we feed TempLab with temperature traces previously recorded in Madrid so that the on-board temperature of the sensor nodes in the testbed experiences the same profile as in the real-world. In our experiments we use a time-lapse factor of 12, so that a day is replayed within 2 hours in our testbed. We further set the width of the transitional region τ = 5 dB and the packet size b = 35 bytes. The remaining model parameter were left at their default values of $\alpha = 0.078\text{dB/K}$, $\beta = 0.078\text{dB/K}$, and $\gamma = 0.037\text{dB/K}$. These values have been empirically determined by earlier experiments [4].

In the first set of experiments we employ the framework to configure the CCA threshold to obtain dependable communication on a *single link*. In particular, we aim to find a configuration that maximizes the CCA threshold while maintaining a PRR between the two nodes of 0.85 at all times. Based on temperature traces from the deployment in Madrid, the configuration framework determines an optimal CCA threshold of -83 dBm for this requirement specification. When trying different possible CCA threshold settings in the TempLab testbed, as shown in Fig. 3, we can see that with a threshold of -83 dBm, the PRR actually stays above 0.85 for the whole duration of the experiment, whilst with a higher CCA threshold this would not be the case.

In a second experiment, we employ the same PRR constraint of 0.85 but only with a probability of 0.6. For this scenario, the framework determines an optimal CCA threshold of -81 dBm. As shown in Fig. 3, the use of a threshold of -81 dBm actually ensures that the PRR stays above 0.85 in four out of the six intervals per day, which fulfills our requirement of sustaining a PRR of at least 0.85 in at least 60% of the cases.

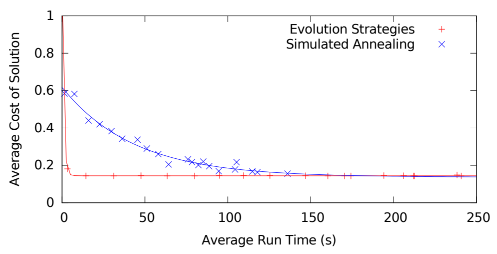

Fig. 5. Time/quality trade off for different optimization strategies.

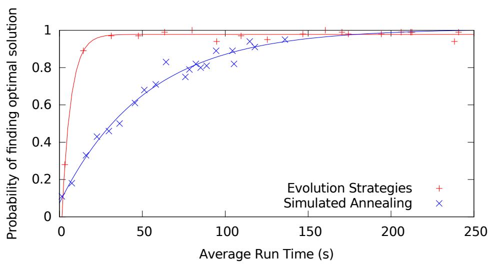

Fig. 6. Probability of finding the optimal solution within a given time.

We can also see that a CCA threshold of -80 dBm would violate the constraint by also dropping below 0.85 in the first interval, hence -81 dBm can be actually assumed to be the most energy conserving configuration that is able to fulfill the constraint.

We now use our framework to configure a network of seven sensor nodes connected to a single gateway node in a star topology. All nodes are exposed to different temperature profiles following the ones recorded on different building façades in Madrid. To configure the network, we employ the user-defined requirements introduced in Sect. V. For this scenario, the framework suggests the use of a CCA threshold of -83 dBm. Based on the results reported in Fig. 4, it can be seen that for the selected CCA threshold, the average PRR stays above 0.85 at all times. For CCA thresholds above -80 dBm, this constraint is clearly violated. Nevertheless, with a threshold of -82 dBm or higher, the PRR drops below 0.95 for at least three out of the 18 intervals, which indicates that the second constraint cannot be met. Consequently, the tool actually picked the best possible CCA value for the given scenario. This demonstrates that our tool is capable to generate useful configurations for realistic deployment scenarios within the selected application area and is able to handle the more complex requirements of typical users.

B. Performance

We now evaluate the basic performance of the system and measure the relative performance of the different optimization algorithms. Employing the same model and similar settings as in the previous section, a globally optimal solution can be found with a CCA threshold of -84 dBm and a cost value $f = 0.14344$.

For the evaluation, we execute the parametrization process with both available optimization strategies. To evaluate different time/quality trade-offs, we artificially limit the maximal number of iterations. To reduce the effect of random events, each algorithm is executed 100 times for each setting.

Fig. 5 presents the trade-off between the runtime of the optimization algorithms and the average quality of the solutions found. To illustrate the observed trend, an exponential function ($x \mapsto a * e^{b*x} + c$) has been fitted to the raw data. It can be seen that with very short run times, the optimal solution is only found in a limited number of runs and the returned solutions are often far away from the optimal one. With a run-time of two minutes or more, both algorithms are very likely to identify the optimal configuration. Incorrect solutions tend to be close to the optimal solution, as shown in Fig. 6. In most applications, finding a good solution that satisfies all constraints is already sufficient. Due to a rather small number of nodes and a limited set of configuration options in this case-study, exhaustive search is also still a feasible option, but its performance degrades quickly if the number of possible configurations is increased.

Both optimization algorithms performed well in this scenario and a run-time in the order of minutes coupled with a high reliability enable an efficient use within a typical WSN development process. For the given model, evolutionary strategies exhibit a slightly superior performance, but both strategies are able to generate suitable configurations within reasonable time.

C. Extensibility

Even though the presented evaluation is limited to a single use case and protocol, the framework can be applied in several other scenarios. As part of the RELYonIT project [10], we have indeed implemented protocol models for a number of additional protocols:

1) A model for TempMAC, a temperature-aware extension of ContikiMAC [11]. This model is essentially an extension of the one introduced in Sect. V;
2) A model capturing the impact of duty cycle configuration on the energy expenditure of a MAC protocol. This model can be used to find the best duty cycle for a specific radio environment and therefore especially targets indoor deployments where radio interference tends to significantly affect network operation;
3) A model to aid the configuration of the jamming-based agreement (JAG) protocol [17]. This model can be used to determine the jamming period length that yields an optimal agreement probability in environments prone to external radio interference;
4) A model to aid the selection of an optimal packet length to minimize energy consumption and reduce latency.

Within RELYonIT the framework has also been applied to determine the configuration of WSN-based smart parking systems in densely populated urban areas where both temperature and radio interference influence the network operation.

VII. Related Work

Support systems for WSN configuration are a surprisingly rarely-considered aspect of deployment support. Few systems exist that support users with the complex task of finding optimal configuration parameters for a given environment.

Existing approaches often rely on simulation [18], [19] and systematically try out different possible configurations in an emulated environment. With existing simulation environments, this is usually a time-consuming task as the high-fidelity models require significant processing power and consequently only allow limited speed-up for larger networks. The long computation times significantly limit the number of configurations that can be evaluated and easily lead to sub-optimal configurations. While our approach shares the same basic strategy, we can significantly reduce the run-time of the model evaluation by using more abstract formal models. This allows to evaluate a larger number of possible configurations and thus increases the likelihood of finding an optimal configuration.

Only a very small number of works apply formal models and mathematical optimization to WSN protocol configuration. A well-known example is the ptunes system developed by Zimmerling et al. [20]. Ptunes employs a formal protocol model and constraint programming to find optimal MAC protocol configurations settings for a specific network topology and radio environment. Ptunes' goals are very similar to our approach, but at least in its current form, ptunes is limited to MAC protocol configuration, while our approach targets protocols at different levels of the network stack. More importantly, ptunes does not explicitly model any environmental effects and only considers internal interference. Instead, ptunes is intended to work online and constantly reconfigure the network, which allows to constantly adapt the configuration to a changing environment, but significantly limits the available run-time for optimization. Our approach of pre-deployment configuration can use more sophisticated models that require a higher run-time, but leads to more precise and dependable results.

Our work builds on simulated annealing and evolution strategies to implement the actual optimization. While our implementation of simulated annealing follows common design strategies found in relevant textbooks, our implementation of evolution strategies employs ideas from Reehuis and Bäck [14] to support integer and floating point parameters.

VIII. Conclusion

In this paper, we introduced a novel framework for the automatic configuration of IoT communication protocols based on different models and user requirements. Our experimental evaluation demonstrates the feasibility of our approach for an exemplary application and an adequate performance of the current prototype. In the future we intend to implement additional protocol models and employ the framework in additional case studies. This will allow us to further assess the performance of the system and to identify means to increase the flexibility of the framework. Ultimately, we intend to enable the configuration of typical WSN and IoT software-stacks without requiring extensive expertise in the area.

Acknowledgment

The research leading to these results has received funding from the European Union 7^{th} Framework Programme (FP7-ICT-2011-8) under grant agreement n° 317826 (RELYonIT).

References

[1] F. J. Oppermann, C. A. Boano, and K. Römer, "A decade of wireless sensing applications: Survey and taxonomy," in *The Art of Wireless Sensor Networks*. Springer, 2014.

[2] C. A. Boano, J. Brown, N. Tsiftes, U. Roedig, and T. Voigt, "The impact of temperature on outdoor industrial sensornet applications," *IEEE Trans. Industr. Inform.*, vol. 6, no. 3, 2010.

[3] H. Wennerström, F. Hermans, O. Rensfelt, C. Rohner, and L.-Å. Nordén, "A long-term study of correlations between meteorological conditions and 802.15.4 link performance," in *Proc. of the* 10^{th} *IEEE SECON Conf.*, 2013.

[4] C. A. Boano et al., "Hot Packets: A systematic evaluation of the effect of temperature on low power wireless transceivers," in *Proc. of the* 5^{th} *ExtremeCom Conf.*, 2013.

[5] K. G. Langendoen, A. Baggio, and O. W. Visser, "Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture," in *Proc. of the* 14^{th} *WPDRTS Workshop*, 2006.

[6] N. Finne, J. Eriksson, A. Dunkels, and T. Voigt, "Experiences from two sensor network deployments – self-monitoring and self-configuration keys to success," in *Proc. of the* 6^{th} *WWIC Conf.*, 2008.

[7] D. M. Doolin and N. Sitar, "Wireless sensors for wildfire monitoring," in *Proc. of the SPIE Symposium*, 2005.

[8] S. Kim, S. Pakzad, D. E. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Health monitoring of civil infrastructures using wireless sensor networks," in *Proc. of the* 6^{th} *IPSN Conf.*, 2007.

[9] M. Ceriotti et al., "Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment," in *Proc. of the* 8^{th} *IPSN Conf.*, 2009.

[10] RELYonIT Consortium, "RELYonIT: Research by Experimentation for Dependability on the Internet of Things," 2015. [Online]. Available: http://www.relyonit.eu

[11] C. A. Boano, K. Römer, and N. Tsiftes, "Mitigating the adverse effects of temperature on low-power wireless protocols," in *Proc. of the* 11^{th} *MASS Conf.*, 2014.

[12] A. Dunkels, "The ContikiMAC radio duty cycling protocol," Swedish Institute of Computer Science, Tech. Rep. T2011:13, 2011.

[13] A. E. Smith and D. W. Coit, "Penalty functions," in *Handbook of Evolutionary Computation*. IOP Publishing Ltd., 1997.

[14] E. Reehuis and T. Bäck, "Mixed-integer evolution strategy using multiobjective selection applied to warehouse design optimization," in *Proc. of the 12th GECCO Conf.*, 2010.

