Enabling Modular Plug&Play Wireless Sensor and Actuator Network Nodes: Software Architecture

Konstantin Mikhaylov and Anton Paatelma
Centre for Wireless Communications
University of Oulu
Oulu, Finland
{konstantin.mikhaylov,anton.paatelma}@ee.oulu.fi

***Abstract*— The implementation of the Internet of Things (IoT) concept introduces fundamentally novel challenges not only at the network level, but also in regards to the design of the devices. In order to enable easy and effective implementation of the new IoT devices earlier we have proposed a modular platform featuring Plug-and-Play (P&P) module connectivity. The nodes are assembled out of the hardware modules encapsulating various peripherals, such as power supply and processing units, transceivers, sensors and actuators, etc. Once a node is built, all the connected peripherals are identified and the operation of the node is optimized accordingly. In this work we present the second component of our solution, i.e. the software intended to operate on top of the highly dynamic hardware. In the paper we first discuss the principal challenges and requirements for such software, then introduce and describe in details our software architecture, and finally report the initial results of its implementation. The lessons learned and the major challenges for the future are summarized in the conclusion.**

Keywords—IoT, Wireless Sensors and Actuators Networks, Modular, Platform, Hardware, Software, Architecture, Design

I. INTRODUCTION

The recent years have brought to the stage a multitude of novel sensing, data processing and communication technologies. These advances lay the basis for further development of the Internet of Things (IoT) concept and have twofold effect. On one hand, they enable significant reductions of cost and energy consumption for the existing devices and systems. On the other hand, they extend the available elemental base thus enabling development of the novel applications. Despite significant positive impact, the latter introduces new challenges by further increasing diversity of IoT devices and makes the network landscape even more heterogeneous.

In order to enable fast and easy development of the new Wireless Sensor and Actuator Network (WSAN) nodes and IoT devices recently we have first proposed the concept [1] and then demonstrated a prototype (Fig. 1) [2] of the modular hardware (HW) platform featuring Plug&Play (P&P) module connection. The nodes are composed out of the *modules* encapsulating the various *peripherals*, such as power supplies, processing units, transceivers, sensors and actuators, etc. Once a node is built, its main processing unit (MPU) identifies all the connected peripherals and uses this information to optimize node's operation and to select the applications to execute.

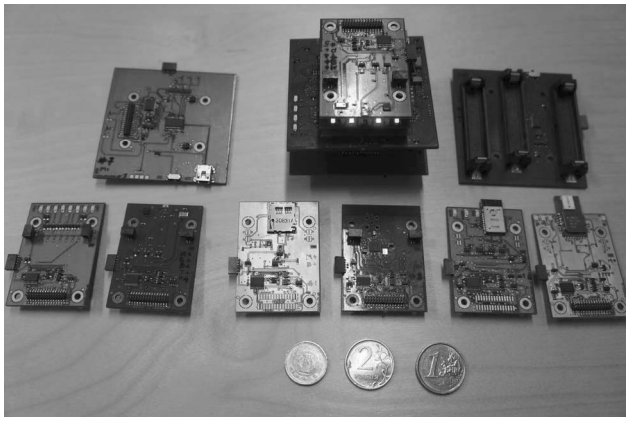

Fig. 1. Proposed modular WSAN HW platform (in the centre – example of an assembled node featuring three 802.15.4 radios, sensors, LED and battery modules, around - various HW modules).

The implementation of the proposed concept is impossible without solving two major challenges. First of all, one needs to come up with HW architecture and develop inter-module interfaces. A possible solution for this problem was first sketched in [1] and then detailed in [3]. In this paper we focus on the second part of the puzzle and propose the software (SW) architecture for a dynamically reconfigurable HW platform.

