On the Monitoring of System-Level Energy Consumption of Battery-Powered Embedded Systems

Arliones Hoeller Jr
Department of Automation and Systems Engineering
Federal University of Santa Catarina
Florianópolis, Brazil
arliones@das.ufsc.br

Antônio Augusto Fröhlich
Software/Hardware Integration Lab
Federal University of Santa Catarina
Florianópolis, Brazil
guto@lisha.ufsc.br

***Abstract*—This paper addresses an approach for accurately measuring energy consumption on battery-powered embedded systems which can be adequately tuned in order to enhance a set of timing and energy consumption requirements for mission critical systems. We introduce a software-based accounting scheme which is calibrated by low-precision battery state-of-charge reads through a battery voltage model. We then perform an offline multi-objective optimization procedure using NSGA-II to find good candidates to the period at which battery consumption information should be updated. Such candidates might guarantee timing constraints (i.e., no deadline misses), minimize residual energy after a pre-defined system lifetime, and maximize system utilization during system lifetime. We considered a simple scheduler which will reserve battery charge to run hard real-time tasks during a pre-defined lifetime and will prevent best-effort tasks from running whenever accounted battery state-of-charge is bellow the current reserve. We evaluated our approach by performing a set of case-studies and show that there is no direct relation between battery information update frequency and the composite of objectives defined here.**

***Index Terms*—accounting, energy, embedded systems, real-time scheduling, adaptive task periods**

I. Introduction

Low energy consumption is an important non-functional requirement for the design of battery-powered embedded systems. Reliable information on the system energy source is of paramount importance to design an energy-efficient system. In order to keep track of exact energy consumption at runtime, traditional approaches rely on continuous measurements of the amount of current drained from the battery. Besides the additional hardware required to perform this task, software support for sampling such circuitry may compromise system performance due to the requirement of fine grained information needed to sample this continuous signal.

To cope with this requirement mobile systems provide means to measure battery voltage by which it is possible to infer battery charge through an approximate discharge model for a given battery. This approach, however, brings limitations to the task of estimating battery charge. Among the reported problems [2], [3] there is one of special interest for the task of precisely estimating battery charge on embedded systems: the low accuracy and long response time of voltage-based battery state-of-charge models. This problem is related to the diminished precision of such voltage measurements implied by low resolution analog-to-digital converters and to the oscillation on the battery voltage-levels due to variations on the applied charge. It can be easily illustrated by Figure 1. The figure shows that during most of system lifetime the battery-voltage response time varies greatly. For instance, if an energy-related decision lowers system utilization on day 50, this decision may last until the next expected voltage drop, around day 100, making an unique decision stay in effect for 14% of system lifetime. We can also see that the scheduler misses the opportunity of raising system utilization because its monitor is not able to rapidly inform that the system is not using the expected amount of energy. Instead of gradually raising system utilization and keeping it higher during a longer period of time, it only acts by the end of a given period, causing bursts in system utilization for short periods of times.

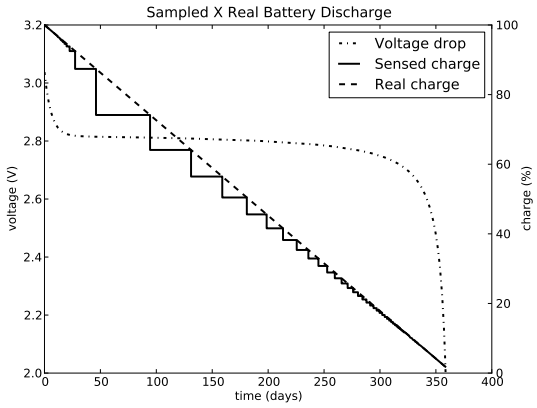


Fig. 1. Sampled discharge plotted against real discharge for a CR2 series Panasonic lithium battery [1].

In this paper we propose a software-based energy consumption accounting scheme. We rely on the fact that energy consumption is a cross-cutting concern, orthogonal to all system components [4], to enable accurate monitoring of energy consumption in a component-based operating system for embedded systems. This scheme monitors the execution

This work have been partially funded by CNPq grant 479009/2009-0. Arliones Hoeller Jr is also supported in part by CAPES grant RH-TVD 006/2008.

of operating system functionalities. In order to make this approach feasible, we derived two different profiles for energy accounting which can be applied to devices with different operational behavior. We evaluate the effect of out approach on system performance.

