# Wireless Embedded Systems Powered by Energy Harvesting

Attila Strba*
Institute of Applied Informatics
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Ilkovičova 3, 842 16 Bratislava, Slovakia
strba@yahoo.com

## Abstract

Ambient intelligence requires that devices integrated to the environment are self-sustaining and maintenance free. To fulfill the vision of ambient intelligence, the design of small wireless embedded systems powered by energy harvesting is needed. Our work raises awareness that generic embedded system and wireless sensor nodes are different from wireless embedded systems powered by energy harvesting (WESPEH) therefore a different design approach is required. The work introduces a new design technique based on rapid prototyping and constraints analysis. From different groups of constraints the software problematic is developed in detail further with the focus on operating system design.

Comprehensive analysis of thin resource operating systems leads to the conclusion that existing systems are not designated to run on a dedicated WESPEH hardware platform. We define the requirements, architecture and design aspects of energy autonomous operating systems. Based on these definitions a new operating system DolphinAPI is designed and implemented. The initial implementation is performed on the EO3000I hardware platform.

Analysis of existing benchmark methods shows that common benchmark methods focus entirely on the computational performance, neglecting other important features of an operating system such as memory needs or influence on the energy consumption. Therefore a new operating system quantification methodology - considering energy consumption, scheduler behavior and application utilization - is introduced . The methodology is based on vector definition and trace-driven performance analysis. Using the methodology TinyOS and Dolphin API is evaluated and compared. For an objective evaluation TinyOS is migrated to the EO3000I hardware platform. As part of the migration the EnOcean protocol stack is implemented in NesC.

## Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design—*Real-time systems and embedded systems*; D.4.8 [**Operating Systems**]: Performance—*Measurements, Modeling and prediction*; D.2.8 [**Software Engineering**]: Metrics—*Product metrics*; C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems; C.3 [**Performance Of Systems**]: Measurement techniques

## Keywords

wireless embedded systems, energy harvesting, rapid prototyping, prototype rating, design technique, operating system design, operating system quantification, performance prediction

## 1. Introduction

With our work we would like to raise the awareness of the research community that to fulfill the vision of ambient intelligence requires the introduction of a new device category. The aim of this research is to examine various problematic aspects of wireless embedded systems powered by energy harvesting and propose solutions. The major goals of the research can be summarized in three categories:

- Introduction of the device category wireless embedded system powered by energy harvesting. Definition of the architecture, application scenarios and powering possibilities of these devices. Discussion and categorization of design problems, proposal of appropriate design methodology.
- Discussion of specific software design issues for energy autonomous wireless devices. Definition of requirements, architecture, design and implementation aspects of operating systems for energy autonomous devices. The feasibility of the definitions has to be demonstrated by implementation.

*Recommended by thesis supervisor: Assoc. Prof. Tibor Krajčovič. Defended at Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava on November 24, 2011.



Strba, A. Wireless Embedded Systems Powered by Energy Harvesting. Information Sciences and Technologies Bulletin of the ACM Slovakia, Vol. 4, No. 1 (2012) 1-13

- Finding an appropriate quantification methodology for energy autonomous operating system evaluation.

## 2. Wireless embedded systems powered by energy harvesting

We introduce the term Wireless Embedded Systems Powered by Energy Harvesting (WESPEH). It is explained why there is a need for new device category. The architecture and properties of WESPEH are described. The use case of WESPEH is presented on various application scenarios. We give an overview of energy harvesting powering possibilities and show how much power can be transformed from ambient energy sources. Furthermore we deal with the design problematic of WESPEH devices and introduce a new design technique based on constraint group analysis and prototype quantification.

### 2.1 Discussion

Rapid miniaturization and performance increases of the integrated circuit leads to the development of tiny embedded systems integrated more and more into our everyday objects and will create a world of smart devices surrounding us. The Information Society and Technology Advisory Group in 1999 defined a concept of future computing called ambient intelligence. This concept is based on the fact that technological progress in the microelectronics makes the increasing miniaturization of embedded systems possible. The ambient intelligence concept expects that the miniaturization trend will soon lead to the development of tiny wireless smart embedded devices, integrated more and more into our everyday objects.

To fulfill the vision of ambient intelligence requires that there are several small-embedded devices integrated in the environment. Each of these devices requires some source of powering possibility and communication interface. While power can be transported with the help of cables or devices could be powered by batteries, neither of these possibilities offers an effective and long-term solution. Embedded systems will fulfill the ambient intelligence concept only if they are self-sustaining and maintenance free. Such a requirement can be achieved only if these devices have a wireless communication interface and are powered from the environment using energy harvesting. Energy harvesting creates the possibility to use the omnipresent energy sources from our surroundings such as, for example, moving objects, vibrating machine parts, temperature changes, electromagnetic waves such as light, radio waves or infrared waves and other energy sources. Pressing a button with 3N force, temperature difference of 12K or just 500 lux of light generates enough energy to power a module equipped with a microcontroller and a RF-transmitter and to transmit a wireless signal.

If we observe those emerging trends in embedded systems - miniaturization, wireless interface, energy harvesting – questions may arise as to whether embedded systems as we know them nowadays are able to cover the requirements of ambient intelligence. To be able to answer these questions let us briefly discuss the term 'embedded system'. Understanding the term embedded is helpful for understanding what application scenarios embedded systems cover. Embedding means to enclose or implant as essential or characteristic. The exact definition of an embedded system, according D. Gajski [5], is as follows:

"*From the viewpoint of computing systems, an embedded system is a special-purpose system in which a computer is entirely encapsulated by the gadget it controls. Unlike a general-purpose computer, an embedded system performs predefined tasks, usually with very specific requirements and constraints.*"

Such a definition is outdated. The target usage of embedded systems has been transformed in the last ten years. Most of the embedded systems of the 21st century perform a variety of different tasks with different requirements. We can demonstrate this fact with the cell phone. The original use of the cell phone was to make phone calls. By technological evolution, the purpose of this device has completely changed from the origin. It is also used as a multifunctional device for taking photographs, browsing the internet, and much more. An anecdote from Bjarne Stroustrup provides a very good description of this status quo:

"*I have always wished that my computer would be as easy to use as my telephone. My wish has come true. I no longer know how to use my telephone.*"

A common element in the definition of embedded systems is that they cover a wide variety of applications with vastly varying requirements. Let us demonstrate this fact with the following example. The term embedded system refers to a line powered Coca-Cola-Automat with TCP/IP networking capabilities, running Windows CE on a 400MHz processor and several MB of RAM. At the same time, a pacemaker device is also categorized as an embedded system powered by lithium iodine battery running a proprietary SW on an 8MHz processor with 48kB flash and 10kB RAM. Despite the fact that both the systems comply with the definition embedded system, the design approach and requirements on these two systems are contradictory. Consequently we can not speak anymore about generic design methodologies, application requirements and research challenges in terms of embedded systems.