II. WSAN APPLICATION DEVELOPMENT AND STATE OF ART (SOA) SOFTWARE ARCHITECTURES

As discussed in [4], the design of a typical WSAN application includes six phases. During the first phase the application requirements are identified. Next, these requirements, design dimensions and respective trade-offs are analyzed. The results of the first two phases define the design of the system as whole and provide specifications for the required HW and SW components. During the fourth phase the new components are developed and integrated with the already existing ones. Finally, the application is prototyped, tested, and deployed. As one can see, the discussed procedure results in highly effective solutions, the target parameters of which can be well optimized (e.g., employing cross-level optimizations). But the cost of this is the poor flexibility, since the resulting HW and SW become application specific. It is hardly surprising that the today's SW architectures for WSAN nodes are designed under the assumption of static HW structure and support reconfiguration for only the topmost layers of stack.

978-1-4799-8203-5/15/$31.00 ©2015 IEEE

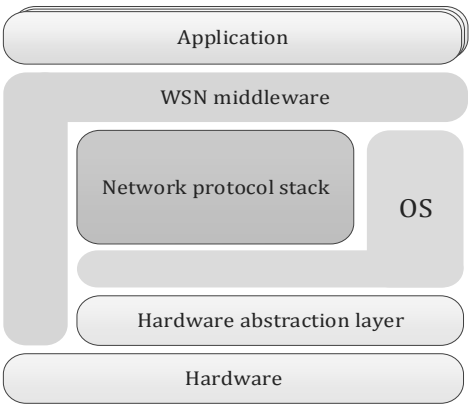

Fig. 2. Software architecture for WSAN node proposed by Kourilehto et al. [4]

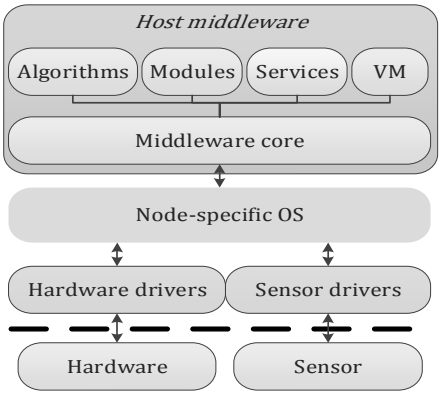

Fig. 3. Software architecture for WSAN node proposed by Blumenthal et al. [5]

A classical SW architecture of a WSAN node is depicted in Fig. 2 [4]. An application is developed on top of the WSAN middleware which glues together the HW, operation system (OS), network stack and applications. The OS manages the system resources usage, timeliness and controls peripherals access of the active tasks. If present, the HW abstraction layer handles standard OS calls to HW thus enabling to make OS's core device-independent. The primary objective of the middleware is to support development, maintenance, deployment and execution of WSAN applications. Based on their purpose, the middlewares may be categorized into programming abstractions, runtime support and interfaces [4], although the recently developed solutions often combine all these functionalities. Typically, only the components of the two upper layers are implemented in dynamic fashion and may be modified after the initial deployment.

Another solution has been proposed by Blumenthal et al. in [5]. As depicted in Fig. 3, the architecture features specific HW drivers to implement OS control over the HW components. The single middleware core controls four optional components, namely algorithms, modules, services and virtual machines. These components can be made dynamic and exchanged between the nodes [5].

As one can see, use of the described architectures with a system featuring dynamic HW reconfiguration is hardly feasible. One of the possible approaches to address this problem is to employ an over-the-air reprogramming technique (e.g., [6-7]) and overwrite or modify the SW image in full each time when the HW is changed. The major drawbacks of this are the significant amount of data to be transferred and the need for the fully reliable and secure protocols to transfer new SW images. Therefore, in order to address this problem, below we propose a new SW architecture for the systems with highly dynamic HW.

III. Proposed Software Architecture

A. Challenges and Requirements

The high level of HW dynamism characterizing the proposed platform introduces a set of very specific problems and challenges when it comes to the SW design. The first critical challenge is the need for flexible, scalable and transferable HW drivers, multiple instances of which could be effectively executed in parallel. Indeed, each node is composed from modules, the functionality and the relative position for which is not known in advance. One of the consequences of this is that the data interfaces between the MPU and each connected peripheral are not known a priori. Also a single node may host multiple identical peripherals (e.g., radio transceivers or sensors). This emphasizes the need for supporting creation of multiple HW driver instances, i.e. driver "cloning". Another critical problem is the selection of the HW modules to be used for each executed task. To give a practical example: if multiple radio transceivers featuring different communication technologies are available on a node, the decision on which ones to use for transferring each particular piece of data should be made. Finally, the list of applications which can be executed by a node depends on the available HW modules and the needs of the network. This requires having a mechanism for deciding which applications should be launched and supporting application and respective driver transfers between the nodes. The list above is not excessive and can be continued.