The remaining of this paper is organized as follows. Section II presents an overview of related work. Section III describes the software-based energy accounting approach proposed in this paper. Section IV describes the implementation of this approach in the EPOS Project. Section V presents a case-study of the proposed approach on a wireless sensor network system. Section VI summarizes this paper and gives some insights on future work.

II. Related work

There are two sorts of third-party developments that are related to the present work: those related to software-based accounting of energy usage and those which address the deployment of multi-objective optimization methods on real-time systems for the control of energy consumption. This section gives an overview of studied related work.

A. Accounting of Battery Usage

We investigated similar battery monitoring schemes in operating systems for wireless sensor networks, including TinyOS [5], MantisOS [6] and Contiki [7]. All three systems provide access to battery voltage measurements through an ADC interface. TinyOS and Contiki, however, have other work either from the original authors or from third-parties that are related to the present work. None of them provide complete voltage models by which it would be possible to infer battery state-of-charge from voltage levels. Also, previous work have already discussed the problems related to the frequency in which operating mode migration happens [8]. These work already address questions concerning time overhead and additional energy consumption during these migrations [9].

Yang et al. [10] built an extension to TinyOS that enables software-based accounting of energy consumption and battery lifetime estimation for the Mica2 sensor note. They monitor the time each hardware component stays in an operating mode by intercepting mode changes and accumulating drawn current during this period, decrementing it from the initial informed battery charge. Their approach, however, requires modifications on every monitored system component and may pose unnecessary overheads in situations where devices change operating mode too often, as it would be the case of a radio transceiver in a low-power listen mode, where the transceiver is periodically switched on and off to check for incoming messages. Weissel and Kellner [11] used an event-based energy accounting mechanism to compute energy on the BTnode platform [12] running TinyOS. They, however, didn't implement the presented concepts, limiting their work to the analysis of the implementation possibilities.

Dunkels et al. [13] implemented a time-based energy accounting system for Contiki running on Tmote Sky [5] platform. Their model, as does the approaches used in TinyOS, demand for modification in several different operating system modules (i.e., drivers), what may make the system difficult to maintain. They also show that the lack of calibration with real information in their system may be the cause of significant errors in energy estimations.

B. Optimization Methods in Energy-Aware Real-Time Systems

Energy optimization for real-time systems has long been a subject of great interest in the real-time community [**?**]. A plurality of works have been published that apply optimization techniques over real-time and system models to find good tradeoffs between energy consumption and operating frequency/voltage of CPUs (DVS - Dynamic Voltage Scaling) [**?**], on/off status of peripheral devices (DPM - Dynamic Power Management) [**?**], or both [**?**]. All these works, although important to the design of energy-aware real-time systems, are outside the scope of this paper for they are orthogonal to the work presented here.

Chantem et al. [14] proposed a generalized elastic scheduling framework for real-time tasks based on Buttazzo's elastic model [15] which may adapt task's elastic periods online based on one specific (generic) performance metric. Optimal period adjustments are then performed by a heuristic proposed by them. Although energy-related metrics may be used as the objective performance metric, authors didn't explore this. Eker et al. [16] and Cervin et al. [17] show the application of optimization theory to solve the period selection problem at runtime by performing adaptive adjustments of periods based on a control performance metric. Bini and Natale [18] devised an optimal search algorithm which minimizes the task frequencies by performing incremental improvements on one specific performance metric by using a branch and bound search over a predefined feasibility region of the domain of task frequencies until the global optimum is reached. The algorithm applies to fixed-priority scheduling schemes and may be only applicable offline due to its high complexity.

To the best of our knowledge, no work explored the effects of the period at which battery state-of-charge information is made available for a real-time energy-aware scheduler. Also, no work was found that applied multi-objective optimization in real-time scheduling taking energy constraints into account.

III. Battery Level Monitoring by Event Accounting

As stated in Section I, traditional voltage-based battery monitoring may lead to an pessimistic bias of an energy-aware task scheduler. In order to enhance the precision of the battery monitor, we propose a scheme to account for energy consumption implemented in software. This scheme is based on the premise that energy is consumed by hardware, not software, but it is the software that controls and monitors hardware activity. Considering that the operating system layer abstracts hardware access to application, it is straightforward to assume that the operating system is the entity with most knowledge about hardware activity, thus being able to monitor system-wide energy consumption.