[15] M. A. Zúñiga and B. Krishnamachari, "Analyzing the transitional region in low-power wireless links," in *Proc. of the* 1^{st} *SECON Conf.*, 2004.

[16] C. A. Boano, M. Zúñiga, J. Brown, U. Roedig, C. Keppitiyagama, and K. Römer, "TempLab: A testbed infrastructure to study the impact of temperature on wireless sensor networks," in *Proc. of the* 13^{th} *IPSN Conf.*, 2014.

[17] C. A. Boano, M. A. Zúñiga, K. Römer, and T. Voigt, "JAG: Reliable and predictable wireless agreement under external radio interference," in *Proc. of the* 33^{rd} *IEEE RTSS Symp.*, 2012.

[18] G. Simon, P. Volgyesi, M. Maróti, and A. Lédeczi, "Simulation-based optimization of communication protocols for large-scale wireless sensor networks," in *Proc. of the IEEE Aerospace Conf.*, 2003.

[19] M. Strübe, F. Lukas, B. Li, and R. Kapitza, "DrySim: Simulation-aided deployment-specific tailoring of mote-class WSN software," in *Proc. of the 17th ACM MSWiM Conf.*, 2014.

[20] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele, "pTunes: Runtime parameter adaptation for low-power MAC protocols," in *Proc. of the* 11^{th} *IPSN Conf.*, 2012.

Bibliography

Abdelzaher, T.; Blum, B.; Cao, Q.; Chen, Y.; Evans, D.; George, J.; George, S.; Gu, L.; He, T.; Krishnamurthy, S.; Luo, L.; Son, S.; Stankovic, J.; Stoleru, R.; and Wood, A. (2004). "EnviroTrack: Towards an Environmental Computing Paradigm for Distributed Sensor Networks." In: *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, pp. 582–589. ISBN: 0-7695-2086-3. DOI: 10.1109/ICDCS.2004.1281625.

Akkaya, Kemal and Younis, Mohamed (2005). "A Survey on Routing Protocols for Wireless Sensor Networks." In: *Ad Hoc Networks* 3.3, pp. 325–349. DOI: 10.1016/j.adhoc.2003.09.010.

Akyildiz, Ian F. and Kasimoglu, Ismail H. (2004). "Wireless Sensor and Actor Networks: Research Challenges." In: *Ad Hoc Networks* 2.4, pp. 351–367. ISSN: 1570-8705. DOI: 10.1016/j.adhoc.2004.04.003.

Al Nahas, Beshr; Duquennoy, Simon; Iyer, Venkatraman; and Voigt, Thiemo (2014). "Low-Power Listening Goes Multi-channel." In: *Proceedings of the 2014 IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*. (Marina Del Rey, CA, USA). IEEE, pp. 2–9. ISBN: 978-1-4799-4618-1. DOI: 10.1109/DCOSS.2014.33.

Aleti, Aldeida; Buhnova, Barbora; Grunske, Lars; Koziolek, Anne; and Meedeniya, Indika (2013). "Software Architecture Optimization Methods: A Systematic Literature Review." In: *IEEE Transactions on Software Engineering* 39.5, pp. 658–683. ISSN: 0098-5589. DOI: 10.1109/TSE.2012.64.

Arora, A.; Dutta, P.; Bapat, S.; Kulathumani, V.; Zhang, H.; Naik, V.; Mittal, V.; Cao, H.; Demirbas, M.; Gouda, M.; Choi, Y.; Herman, T.; Kulkarni, S.; Arumugam, U.; Nesterenko, M.; Vora, A.; and Miyashita, M. (2004). "A Line in the Sand: A Wireless Sensor Network for Target Detection, Classification, and Tracking." In: *Computer Networks* 46.5, pp. 605–634. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2004.06.007.

Aslam, Faisal; Fennell, Luminous; Schindelhauer, Christian; Thiemann, Peter; Ernst, Gidon; Haussmann, Elmar; Rührup, Stefan; and Uzmi, Zastash A. (2010). "Optimized Java Binary and Virtual Machine for Tiny Motes." In: *Proceedings of the 6th IEEE International Conference on Distributed Computing in Sensor Systems*

(DCOSS). (Santa Barbara, CA). Springer, pp. 15–30. ISBN: 978-3-642-13650-4. DOI: 10.1007/978-3-642-13651-1_2.

Bagree, Ravi; Jain, VishwasRaj; Kumar, Aman; and Ranjan, Prabhat (2010). "TigerCENSE: Wireless Image Sensor Network to Monitor Tiger Movement." In: *Proceedings of the 4th International Workshop on Real-World Wireless Sensor Networks (REALWSN).* (Colombo, Sri Lanka). Springer, pp. 13–24. ISBN: 978-3-642-17519-0. DOI: 10.1007/978-3-642-17520-6_2.

Bakshi, Amol; Ou, Jingzhao; and Prasanna, Viktor K. (2002). "Towards Automatic Synthesis of a Class of Application-specific Sensor Networks." In: *Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES).* (Grenoble, France). ACM, pp. 50–58. ISBN: 1-58113-575-0. DOI: 10.1145/581630.581639.

Balsamo, Simonetta; Di Marco, Antinisca; Inverardi, Paola; and Simeoni, Marta (2004). "Model-Based Performance Prediction in Software Development: A Survey." In: *IEEE Transactions on Software Engineering* 30.5, pp. 295–310. ISSN: 0098-5589. DOI: 10.1109/TSE.2004.9.

Barik, Runa and Divakaran, Dinil Mon (2013). "Evolution of TCP's initial window size." In: *Proceedings of the 38th IEEE Conference on Local Computer Networks (LCN).* (Sydney, Australia), pp. 500–508. ISBN: 978-1-4799-0536-2. DOI: 10.1109/LCN.2013.6761284.

Baronti, Paolo; Pillai, Prashant; Chook, Vince W. C.; Chessa, Stefano; Gotta, Alberto; and Hu, Y. Fun (2007). "Wireless Sensor Networks: A Survey on the State of the Art and the 802.15.4 and ZigBee Standards." In: *Computer Communications* 30.7, pp. 1655–1695. ISSN: 0140-3664. DOI: 10.1016/j.comcom.2006.12.020.

Barrenetxea, Guillermo; Ingelrest, François; Schaefer, Gunnar; and Vetterli, Martin (2008). "The Hitchhiker's Guide to Successful Wireless Sensor Network Deployments." In: *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys).* (Raleigh, NC, USA). ACM, pp. 43–56. ISBN: 978-1-59593-990-6. DOI: 10.1145/1460412.1460418.

Batalin, Maxim A.; Sukhatme, Gaurav S.; and Hattig, Myron (2004). "Mobile Robot Navigation Using a Sensor Network." In: *Proceedings of the 2004 IEEE International Conference on Robotics and Automation (ICRA).* (New Orleans, LA, USA). IEEE, pp. 636–641. ISBN: 0-7803-8232-3. DOI: 10.1109/ROBOT.2004.1307220.

Baumgartner, Tobias; Chatzigiannakis, Ioannis; Fekete, Sándor; Koninis, Christos; Kröller, Alexander; and Pyrgelis, Apostolos (2010). "Wiselib: A Generic Algorithm Library for Heterogeneous Sensor Networks." In: *Proceedings of the 7th European Conference on Wireless Sensor Networks (EWSN).* (Coimbra, Portugal). Springer, pp. 162–177. ISBN: 978-3-642-11916-3. DOI: `10.1007/978-3-642-11917-0_11`.

Becker, Steffen; Koziolek, Heiko; and Reussner, Ralf (2009). "The Palladio Component Model for Model-driven Performance Prediction." In: *Journal of Systems and Software* 82.1, pp. 3–22. ISSN: 0164-1212. DOI: `10.1016/j.jss.2008.03.066`.

Beutel, Jan; Gruber, Stephan; Hasler, Andreas; Lim, Roman; Meier, Andreas; Plessl, Christian; Talzi, Igor; Thiele, Lothar; Tschudin, Christian; Woehrle, Matthias; and Yuecel, Mustafa (2009a). "Demo Abstract: Operating a Sensor Network at 3500m Above Sea Level." In: *Proceedings of the 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN).* (San Francisco, CA, USA). IEEE, pp. 405–406. ISBN: 978-1-4503-3475-4.

Beutel, Jan; Römer, Kay; Ringwald, Matthias; and Woehrle, Matthias (2009b). "Deployment Techniques for Wireless Sensor Networks." In: *Sensor Networks: Where Theory Meets Practice.* Ed. by Gianluigi Ferrari. Springer. ISBN: 978-3-642-01341-6.

Bijwaard, Dennis J. A.; van Kleunen, Wouter A. P.; Havinga, Paul J. M.; Kleiboer, Leon; and Bijl, Mark J. J. (2011). "Industry: Using Dynamic WSNs in Smart Logistics for Fruits and Pharmacy." In: *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys).* (Seattle, WA, USA). ACM, pp. 218–231. ISBN: 978-1-4503-0718-5. DOI: `10.1145/2070942.2070965`.

Boano, Carlo Alberto; Oppermann, Felix Jonathan; and Römer, Kay (2013). "The Use of Body Sensor Networks in Clinical Settings and Medical Research." In: *Sensor Networks for Sustainable Development.* Ed. by Mohammad Ilyas; Sami S. Al-Wakeel; Mohammed M. Alwakeel; and El-Hadi M. Aggoune. CRC Press, pp. 215–252. ISBN: 978-1466582064.

Boano, Carlo Alberto; Römer, Kay; and Tsiftes, Nicolas (2014). "Mitigating the Adverse Effects of Temperature on Low-Power Wireless Protocols." In: *Proceedings of the 11th International Conference on Mobile Ad hoc and Sensor Systems (MASS).* (Philadelphia, PE, USA). IEEE, pp. 336–344. ISBN: 978-1-4799-6035-4. DOI: `10.1109/MASS.2014.14`.

Boano, Carlo Alberto; Wennerström, Hjalmar; Zúñiga, Marco Antonio; Brown, James; Keppitiyagama, Chamath; Oppermann, Felix Jonathan; Roedig, Utz; Nordén, Lars-Åke; Voigt, Thiemo; and Römer, Kay (2013). "Hot Packets: A Systematic Evaluation of the Effect of Temperature on Low Power Wireless Transceivers." In: *Proceedings of the 5th Extreme Conference on Communication (ExtremeCom).* (Thórsmörk, Iceland). ACM, pp. 7–12. ISBN: 978-1-4503-2171-6.

Boano, Carlo Alberto; Zúñiga, Marco; Brown, James; Roedig, Utz; Keppitiyagama, Chamath; and Römer, Kay (2014). "TempLab: A Testbed Infrastructure to Study the Impact of Temperature on Wireless Sensor Networks." In: *Proceedings of the 13th International Conference on Information Processing in Sensor Networks (IPSN).* (Berlin, Germany). IEEE, pp. 95–106. ISBN: 978-1-4799-3146-0. DOI: 10.1109/IPSN.2014.6846744.