B. Proposed Architecture

The proposed SW architecture is depicted in Fig. 4. The core component of the middleware is the *Resource manager* which is composed of the three major units. The first one is the *Module manager*, which is a low-level entity responsible for identification of the peripherals and modules (i.e., obtaining *HW resource descriptors* - refer to [1] for details), controlling power supply of the peripherals and of the whole node (e.g., dynamic voltage-frequency scaling), interpretation and prioritization of the interrupts coming from various peripherals. The second component is the *Communication manager* which handles all the communication of a node with external world. Based on the available modules (i.e., wired/wireless transceivers), the manager decides which communication technology and which parameters to use. Depending on the resources available for the manager and on the range of modules and applications supported, the communication parameters may be assigned either statically or can be defined dynamically for each transmitted message.

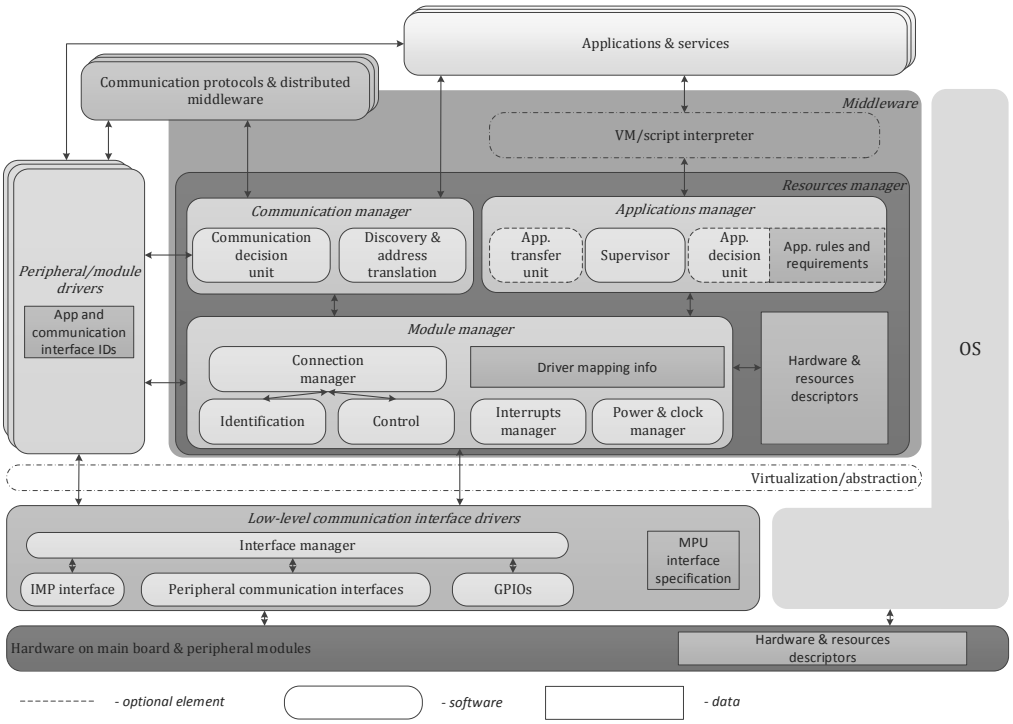

Fig. 4. Proposed SW architecture and interfaces between SW components

The other functionalities which must be supported by a Communication manager include: discovery of devices and networks, translation of addresses and routing between the different communication technologies, prevention of interference between the wireless transceivers and handling of communication interface change if the one currently used gets blocked. Finally, the *Application manager* decides which applications and services can be launched and supervises their operation. Depending on the presence of control entities in the network, the applications executed by each node may be either defined by a node independently or dictated from remote. The former option requires a node to have a decision unit which will come up with the list of applications to run based on the specially formulated set of rules. Also, in order to enable transferring the tasks between the nodes, the applications should be written in HW independent manner and handled by a virtual machine (VM) or some other sort of interpreter.