A. Energy consumption profiles

We analyzed usual hardware behavior and modeled three different ways to account for energy consumption: time-based measurement, event-based measurement, and combined measurement. The time-based approach is used for accounting energy consumption of devices which drain constant current over time when in a specific operating mode. In this scenario, timestamps are stored when an operating mode transition is performed and energy consumption is computed by multiplying the time the device stayed in the previous operating mode by the current drain during that period (Equation 1).

$$E_{tm}(dev) = (t_{end} - t_{begin}) \times I_{dev,mode} \quad (1)$$

For some devices, however, energy consumption is better measured on an event-basis. It is the case of, for instance, sensor sampling. Given a set of operational parameters (e.g., ADC frequency, sensor characteristics) it is possible to determine an energy cost per sample, either analytically or by means of measurement, thus enabling energy accounting to be linked to event monitoring. In this model, system energy consumption is updated for every event. The notation used is in Equation 2.

$$E_{ev}(dev) = \sum_{event\_counters} E * counter \quad (2)$$

Additionally, for some devices, both approaches may be used. For instance, a radio that stays in a low-power listen mode waiting for data to arrive have a base energy consumption which may be computed using the time-based approach. When data actually arrives, we may know the amount of energy that will be consumed to process the reception of each byte. Energy consumption for data reception can then be computed with the event-based approach. The notation used for this is in Equation 3.

$$E_{tot}(dev) = E_{tm}(dev) + E_{ev}(dev) \quad (3)$$

B. Battery state-of-charge monitoring

At runtime, battery charge is updated with the accounted data of each device. These accounting information, however, need to be collected in order to update battery charge. The frequency in which these data are collected directly affects the accuracy of the proposed battery charge monitor. Recalling the curves in Figure 1, we may say that high update frequencies would draw a curve close to the "real charge", while low update frequencies would approximate the "sensed charge" curve. In this section we describe how to achieve a satisfactory curve close to the "real charge" curve of Figure 1 by adequately adjusting the battery update frequency.

We start by analyzing the update frequency for the time-based profile. It is not the intent of the present work to investigate issues related to the frequency of operating mode migrations or their time and energy overheads, for such problems have already been extensively addressed [8], [9]. Thus, this profile can adhere to such migration models by the inclusion of an extra routine like $migration\_update\_energy$ on Figure 2 to compute elapsed time and consumed energy during these migrations as described by Equation 1. The execution time for this routine, shown in Table I is constant and can be easily obtained and integrated to any transition model, being either real-time or not. Additionally, to prevent the system from loosing control of battery discharge when devices stays in a certain operating mode for long periods, an active component periodically collects energy consumption information from all devices and updates battery charge.

For the event-based profile, however, the battery charge update approach needs to be different to avoid unnecessary processing overheads. For instance, suppose that a hypothetic system monitors an event that is the reception of a byte from a network interface. Network protocols will seldom use only one byte to perform communications, thus, as can be seen in Table I, system performance may benefit from periodic updates of an accumulated counter. In order to do that, an

```
Battery:      battery charge
I[modes]:     array of current for each mode
E[events]:    array of energy for each event
C[counters]:  array of integer for each event
time_begin:   timestamp
mode      :   operating mode


procedure energy_migration_update()
    now = SystemTime();
    // Equation 1
    Battery −= (now − time_begin) ∗ I[mode];
    time_begin = now;
end procedure;

procedure energy_event_update(event)
    // Equation 2
    Battery −= E[event] ∗ C[event];
    C[event] = 0;
end procedure;

procedure energy_update_total()
    for each device do
        // Equation 3
        for each event of device do
            energy_event_update(event);
        energy_migration_update();
    end for;

    Battery_Charge = from battery voltage model;
    // Equation 4
    Battery = max(Battery_Voltage, Battery);
end procedure;
```

Fig. 2. Algorithms for energy accounting.

TABLE I
PROCESSING OVERHEAD OF THE ENERGY ACCOUNTER ON AN ARM7-TDMI PROCESSOR.

Procedure	Time overhead
migration_update_energy	73 cycles
event_update_energy (1 repetition)	29 cycles
event_update_energy (n repetitions)	63 cycles

active object was modeled as an extra task on the system which is responsible for collecting accounted information of the event-based profile.