### 2.2 WESPEH Definition

The previous discussion leads us to the conclusion that categorizing ambient intelligence devices as embedded systems is not a good solution. There is a gap in the current understanding of these devices. Moreover existing embedded system design techniques does not necessarily ensures a successful design. This gap needs to be filled. Ambient intelligence devices need specific state-of-the-art design methodologies and solutions. In order the vision of ambient intelligence is fulfilled a new category of devices has to be introduced. Paper [1] discusses the vision of ambient intelligence in detail. Based on this paper we can derive the requirements of devices fulfilling ambient intelligence. Devices fulfilling the vision of ambient intelligence:

- are powered from the environment with the help of energy harvesting
- are maintenance free
- have a wireless communication interface
- have a sensor/actuator interface
- can interact with real-time events
- have unobtrusive hardware

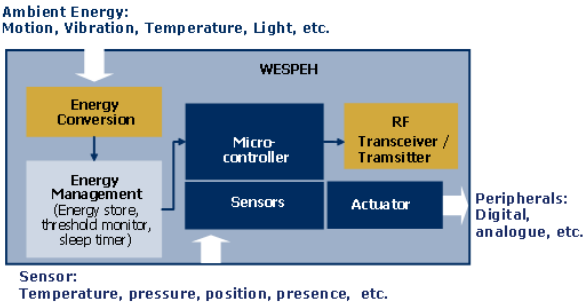


**Figure 1: Architecture of WESPEH**

From the requirements we can derive the name of this device category as following Wireless Embedded Systems Powered by Energy Harvesters (WESPEH). The architecture of a WESPEH is shown in Figure 1. The device consists of an energy conversion unit that is the energy harvester itself. This unit converts the ambient energy to electrical energy. The energy management block stores the electrical energy to storage capacitor. In addition, the energy management unit is responsible for converting the voltage using a step down or step up converter to lower or higher value. Using this method the capacitor can be loaded more effectively. For further functions, the energy management block provides ultra low sleep timers to control the CPU and a voltage threshold detector. The microcontroller communicates using the RF interface and interacts with the environment using sensor or actuator interfaces. From the above description, we can formulate the following definition of WESPEH.

*Definition 1. Wireless Embedded System Powered by Energy Harvester - designated as WESPEH - is a standalone; maintenance-free special-purpose computing system powered by ambient energy; equipped with an uni- or bidirectional wireless communication interface; capable of hard or soft real time interaction with the environment using sensor and actuator interfaces.*

Considering WESPEH device architecture and comparing it to a wireless sensor node we can see several similarities. A question may arise if wireless sensor node corresponds to a wireless embedded system. To answer this question there is a need to understand the difference between wireless embedded systems powered by energy harvesters and wireless sensor networks. According Thomas Haenselmann [6] the wireless sensor network is defined as following:

"*A sensor network is a set of small autonomous systems, called sensor nodes which cooperate to solve at least one common application. Their tasks include some kind of perception of physical parameters.*"

One of the main aspects in which embedded systems and wireless sensor networks differ is that embedded systems are designed to work standalone. Wireless sensor network devices core design space is the network infrastructure [11]. The target application of wireless sensor network is to build a large-scale ad hoc, multi-hop infrastructure composed of several thousands of nodes. A standalone wireless sensor node has no relevant functionality. Another aspect in which embedded systems and wireless sensor networks differ is the purpose for which they use their communication interface. The embedded system wireless interface's purpose is to enable the human to access information or trigger action. The sensor node wireless interface's primary focus is to interact with other nodes. In the case of the wireless sensor node, the bidirectional wireless communication interface is obligatory, while WESPEH may use unidirectional communication interfaces. Wireless sensor nodes differ from WESPEH devices also in having only sensing interfaces. WESPEH can be equipped also with an actuator as described in the application scenarios. The last major difference between WESPEH and WSN is the powering option. While in certain wireless sensor network scenarios battery is an acceptable option WESPEH must be maintenance-free, and energy autonomous. Part of the wireless sensor network has a lot in common with wireless embedded systems powered by energy harvesting. Especially in applications where WESPEH devices have a sensing role. Still the design problematic of wireless sensor networks does not fully cover the problematic of WESPEH. WESPEH has to be handled as a separate device category.

## 2.3 Powering possibility

Understanding the power problematic of WESPEH is a key objective to understand the device and software design decisions discussed in later chapters. For this purpose we have analyzed various ambient energy sources and harvesting possibilities. We have performed measurements to determine how many energy is available from which type of energy harvesters, what is the typical conversion efficiency and dimension of the energy harvesters. All these facts are summarized in the thesis. From all the energy powering possibilities most proven are the photovoltaic energy harvesters.

## 2.4 Application scenarios

WESPEH find the usage in various application fields. The definition of scenarios should help to understand the design problematic of WESPEH. The thesis discusses various WESPEH application scenarios in detail. This extended dissertation abstract contains only the summarization of the relevant WESPEH application categories as follows:

**Intelligent buildings and homes** - The most widespread applications of WEPSEH nowadays are in intelligent buildings and homes. The precondition of these scenarios is that all WESPEH devices are interoperable and are using the same wireless protocol. Usage of WESPEH saves wire installation costs in buildings and homes. Additional building equipped with WESPEH gives the possibility for further energy saving, by intelligent heating and air condition control, switching of unnecessary light sources, etc. Typical ambient powering possibilities are the following: light, mechanical, temperature difference. Applications examples are: solar powered blind control, thermal powered heating regulator, solar powered air ventilation.

**Safety critical applications** - Using WESPEH various safety critical applications can be realized as in everyday life also in disaster cases. The precondition of these devices is long lifetime and robustness. WESPEH used in safety applications are equipped with various sensor interfaces and with unidirectional communication interface. WESPEH in these scenarios are most of their life-

time in an inactive power saving mode until an occurrence of a critical event. The critical design aspect in these devices is that during inactive mode the collection of maximum power reserves has to be ensured. In case of a critical event occurrence the WESPEH wakes up and starts to transmit the sensor information periodically. The receivers are always line powered. In some scenarios the information transmission carries on until all the energy reserves of the device are exploited. For successful functionality the information has to be transmitted in real time. Typical ambient powering possibilities are the following: light, vibrations. Applications examples are: solar powered cooker extractor sensor, solar powered intelligent smoke detector, solar powered geological rock fall guard, vibration powered tunnel disaster monitor.

**Health monitor applications** - WESPEH can be used effectively in a health care. Many cables for vital monitoring attached to patients' body causes a lot of discomfort in a hospital. The cables can be replaced by a WESPEH attached to the patient body that monitors various vital function like temperature, blood pressure, pulse rate. Health monitor WESPEH application requires fault safe functionality and real time reaction on critical events. Typical ambient powering possibilities are the following: breathing tension of the patient, thermal difference between human body and the surrounding, mechanical per finger press, solar power. Application examples are: temperature powered vital monitor, mechanical powered emergency button.

**Wire replacement applications** - WESPEH can be efficient used in applications where wire installation is very difficult or even impossible. Modern cars have many sensors that are attached to the control unit of the car using wires. The average wiring length in a modern car is around 2km. The lengthy wire installation raises the risk of a system malfunction as wires exposed longer to environmental effects can be damaged. Most of these sensors could be replaced using WESPEH devices. Similar as in cars airplane wire installation is also a significant issue. As an example to reduce the costs of the installation WESPEH device can be used to replace blinds control of the airplane window. Wire installations are also an issue in industry halls and productions. Mechanical and solar power can be used effectively as ambient energy sources. Position switches with an autonomous energy generator are easily fitted in areas difficult of access and flexible when it comes to moving them elsewhere. Typical ambient powering possibilities are the following: thermal difference, vibration, motion energy.

## 2.5 WESPEH design

The WESPEH scenarios and use-cases are showing that the applications can be heterogeneous. For their successful implementation an appropriate design methodology is required. Embedded system design is a very complex problematic as the system consists of hardware and software. This brings the complexity to the design problematic since the method of hardware and software design has a significant difference. Hardware systems are designed as the composition of parallel components represented as analytical models. Software systems by contrast are designed from sequential components (objects, threads) represented by computation models [7]. For a successful design, it is crucial to have good synchronization between hardware and software design steps. Failure to do so can lead to a hardware/software design gap problem.