Bocchino, Stefano; Fedor, Szymon; and Petracca, Matteo (2015). "PyFUNS: A Python Framework for Ubiquitous Networked Sensors." In: *Proceedings of the 12th European Conference on Wireless Sensor Networks (EWSN).* (Porto, Portugal). Springer, pp. 1–18. ISBN: 978-3-319-15581-4. DOI: 10.1007/978-3-319-15582-1_1.

Brouwers, Niels; Corke, Peter; and Langendoen, Koen (2008). "Darjeeling, a Java Compatible Virtual Machine for Microcontrollers." In: *Proceedings of the 9th ACM/IFIP/USENIX Middleware Conference.* (Leuven, Belgium). ACM, pp. 18–23. ISBN: 978-1-60558-369-3. DOI: 10.1145/1462735.1462740.

Brown, James; Roedig, Utz; Boano, Carlo Alberto; and Römer, Kay (2014). "Estimating Packet Reception Rate in Noisy Environments." In: *Workshop Proceedings of the 39th IEEE Conference on Local Computer Networks (LCN).* (Edmonton, AB, Canada). IEEE, pp. 583–591. ISBN: 978-1-4799-3782-0. DOI: 10.1109/LCNW.2014.6927706.

Brunnert, Andreas; Danciu, Alexandru; Vögele, Christian; Tertilt, Daniel; and Krcmar, Helmut (2013). "Integrating the Palladio-Bench into the Software Development Process of a SOA Project." In: *Proceedings of the Symposium on Software Performance: Joint Kieker/Palladio Days 2013 (KPDAYS).* (Karlsruhe, Germany).

Buettner, Michael; Yee, Gary V.; Anderson, Eric; and Han, Richard (2006). "X-MAC: A Short Preamble MAC Protocol for Duty-cycled Wireless Sensor Networks." In: *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys).* (Boulder, CO, USA). ACM, pp. 307–320. ISBN: 1-59593-343-3. DOI: 10.1145/1182807.1182838.

Buonadonna, Phil; Gay, David; Hellerstein, Joseph M.; Hong, Wei; and Madden, Samuel (2005). "TASK: Sensor Network in a Box." In: *Proceedings of the 2nd European Workshop on Sensor Networks (EWSN).* (Istanbul, Turkey). IEEE, pp. 133–144. ISBN: 0-7803-8801-1. DOI: 10.1109/EWSN.2005.1462005.

Burrell, Jenna; Brooke, Tim; and Beckwith, Richard (2004). "Vineyard Computing: Sensor Networks in Agricultural Production." In: *IEEE Pervasive Computing* 3.1, pp. 38–45. ISSN: 1536-1268. DOI: 10.1109/MPRV.2004.1269130.

Casati, Fabio; Daniel, Florian; Dantchev, Guenadi; Eriksson, Joakim; Finne, Niclas; Karnouskos, Stamatis; Montero, Patricio Moreno; Mottola, Luca; Oppermann, Felix Jonathan; Picco, Gian Pietro; Quartulli, Antonio; Römer, Kay; Spiess, Patrik; Tranquillini, Stefano; and Voigt, Thiemo (2012). “Towards Business Processes Orchestrating the Physical Enterprise with Wireless Sensor Networks.” In: *Proceedings of the 34th International Conference on Software Engineering (ICSE).* (Zurich, Switzerland). IEEE, pp. 1357–1360. ISBN: 978-1-4673-1066-6. DOI: `10.1109/ICSE.2012.6227080`.

Ceriotti, Matteo; Corra, Michele; D’Orazio, Leandro; Doriguzzi, Roberto; Facchin, Daniele; Guna, Stefan T.; Jesi, Gian Paolo; Lo Cigno, Renato; Mottola, Luca; Murphy, Amy L.; Pescalli, Massimo; Picco, Gian Pietro; Pregnolato, Denis; and Torghele, Carloalberto (2011). “Is There Light at the Ends of the Tunnel? Wireless Sensor Networks for Adaptive Lighting in Road Tunnels.” In: *Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN).* (Chicago, IL, USA). IEEE, pp. 187–198. ISBN: 978-1-61284-854-9.

Chatterjee, Krishnendu; Pavlogiannis, Andreas; Kößler, Alexander; and Schmid, Ulrich (2014). “A Framework for Automated Competitive Analysis of On-line Scheduling of Firm-Deadline Tasks.” In: *Proceedings of the 35th IEEE Real-Time Systems Symposium (RTSS).* (Rome, Italy). IEEE, pp. 118–127. DOI: `10.1109/RTSS.2014.9`.

Chu, David; Popa, Lucian; Tavakoli, Arsalan; Hellerstein, Joseph M.; Levis, Philip; Shenker, Scott; and Stoica, Ion (2007). “The Design and Implementation of a Declarative Sensor Network System.” In: *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys).* (Sydney, Australia). ACM, pp. 175–188. ISBN: 978-1-59593-763-6. DOI: `10.1145/1322263.1322281`.

Clements, Paul and Northrop, Linda M. (2006). *Software Product Lines: Practices and Patterns.* 6th ed. Pearson Education. ISBN: 978-0201703320.

Coleri, Sinem; Ergen, Mustafa; and Koo, T. John (2002). “Lifetime Analysis of a Sensor Network with Hybrid Automata Modelling.” In: *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA).* (Atlanta, GA, USA). ACM, pp. 98–104. ISBN: 1-58113-589-0. DOI: `10.1145/570738.570752`.

Culler, David E. (2006). “TinyOS: Operating System Design for Wireless Sensor Networks.” In: *Sensors* 23.5, pp. 14–20.

Culler, David E.; Hill, Jason; Buonadonna, Philip; Szewczyk, Robert; and Woo, Alec (2001). “A Network-Centric Approach to Embedded Software for Tiny Devices.” In: *Embedded Software.* Ed. by Thomas A. Henzinger and Christoph M. Kirsch.

Vol. 2211. Lecture Notes in Computer Science. Springer, pp. 114–130. ISBN: 978-3-540-42673-8. DOI: 10.1007/3-540-45449-7_9.

van Dam, Tijs and Langendoen, Koen (2003). "An Adaptive Energy-efficient MAC Protocol for Wireless Sensor Networks." In: *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys).* (Los Angeles, CA, USA). ACM, pp. 171–180. ISBN: 1-58113-707-9. DOI: 10.1145/958491.958512.

Daniel, Florian; Eriksson, Joakim; Finne, Niclas; Fuchs, Harald; Gaglione, Andrea; Karnouskos, Stamatis; Montero, Patricio Moreno; Mottola, Luca; Oppermann, Felix Jonathan; Picco, Gian Pietro; Römer, Kay; Spieß, Patrik; Tranquillini, Stefano; and Voigt, Thiemo (2013). "makeSense: Real-world Business Processes through Wireless Sensor Networks." In: *Proceedings of 4th International Workshop on Networks of Cooperating Objects for Smart Cities.* (Philadelphia, PA, USA), pp. 58–72.

Deligiannis, Nikos; Mota, João F. C.; Smart, George; and Andreopoulos, Yiannis (2015). "Decentralized Multichannel Medium Access Control: Viewing Desynchronization As a Convex Optimization Method." In: *Proceedings of the 14th International Conference on Information Processing in Sensor Networks (IPSN).* (Seattle, WA, USA). ACM, pp. 13–24. ISBN: 978-1-4503-3475-4. DOI: 10.1145/2737095.2737108.

Diaz, Javier; Muñoz-Caro, Camelia; and Niño, Alfonso (2012). "A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era." In: *IEEE Transactions on Parallel and Distributed Systems* 23.8, pp. 1369–1386. ISSN: 1045-9219. DOI: 10.1109/TPDS.2011.308.

Doerig, Karl (1998). *Espresso, A Java Compiler Written in Java.* Tech. rep. Boston University. URL: http://types.bu.edu/Espresso/report/espresso.pdf.

Dunkels, Adam (2011). *The ContikiMAC Radio Duty Cycling Protocol.* Tech. rep. T2011:13. Swedish Institute of Computer Science. URL: http://dunkels.com/adam/dunkels11contikimac.pdf.

Dunkels, Adam; Grönvall, Björn; and Voigt, Thiemo (2004). "Contiki – a Lightweight and Flexible Operating System for Tiny Networked Sensors." In: *Proceedings of the 1st IEEE Workshop on Embedded Networked Sensors (EmNets).* (Tampa, FL, USA). IEEE, pp. 455–462. ISBN: 0-7695-2260-2. DOI: 10.1109/LCN.2004.38.

Dunkels, Adam; Österlind, Fredrik; and He, Zhitao (2007). "An Adaptive Communication Architecture for Wireless Sensor Networks." In: *Proceedings of the 5th ACM Conference on Embedded Networked Sensor Systems (SenSys).* (Sydney, Australia). ACM, pp. 335–349. ISBN: 978-1-59593-763-6. DOI: 10.1145/1322263.1322295.

Dunkels, Adam; Feeney, Laura Marie; Grönvall, Björn; and Voigt, Thiemo (2004). "An Integrated Approach to Developing Sensor Network Solutions." In: *Proceedings of the Second International Workshop on Sensor and Actor Network Protocols and Applications (SANPA) in conjunction with the 13th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous).* (Boston, MA, USA).

Dunkels, Adam; Schmidt, Oliver; Voigt, Thiemo; and Ali, Muneeb (2006). "Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems." In: *Proceedings of the 4th ACM Conference on Embedded Networked Sensor Systems (SenSys).* (Boulder, CO, USA). ACM, pp. 29–42. ISBN: 1-59593-343-3. DOI: 10.1145/1182807.1182811.

Duquennoy, Simon; Landsiedel, Olaf; and Voigt, Thiemo (2013). "Let the Tree Bloom: Scalable Opportunistic Routing with ORPL." In: *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys).* (Rome, Italy). ACM, 2:1–2:14. ISBN: 978-1-4503-2027-6. DOI: 10.1145/2517351.2517369.

Duquennoy, Simon; Al Nahas, Beshr; Landsiedel, Olaf; and Watteyne, Thomas (2015). "Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH." In: *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems (SenSys).* (Seoul, South Korea). ACM, pp. 337–350. ISBN: 978-1-4503-3631-4. DOI: 10.1145/2809695.2809714.

Durvy, Mathilde; Abeillé, Julien; Wetterwald, Patrick; O'Flynn, Colin; Leverett, Blake; Gnoske, Eric; Vidales, Michael; Mulligan, Geoff; Tsiftes, Nicolas; Finne, Niclas; and Dunkels, Adam (2008). "Making Sensor Networks IPv6 Ready." In: *Proceedings of the 6th ACM Conference on Networked Embedded Sensor Systems (SenSys).* (Raleigh, NC, USA). ACM, pp. 421–422. ISBN: 978-1-59593-990-6. DOI: 10.1145/1460412.1460483.