It is important to note that the accounting mechanism employed in this scheme, although more accurate, is still pessimistic once it is based on the worst-case energy consumption (WCEC) of events and components' operating modes. Thus, it is expected that the accounted energy consumption reaches the value read from the battery voltage model short before it shows a drop in voltage. It is safe to assume that the information from the voltage model is a secure bound to battery charge, although conservative. Then, we may correct the battery charge to the maximum value between the accounted charge and the one estimated by the voltage model (Equation 4).

$$E_{batt} = max\left(E_{volt}, E_{batt} - \sum_{i=0}^{\#devs} E_{tot}(i)\right) \quad (4)$$

Finally, the active component with the task of periodically updating the battery charge is responsible for collecting accounted information from both event-based and time-based profiles. The algorithm is the one at the procedure $energy\_update\_total$ of Figure 2. It is important to note that timestamps and event counters are reset every time energy accounting is updated (by $migration\_update\_energy$ and $event\_update\_energy$), thus making sure that no energy consumed is accounted for twice. This algorithmic approach assumes initialization of $Battery$ with the nominal capacity of the battery in use.

C. On the Freshness of Battery Information

To understand the accounter behavior further we performed an exploration of system design space to be able to determine the frequency in which accounted data should be gathered provided that the system has a pre-defined requirement of operation lifetime. We used a multi-objective optimization method based on the *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) [19] which is able to find good solution candidates for the frequency of the collector task, i.e., those closer to the Pareto front. The optimization is executed with two objectives: to minimize residual energy and to maximize the execution rate of best-effort tasks. We use a simple scheduling mechanism in which hard real-time tasks execute regardless of system energy availability and best-effort tasks run only when the system is still able to guarantee energy availability for hard real-time tasks. Priorities in the scheduling queue are assigned through a Rate Monotonic policy provided that all hard real-time tasks have higher priorities than any best-effort task, regardless of their period. We also assume that initial battery charge is enough to guarantee hard real-time tasks' executions during the expected lifetime.

Now we analyze a hypothetic application. Table II shows the parameters for this application, comprised by two hard real-time tasks (H_1 and H_2), one best-effort task (B_1) and the energy accounter collector task (H_C), ordered according to their priority. We kept the collector task as a hard real-time task for two reasons. First, it is the only way to guarantee that the defined period for the collector task will be respected, once best-effort tasks may be prevented from executing. Second, if eventually the system stops the execution of best-effort tasks and the collector task is a best-effort task, the battery information will no longer be updated, thus being this a dead-end for the energy-aware scheduler.

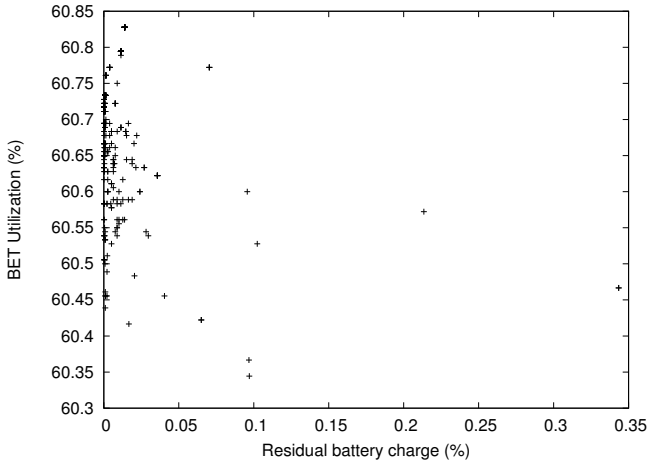


Fig. 3. All solutions for the hypothetical application.

TABLE III
SOLUTIONS FOR THE FREQUENCY OF THE COLLECTOR TASK FOR THE HYPOTHETICAL APPLICATION.

Frequency (Phenotype)	Period (s)	Best-effort rate	Remaining Charge
11.790 Hz	0.085	60.794%	0.114 ‰
11.793 Hz	0.085	60.761%	0.014 ‰
13.089 Hz	0.076	60.728%	0.001 ‰
13.161 Hz	0.076	60.717%	0.001 ‰
14.468 Hz	0.069	60.828%	0.139 ‰
15.812 Hz	0.063	60.772%	0.039 ‰
16.455 Hz	0.061	60.733%	0.007 ‰