Several known design approaches try to solve these problems with the help of model abstraction. Although these models seem to offer a generic solution, they are often diametrically opposed. A certain model can only be applicable for a certain type of embedded system. This problem has already been recognized by both the industrial and scientific communities and there are several ongoing research studies which are attempting to address these problems [9]. Other embedded system design approach is through critical system engineering. This approach tries to guarantee system safety at all costs, and is achieved mainly by using massive redundancy of resources. WESPEH can't be characterized as critical systems and the system has limited energy resources for redundancy.Best-effort system engineering design approach tries to optimize system performance when the system operates under expected conditions. Best-effort system engineering is based on average case analysis and on dynamic resource allocation. In case of a WESPEH system an average case analysis doesn't resolve critical factors such as critical energy state.

WESPEH devices are heterogeneous as the powering possibility dictates the critical requirements on the system. This heterogeneity makes it difficult to follow a generic design model. Design process of WESPEH with similar functionality but different harvesting method may completely differ. There is a different proceeding for a system powered by an electro dynamic converter where the main emphasis is placed on the optimization of the energy generator as for a solar powered device where the emphasis is concentrated on the long-term energy storage. We believe that best WESPEH design proceeding are practical straight-forward techniques.

### 2.5.1 Design approach

Based on several years of experience and observation of WESPEH devices development we conclude that the most successful WESPEH projects were those where several functional prototypes with hardware and software were built and evaluated. Therefore we propose a straightforward design technique based on prototype quantification. Prototyping of WESPEH devices has several advantages:

- gives the opportunity to test various design options
- helps to determine design flaws earlier
- simplifies the energy budget estimation
- verifies the product feasibility
- gives the possibility for radio performance test
- enables the definition of test in the early project phase

It is important that during the development lifecycle not only one but several prototypes are built in parallel. This makes the evaluation easier and ensures to find most optimal technical solution.

There are many different possibilities how to build relative fast working WESPEH prototype. On the other hand just by prototyping is not guaranteed that the final system will be feasible to become a product for mass production. Lot of universities and research institutes has succeeded in creating their own WESPEH prototypes. However, most of these projects remained in the research

phase and were not realized as commercial products. We suggest that the explanation of this phenomenon lies in a common problem of underestimating the complexity of WESPEH design problematic. The difficulty of the WESPEH design is given by the fact that there are usually opposing requirements placed on the device. In order to fulfill these requirements oft critical design constraints are being violated. As long as a device is developed as a prototype, design constraints like the number of components, component availability, form factor, radio performance, device lifetime, temperature dependency, production stability, aging of energy storage and price are disregarded. All these minor problems will turn to unsolvable issues in the moment the device has to become a product ready for mass production. To overcome these problems our proposal is that prototypes have to be evaluated as the subject of design constraint. Based on this knowledge we define a straight forward design approach that consist of the following steps:

1. Requirement definition
2. Design constraints definition
3. Preliminary specification
4. Prototype creation
5. Prototype evaluation
6. Prototype quantification
7. Technical specification
8. Product development

As the first step of the development the requirements on the WESPEH device has to be settled down. An important second step for a WESPEH development is to identify the major design constraints. For a typical WESPEH system – based on the architecture and application scenarios in the previous chapter – we can identify five typical design constraint groups:

- Energy aspect
- Radio platform
- Microcontroller platform
- Software aspect
- Component aspect
- Price aspect

With design constraints defined a preliminary specification of various technical solutions is created. The specification is preliminary as it is not yet clear which technological solution will be the most suitable. In the following design step functional prototypes - inclusive hardware and software - are created with various technical solutions. As the next step each prototype is evaluated and compared to the defined requirements. The sixth step includes the quantification of the prototypes as the subject of design constraint. The prototypes are given a rate according the evaluation of the design constraint groups. The best rated prototype is selected and a specification created. Finally the product development is started which can follow any common design procedure.

### 2.5.2 Prototypes quantification

In order to determine which prototype has the best assumption for a successful design the prototypes has to be quantified. This is done in two steps:

- prototype rating using decision matrix
- constraint group equilibrium determination

The prototype quantification process is the following. The number of prototypes to evaluate is *i,* the number of constraint groups are designated as *j*. Each evaluated prototypes constraint group is assigned a rating value designated as $x_{ij}$. From this we can define the decision matrix as follows:

$$X_{ij} = \begin{pmatrix} x_{11} & \cdots & x_{1j} \\ \vdots & \dots & \vdots \\ x_{i1} & \cdots & x_{ij} \end{pmatrix}, x_{ij} \in \{1, ..., 5\} \quad (1)$$

where $x_{ij}$ is the prototype design constraint rate. 1 is the lowest rate and 5 is the highest rate. As the design constraint groups can have different importance dependent on the application, each group is assigned a weighting factor satisfying the condition that is:

$$\sum_{j=1}^{n} w_j = 1 \quad (2)$$

where $w_j$is the weight of the constraint. The prototypes summary rate designated as $\overline{R}_i$ is calculated using the weighted average of the constraints in following way:

$$\overline{R}_i = \sum_{j=1}^{n} w_j x_{ij} \quad (3)$$

where $w_j$ is the weight of the constraint and $x_{ij}$ is the rate of the prototype design constraint.

### 2.5.3 Design constraint equilibrium

Design constraints place an overall boundary on the design process. All these parameters are tied together and for a successful design, they must be properly balanced. Although prototype design with unbalanced constraint groups may fulfill all the core requirements of the system, it is an indication that for long term the selected choice may lead to failure. We can demonstrate this fact on example.

Let's consider a design of WESPEH solar window contact. The core requirement of the device is appropriate radio signal strength to cover an area of 300 meters. The prototype design for this purpose requires energy for the transmission with 6dBm signal strength. A solar cell with the size of 5cm$^2$ could deliver the right amount of energy but the core requirement of unobtrusive hardware limits the size of the cell. The selected microcontroller power requirements are relative high (still in target range), but it has significant RAM, FLASH and I/O reserves. One solution to reach the design objective is to change the type of microcontroller with lower power consumption. The microcontroller with lower power consumption has less RAM, FLASH and no I/O reserves. Naturally microcontroller is rated lower. Although the constraints rating is not balanced, with the new microcontroller the power consumption is in the target range. The decision is made to use this prototype for the final device design. As the project enters its final phase integration test shows that at high temperature the microcontroller A/D converter is not linear. This issue could be easily fixed using a software workaround with a conversion table. Unfortunately the new microcontroller has no FLASH reserves

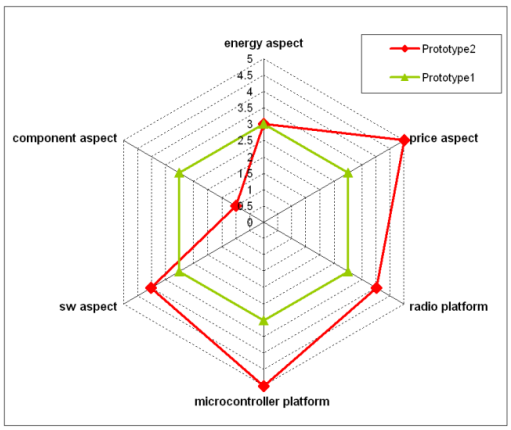


**Figure 2: Equilibrium demonstration**

to implement this solution. The design fails and a new development cycle has to be started. The optimal solution in this case would be instead changing the microcontroller to adjust the radio constraint. One possibility would be to introduce a better antenna thus leading to a lower transmission power need. This would fulfill the power requirement. The unbalanced design constraints is an indication for a not optimized design. From these facts, we can define the following statement:

*Definition 2. A WESPEH design is successful when the rating of constraint groups are in equilibrium.*

For the visual determination of the constraint group equilibrium the rating values represented in a radar chart are useful. For demonstration purpose Figure 2 shows the evaluation of two WESPEH prototypes. Prototype-2 has overall better rating except of the components aspect that is very low. Prototype-1 has a worse overall rating but offers a solution with balanced design constraints.