Dutta, Prabal; Dawson-Haggerty, Stephen; Chen, Yin; Liang, Chieh-Jan Mike; and Terzis, Andreas (2010). "Design and Evaluation of a Versatile and Efficient Receiver-initiated Link Layer for Low-power Wireless." In: *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys).* (Zurich, Switzerland). ACM, pp. 1–14. ISBN: 978-1-4503-0344-6. DOI: 10.1145/1869983.1869985.

Dyer, Matthias; Beutel, Jan; Kalt, Thomas; Oehen, Patrice; Thiele, Lothar; Martin, Kevin; and Blum, Philipp (2007). "Deployment Support Network: A Toolkit for the Development of WSNs." In: *Proceedings of the 4th European Conference on Wireless Sensor Networks (EWSN).* (Delft, Netherlands). Springer, pp. 195–211. ISBN: 978-3-540-69829-6.

Dyo, Vladimir; Ellwood, Stephen A.; Macdonald, David W.; Markham, Andrew; Mascolo, Cecilia; Pásztor, Bence; Trigoni, Niki; and Wohlers, Ricklef (2009). "Wildlife and Environmental Monitoring Using RFID and WSN Technology." In: *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys).* (Berkeley, CA, USA). ACM, pp. 371–372. ISBN: 978-1-60558-519-2. DOI: `10.1145/1644038.1644107`.

Dyo, Vladimir; Ellwood, Stephen A.; Macdonald, David W.; Markham, Andrew; Mascolo, Cecilia; Pásztor, Bence; Scellato, Salvatore; Trigoni, Niki; Wohlers, Ricklef; and Yousef, Kharsim (2010). "Evolution and Sustainability of a Wildlife Monitoring Sensor Network." In: *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys).* (Zurich, Switzerland). ACM, pp. 127–140. ISBN: 978-1-4503-0344-6. DOI: `10.1145/1869983.1869997`.

El-Hoiydi, Amre (2002). "Aloha with Preamble Sampling for Sporadic Traffic in Ad Hoc Wireless Sensor Networks." In: *Proceedings of the 2002 IEEE International Conference on Communications (ICC).* (New York, NY, USA). IEEE, pp. 3418–3423. ISBN: 0-7803-7400-2. DOI: `10.1109/ICC.2002.997465`.

El-Hoiydi, Amre and Decotignie, Jean-Dominique (2004). "WiseMAC: An Ultra Low Power MAC Protocol for Multi-hop Wireless Sensor Networks." In: *Proceedings of the 1st International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS).* (Turku, Finland). Springer, pp. 18–31. ISBN: 978-3-540-22476-1. DOI: `10.1007/978-3-540-27820-7_4`.

Eriksson, Joakim; Österlind, Fredrik; Finne, Niclas; Tsiftes, Nicolas; Dunkels, Adam; Voigt, Thiemo; Sauter, Robert; and Marrón, Pedro José (2006). "Cross-Level Sensor Network Simulation with COOJA." In: *Proceedings of the 31st IEEE Conference on Local Computer Networks (LCN).* (Tampa, FL, USA), pp. 641–648. ISBN: 1-4244-0418-5. DOI: `10.1109/LCN.2006.322172`.

Eriksson, Joakim; Finne, Nicla; Mottola, Luca; Voigt, Thiemo; Pettinato, Paolo; Casati, Fabio; Daniel, Florian; Gaglione, Andrea; Molteni, Davide; Picco, Gian Pietro; Quartulli, Antonio; Tranquillini, Stefano; Oppermann, Felix Jonathan; Römer, Kay; Dantchev, Guenadi; Karnouskos, Stamatis; Spieß, Patrik; and Montero, Patricio Moreno (2012a). *D-5.1 – Initial Integrated System.* Tech. rep. `make`*Sense*: Easy Programming of Integrated Wireless Sensor Networks.

Eriksson, Joakim; Finne, Niclas; Mottola, Luca; Voigt, Thiemo; Casati, Fabio; Daniel, Florian; Gaglione, Andrea; Molteni, Davide; Picco, Gian Pietro; Quartulli, Antonio; Tranquillini, Stefano; Oppermann, Felix Jonathan; Römer, Kay; Fuchs, Harald; Oertel, Nina; Spiess, Patrik; and Montero, Patricio Moreno (2012b). *D-5.2 – Report from Application Implementations and Evaluation.* Tech. rep. `make`*Sense*: Easy Programming of Integrated Wireless Sensor Networks.

Eriksson, Joakim; Finne, Niclas; Mottola, Luca; Voigt, Thiemo; Casati, Fabio; Daniel, Florian; Gaglione, Andrea; Molteni, Davide; Picco, Gian Pietro; Tranquillini, Stefano; Holländer, Bengt-Ove; Oppermann, Felix Jonathan; Römer, Kay; Doeweling, Sebastian; Oertel, Nina; Probst, Florian; Spieß, Patrik; and Montero, Patricio Moreno (2013). *D-3.6 & D-5.4 – Final application implementations and evaluation of system & Final evaluation of the programming model.* Tech. rep. `make`*Sense*: Easy Programming of Integrated Wireless Sensor Networks.

Evans, Dave (2011). *The Internet of Things – How the Next Evolution of the Internet Is Changing Everything.* White Paper. Cisco Internet Business Solutions Group (IBSG). URL: `http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf`.

Ferrari, Federico; Zimmerling, Marco; Thiele, Lothar; and Saukh, Olga (2011). "Efficient Network Flooding and Time Synchronization with Glossy." In: *Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN).* (Chicago, IL, USA). IEEE, pp. 73–84. ISBN: 978-1-61284-854-9.

Flury, Roland and Wattenhofer, Roger R. (2008). "Randomized 3D Geographic Routing." In: *Proceedings of the 27th Annual IEEE Conference on Computer Communications (INFOCOM).* (Phoenix, AZ, USA). DOI: `10.1109/INFOCOM.2008.135`.

Frank, Christian and Römer, Kay (2005). "Algorithms for Generic Role Assignment in Wireless Sensor Networks." In: *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys).* (San Diego, CA, USA). ACM, pp. 230–242. ISBN: 1-59593-054-X. DOI: `10.1145/1098918.1098944`.

Gaglione, Andrea; Molteni, Davide; Picco, Gian Pietro; Oppermann, Felix Jonathan; and Römer, Kay (2013). *D-3.4/D-3.5 – Final* `make`*Sense Programming Model and Final Language Core Implementation.* Tech. rep. `make`*Sense*: Easy Programming of Integrated Wireless Sensor Networks.

Gamma, Erich; Helm, Richard; Johnson, Ralph; and Vlissides, John (1995). *Design Patterns.* Addison Wesley. ISBN: 0-201-63361-2.

Ganti, Raghu K.; Pham, Nam; Ahmadi, Hossein; Nangia, Saurabh; and Abdelzaher, Tarek F. (2010). "GreenGPS: A Participatory Sensing Fuel-efficient Maps Application." In: *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys).* (San Francisco, CA, USA). ACM, pp. 151–164. ISBN: 978-1-60558-985-5. DOI: `10.1145/1814433.1814450`.

Gay, David; Levis, Philip; von Behren, Robert; Welsh, Matt; Brewer, Eric; and Culler, David (2003). "The nesC Language: A Holistic Approach to Networked Embedded Systems." In: *Proceedings of the ACM SIGPLAN 2003 Conference on*

Programming Language Design and Implementation (PLDI). (San Diego, CA, USA). ACM, pp. 1–11. ISBN: 1-58113-662-5. DOI: 10.1145/781131.781133.

Gosling, James; Joy, Bill; Steele, Guy; and Bracha, Gilad (2005). *The Java™ Language Specification*. 3rd. Addison-Wesley. ISBN: 978-0321246783.

Gummadi, Ramakrishna; Gnawali, Omprakash; and Govindan, Ramesh (2005). "Macro-programming Wireless Sensor Networks Using Kairos." In: *Proceedings of the 1st IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*. (Marina del Rey, CA, USA). Springer, pp. 126–140. ISBN: 3-540-26422-1. DOI: 10.1007/11502593_12.

Gună, Ştefan; Mottola, Luca; and Picco, Gian Pietro (2014). "DICE: Monitoring Global Invariants with Wireless Sensor Networks." In: *ACM Transactions on Sensor Networks* 10.4, 54:1–54:34. ISSN: 1550-4859. DOI: 10.1145/2509434.

Guo, Jianmei; White, Jules; Wang, Guangxin; Li, Jian; and Wang, Yinglin (2011). "A Genetic Algorithm for Optimized Feature Selection with Resource Constraints in Software Product Lines." In: *Journal of Systems and Software* 84.12, pp. 2208–2221. ISSN: 0164-1212. DOI: 10.1016/j.jss.2011.06.026.

Guttman, Erik (2001). "Autoconfiguration for IP Networking: Enabling Local Communication." In: *Internet Computing* 5.3, pp. 81–86. ISSN: 1089-7801. DOI: 10.1109/4236.935181.

Harman, M.; Jia, Y.; Krinke, J.; Langdon, W. B.; Petke, J.; and Zhang, Y. (2014). "Search Based Software Engineering for Software Product Line Engineering: A Survey and Directions for Future Work." In: *Proceedings of the 18th International Software Product Line Conference (SPLC)*. (Florence, Italy). ACM, pp. 5–18. ISBN: 978-1-4503-2740-4. DOI: 10.1145/2648511.2648513.

Hayes, Jer; Beirne, Stephen; Lau, King-Tong; and Diamond, Dermot (2008). "Evaluation of a Low Cost Wireless Chemical Sensor Network for Environmental Monitoring." In: *Proceeding of the 7th IEEE Conference on Sensors (Sensors)*. (Lecce, Italy). IEEE, pp. 530–533. ISBN: 978-1-4244-2580-8. DOI: 10.1109/ICSENS.2008.4716494.

Heinzelman, W. B.; Chandrakasan, A. P.; and Balakrishnan, H. (2002). "An Application-specific Protocol Architecture for Wireless Microsensor Networks." In: *Transactions on Wireless Communications* 1.4, pp. 660–670. ISSN: 1536-1276. DOI: 10.1109/TWC.2002.804190.

Hill, Jason; Szewczyk, Robert; Woo, Alec; Hollar, Seth; Culler, David E.; and Pister, Kristofer (2000). "System Architecture Directions for Networked Sensors." In: *ACM SIGPLAN Notices* 35.11, pp. 93–104. DOI: 10.1145/356989.356998.

Hossain, Mohammad Sajjad; Islam, A.B.M. Alim Al; Kulkarni, Milind; and Raghunathan, Vijay (2011). “μSETL: A Set Based Programming Abstraction for Wireless Sensor Networks.” In: *Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN).* (Chicago, IL, USA). IEEE, pp. 354–365. ISBN: 978-1-61284-854-9.

Hui, Jonathan W. and Culler, David E. (2004). “The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale.” In: *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys).* (Baltimore, MD, USA). ACM, pp. 81–94. ISBN: 1-58113-879-2. DOI: 10.1145/1031495.1031506.

Hui, Jonathan W. and Culler, David E. (2008). “IP is Dead, Long Live IP for Wireless Sensor Networks.” In: *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys).* (Raleigh, NC, USA). ACM, pp. 15–28. ISBN: 978-1-59593-990-6. DOI: 10.1145/1460412.1460415.