In order to evaluate the system instances (individuals) generated during the optimization process we integrated a real-time simulator to the optimizer. This system simulates a rate monotonic queue with two levels of priorities, being the first one the task's class and the second one the rate monotonic priority itself. This made it possible the separation between hard real-time and best-effort tasks. The simulator also monitors energy consumption of tasks and controls battery discharge based on informed worst-case energy consumption (WCEC) of tasks. In order to achieve a more realistic behavior we consider that all tasks actually use their WCEC for 75% of the jobs. The remaining 25% of the jobs have a random energy consumption uniformly distributed between 50% and 100% of the WCEC. This generates a slack on the energy budget that can be used by the best-effort tasks, as would actually happen on real systems. Although naive, this simple probabilistic assumption helps to understand and analyze the approach. More consistent approaches will be considered later in Section III-D and in the case study of Section V.

We ran NSGA-II with a population size of 100 individuals,

TABLE II
HYPOTHETIC APPLICATION TASKS' PARAMETERS.

Task	Period (ms)	WCET (ms)	WCEC (ηAh)	1-hour energy (mAh)
H_1	100	25	1.0	36×10^{-3}
H_2	150	25	1.4	33.6×10^{-3}
H_C	to be defined (worst-case: 50)	0.0453×10^{-3}	41.56×10^{-6}	will depend on the period (worst-case: 2.99×10^{-6})
B_1	200	50	2.0	36×10^{-3}
Energy consumption of mandatory tasks				69.603×10^{-3}

TABLE IV
PRE-CALCULATED FREQUENCIES IN DIFFERENT WCEC SCENARIOS FOR THE HYPOTHETICAL APPLICATION.

WCEC	Frequency (Hz)	Period (s)	BE rate	Residual Charge
1%	152.382	0.007	100.0%	986.83 ‰
10%	165.315	0.006	100.0%	867.98 ‰
20%	148.304	0.007	100.0%	735.96 ‰
30%	118.375	0.008	100.0%	603.94 ‰
40%	117.894	0.008	100.0%	471.93 ‰
50%	132.770	0.007	100.0%	339.91 ‰
60%	156.458	0.006	100.0%	207.89 ‰
70%	122.379	0.006	100.0%	75.87 ‰
75%	161.306	0.006	100.0%	9.86 ‰
80%	5.659	0.177	84.44%	0.083 ‰
85%	5.654	0.177	68.11%	0.126 ‰
90%	6.050	0.165	53.58%	0.00099 ‰
95%	5.654	0.177	40.58%	0.036 ‰
100%	5.654	0.177	28.89%	0.104 ‰

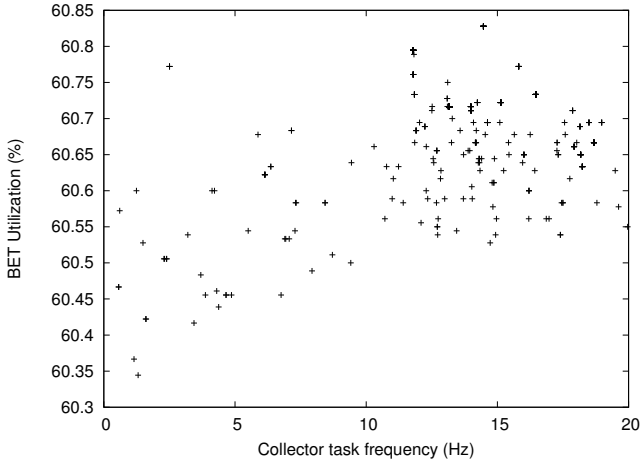
being 20 of them selected as parents, generating 20 offsprings, repeating the process during 50 generations (iterations) ($\alpha = 100, \mu = 20, \lambda = 20, g = 50$). Figure 3 shows all simulated solutions. The best solutions, i.e., those at the Pareto front, are shown in Table III. It is possible to observe here the wide spread of results obtained from the optimization process. Moreover, Figures III-C and III-C show the non-linear behavior of the observed parameters (respectively, best-effort tasks' execution rate and residual battery) in relation to frequency, showing why the solution to this problem benefits from the application of meta-heuristic methods like NSGA-II.

D. On the Impact of Actual Energy Consumption

In these experiments we observed that the optimization results, as expected, are directly dependent on the actual energy consumption of tasks. In Section III-C, with the goal of illustrating the concepts, we considered that tasks only use their WCEC for 75% of their jobs, while the remaining 25% of the jobs have its energy consumption uniformly distributed between 50% and 100% of the WCEC. In a real system, however, it is too difficult, if not impossible, to derive such assumptions. In order to cope with that one needs to adapt the collector task's frequency to the actual execution rate of tasks during runtime.