When several prototypes are showen in a radar chart, it is difficult to evaluate the equilibrium state. For this reason to determine which prototype has the closest to the equilibrium state we define a balance rate calculated using the standard deviation $s_i$ of the rating values in a following way:

$$s_i = \sqrt{\sum_{j=1}^{n} w_j (x_{ij} - \overline{x}_i)^2} \qquad (4)$$

where $w_j$ is the weight of the design constraint, $\overline{x}_{ij}$ is the rate of the design constraint, $x_i$ is the average rate of the prototype $i$. The quantification of the prototypes is performed by the evaluation of the summary rate and the balance rate. The significant evaluation parameter is the summary rate. The prototype with the highest summary rate is taken and the balance rate is evaluated. The following results are possible:

$$\begin{aligned} R_k > R_l, \quad s_k > s_l \\ R_k > R_l, \quad s_k = s_l \\ R_k > R_l, \quad s_k < s_l \end{aligned}$$

where $R_k$ and $R_l$ is the prototype summary raiting, and $s_k$ and $s_l$ is the balance rate of prototype $k$, $l$. In case the summary rate between two prototypes is higher but the balance rate is lower, it indicates a problem. Although the prototype is rated the best, unbalanced design constraints represent a risk. In such case there are three possibilities:

- the prototype is redesigned in a manner to have balanced design constraints
- the prototype with lower summary rate but better balance rate is used as the final design
- if the unbalanced prototype represents no rational risk the prototype with the highest summary rate is used

The success of the design methodology is dependent on the detailed analysis of the constraint groups. In the thesis we discuss briefly the problematic of each design constraint and place the focus on the software aspect.

## 3. Operating system for wireless embedded systems powered by energy harvesters

The following aim of our research was to analyze the WESPEH software problematic. Based on the application scenarios, we lay down the core requirements for WESEPEH operating systems.

- Abstract the hardware
- Abstract the power management
- Abstract wireless interface and implement a wireless protocol stack
- Thin resourced
- Modular
- Hard real-time
- Support parallel task execution
- Enables remote configuration

All these requirements are discussed in the thesis in detail. Our further research aim was to find an appropriate operating system that is capable of running on WESPEH platform with the limited resources and fulfill the defined requirements. A comprehensive analysis of thin resourced operating system was performed. The analysis results are discussed in the thesis. We have come to the conclusion that neither of the OS fulfills all the requirements. The main problematic of the analyzed operating systems are the RAM and ROM resources. Furthermore most of the systems were supporting a cooperative scheduler thus violating the hard RT requirement. The analysis results led us to the conclusion that there is a need to discuss the problematic of operating system design and to define architecture for a WESPEH operating system.

The thesis discusses the WESPEH OS design problematics in detail. In this extended abstract we summarize the most important facts: In a typical WEPSEH OS architecture, the application is tightly coupled with the OS. A generic architecture of WESPEH OS is showen on Figure 3. Either the application source code is compiled with the OS or the application objects are linked to the OS objects. Furthermore the application has the possibility through system calls directly access the hardware abstraction layer. For a successful WESPEH OS implementation the listed design consideration has to be followed:

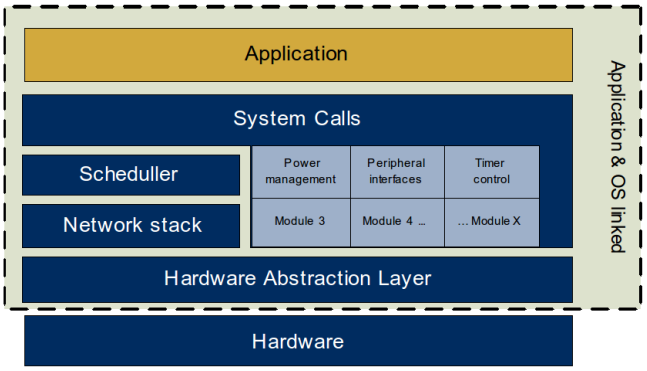


**Figure 3: WESPEH OS Architecture**

- the WESPEH OS implementation language is C
- the OS has no kernel or user space, the OS code is statically linked to the application
- the OS has no memory management
- the OS implements no file system
- the OS provides the abstraction of peripheral interfaces
- the OS is modular, the OS modules are linked only if they are used by the application
- the OS implements no device drivers, the drivers are replaced with a thin hardware abstraction layer
- the OS implements a light-weight preemptive priority scheduler
- the scheduler algorithm and type and priority can be adjusted dependent on the application needs
- the radio protocol stack is part of the OS implementation
- the power management is part of the OS implementation
- the power management module provides interfaces for to the application to control and monitor energy resources

Based on these design consideration we have implemented an operating system DolphinAPI. The initial implementation of the system was done for the EO3000I hardware platform. DolphinAPI design and implementation is discussed in the thesis. The next step in our research is to evaluate the DolphinAPI performance of the operating system and compare it to other operating systems. This requires that another benchmarked operating system runs on the EO3000I HW platform and implements the EnOcean Radio Protocol stack. The analysis showed that the best OS candidate for WESPEH systems was TinyOS. To be able to provide evaluation measurements between TinyOS 2.x and DolphinAPI we have ported TinyOS 2.x to the EO3000I platform and implemented the EnOcean Radio Protocol stack on this system. The details of the porting is discussed in the thesis.

## 4. WESPEH operating system quantification

Following aim of our research work was to find an appropriate evaluation method to compare WESPEH operating system.

### 4.1 Related work

Measuring operating system performance and comparing the systems to each other in an objective way is a difficult problem. The difficulty resides primarily in the fact that various systems, even if intended for the same application area, can exhibit different functionalities. No standard definition exists how the features shall be measured or compared in an objective manner. Even using the same benchmark package on two different systems does not ensure that the information provides the answer to which system is better, as is shown in a case study from paper [12].

Typical OS selection method is the pragmatic approach – by observing and comparing OS resource requirements and architecture. Such proceeding makes it possible to exclude systems based on the HW resource constraint; on the other hand, this approach does not give an answer to the question how resource efficient does the application run on the operating system. Therefore a quantitative analysis method is needed through which the performance of the operating systems can be determined. There are three common approaches for quantifying OS performance: Macro-benchmarks, Micro-benchmarks, Kernel Profiling.

Macro-benchmarking method is suitable for complex systems where system components are not accessible and the operating system appears as a black box. In such system the benchmark of single system parts is not meaningful. An example is parallel computer systems where benchmarking a single computer would be misleading. The drawback of macro-benchmark method for WESPEH OS is that it involves too many factors and the benchmark result does not reveal why a system performs well or badly. In a WESPEH system the optimization is a key factor. For a successful design therefore the reason of bad performance has to be possible to deduce from the evaluation results. Evaluating of a system just by micro-benchmarks has the drawback that micro-benchmarks results can be correctly interpreted only with an understanding what role each micro-workload plays in the complete system. While kernel profiling is useful in spotting the performance bottlenecks of an operating system running a specific workload, it often requires unavailable kernel source code of the operating system.