Another important feature of the proposed architecture is the design of *low-level communication interface drivers* used by the peripheral drivers for communicating with peripherals. The drivers for each interface (e.g. GPIOs, SPI, I2C, UART) are implemented by specific SW instances, which are accessed by the peripheral drivers through the standardized access interfaces. The identifiers of the interfaces for each particular peripheral connected to a node are defined as a part of peripheral identification procedure and are reported to the peripheral drivers by the module manager. After this, when a driver desires to communicate with a peripheral, it calls the respective low-level interface driver and provides the identifier of the interface to use. On one hand, this enables to abstract from the specifics of HW interface implementation for particular MPU and to leave aside the details of peripheral's connection when implementing the peripheral drivers. On the other hand, this can be used to enable sharing of communication interfaces between the peripherals and interface access prioritization.

The *peripheral divers* should be implemented as independent threads which are interfaced to the respective communication interface drivers and applications. In addition, the drivers of wired and wireless transceivers should be registered at the communication manager. If desired, the drivers and the protocol stacks may be implemented as scripts executed by an interpreter. One of the principal questions is where and how the peripheral drivers are stored. The possible options include: a remote network location, the internal memory of MPU, and memory of HW modules.

C. Implementation

Although the development of the SW is still in progress, the most critical components of the proposed architecture for the HW platform depicted in Fig. 1 are already implemented. The SW is written in C language and operates on top of FreeRTOS [8] embedded OS. The simplified high-level algorithm of the developed SW solution is depicted in Fig. 5. The implemented SW includes the following components:

- the low level drivers for IMP,SPI,I2C,UART interfaces and GPIOs;
- the full featured peripheral drives (see Table 1) for LED modules, temperature, pressure, humidity and light sensors, IEEE 802.15.4 and IEEE 802.15.4a UWB radio transceivers and the basic drivers for Bluetooth Smart transceiver;

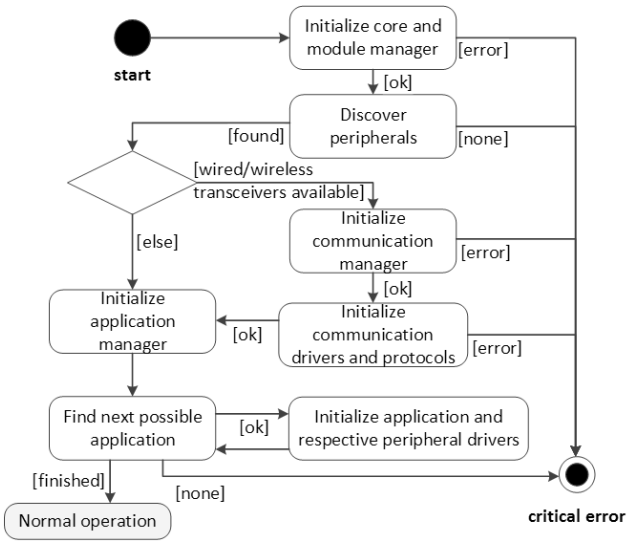

Fig. 5. Simplified algorithm of operation of implemented SW.

Table I. Status of HW modules and respective peripheral SW drivers

HW Module	Description	SW driver status
Main	STM32F206 microcontroller, FRAM and dynamic voltage control system	Basic functionality
Battery	3 x AAA batteries	Not required
USB	USB-UART interface	Full featured
Environment sensors	Pressure, temperature, humidity and light sensors	Full featured
LED	8 LEDs	Full featured
Radio 1	IEEE 802.15.4 2450 DSSS radio transceiver	Full featured
Radio 2	IEEE 802.15.4a (UWB) radio transceiver	Full featured
Radio 3	BLE (Bluetooth Smart) radio transceiver	Basic functionality
Radio 4	433/868 MHz ISM band radio transceiver	In progress
Radio 5	IEEE 802.11 b/g/n	In progress
microSD card	MicroSD card adapter	In progress

- module, application and communication managers;
- applications implementing collection of the information about node's structure and periodically reporting these data to a remote server.