Instead making such statistical assumptions during simulations, we fixed the energy consumption to several different values, ranging from 1.00% to 100% of the WCEC to observe the results. This led to the configurations shown at Table IV. As can be seen, the frequency of the collector task varies

Execution rate of best-effort tasks.



Residual energy after projected lifetime.

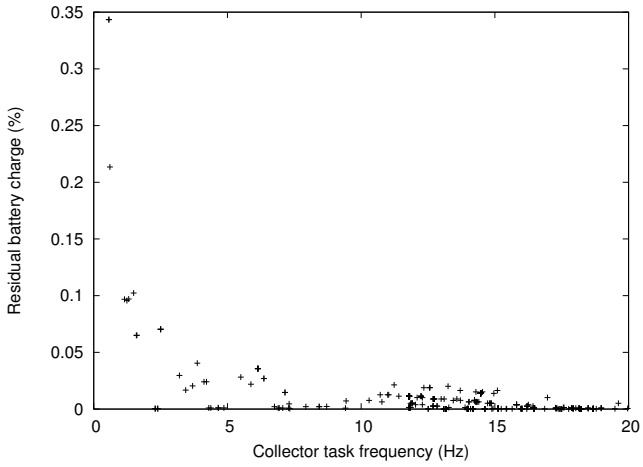


Fig. 4. Optimization objectives plotted against variations on the frequency of the collector task for the hypothetical application.

greatly when WCEC goes above circa 75.76%[1]. That means that target battery lifetime will only be at risk if system is consuming more than this limit. With this information in hand, we adapted the simulator to be able to change frequency of the collector task frequency according to the actual system

[1]This limit is determined by $K_{WCEC} = \frac{B_0}{T_t * E_1}$, where B_0 is the initial battery charge, T_t is the target system lifetime (in seconds), and E_1 is the energy consumed by all system tasks (including best-effort) during one second. B_0 and E_1 need to be at the same unit (e.g., mAh).

energy consumption based on the information at Table IV. For this setup, we set a transition point when energy consumption crosses the 75% limit. If consumption is low, the period of the collector task is set to 7 ms, if consumption is high, the period changes to 177 ms.

Simulation showed that the proposed heuristic resulted in a best-effort task execution rate of 60.66% with a residual energy of 0.2781 $‰$ with the same probabilistic assumptions made on the actual energy consumption of tasks. If compared with the solutions found by the experiment of Section III-C, the heuristic was able to sustain a similar best-effort execution rate while presenting more residual energy, as expected.

If we change the probabilistic assumptions, however, we see that the heuristic is still able to get results close to the previous setup. We first relaxed the task's energy consumption so that only 25.0% of the jobs consume the whole WCEC. This resulted in a best-effort execution rate of 80.08% and a residual energy of 0.2674 $‰$. When we restricted the taks's energy consumption further, to 90.0%, the results were 34.64% for the best-effort task execution rate and 0.2770 $‰$ for the residual energy. Running the optimization process using these variations also resulted in similar values: 80.36% and 0.103 $‰$ for 25%; and 34.74 and 1.826 $‰$ for 90%. This shows that the heuristic approach is preferred to the use of a unique collector task frequency.

IV. Implementation for the EPOS Project

The EPOS Project (Embedded Parallel Operating System) aims at automating the development of embedded systems so that developers can concentrate on the applications. EPOS relies on the Application-Driven Embedded System Design (ADESD) [20] method to guide the development of both software and hardware components that can be automatically adapted to fulfill the requirements of particular applications. EPOS features a set of tools to support developers in selecting, configuring, and plugging components into its application-specific framework. The combination of methodology, components, frameworks, and tools enable the automatic generation of application-specific embedded system instances [21].