An application specific benchmarking based on vector and trace methodology is described in the paper [12]. The paper states that standard benchmarking methods as microbenchmarks or macro-benchmarks often provide only partial information on how well a particular system will handle an application. The authors of the paper argue that the system performance should always be measured in the context of a particular application. The paper discusses application-directed benchmarking techniques and introduces three different approaches: vector based, trace based and hybrid based methodology. Techniques from these paper are used for the evaluation in various other works. Leopold Martin, the creator of the TinyOS 8051 workgroup, has adapted the vector-based methodology in his work [10] and evaluated Mote performances on different HW platforms. The vector-based methodology was also used by A. Brown to evaluate the Apache web server performance described in his thesis [2]. The methodology was furthermore used for desktop operating system and [3] Java Virtual Machines [15] evaluation.

As WESPEH operating systems are coupled tight with the application therefore the evaluation of the system has

to be made in the context of the application. Common benchmark methods focus entirely on the computational performance, neglecting other important features of an OS such as energy consumption. These features are crucial for WESPEH systems therefore they have to be considered during the evaluation. To derive an appropriate quantification methodology for WESPEH operating system, we can summarize the result of this discussion and define the following requirements:

- the methodology must characterize the WESPEH OS performance, energy requirements, memory footprint
- the methodology must consider the application influence
- the methodology has to give the possibility to determine which components of the OS behaves badly
- the methodology must give clear quantification result from the complete WESPEH operating system in order to make the results comparable
- the methodology must give the possibility to predict performance of other applications

From the discussed benchmark methods Seltzer method was the one that performed the performance evaluation in the context of the application. Consequently we adapted Seltzer methodology and enhanced with the energy, memory footprint and parallel execution aspect. In our proceeding we were motivated by Leopold Martin work [10] who similarly adapted Seltzer methodology to evaluate the Mote performance. Leopold in his work was considering the HW influences in his methodology while in our proceeding we will consider only the performance from the operating system point of view. Our work is the first to propose a performance methodology in the context of WESPEH operating systems inclusive energy consideration.

### 4.2 Definition of the methodology

The first step for finding a methodology for WEPSEH OS quantification is to define the benchmark parameters that characterize the system. We can define these parameters by deriving them from the key requirements placed on WESPEH OS. The key parameters that characterize a WESPEH OS are as follows: *execution time [ms], power consumption [mW], memory footprint [kBytes].*

Execution time is derived from the hard real-time and parallel task requirements and gives overall information about the operating system execution speed. The power consumption is derived from the power management requirement. It defines how power efficiently the operating system controls the application execution and how it abstracts and optimally uses the HW power management blocks. The footprint parameter is derived from the limited HW resources requirement. It indicates the OS capability to run on limited memory microcontrollers.

To understand the performance of a specific operating system the system has to be decomposed to primitives. Each primitive operation in an operating system takes a fixed amount of time to be completed. By characterizing the performance of these primitives, it is possible to relate the analysis results to the complete system performance. A primitive operation in a WESPEH OS is either a system call to an operating system function or an event. As the basis of the breakdown of the WESPEH OS, we will use the generic WESPEH OS architecture showen in Figure 3. We will consider only the high level layers of the operating system in order to avoid the influence of latencies introduced by the hardware abstraction layer that is HW platform specific. We can derive the primitive operations of the WESPEH OS by looking at the common operations executed in application scenarios. These operations are the following: *radio communication, peripheral control, power management, task switching.*

The primitive operation performances are represented as a vector quantity and the vector collecting this information is called system characterization vector. The primitive performance is measured by running an appropriate micro benchmark. The vector itself is meaningless - it represents only summarized micro benchmark values. To obtain a meaningful result, the system characterization vector has to be combined with a corresponding application vector. Each component in the application vector represents the application's utilization of the corresponding primitive operation as either a frequency, number of bytes or other quantity. Once the application vector is determined, it can be used with different system characterization vectors. The final benchmark of the system can be calculated as a dot product of the system characterization vector with the application vector.

$$System\,performance = \overline{s}.\overline{a} \tag{5}$$

where $\overline{s}$ is the system characterization vector and $\overline{a}$ is the application vector. In the next chapters we will derive the vector parameters for WESPEH OS.

#### 4.2.1 System characterization vector

To apply the Seltzer method to a WESPEH operating system requires in the first place to determine the parts of the system characterization vector. For the initial step we can use the set of primitive operations defined in the previous chapter. We have to select the appropriate benchmark parameter for these primitive operations. From the defined benchmark parameters, the best parameter that characterizes the operating system primitive is the execution time. The execution time parameter gives overall information about the operating system reaction time. The longer the OS overhead latencies take, the more energy is consumed by the application.

The exact time measurement of WESPEH OS can be performed using execution profiling. WESPEH OS source code is available as the applications have a strong dependency on the OS. Using a simulator and an execution profiler the execution time of functions composing the primitive operation can be measured. In a measurement performed by a simulator, all parallel function influences can be eliminated, thus the result is very precise. At least two measurements have to be performed. In the first measurement the target primitive operation has to be set up for the best case execution; in the second measurement the target has to be set up for the worst case execution time.

$$\vec{t} = \begin{bmatrix} \frac{t_{RxBest}+t_{RxWorst}}{2} \\ \frac{t_{TxBest}+t_{TxWorst}}{2} \\ \frac{t_{schedOverheadWorst}+t_{schedOverheadBest}}{2} \\ \frac{t_{powerm_1Worst}+t_{powerm_1Best}}{2} \\ \vdots \\ \frac{t_{powerm_NWorst}+t_{powerm_NBest}}{2} \\ \frac{t_{peripheral_1Worst}+t_{peripheral_1Best}}{2} \\ \vdots \\ \frac{t_{peripheral_NWrost}+t_{peripheral_NBest}}{2} \end{bmatrix} \quad (6)$$

An additional advantage of this method is that execution profiles apart the time measurement can give additional useful statistical data like the percentage of code coverage, depth of stack usage, etc. We can use these data to get a better picture of the characterized primitive operation. The further advantage of this method is that the measurement focuses on the performance of the implemented code and the latencies introduced by the HW are not considered in these measurements. By having the best and worst execution time of the primitive the typical execution time of the primitive has to be determined. The typical execution time can be calculated using: arithmetic mean, harmonic mean, geometric mean, weighted harmonic mean. In paper the problematic of means are discussed and the conclusion is made that after analyzing several hundreds of benchmark results the influence of various mean is small. Based on this paper and on the fact that in our evaluation method the execution time is not the only performance parameter, we can conclude that the selection of the mean algorithm doesn't have a significant impact on the result of our evaluation. Therefore for the execution time calculation we will select algorithm of the arithmetic mean. Applying this method we can get the characterization vector of WESPEH OS as showed in equation (6) where $t$ is the time characterizing the primitive execution.

### 4.2.2 Power consumption vector

The way in which application and operating system utilizes the underlying hardware has a significant impact on the complete power dissipation of the system. Different studies in papers [14] [13] observed that the choice of algorithm and other high level programming decisions measurably affect the system power. The same effect can be observed if the operating system manages the HW power resources effectively. For example, a floating-point division operation executed with active radio interface can consume three times as much energy as executing the same operation with inactive radio interface. All these examples indicate that for WESPEH OS evaluation the power consumption parameter has to be considered.