In the current version of the SW the drivers and applications are implemented as pre-compiled machine code residing in the MPU internal program memory. In future, they can be changed to scripts which can be downloaded over-the-air. Similarly, the sets of rules used by the communication and application managers are currently hardcoded and cover all possible node structures. The communication manager has quite simplistic structure and is configured to broadcast all the messages via every communication technology available to a node. The proposed SW architecture and its implementation have been used as basis for demonstrations reported in [2] and [9].

IV. Conclusions and Further Work

In the paper we have proposed the software architecture for the modular WSAN node hardware platform introduced earlier. The proposed architecture is designed to support and maximally benefit from the dynamism of underlying hardware platform. For this, we have introduced as a part of node's middleware a special resource manager, which is composed of the three components, namely the module manager, the communication manager and the application manager. The former one is responsible for collecting the information about the peripherals available on each node, while the two latter make decisions on the communication interfaces to use and applications to launch. Each of the peripherals available on the attached modules is controlled by a special peripheral driver thread. In order to enable launching multiple driver instances and handling various connections of the modules, the drivers operate on top the specially designed low level communication interface driver solutions.

Although the implementation of the full featured SW architecture is still in its early phase, even the development of the core components implementing just the most basic functionalities has dramatically increased the capabilities of the hardware platform. Nonetheless, there are still many unsolved challenges. The first critical question which needs to be answered is where and in which form the drivers and the applications will be stored. The second major challenge is who and how will decide which of the possible applications each node will execute. Finally, the optimization of node's operation and data transfer in respect of node's task, resources and applications is a challenging problem of extreme importance. In the future we plan to continue developing and implementing the proposed software architecture and will try to address some aspects of the listed challenges.

References

[1] K. Mikhaylov and M. Huttunen, "Modular Wireless Sensor and Actuator Network Nodes with Plug-and-Play Module Connection", in *Proc. IEEE SENSORS*, Valencia, Spain, Nov. 2-5, 2014, pp. 470-473.

[2] K. Mikhaylov et al., "Extensible Modular Wireless Sensor and Actuator Network and IoT Platform with Plug&Play Module Connection" in *Proc. ACM/IEEE Int. Conf. Inf. Proc. Sensor Netw.*, Seattle, USA, Apr. 13-16, 2015, pp. 386-387.

[3] K. Mikhaylov et al., Design and Implementation of Modular Wireless Sensor and Actuator Network Nodes with Plug-and-Play Module Connection. In press.

[4] M. Kourilehto et al., Ultra-Low Energy Wireless Sensor Networks in Practice, Chichester: John Wiley & Sons Ltd, 2007.

[5] J. Blumenthal et al., "Wireless sensor networks - new challenges in software engineering", in *Proc. IEEE Conf. Emerging Tech. Factory Automation*, Lisbon, Portugal, 16-19 Sept., 2003, pp. 551-556.

[6] M. Rossi et al., "SYNAPSE++: Code Dissemination in Wireless Sensor Networks Using Fountain Codes ", *IEEE Trans. Mob. Comp.*, vol. 9, no. 12, pp. 1749-1765.

[7] A. Hagedorn, D. Starobinski, and A. Trachtenberg, "Rateless Deluge: Over-the-Air Programming of Wireless Sensor Networks using Random Linear Codes", in *Proc. ACM/IEEE Int. Conf. Inf. Proc. Sensor Netw.*, St. Louis, USA, Apr. 22-24, 2008, pp. 457-466.

[8] FreeRTOS [Online]. Available: http://www.freertos.org/

[9] K. Mikhaylov et al., " Modular Multi-radio Wireless Sensor Platform for IoT Trials with Plug&Play Module Connection" to be presented at *Annu. Int. Conf. Mobile Comp. Netw.*, Paris, France, Sept. 7-11, 2015.