A. The EPOSMote

Besides run time support system and tools, the EPOS Project has driven the development of hardware platforms, being the EPOSMOTE among them. EPOSMOTE is a modular platform for wireless sensor network applications. It has three different modules as shown in Figure 5. The *Processing Module* incorporates the core processing and communication components of the system. There are two different versions of this module: one based on Atmel's ZigBit package and another, used in this work, based on Freescales' System-on-Chip MC13224V. Both present RF transceivers compatible with IEEE 802.15.4 standard. Power supply and I/O interfaces were factored out on EPOSMOTE's design to allow adaptation to specific applications. The power interface has separated signals for the power source (V_{cc}, V_{dd} and Gnd) and an I^2C interface for communication with the processing module. The I/O interface

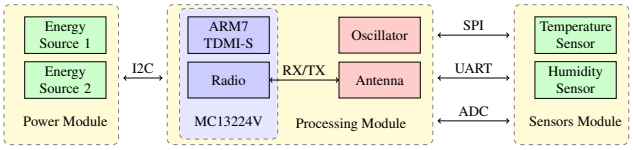


Fig. 5. EPOSMote's block diagram.

TABLE V
CURRENT DRAWN BY EPOSMOTE'S COMPONENTS IN DIFFERENT OPERATING MODES FOR TIME-BASED ACCOUNTING.

Device/Mode	Current (μA)
CPU	
Active	$2,200$
Sleep	40
Power Down	3.4
Radio	
Tx only	$19,300$
Rx only	16
Low Power Listen	0.8

make 32 pins available for custom designs including a bypass of the power source, all ADC channels, SPI, $UART$ and several $GPIO$ pins. The EPOSMOTE Project developed a *Start-Up* board to be connected to the I/O interface featuring USB converter, a thermistor, a 3-axis accelerometer, LEDs, and buttons that was also used in this work.

To be able to account for energy consumption on the EPOSMOTE we first need to analyze its power characteristics and build its energy model. These information were collected from the devices' datasheets, when available, or measured using an oscilloscope in current mode. Table V shows values of current drains of system devices in different operating modes to be used by time-based accounters, while Table VI show drained battery charge for monitored events to be used by event-based accounters.

B. The EPOS Power Manager

Once power management is a non-functional property of computing systems [4] the EPOS' power manager was modeled as an software aspect [22], thus being its implementation orthogonal to the implementation of other components in EPOS' framework. In EPOS, aspects are implemented as constructs called *Scenario Adapters* [23], which relies on C++'s static metaprogramming capability (templates) and doesn't imply in the use of extra tools such as aspect weavers.

Figure 6 shows a class diagram for the EPOS' power manager modeled as a scenario adapter.

The base `Power_Manager` wraps the target class (to which the aspect may be applied) by inheritance and function overriding (the *Adapter Design Pattern*). Additional methods

TABLE VI
BATTERY CHARGE USED BY EPOSMOTE'S COMPONENTS IN MONITORED EVENTS FOR EVENT-BASED ACCOUNTING.

Event	Drawn charge (nAh)
Read temperature sensor	10.584
Sample battery voltage	0.462