Several techniques exist to determine the software power consumptions. In the simulation-based approaches, the processor architectural model is used and the software power consumption is estimated through simulation at different abstraction levels [4]. This method offers high flexibility for experimenting, nevertheless, commercial microprocessors' circuit-level implementation is not available. Another common approach is to determine the instruction power consumption and by combining this information with the SW analysis of the instruction level [8] the complete SW power consumption can be estimated. Using such a method does not cover the measurement of dynamic power management of the SW where the CPU is entered into standby mode or unnecessary peripherals are switched off in order to save power. The most accurate results can be obtained by measuring the physical energy consumption of the application executed on the real HW platform. We can apply this measurement method by measuring the power consumption of each operating system primitive operation. A special application can be created that utilizes only one primitive operation in a loop with default parameters. During the implementation of these applications, we have to make sure that there is no other parallel task execution. We measure the power consumption of the primitive as the function of time. The measured values we get are summarized in a power consumption vector:

$$\vec{p} = \begin{bmatrix} p_{RadioRx} \\ p_{RadioTx} \\ p_{Scheduler} \\ p_{Powerm_1} \\ \vdots \\ p_{Powerm_N} \\ p_{Peripheral_1} \\ \vdots \\ p_{Peripheral_N} \end{bmatrix} \quad (7)$$

where p is the energy cost of the primitive in milliwatt-seconds [mWs]. One could argue that the power consumption vector represents the HW blocks constant power consumption and these values are HW dependent thus constant for different WESPEH OS. In fact, this may be the case if the analyzed OS primitive operation would not adjust the HW power modes during execution. An example of this scenario would be if the Radio Tx primitive would immediately turn the radio Tx state machine on and would turn it off after the packet transmission was completed. In such case, the power consumption parameter would equal to the radio Tx HW power consumption. On the other hand, in a WESPEH OS the SW tries to reduce the power consumption by applying different algorithm. For instance, the OS can put the system in standby while waiting for the radio Tx state machine to start. Alternatively, the OS can switch on the Tx state machine immediately before the packet transmission. The ways in which these algorithms are applied are operating system specific and have an influence on the power consumption. Furthermore each primitive makes various function and system calls to the OS. All these calls have a certain overhead dependent on the OS driver and interfaces implementation. As the power consumption vector incorporates these overhead latencies it gives more precise information how efficient the OS implementation is.

### 4.2.3 System matrix

We have defined a methodology where three vectors characterize the WESPEH OS. The weakness of the current methodology is that it does not reflect the strategy of parallel primitive operation execution. In all vectors, we consider that the primitive operation's execution happens in a consecutive manner and the actual primitive operation is the only running task in the operating system. Latencies introduced by scenarios where primitive operations

are blocking each other or where one awaits the output of another are not modeled in the current vector definition.

The strategy of primitive operations' execution significantly influences the WESPEH OS evaluation result whereas it has a major influence on both execution time and power consumption. We define a system matrix M of dimension N x N where N is the number of primitive operations in the system characterization vector. The system matrix models the strategy of primitive operation execution by encoding parallel states of primitive operations. The rows and columns represent the primitive operations. The elements of the matrix represent the delay or enhancement factor of the parallel execution of two primitive operations. If all diagonal elements equals 1 and other elements equal 0 we talk about sequential execution of tasks. The system matrix is always multiplied with the system characterization vector. By the definition of system matrix we should note that the matrix is both application and operating system dependent. The behavior described in the system matrix depends on the application flow.

A certain scheduler type can give a good performance with a certain application while the same type of scheduler can have a bad performance with other application flow. To model the whole system precisely the whole application flow with all combination of the application states would have to be represented as different system matrixes. This would make the model very complex, therefore for the sake of simplicity we will consider that each element of the system matrix models the average behavior of the application flow. The system matrix can be determined by analyzing the operating system scheduler strategy and by analyzing the trace log of the current application.

#### 4.2.4 Definition

Based on the discussion in previous chapters we can define the performance of the WESPEH operating system.

*Definition 3.* Let $O = \{o_1, o_2, ..., o_n\}$ be a WESPEH operating system defined as a tuple of primitive operations and $o_i, i \in \{1, ..., n\}$is the primitive operation, let

$$\vec{s} = \begin{pmatrix} s_{o_1} \\ \vdots \\ s_{o_n} \end{pmatrix} = \begin{pmatrix} \frac{t_{o_1 best} + t_{o_1 worst}}{2} \\ \vdots \\ \frac{t_{o_n best} + t_{o_n worst}}{2} \end{pmatrix} \tag{8}$$

be the system characterization vector dependent on the WESPEH OS where the component $s_{o_i}$is the arithmetic mean of the primitive operation execution time $o_i, i \in \{1, ..., n\}$ expressed in milliseconds [ms].

Let

$$\vec{p} = \begin{pmatrix} p_{o_1} \\ \vdots \\ p_{o_n} \end{pmatrix} \tag{9}$$

be the power consumption vector dependent on the WESPEH OS where the component $p_{o_i}$is the energy cost of primitive operation $o_i, i \in \{1, ..., n\}$ expressed in milliwatt-seconds [mWs]. Let $M \in \Re^{n \times n}$

$$M = \begin{pmatrix} m_{11} & \cdots & m_{1n} \\ \vdots & \ldots & \vdots \\ m_{n1} & \cdots & m_{nn} \end{pmatrix} \tag{10}$$

be the system matrix where $m_{ij}$characterizes the parallel execution of primitive operation $o_i \, and \, o_j, i, j \in \{1, ..., n\}$. The matrix is dependent on the WESPEH OS scheduler architecture and application execution trace. Let

$$\vec{a} = \begin{pmatrix} a_{o_1} \\ \vdots \\ a_{o_n} \end{pmatrix} \tag{11}$$

be the application vector where $a_{o_i}$is the call frequency of the primitive operation $o_i, i \in \{1, ..., n\}$. Then we know that

$$\Pi = a^T \left[ \left( s^T M \right)^T \otimes p \right) \tag{12}$$

is called the performance of the operating system composed of primitives operation $o_i, i \in \{1, ..., n\}$. $\otimes$is the direct product of the vectors. The triplet ($\Pi$, $m_{ram}$, $m_{rom}$) are the quantification parameters of the WESPEH operating system. Using these parameters a certain application runtime on different WESPEH OS can be evaluated and compared. To get precise and objective evaluation results the measurement of the vector parameters has to be done on the same HW.

### 4.3 Experimental results

We have applied the methodology to compare the DolphinAPI and TinyOS. For the operating system evaluation we have used a simplified version of a WESPEH heating regulator described in the heating automation scenario. The details of the measurement is described in the thesis.

#### 4.3.1 Applying the methodology

With the data available the methodology can be applied to determine on which OS the application behaves better. The first step of the evaluation is to determine the relevant OS primitives the application uses. From the application we can derive the following primitive operations:

$O = \{o_{radio\_rx}, o_{radio\_tx}, o_{standby}, o_{uartTx}, o_{analog}\}$

$$\begin{aligned} \vec{s}_{DolphinAPI} &= \begin{pmatrix} 1949 \\ 1241 \\ 25 \\ 861 \\ 10000 \end{pmatrix} \\ \vec{s}_{TinyOS} &= \begin{pmatrix} 2267 \\ 1016 \\ 27 \\ 634 \\ 10000 \end{pmatrix} \end{aligned} \tag{13}$$

The second step of the evaluation is the determination of the system characterization vector. For this vector the worst and best case execution time parameter of each