Intanagonwiwat, Chalermek; Govindan, Ramesh; and Estrin, Deborah (2000). “Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks.” In: *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom).* (Boston, MA, USA). ACM, pp. 56–67. ISBN: 1-58113-197-6. DOI: 10.1145/345910.345920.

Iyer, Venkatraman; Woehrle, Matthias; and Langendoen, Koen (2011). “Chrysso – A Multi-channel Approach to Mitigate External Interference.” In: *Proceedings of the 8th IEEE Communications Society Conference on Sensor, Mesh, and Ad Hoc Communications and Networks (SECON).* (Salt Lake City, UT, USA). IEEE, pp. 422–430. ISBN: 978-1-4577-0094-1. DOI: 10.1109/SAHCN.2011.5984929.

Jóźwiak, Lech; Nedjah, Nadia; and Figueroa, Miguel (2010). “Modern Development Methods and Tools for Embedded Reconfigurable Systems: A Survey.” In: *Integration, the VLSI Journal* 43.1, pp. 1–33. ISSN: 0167-9260. DOI: 10.1016/j.vlsi.2009.06.002.

Juang, Philo; Oki, Hidekazu; Wang, Yong; Martonosi, Margaret; Peh, Li-Shiuan; and Rubenstein, Daniel (2002). “Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet.” In: *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).* (San Jose, CA, USA). ACM, pp. 96–107. ISBN: 1-58113-574-2. DOI: 10.1145/605397.605408.

Jung, Deokwoo; Teixeira, Thiago; Barton-Sweeney, Andrew; and Savvides, Andreas (2007). “Model-Based Design Exploration of Wireless Sensor Node Lifetimes.” In: *Proceedings of the 4th European Conference on Wireless Sensor Networks*

(EWSN). (Delft, Netherlands). Springer, pp. 277–292. ISBN: 978-3-540-69829-6. DOI: 10.1007/978-3-540-69830-2_18.

Kellner, Simon; Pink, Mario; Meier, Detlev; and Blaß, Erik-Oliver (2008). “Towards a Realistic Energy Model for Wireless Sensor Networks.” In: *Proceedings of the 5th Annual Conference on Wireless On Demand Network Systems and Services (WONS).* (Garmisch-Partenkirchen, Germany). IEEE, pp. 97–100. ISBN: 978-1-4244-1959-3. DOI: 10.1109/WONS.2008.4459362.

Kim, Hyunbum and Cobb, Jorge A. (2012). “Optimization Trade-offs in the Design of Wireless Sensor and Actor Networks.” In: *Proceedings of the 37th IEEE Conference on Local Computer Networks (LCN).* (Clearwater, FL, USA). IEEE, pp. 559–567. ISBN: 978-1-4673-1565-4. DOI: 10.1109/LCN.2012.6423675.

Kim, Sukun; Pakzad, Shamim; Culler, David; Demmel, James; Fenves, Gregory; Glaser, Steven; and Turon, Martin (2007). “Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks.” In: *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN).* (Cambridge, MA, USA). ACM, pp. 254–263. ISBN: 978-1-59593-638-X. DOI: 10.1145/1236360.1236395.

King, Alex; Brown, James; Vidler, John; and Roedig, Utz (2015). “Estimating Node Lifetime in Interference Environments.” In: *Workshop Proceedings of the 40th IEEE Conference on Local Computer Networks (LCN).* (Clearwater, FL, USA). IEEE. ISBN: 978-1-4673-6772-1.

Kirkpatrick, S.; Gelatt Jr., C. D.; and Vecchi, M. P. (1983). “Optimization by Simulated Annealing.” In: *Science* 220.4598, pp. 671–680. DOI: 10.1126/science.220.4598.671.

Klues, Kevin; Xing, Guoliang; and Lu, Chenyang (2010). “Link Layer Driver Architecture for Unified Radio Power Management in Wireless Sensor Networks.” In: *ACM Transactions on Embedded Computing Systems (TECS)* 9.4, 41:1–41:28. ISSN: 1539-9087. DOI: 10.1145/1721695.1721707.

Klues, Kevin; Hackmann, Gregory; Chipara, Octav; and Lu, Chenyang (2007). “A Component-based Architecture for Power-efficient Media Access Control in Wireless Sensor Networks.” In: *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys).* (Sydney, Australia). ACM, pp. 59–72. ISBN: 978-1-59593-763-6. DOI: 10.1145/1322263.1322270.

Ko, JeongGil; Eriksson, Joakim; Tsiftes, Nicolas; Dawson-Haggerty, Stephen; Vasseur, Jean-Philippe; Durvy, Mathilde; Terzis, Andreas; Dunkels, Adam; and Culler, David (2011). “Industry: Beyond Interoperability: Pushing the Performance of Sensor Network IP Stacks.” In: *Proceedings of the 9th ACM Conference* *on Embedded Networked Sensor Systems (SenSys).* (Seattle, Washington). ACM, pp. 1–11. ISBN: 978-1-4503-0718-5. DOI: 10.1145/2070942.2070944.

Kogekar, Sachin; Neema, Sandeep; Eames, Brandon; Koutsoukos, Xenofon; Ledeczi, Akos; and Maroti, Miklos (2004). "Constraint-guided Dynamic Reconfiguration in Sensor Networks." In: *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN).* (Berkeley, CA, USA). IEEE, pp. 379–387. ISBN: 1-58113-846-6. DOI: 10.1109/IPSN.2004.1307359.

Kothari, Nupur; Gummadi, Ramakrishna; Millstein, Todd; and Govindan, Ramesh (2007). "Reliable and Efficient Programming Abstractions for Wireless Sensor Networks." In: *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI).* (San Diego, CA, USA). ACM, pp. 200–210. ISBN: 978-1-59593-633-2. DOI: 10.1145/1250734.1250757.

Kumagai, Jean (2004). "The Secret Life of Birds." In: *IEEE Spectrum* 41.4, pp. 42–48. ISSN: 0018-9235. DOI: 10.1109/MSPEC.2004.1279193.

Künzli, S.; Thiele, L.; and Zitzler, E. (2005). "Modular Design Space Exploration Framework for Embedded Systems." In: *IEE Proceedings – Computers and Digital Techniques* 152 (2), pp. 183–192. ISSN: 1350-2387. DOI: 10.1049/ip-cdt:20045081.

Lai, Farley; Hasan, Syed Shabih; Laugesen, Austin; and Chipara, Octav (2014). "CSense: A Stream-processing Toolkit for Robust and High-rate Mobile Sensing Applications." In: *Proceedings of the 13th International Symposium on Information Processing in Sensor Networks (IPSN).* (Berlin, Germany). IEEE, pp. 119–130. ISBN: 978-1-4799-3146-0. DOI: 10.1109/IPSN.2014.6846746.

Landsiedel, Olaf; Ferrari, Federico; and Zimmerling, Marco (2013). "Chaos: Versatile and Efficient All-to-all Data Sharing and In-network Processing at Scale." In: *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys).* (Rome, Italy). ACM, 1:1–1:14. ISBN: 978-1-4503-2027-6. DOI: 10.1145/2517351.2517358.

Langendoen, Koen (2008). "Medium Access Control in Wireless Sensor Networks." In: *Medium Access Control in Wireless Networks.* Ed. by Hongyi Wu and Yi Pan. Nova Science Publishers Inc. Chap. 20, pp. 535–560. ISBN: 978-1-60021-944-3.

Langendoen, Koen; Baggio, Aline; and Visser, Otto (2006). "Murphy Loves Potatoes: Experiences from a Pilot Sensor Network Deployment in Precision Agriculture." In: *Proceedings of the 20th IEEE International Parallel & Distributed Processing Symposium (IPDPS).* (Rhodes Island, Greece). IEEE Computer Society, pp. 174–181. ISBN: 1-4244-0054-6. DOI: 10.1109/IPDPS.2006.1639412.

Langendoen, Koen and Halkes, Gertjan (2005). “Energy-Efficient Medium Access Control.” In: *Embedded Systems Handbook*. Ed. by Richard Zurawski. CRC press. Chap. 34, pp. 34.1–34.29. ISBN: 978-0-8493-2824-4. DOI: `10.1201/9781420038163.ch34`.

Langendoerfer, Peter; Peter, Steffen; Piotrowski, Krzysztof; Nunes, Renato; and Casaca, Augusto (2007). “A Middleware Approach to Configure Security in WSN.” In: *Proceedings of the 1st ERCIM Workshop on eMobility in conjunction with the 5th International Conference on Wired/Wireless Internet Communications (WWIC)*. (Coimbra, Portugal). Springer. ISBN: 978-3-540-72694-4.

Levis, P.; Madden, S.; Polastre, J.; Szewczyk, R.; Whitehouse, K.; Woo, A.; Gay, D.; Hill, J.; Welsh, M.; Brewer, E.; and Culler, D. (2005). “TinyOS: An Operating System for Sensor Networks.” In: *Ambient Intelligence*. Ed. by Werner Weber; Jan M. Rabaey; and Emile Aarts. Springer, pp. 115–148. ISBN: 978-3-540-23867-6. DOI: `10.1007/3-540-27139-2_7`.

Levis, Philip and Culler, David (2004). “The Firecracker Protocol.” In: *Proceedings of the 11th ACM SIGOPS European Workshop (EW)*. (Leuven, Belgium). ACM. DOI: `10.1145/1133572.1133589`.

Levis, Philip; Patel, Neil; Culler, David; and Shenker, Scott (2004). “Trickle: A Self-regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks.” In: *Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*. (San Francisco, CA, USA). USENIX Association, pp. 15–28.

Liang, Sheng (1999). *Java™ Native Interface: Programmer’s Guide and Specification*. Addison-Wesley. ISBN: 0-201-32577-2.

Lorincz, Konrad; Malan, David J.; Fulford-Jones, Thaddeus R.F.; Nawoj, Alan; Clavel, Antony; Shnayder, Victor; Mainland, Geoffrey; and Welsh, Matt (2004). “Sensor Networks for Emergency Response: Challenges and Opportunities.” In: *IEEE Pervasive Computing* 3.4, pp. 16–23. ISSN: 1536-1268. DOI: `10.1109/MPRV.2004.18`.

Luo, Liqian; Abdelzaher, Tarek F.; He, Tian; and Stankovic, John A. (2006). “EnviroSuite: An Environmentally Immersive Programming Framework for Sensor Networks.” In: *Transactions on Embedded Computing Systems* 5.3, pp. 543–576. ISSN: 1539-9087. DOI: `10.1145/1165780.1165782`.

Madden, Sam R.; Franklin, Michael J.; Hellerstein, Joseph M.; and Hong, Wei (2005). “TinyDB: An Acquisitional Query Processing System for Sensor Networks.” In: *ACM Transactions on Database Systems* 30.1, pp. 122–173. ISSN: 0362-5915.