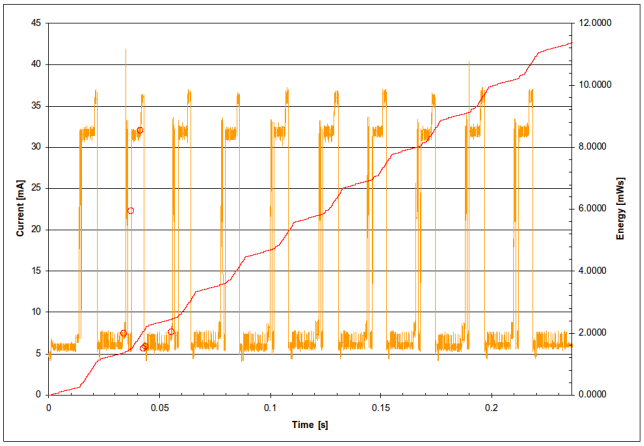


**Figure 4: Current flow of WESPEH running the application on DolphinAPI**

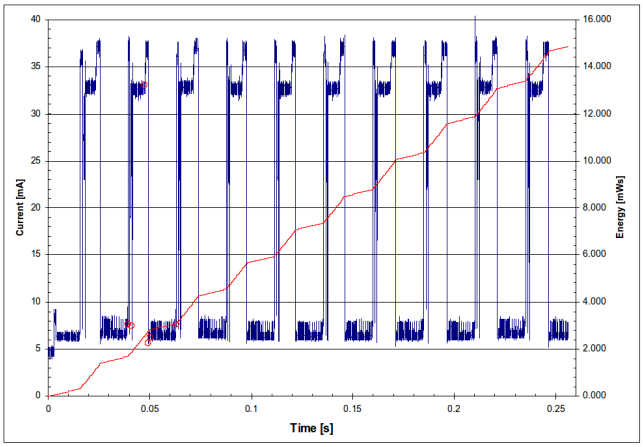


**Figure 5: Current flow of WESPEH running the application on TinyOS**

primitive is needed. As discussed in previous chapter the best method to determine these values using an execution profiler. We use the Keil simulator, debugger environment. With the help of the execution profiler the worst case and best case execution time of primitive operations can be determined. To have a precise result, we have created various test cases that apply various loads on the primitive functions. From these results, we can create the system characterization vector for both operating systems as showen on 13.

The following step of the quantification methodology is to determine the application vector and system matrix. The sampled data from the logic analyzer are analyzed and processed. Based on the trace log analysis and by knowing the scheduler strategy we can create the system matrix of both operating systems. First we will determine all primitive operations' parallel execution states. We create the system matrix by weighting the parallel execution states. Note that during the calculation if two primitive operations run parallel we consider in the matrix the primitive operation having the higher power consumption. This can be interpreted by populating the row of the primitive operation with the higher power consumption. For the sake of simplicity, we consider fair CPU arbitration between parallel-executed primitive operations by populating matrix elements with 1/2 value. The behaviors of parallel tasks that may block each other caused by the cooperative scheduler in TinyOS are represented as an increase of the element value by 1/2. We apply this value as an addition to elements where this behavior can significantly increase the power consumption, in our case on the parallel execution of Analog & UartTx and on RadioTx & UartTx. The result system matrixes are the following:

$$M_{DolphinAPI} = \begin{bmatrix} 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0.05 & 0 & 0 & 0 \\ 0.05 & 0 & 0.05 & 0 & 0 \\ 0 & 0.05 & 0 & 0.13 & 0.05 \\ 0 & 0 & 0 & 0 & 0.05 \end{bmatrix}$$

$$M_{TinyOS} = \begin{bmatrix} 0.09 & 0 & 0 & 0 & 0 \\ 0 & 0.145 & 0 & 0 & 0 \\ 0.045 & 0 & 0.05 & 0 & 0 \\ 0 & 0.045 & 0 & 0.095 & 0.08 \\ 0 & 0 & 0 & 0 & 0.045 \end{bmatrix}$$

The application vector determination is in this simple scenario straightforward as each primitive was executed exactly 10 times.

$$\vec{a} = \begin{pmatrix} 10 \\ 10 \\ 10 \\ 10 \\ 10 \end{pmatrix}$$

The power consumption vector determination is based on the current measurement performed on the shunt-resistor. The current flow of both operating system is showed in Figure 4 and Figure 5.

To get the energy consumption of the system the current consumption has to be integrated. The result of the integral is designated as a red line. For the methodology the energy needs of the primitives has to be determined. This can be calculated the following way. The trace log timing has to be synchronized with the energy measurements to see in which moment which primitive operation was executed. This way the time sequences of the primitive operations can be determined. Following step is to calculate the integral of these time sequences. In such way we get the energy consumption of the primitive which will be the value for our power consumption vector. The red circles in the diagram represent the time synchronization points of the primitives.

The final power consumption vectors derived from the measurements for both operating systems are the following:

$$\vec{p}_{DolphinAPI} = \begin{pmatrix} 0.1382 \\ 0.4196 \\ 0.0154 \\ 0.2612 \\ 0.1428 \end{pmatrix}$$

$$\vec{p}_{TinyOS} = \begin{pmatrix} 0.2237 \\ 0.6690 \\ 0.0216 \\ 0.2835 \\ 0.1982 \end{pmatrix} \quad (14)$$

Now when all vectors are known we can perform the performance calculation based on the equation . The performance parameter results are the following:

$$\begin{aligned} \Pi_{DolphinAPI} &= 1.782 \\ \Pi_{TinyOS} &= 2.799 \end{aligned} \tag{15}$$

Additional to these measurements we have performed analysis of the memory consumption. The details of these analysis can be found in the thesis.

### 4.3.2 Results analysis

From the results we can see that the application executed on DolphinAPI behaves better than on TinyOS. The application performance running on DolphiAPI is around 1.5x better than TinyOS. This result was already possible to predict from the trace log and energy measurements. By comparing the current flow in diagrams shown in Figure 4 and Figure 5 can be determined that the execution time of the application in TinyOS took 28ms longer than in DolphinAPI. This is an interesting fact, as when looking at the system characterization vector (13) of both operating systems we can see that the vector values indicates that the execution duration of some primitives in TinyOS are faster than in DolphinAPI (radioRx or Standby). The system characterization vector is created based on micro-benchmark simulation method. Therefore the scheduler, event processing overhead and other influences are not embedded in these values. This is the reason why standalone micro-benchmark results can be misleading. In our evaluation method the real application execution and the whole system influences are embedded in the power consumption vector. Thus all these influences are visible in the final evaluation.

The major energy needs in a WESPEH has the radio receiving interface. From the measurements the power consumption vector (14) indicates that in both systems the most energy was consumed by the radio packet transmission. This is explained by the fact that the transmitter $A$ responded on the packet very fast therefore the radio Rx interface was active only a very short time. This proves the fact that it is important to measure the system always with the executed application. In a macro benchmark where only the HW power consumption values would be considered, this fact would not be visible.

Further measurement shows that the way TinyOS has controlled the power resources of the microcontroller was not as energy efficient as DolphinAPI did. The energy difference can be explained with the operating systems architecture and scheduler difference. The scheduler in TinyOS is used more often as it is responsible for the management of events and tasks. Therefore the scheduler overhead in TinyOS is more significant than in DolphinAPI. Every time a new event occurs or a task is started or stopped the scheduler is invoked. In DolphinAPI the scheduler is invoked once in 1ms. As the tasks are constant and scheduling happens only on the system level no extra overhead is represented. Further reason of the increased power requirements in TinyOS can be explained with the cooperative behavior of the scheduler. Cooperative tasks may block the system for a certain time that may cause that turning off the radio interface can take longer as in DolphinAPI. Also the switch of microcontroller power modes can be realized only in the moment when the application receives the processor time.

To prove our evaluation correctness and to indicate this bottleneck we have performed additional micro- macro-benchmark measurements. These measurement results are discussed in the thesis in detail. The results indicate the cooperative scheduler as the major bottleneck of TinyOS.

### 4.3.3 Performance prediction

We have to note that the evaluation results tell how the current application behaves on the operating system. Although we have proven the fact that DolphinAPI radio processing is more optimized, the performance evaluation doesn't reveal if DolphinAPI in general is better than TinyOS. Comparing operating system in generic manner is not an objective proceeding as it always comes to the application and the scenario where the operating system is used. By changing the application implementation or measurements performed with different application could give a result where the TinyOS performance would be better.