Mainwaring, Alan; Polastre, Joseph; Szewczyk, Robert; Culler, David; and Anderson, John (2002). “Wireless Sensor Networks for Habitat Monitoring.” In: *Proceedings of the 1st International Workshop on Wireless Sensor Networks and Applications (WSNA).* (Atlanta, GA, USA). ACM, pp. 88–97. ISBN: 1-58113-589-0. DOI: 10.1145/570738.570751.

make*Sense* Consortium (2014). make*Sense: Easy Programming of Integrated Wireless Sensor Networks.* URL: http://makesense.sics.se (visited on 2015-12-13).

Maróti, Miklós; Völgyesi, Péter; Simon, Gyula; Karsai, Gábor; and Lédeczi, Ákos (2003). “Distributed Middleware Services Composition and Synthesis Technology.” In: *Proceedings of the 2003 IEEE Aerospace Conference.* (Big Sky, MT, USA). Vol. 6. IEEE, 6_2855–6_2862. ISBN: 0-7803-7651-X. DOI: 10.1109/AERO.2003.1235211.

Marrón, Pedro José; Lachenmann, Andreas; Minder, Daniel; Hähner, Jörg; Sauter, Robert; and Rothermel, Kurt (2005). “TinyCubus: A Flexible and Adaptive Framework for Sensor Networks.” In: *Proceedings of the 2nd European Workshop on Wireless Sensor Networks (EWSN).* (Berlin, Germany). IEEE, pp. 278–289. ISBN: 0-7803-8801-1. DOI: 10.1109/EWSN.2005.1462020.

Martinez, Kirk; Ong, Royan; and Hart, Jane K. (2004). “GLACSWEB: A Sensor Network for Hostile Environments.” In: *Proceedings of the 1st Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON).* (Santa Clara, CA, USA). IEEE, pp. 81–87. ISBN: 0-7803-8796-1. DOI: 10.1109/SAHCN.2004.1381905.

Meier, Andreas; Woehrle, Matthias; Zimmerling, Marco; and Thiele, Lothar (2010). “Zerocal: Automatic MAC Protocol Calibration.” In: *Proceedings of the 6th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS).* (Santa Barbara, CA, USA). IEEE, pp. 31–44. ISBN: 978-3-642-13650-4. DOI: 10.1007/978-3-642-13651-1_3.

Moss, David and Levis, Philip (2008). *BoX-MACs: Exploiting Physical and Link Layer Boundaries in LowPower Networking.* Tech. rep. SING-08-00. Stanford University. URL: http://csl.stanford.edu/~pal/share/spots08.pdf.

Moteiv (2006). *Tmote Sky Datasheet.* URL: http://www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf.

Mottola, Luca and Picco, Gian Pietro (2006). “Programming Wireless Sensor Networks with Logical Neighborhoods.” In: *Proceedings of the 1st International Conference on Integrated Internet Ad Hoc and Sensor Networks (InterSense).* (Nice, France). ACM. ISBN: 1-59593-427-8. DOI: 10.1145/1142680.1142691.

Mottola, Luca and Picco, Gian Pietro (2011). "Programming Wireless Sensor Networks: Fundamental Concepts and State of the Art." In: *ACM Computing Surveys* 43.3, 19:1–19:51. ISSN: 0360-0300. DOI: `10.1145/1922649.1922656`.

Mottola, Luca; Picco, Gian Pietro; and Amjad Sheikh, Adil (2008). "FiGaRo: Fine-Grained Software Reconfiguration for Wireless Sensor Networks." In: *Proceedings of the 5th European Conference on Wireless Sensor Networks (EWSN).* (Bologna, Italy). Springer, pp. 286–304. ISBN: 978-3-540-77689-5. DOI: `10.1007/978-3-540-77690-1_18`.

Mottola, Luca; Eriksson, Joakim; Finne, Niclas; Voigt, Thiemo; Oppermann, Felix Jonathan; Römer, Kay; Casati, Fabio; Daniel, Florian; Picco, Gian Pietro; Tranquillini, Stefano; Valleri, Paolo; Karnouskos, Stamatis; Oertel, Nina; Spieß, Patrik; and Montero, Patricio Moreno (2011a). *D-1.3 – Initial System Architecture.* Tech. rep. `make`*Sense*: Easy Programming of Integrated Wireless Sensor Networks.

Mottola, Luca; Picco, Gian Pietro; Valleri, Paolo; Oppermann, Felix Jonathan; and Römer, Kay (2011b). *D-3.1 – The* `make`*Sense Programming Model.* Tech. rep. `make`*Sense*: Easy Programming of Integrated Wireless Sensor Networks.

Mottola, Luca; Pettinato, Paolo; Picco, Gian Pietro; Quartulli, Antonio; Oppermann, Felix Jonathan; and Römer, Kay (2011c). *D-3.2 – Language Core Implementation.* Tech. rep. `make`*Sense*: Easy Programming of Integrated Wireless Sensor Networks.

Mottola, Luca; Voigt, Thiemo; Oppermann, Felix Jonathan; Römer, Kay; and Langendoen, Koen (2013). *D-3.1 – Report on Specification Language.* Tech. rep. RELYonIT: Research by Experimentation for Dependability on the Internet of Things.

Movahedi, Zeinab; Ayari, Mouna; Langar, Rami; and Pujolle, Guy (2012). "A Survey of Autonomic Network Architectures and Evaluation Criteria." In: *IEEE Communications Surveys & Tutorials* 14.2, pp. 464–490. ISSN: 1553-877X. DOI: `10.1109/SURV.2011.042711.00078`.

Mulligan, Geoff (2007). "The 6LoWPAN Architecture." In: *Proceedings of the 4th Workshop on Embedded Networked Sensors (EmNets).* (Cork, Ireland). ACM, pp. 78–82. ISBN: 978-1-59593-694-3. DOI: `10.1145/1278972.1278992`.

Na, Keewook; Kim, Yungeun; and Cha, Hojung (2009). "Acoustic Sensor Network-Based Parking Lot Surveillance System." In: *Proceedings of the 6th European Conference on Wireless Sensor Networks (EWSN).* (Cork, Ireland). Springer, pp. 247–262. ISBN: 978-3-642-00223-6. DOI: `10.1007/978-3-642-00224-3_16`.

Newton, R.; Morrisett, G.; and Welsh, M. (2007). "The Regiment Macroprogramming System." In: *Proceedings of the 6th International ACM/IEEE Conference* *on Information Processing in Sensor Networks (IPSN).* (Cambridge, MA, USA). ACM, pp. 489–498. ISBN: 978-1-59593-638-7. DOI: `10.1145/1236360.1236422`.

Object Management Group (2011). *Business Process Model and Notation (BPMN).* OMG specification formal/2011-01-03. Version 2. URL: `http://www.omg.org/spec/BPMN/2.0`.

Oppermann, Felix Jonathan (2009). “Towards an End-User-Requirements-Driven Design and Deployment of Wireless Sensor Networks.” In: *Adjunct Proceedings of the Work in Progress Session 17th EUROMICRO International Conference on Parallel, Distributed, and Network-based Processing (PDP).* (Weimar, Germany). Ed. by Erwin Grosspietsch and Konrad Klöckner. SEA-Publications of the Institute for Systems Engineering and Automation SEA-SR-21. J. Kepler University Linz. ISBN: 978-3-902457-21-9.

Oppermann, Felix Jonathan; Boano, Carlo Alberto; and Römer, Kay (2013). “A Decade of Wireless Sensing Applications: Survey and Taxonomy.” In: *The Art of Wireless Sensor Networks.* Ed. by Habib M. Ammari. Vol. 1. Springer, pp. 11–50. ISBN: 978-3-642-40008-7. DOI: `10.1007/978-3-642-40009-4_2`.

Oppermann, Felix Jonathan and Peter, Steffen (2010). “Inferring Technical Constraints of a Wireless Sensor Network Application from End-User Requirements.” In: *Proceedings of the 6th International Conference on Mobile Ad-hoc and Sensor Networks (MSN).* (Hangzhou, China). IEEE, pp. 169–175. ISBN: 978-1-4244-9456-9. DOI: `10.1109/MSN.2010.32`.

Oppermann, Felix Jonathan; Boano, Carlo Alberto; Zimmerling, Marco; and Römer, Kay (2014a). “Automatic Configuration of Controlled Interference Experiments in Sensornet Testbeds.” In: *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems (SenSys).* (Memphis, Tennessee, USA). ACM, pp. 342–343. ISBN: 978-1-4503-3143-2. DOI: `10.1145/2668332.2668355`.

Oppermann, Felix Jonathan; Römer, Kay; Mottola, Luca; Picco, Gian Pietro; and Gaglione, Andrea (2014b). “Design and Compilation of an Object-Oriented Macroprogramming Language for Wireless Sensor Networks.” In: *Workshop Proceedings of the 39th IEEE Conference on Local Computer Networks (LCN).* (Edmonton, AB, Canada). IEEE, pp. 574–582. ISBN: 978-1-4799-3782-0. DOI: `10.1109/LCNW.2014.6927705`.

Oppermann, Felix Jonathan; Boano, Carlo Alberto; Zúñiga, Marco Antonio; and Römer, Kay (2015a). “Automatic Protocol Configuration for Dependable Internet of Things Applications.” In: *Workshop Proceedings of the 40th IEEE Conference on Local Computer Networks (LCN).* (Clearwater, FL, USA). IEEE. ISBN: 978-1-4673-6772-1.

Oppermann, Felix Jonathan; Boano, Carlo Alberto; Baunach, Marcel; Römer, Kay; Aslam, Faisal; Zúñiga, Marco; Protonotarios, Ioannis; Langendoen, Koen; Finne, Niclas; Tsiftes, Nicolas; and Voigt, Thiemo (2015b). *D-3.2 – Report on Protocol Selection, Parameterization, and Runtime Adaptation.* Tech. rep. RELYonIT: Research by Experimentation for Dependability on the Internet of Things.

Oracle (2015a). *Java Card Technology.* URL: `http://www.oracle.com/technetwork/java/embedded/javacard/overview/index.html` (visited on 2015-11-15).

Oracle (2015b). *Java Platform, Micro Edition (Java ME).* URL: `http://www.oracle.com/technetwork/java/javame/index.html` (visited on 2015-11-15).

Pantazis, Nikolaos A.; Nikolidakis, Stefanos A.; and Vergados, Dimitrios D. (2013). "Energy-Efficient Routing Protocols in Wireless Sensor Networks: A Survey." In: *Communications Surveys & Tutorials* 15.2, pp. 551–591. ISSN: 1553-877X. DOI: `10.1109/SURV.2012.062612.00084`.

Parr, Terence (2007). *The Definitive ANTLR Reference: Building Domain-Specific Languages.* Pragmatic Bookshelf. ISBN: 978-0978739256.

Parr, Terence and Fisher, Kathleen S. (2011). "LL(∗): The Foundation of the ANTLR Parser Generator." In: *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI).* (San Jose, CA, USA). ACM, pp. 425–436. ISBN: 978-1-4503-0663-8. DOI: `10.1145/1993498.1993548`.