One of the biggest advantages of the suggested methodology is that once all the vector values are known additional performance estimation can be performed without extra measurement effort. By knowing the values of the system characterization vector, power consumption vector and system matrix for both DolphinAPI and TinyOS we can calculate and predict any application performance and compare its runtime to the analyzed operating systems by determining the application vector. For demonstration purpose let's take the following example. We will consider the same application scenario but with slightly modified execution. The application will perform only four packet transmissions, and one reception. Furthermore there will be an intensive UART communication where 30 packets will be transferred to control the heating valve. In addition only 5 analog measurements will be performed. In such case the application vector definition is the following:

$$\vec{a} = \begin{pmatrix} 4 \\ 1 \\ 5 \\ 30 \\ 5 \end{pmatrix}$$

Applying the methodology on this modified application we will get the following performance results:

$$\begin{aligned} \Pi_{DolphinAPI} &= 1.418 \\ \Pi_{TinyOS} &= 1.310 \end{aligned} \tag{16}$$

The results indicate that TinyOS would behave slightly better as DolphinAPI with the described application. The methodology makes it possible to predict how the optimization of the operating system routines influences the overall system performance.

## 5. Conclusion

This work deals with the problematic of the design and development of wireless energy autonomous devices with the focus placed on the software problematic. Based on observation and experience we claim that energy autonomous systems cannot be treated as generic embedded systems nor as conventional wireless sensor nodes. Hence a new device category called wireless embedded systems powered by energy harvesting was introduced. The architecture of these devices was established and application scenarios were presented.

We presented a design technique based on constraints analysis and prototype quantification. It is concluded that for successful WESPEH implementation the design constraints have to be in equilibrium. From the discussed design constraint group the work deals with the SW problematic in detail.

The architecture and design aspects of a WESPEH OS is elaborated and defined. Based on these design aspects the successful implementation of a new WESPEH operating system called DolphinAPI is presented. The initial implementation of the operating system was done for the EO3000I HW platform. To prove the implementation feasibility TinyOS is ported to the EO3000I platform and the implementation of the EnOcean protocol stack is performed in NesC.

We have introduced a new quantification methodology intended to be used for WESPEH operating systems. The methodology is based on vector definition and trace-driven performance analysis. The main contribution of the methodology is that it can be used for prediction of application running on various WESPEH operating systems. Using the methodology we have performed measurements of an application running on DolphinAPI and TinyOS. The results showed that the application running on DolphinAPI had a slightly better score as running on TinyOS. Using the methodology the TinyOS performance bottlenecks were identified.

### 5.1 Contributions

The title of the work suggests that the objective of this work was to cover the whole problematic of Wireless Embedded Systems Powered by Energy Harvesting. As this is a very wide topic, single research thesis can't fulfill such an advantageous aim. Still the title of the work was selected intentionally. Despite the fact that the main focus in the work is placed on the software aspect, the primary objective was to raise the awareness of the research community on the problematic of WESPEH.

The work brings several new theoretical and practical contributions to the scientific community in the field of applied informatics by the definition WESPEH architecture, introducing application scenarios, definition of design methodology and constraint group. Furthermore the work lays down the requirements and design aspects of operating system designated to be used by WEPSEH. The work also contributes the area of operating system evaluation by the introduction of a quantification method based on vector definition and trace-driven performance analysis. Further on a significant contribution to the research community is the porting of TinyOS to a new hardware platform and by the implementation of the EnOcean radio protocol stack.

## References


[1] J. Bohn, V. Coroama, M. Langheinrich, F. Mattern, and M. Rohs. Social, economic, and ethical implications of ambient intelligence and ubiquitous computing. http://www.vs.inf.ethz.ch/publ/papers/socialambient.pdf, 2004. institute for pervasive computing. In *In*. Springer-Verlag, 2004.

[2] A. B. Brown. A decompositional approach to computer system performance evaluation, 1997.

[3] J. B. Chen, Y. Endo, K. Chan, D. Mazières, A. Dias, M. Seltzer, and M. D. Smith. The measured performance of personal computer operating systems. *ACM Trans. Comput. Syst.*, 14:3–40, February 1996.

[4] R. Y. Chen, M. J. Irwin, and R. S. Bajwa. Architecture-level power estimation and design experiments. *ACM Trans. Des. Autom. Electron. Syst.*, 6:50–66, January 2001.

[5] D. D. Gajski and F. Vahid. Specification and design of embedded software/hardware systems. *IEEE Design and Test of Computers*, 12:53–67, 1995.

[6] T. Haenselmann. Gfdl wireless sensor network textbook. `www.informatik.unimannheim.de/haensel/snbook`. [Online; accessed 19-05-2011].

[7] T. A. Henzinger and J. Sifakis. The embedded systems design challenge. In *Proceedings of the 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*, pages 1–15. Springer, 2006.

[8] N. Kavvadias, P. Neofotistos, S. Nikolaidis, K. Kosmatopoulos, and T. Laopoulos. Measurements analysis of the software-related power consumption of microprocessors. *IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT*, 53(4):1106–1112, 2004.

[9] D. Lammers:. Embedded system design methodologies. `http://www12.informatik.uni-erlangen.de/research/meat/`. [Online; accessed 19-05-2011].

[10] M. Leopold, M. Chang, and P. Bonnet. haracterizing mote performance: A vector-based methodology. In *Proc. of the 5th European Workshop on Wireless Sensor Networks (EWSN)*. Springer-Verlag, Jan. 2008.

[11] K. Romer and F. Mattern. The design space of wireless sensor networks. *Wireless Communications, IEEE*, 11(6):54 – 61, dec. 2004.

[12] M. Seltzer, D. Krinsky, K. Smith, and X. Zhang. The case for application-specific benchmarking. In *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, pages 102 –107, 1999.

[13] A. Sinha and A. Chandrakasan. Dynamic power management in wireless sensor networks. *Design Test of Computers, IEEE*, 18(2):62 –74, mar/apr 2001.

[14] V. Tiwari, S. Malik, A. Wolfe, and M.-C. Lee. Instruction level power analysis and optimization of software. In *VLSI Design, 1996. Proceedings., Ninth International Conference on*, pages 326 –328, jan 1996.

[15] X. Zhang and M. Seltzer. Hbench:java: An application-specific benchmarking framework for java virtual machines. In *ACM Java Grande*, pages 62–70. ACM Press, 2000.


## Selected Papers by the Author


A. Strba, T. Krajcovic. Operating System for Wireless Embedded Systems Powered by Energy Harvesters. In *International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering*, December, 2010. Springer.

A. Strba. Operating system design challenges for wireless embedded systems powered by energy harvesters. In *Proceedings of the 7th International Symposium on Applied Machine Intelligence and Informatics*, Kosice, Slovakia, January, 2009. IEEE Computer society.

A. Strba. Design and quantification approach of Wireless Embedded Systems powered by Energy Harvesters. In *Proceedings of the Hungarian PhD Conference 2008*, Budapest, Hungary, November 10, 2008.

A. Strba, T. Krajcovic. Software Design challenges for self-sustaining Embedded Systems. In *Proceedings of the IEEE International Conference on Applied Electronics 2008*, Pilsen, Czech Republic, June, 2008.

A. Strba, T. Krajcovic. Embedded Systems with Limited Power Source. In *Proceedings of the 9th International Conference on Informatics*, Bratislava, Slovakia, 2007.

A. Strba. Design Approach of Embedded Systems Gaining Power from Energy Harvesting. In *Proceedings in Informatics and Information Technologies Student Research Conference* , Bratislava, Slovakia, April 18, 2007.