Pazurkiewicz, Tomasz; Gregorczyk, Michal; and Iwanicki, Konrad (2014). "NarrowCast: A New Link-Layer Primitive for Gossip-Based Sensornet Protocols." In: *Proceedings of the 11th European Conference on Wireless Sensor Networks (EWSN).* (Oxford, UK). Springer, pp. 1–16. ISBN: 978-3-319-04650-1. DOI: `10.1007/978-3-319-04651-8_1`.

Peter, Steffen (2011). "Tool-supported Development of Secure Wireless Sensor Networks." PhD thesis. Brandenburgische Technische Universität Cottbus.

Peter, Steffen; Piotrowski, Krzysztof; and Langendörfer, Peter (2008). "In-Network-Aggregation as Case Study for a Support Tool Reducing the Complexity of Designing Secure Wireless Sensor Networks." In: *Workshop Proceedings of the 33rd IEEE Conference on Local Computer Networks (LCN).* (Montreal, Canada). IEEE, pp. 778–785. ISBN: 978-1-4244-2412-2. DOI: `10.1109/LCN.2008.4664280`.

Pister, Kris S. (2001). *Tracking Vehicles with a UAV-Delivered Sensor Network.* Tech. rep. UC Berkeley and MLB. URL: `http://robotics.eecs.berkeley.edu/~pister/29Palms0103/`.

Polastre, Joseph; Hill, Jason; and Culler, David (2004). "Versatile Low Power Media Access for Wireless Sensor Networks." In: *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys).* (Baltimore, MD, USA). ACM, pp. 95–107. ISBN: 1-58113-879-2. DOI: 10.1145/1031495.1031508.

Polastre, Joseph; Szewczyk, Robert; and Culler, David (2005). "Telos: Enabling Ultra-low Power Wireless Research." In: *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN).* (Los Angeles, CA, USA). IEEE, pp. 364–369. ISBN: 0-7803-9202-7. DOI: 10.1109/IPSN.2005.1440950.

Rajendran, Venkatesh; Obraczka, Katia; and Garcia-Luna-Aceves, J. J. (2003). "Energy-efficient Collision-free Medium Access Control for Wireless Sensor Networks." In: *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys).* (Los Angeles, CA, USA). ACM, pp. 181–192. ISBN: 1-58113-707-9. DOI: 10.1145/958491.958513.

Rajkumar, R.; Lee, C.; Lehoczky, J.; and Siewiorek, Dan (1997). "A Resource Allocation Model for QoS Management." In: *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS).* (San Francisco, CA, USA). IEEE, pp. 298–307. ISBN: 0-8186-6600-5. DOI: 10.1109/REAL.1997.641291.

Raman, Bhaskaran; Chebrolu, Kameswari; Bijwe, Sagar; and Gabale, Vijay (2010). "PIP: A Connection-oriented, Multi-hop, Multi-channel TDMA-based MAC for High Throughput Bulk Transfer." In: *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys).* (Zurich, Switzerland). ACM, pp. 15–28. ISBN: 978-1-4503-0344-6. DOI: 10.1145/1869983.1869986.

Ramanathan, Nithya; Chang, Kevin; Kapur, Rahul; Girod, Lewis; Kohler, Eddie; and Estrin, Deborah (2005). "Sympathy for the Sensor Network Debugger." In: *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys).* (San Diego, CA, USA). ACM, pp. 255–267. ISBN: 1-59593-054-X. DOI: 10.1145/1098918.1098946.

Rechenberg, Ingo (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* German. Frommann-Holzboog. ISBN: 978-3772803734.

Reehuis, Edgar and Bäck, Thomas (2010). "Mixed-integer Evolution Strategy Using Multiobjective Selection Applied to Warehouse Design Optimization." In: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO).* (Portland, OR, USA). ACM, pp. 1187–1194. ISBN: 978-1-4503-0072-8. DOI: 10.1145/1830483.1830700.

Reinhardt, Andreas and Renner, Christian (2015). “RoCoCo: Receiver-Initiated Opportunistic Data Collection and Command Multicasting for WSNs.” In: *Proceedings of the 12th European Conference on Wireless Sensor Networks (EWSN).* (Porto, Portugal). Springer, pp. 218–233. ISBN: 978-3-319-15581-4. DOI: `10.1007/978-3-319-15582-1_14`.

RELYonIT Consortium (2015). *RELYonIT: Research by Experimentation for Dependability on the Internet of Things.* URL: `http://www.relyonit.eu` (visited on 2015-03-25).

Riem-Vis, Ruud (2004). “Cold Chain Management using an Ultra Low Power Wireless Sensor Network.” In: *Proceedings of the Workshop on Applications of Mobile Embedded Systems (WAMES) in Conjuction with the 2nd International Conference on Mobile Systems, Applications, and Services (MobiSys).* (Boston, MA, USA).

Ringwald, Matthias and Römer, Kay (2007). “Deployment of Sensor Networks: Problems and Passive Inspection.” In: *Proceedings of the 5th Workshop on Intelligent Solutions in Embedded Systems (WISES).* (Leganes, Spain). IEEE, pp. 179–192. ISBN: 978-84-89315-47-1. DOI: `10.1109/WISES.2007.4408504`.

Römer, Kay (2004). “Programming Paradigms and Middleware for Sensor Networks.” In: *Proceedings of the 2. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze.* (Karlsruhe, Germany), pp. 49–54.

Römer, Kay and Mattern, Friedemann (2004). “The Design Space of Wireless Sensor Networks.” In: *Wireless Communications* 11.6, pp. 54–61. ISSN: 1536-1284. DOI: `10.1109/MWC.2004.1368897`.

Rosenmüller, Marko and Siegmund, Norbert (2010). “Automating the Configuration of Multi Software Product Lines.” In: *Proceedings of the 4th International Workshop on Variability Modelling of Software-intensive Systems (VAMOS).* ICB-Research Report 37. Universität Duisburg-Essen.

van Rossum, Guido and de Boer, Jelke (1991). “Interactively Testing Remote Servers Using the Python Programming Language.” In: *CWI Quarterly* 4.4, pp. 283–303. ISSN: 0922-5366.

San Francisco Municipal Transportation Agency (2011). *SFpark: Putting Theory Into Practice.* URL: `http://sfpark.org/wp-content/uploads/2011/09/sfpark_aug2011projsummary_print-2.pdf` (visited on 2015-12-13).

Sasnauskas, Raimondas; Landsiedel, Olaf; Alizai, Muhammad Hamad; Weise, Carsten; Kowalewski, Stefan; and Wehrle, Klaus (2010). “KleeNet: Discovering Insidious Interaction Bugs in Wireless Sensor Networks Before Deployment.” In: *Proceedings of the 9th ACM/IEEE International Conference on Information*

Processing in Sensor Networks (IPSN). (Stockholm, Sweden). ACM, pp. 186–196. ISBN: 978-1-60558-988-6. DOI: 10.1145/1791212.1791235.

Schmid, Stefan and Wattenhofer, Roger (2006). "Algorithmic Models for Sensor Networks." In: *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS).* (Rhodes Island, Greece). IEEE, pp. 160–171. ISBN: 1-4244-0054-6. DOI: 10.1109/IPDPS.2006.1639417.

Sha, Mo; Dor, Rahav; Hackmann, Gregory; Lu, Chenyang; suk Kim, Tae; and Park, Taerim (2013). "Self-Adapting MAC Layer for Wireless Sensor Networks." In: *Proceedings of the 34th IEEE Real-Time Systems Symposium (RTSS).* (Vancouver, BC, USA). IEEE, pp. 192–201. DOI: 10.1109/RTSS.2013.27.

Shaylor, Nik; Simon, Douglas N.; and Bush, William R. (2003). "A Java Virtual Machine Architecture for Very Small Devices." In: *Proceedings of the 2003 ACM SIGPLAN Conference on Language, Compiler, and Tool Support for Embedded Systems (LCTES).* (San Diego, CA, USA). ACM, pp. 34–41. ISBN: 1-58113-647-1. DOI: 10.1145/780732.780738.

Shnayder, Victor; Chen, Bor-rong; Lorincz, Konrad; Fulford-Jones, Thaddeus R.F.; and Welsh, Matt (2005). *Sensor Networks for Medical Care.* Tech. rep. TR-08-05. Harvard University.

Shu, Tong; Wu, C.Q.; and Yun, Daqing (2013). "Advance Bandwidth Reservation for Energy Efficiency in High-Performance Networks." In: *Proceedings of the 38th IEEE Conference on Local Computer Networks (LCN).* (Sydney, Australia), pp. 541–548. ISBN: 978-1-4799-0536-2. DOI: 10.1109/LCN.2013.6761289.

Simon, Doug; Cifuentes, Cristina; Cleal, Dave; Daniels, John; and White, Derek (2006). "Java™ on the Bare Metal of Wireless Sensor Devices: The Squawk Java Virtual Machine." In: *Proceedings of the 2nd International Conference on Virtual Execution Environments (VEE).* (Ottawa, ON, Canada). ACM, pp. 78–88. ISBN: 1-59593-332-8. DOI: 10.1145/1134760.1134773.

Simon, Gyula; Volgyesi, Péter; Maróti, Miklós; and Lédeczi, Ákos (2003). "Simulation-based Optimization of Communication Protocols for Large-scale Wireless Sensor Networks." In: *Proceedings of the 2003 IEEE Aerospace Conference.* (Big Sky, MT, USA). Vol. 3. IEEE, 3_1339–3_1346. DOI: 10.1109/AERO.2003.1235250.

Smith, Alice E. and Coit, David W. (1997). "Penalty Functions." In: *Handbook of Evolutionary Computation.* Ed. by Thomas Bäck; David B. Fogel; and Zbigniew Michalewicz. IOP Publishing, C5.2:1–C5.2:6. ISBN: 978-0750303927.

Sommer, Philipp and Kusy, Branislav (2013). "Minerva: Distributed Tracing and Debugging in Wireless Sensor Networks." In: *Proceedings of the 11th ACM Con-*

ference on Embedded Networked Sensor Systems (SenSys). (Rome, Italy). ACM, 12:1–12:14. ISBN: 978-1-4503-2027-6. DOI: 10.1145/2517351.2517355.

Stroustrup, Bjarne (1989). "Multiple Inheritance for C++." In: *Computing Systems* 2.4, pp. 367–395. ISSN: 0895-6340.

Strübe, Moritz; Lukas, Florian; Li, Bijun; and Kapitza, Rüdiger (2014). "DrySim: Simulation-aided Deployment-specific Tailoring of Mote-class WSN Software." In: *Proceedings of the 17th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM).* (Montreal, QC, Canada). ACM, pp. 3–11. ISBN: 978-1-4503-3030-5. DOI: 10.1145/2641798.2641838.

Sugihara, Ryo and Gupta, Rajesh K. (2008). "Programming Models for Sensor Networks: A Survey." In: *ACM Transactions on Sensor Networks* 4.2, 8:1–8:29. ISSN: 1550-4859. DOI: 10.1145/1340771.1340774.

Sun, Jiyan; Zhang, Yan; Tang, Ding; Zhang, Shuli; Xu, Zhen; and Ge, Jingguo (2015). "Improving TCP Performance in Data Center Networks with Adaptive Complementary Coding." In: *Proceedings of the 40th IEEE Conference on Local Computer Networks (LCN).* (Clearwater, FL, USA). IEEE. ISBN: 978-1-4673-6772-1.

Szewczyk, Robert; Mainwaring, Alan; Polastre, Joseph; Anderson, John; and Culler, David (2004). "An Analysis of a Large Scale Habitat Monitoring Application." In: *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys).* (Baltimore, MD, USA). ACM, pp. 214–226. ISBN: 1-58113-879-2. DOI: 10.1145/1031495.1031521.

Tang, Lei; Sun, Yanjun; Gurewitz, Omer; and Johnson, David B. (2011). "EM-MAC: A Dynamic Multichannel Energy-efficient MAC Protocol for Wireless Sensor Networks." In: *Proceedings of the 12th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc).* (Paris, France). ACM, 23:1–23:11. ISBN: 978-1-4503-0722-2. DOI: 10.1145/2107502.2107533.

Texas Instruments (2006). *MSP430x1xx Family User's Guide (Rev. F).* URL: http://www.ti.com/lit/pdf/slau049.

Texas Instruments (2013). *CC2420 Datasheet – 2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver (Rev. C).* URL: http://www.ti.com/lit/gpn/cc2420.

Tranquillini, Stefano; Spieß, Patrik; Daniel, Florian; Karnouskos, Stamatis; Casati, Fabio; Oertel, Nina; Mottola, Luca; Oppermann, Felix Jonathan; Picco, Gian Pietro; Römer, Kay; and Voigt, Thiemo (2012). "Process-Based Design and Integration of Wireless Sensor Network Applications." In: *Proceedings of the 10th International Conference on Business Process Management (BPM).* (Tallinn, Estonia). Springer, pp. 134–149. ISBN: 978-3-642-32884-8. DOI: `10.1007/978-3-642-32885-5_10`.

Tsiftes, Nicolas; Eriksson, Joakim; and Dunkels, Adam (2010). "Poster Abstract: Low-power Wireless IPv6 Routing with ContikiRPL." In: *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN).* (Stockholm, Sweden). ACM, pp. 406–407. ISBN: 978-1-60558-988-6. DOI: `10.1145/1791212.1791277`.

Tu, Yi-Hsuan; Li, Yen-Chiu; Chien, Ting-Chou; and Chou, Pai H. (2011). "EcoCast: Interactive, Object-oriented Macroprogramming for Networks of Ultra-compact Wireless Sensor Nodes." In: *Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN).* (Chicago, IL, USA). IEEE, pp. 366–377. ISBN: 978-1-61284-854-9.

Van Laerhoven, Kristof; Lo, Benny P.L.; Ng, Jason W.P.; Thiemjarus, Surapa; King, Rachel; Kwan, Simon; Gellersen, Hans-Werner; Sloman, Morris; Wells, Oliver; Needham, Phil; Peters, Nick; Darzi, Ara; Toumazou, Chris; and Yang, Guang-Zhong (2004). "Medical Healthcare Monitoring with Wearable and Implantable Sensors." In: *Proceedings of the 3rd International Workshop on Ubiquitous Computing for Pervasive Healthcare Applications (UbiHealth) in Conjunction with the 6th International Conference on Ubiquitous Computing (UbiComp).* (Nottingham, UK).

Voigt, Thiemo and Österlind, Fredrik (2008). "CoReDac: Collision-Free Command-Response Data Collection." In: *Proceedings of the 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA).* (Hamburg, Germany). IEEE, pp. 967–973. ISBN: 978-1-4244-1505-2. DOI: `10.1109/ETFA.2008.4638511`.

Wang, Limin (2004). "MNP: Multihop Network Reprogramming Service for Sensor Networks." In: *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys).* (Baltimore, MD, USA). ACM, pp. 285–286. ISBN: 1-58113-879-2. DOI: `10.1145/1031495.1031538`.

Warneke, Brett; Last, Matt; Liebowitz, Brian; and Pister, Kristofer S.J. (2001). "Smart Dust: Communicating with a Cubic-Millimeter Computer." In: *Computer* 34.1, pp. 44–51. ISSN: 0018-9162. DOI: `10.1109/2.895117`.

Welsh, Matt and Mainland, Geoff (2004). "Programming Sensor Networks Using Abstract Regions." In: *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI).* (San Francisco, CA, USA). USENIX Association, pp. 29–42.

Whang, Daniel H.; Xu, Ning; Rangwala, Sumit; Chintalapudi, Krishna; Govindan, Ramesh; and Wallace, John W. (2004). “Development of an Embedded Networked Sensing System for Structural Health Monitoring.” In: *Proceedings of the 1st International Workshop on Smart Materials and Structures Technology (SM&S)*. (Waikiki, HI, USA). DEStech Publications, pp. 461–468. ISBN: 1932078401.

Whitehouse, Kamin; Zhao, Feng; and Liu, Jie (2006). “Semantic Streams: A Framework for Composable Semantic Interpretation of Sensor Data.” In: *Proceedings of the 3rd European Conference on Wireless Sensor Networks (EWSN)*. (Zurich, Switzerland). Springer, pp. 5–20. ISBN: 978-3-540-32158-3. DOI: `10.1007/11669463_4`.

Whitehouse, Kamin; Sharp, Cory; Brewer, Eric; and Culler, David (2004). “Hood: a Neighborhood Abstraction for Sensor Networks.” In: *Proceedings of the 2nd International Conference in Mobile Systems, Applications, and Services (MobiSys)*. (Boston, MA, USA). ACM, pp. 99–110. ISBN: 1-58113-793-1. DOI: `10.1145/990064.990079`.

Winter, T.; Thubert, A. B. P.; Clausen, T.; Hui, J.; Kelsey, R.; Levis, P.; Pister, K.; Struik, R.; and Vasseur, J. (2012). *RPL: IPv6 Routing Protocol for Low Power and Lossy Networks*. RFC 6550. URL: `http://tools.ietf.org/html/rfc6550`.

Xiang, Yun; Bai, Lan; Piedrahita, Ricardo; Dick, Robert P.; Lv, Qin; Hannigan, Michael; and Shang, Li (2012). “Collaborative Calibration and Sensor Placement for Mobile Sensor Networks.” In: *Proceedings of the 11th International Conference on Information Processing in Sensor Networks (IPSN)*. (Beijing, China). ACM, pp. 73–84. ISBN: 978-1-4503-1227-1. DOI: `10.1145/2185677.2185687`.

Xu, Ning; Rangwala, Sumit; Chintalapudi, Krishna Kant; Ganesan, Deepak; Broad, Alan; Govindan, Ramesh; and Estrin, Deborah (2004). “A Wireless Sensor Network for Structural Monitoring.” In: *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*. (Baltimore, MD, USA). ACM, pp. 13–24. ISBN: 1-58113-879-2. DOI: `10.1145/1031495.1031498`.

Yang, Guang-Zhong, ed. (2006). *Body Sensor Networks*. Springer London. ISBN: 978-1-84628-272-0. DOI: `10.1007/1-84628-484-8`.

Yao, Yong and Gehrke, Johannes (2002). “The Cougar Approach to In-Network Query Processing in Sensor Networks.” In: *Sigmod Record* 31.3. ISSN: 0163-5808.

Ye, Wei; Heidemann, John; and Estrin, Deborah (2002). “An Energy-efficient MAC Protocol for Wireless Sensor Networks.” In: *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*. Vol. 3. IEEE, pp. 1567–1576. ISBN: 0-7803-7476-2. DOI: `10.1109/INFCOM.2002.1019408`.

Ye, Wei; Silva, Fabio; and Heidemann, John (2006). "Ultra-low Duty Cycle MAC with Scheduled Channel Polling." In: *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys).* (Boulder, CO, USA). ACM, pp. 321–334. ISBN: 1-59593-343-3. DOI: 10.1145/1182807.1182839.

Yoon, Suk-Un; Murawski, Robert; Ekici, Eylem; Park, Sangjoon; and Mir, Zeeshan Hameed (2010). "Adaptive Channel Hopping for Interference Robust Wireless Sensor Networks." In: *Proceedings of the IEEE International Conference on Communications (ICC).* (Cape Town, South Africa), pp. 432–439. ISBN: 978-1-4244-6402-9. DOI: 10.1109/ICC.2010.5502780.

Yun, Daqing; Wu, Qishi; Brown, Patrick; and Zhu, Mengxia (2012). "Modeling and Optimizing Transport-Support Workflows in High-Performance Networks." In: *Proceedings of the 39th IEEE Conference on Local Computer Networks (LCN).* (Clearwater, FL, USA). IEEE, pp. 384–391. ISBN: 978-1-4673-1565-4. DOI: 10.1109/LCN.2012.6423652.

Zhang, Lifeng; Jin, Beihong; Li, Keqin; and Zhang, Fusang (2013). "An Adaptive Channel Coordination Mechanism for Vehicular Ad Hoc Networks." In: *Proceedings of the 38th IEEE Conference on Local Computer Networks (LCN).* (Sydney, Australia), pp. 557–564. ISBN: 978-1-4799-0536-2. DOI: 10.1109/LCN.2013.6761291.

Zhang, Pei; Sadler, Christopher; Lyon, Stephen; and Martonosi, Margaret (2004). "Hardware Design Experiences in ZebraNet." In: *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys).* (Baltimore, MD, USA). ACM, pp. 227–238. ISBN: 1-58113-879-2. DOI: 10.1145/1031495.1031522.

Zhou, Jiangwei; Chen, Yu; Leong, Ben; and Sundaramoorthy, Pratibha Sundar (2010). "Practical 3D Geographic Routing for Wireless Sensor Networks." In: *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys).* (Zurich, Switzerland). ACM, pp. 337–350. ISBN: 978-1-4503-0344-6. DOI: 10.1145/1869983.1870016.

Zimmerling, Marco (2015). "End-to-end Predictability and Efficiency in Low-power Wireless Networks." PhD thesis. Eidgenössische Technische Hochschule (ETH) Zürich. DOI: 10.3929/ethz-a-010531577.

Zimmerling, Marco; Ferrari, Federico; Mottola, Luca; Voigt, Thiemo; and Thiele, Lothar (2012). "pTunes: Runtime Parameter Adaptation for Low-power MAC Protocols." In: *Proceedings of the 11th International Conference on Information Processing in Sensor Networks (IPSN).* (Beijing, China). ACM, pp. 173–184. ISBN: 978-1-4503-1227-1. DOI: 10.1145/2185677.2185730.

Zúñiga, Marco Antonio; Protonotoarios, Ioannis; Li, Si; Langendoen, Koen; Boano, Carlo Alberto; Oppermann, Felix Jonathan; Römer, Kay; Brown, James; Roedig, Utz; Mottola, Luca; and Voigt, Thiemo (2014). *D-2.2 & D-2.3 – Report on Protocol Models & Validation and Verification.* Tech. rep. RELYonIT: Research by Experimentation for Dependability on the Internet of Things.