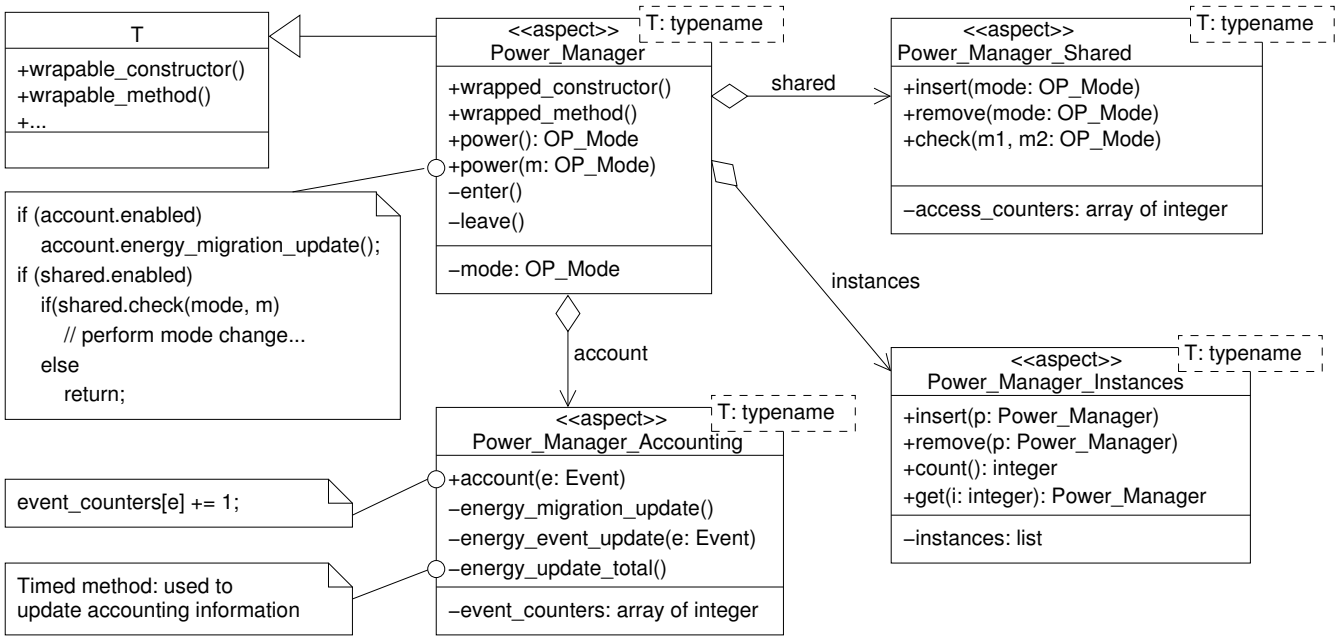


Fig. 6. UML class diagram of EPOS' power manager.

may them be easily included, as is the case of the `power` methods in the base `Power_Manager` class. `Power_Manager` is, in turn, a facade (the *Facade Design Pattern*) to other functionalities related to power management being implemented by other components. For instance, `Power_Manager_Shared` and `Power_Manager_Instances` are responsible for, respectively, controlling of operating modes for shared components, and keeping of object references for system-wide power management actions [8].

In this work, EPOS' power manager was extended to include the energy consumption accounting functionality. This was done by aggregating the `Power_Manager_Accounter` class which, in turn, implements the energy consumption profiles described at Section III-A with the algorithms listed in Figure 2. Accounting of events on the event-based profile are performed through the `account(e:Event)` method, which may be called in a `wrapped_method()` if the event generated by such a method is a monitored one. As concerning overhead issues, it is important to note that the `account(e:Event)` method, which increments an event counter, is an inline function, thus incurring in no overheads related to function calls at runtime. Also, the branches that implement the facade at the `power(m:OP_Mode)` method of `Power_Manager` use constant boolean values which, in EPOS, are defined at configuration time, before compilation. As such, these are subject to compiler optimizations which are able to remove these branches in the final system binary. Table VII presents the impact of the proposed accounter on EPOS in terms of code and data memory usage, showing that the accounting mechanism aggregates 2,768 bytes of code (ROM) and 70 bytes of data (RAM) over the original fully functional `Power_Manager`.

TABLE VII
MEMORY FOOTPRINT OVERHEAD OF THE EPOS POWER MANAGER.(?COMO TIRA A HLINE DE BAIXO DO SETUP?)

Setup	**Sizes in bytes**			
	.text	.data	.bss	**Total**
Original	29,146	396	325	**29,867**
with accounting	31,914	454	337	**32,705**
Increase	9.5%	9.71%		**9.5%**

V. CASE STUDY

In this section we present the deployment of the approach described in this paper in a mobility-enabled wireless sensor network running the Ant-based Dynamic Hop Optimization Protocol (ADHOP) over an IP network using IEEE 802.15.4. ADHOP is a self-configuring, reactive routing protocol inspired by the HOPNET protocol for *Mobile Ad Hoc Networks* (MANETs) and designed with the typical limitations of sensor nodes in mind, energy in particular [24]. ADHOP's reactive component relies on an *Ant Colony Optimization* algorithm to discover and maintain routes. Ants are sent out to track routes, leaving a trail of pheromone on their way back. Routes with a higher pheromone deposit are preferred for data exchange.

With the purpose of corroborating the approach presented in this paper we made a few modifications to the ADHOP in order to make it energy-aware. In order to do that, we separated ADHOP tasks between mandatory (hard real-time) tasks and optional (best-effort) tasks. The main idea behind this setup was to homogenize the battery discharge for every node in the network to enhance the lifetime of the network as a whole. Considering the radio the most energy-hungry component in a wireless sensing node, we toke the design decision of modeling

TABLE VIII
ADHOP CASE-STUDY TASKS' PARAMETERS.

Task	Period (ms)	WCET (ms)	WCEC (ηAh)	25-days energy (mAh)
$Sense$	$1,000$	2	10.584	441.50
$Forward$	$1,000$	50	268.056	378.43
$Collector$	to be defined (worst-case: 5)	0.0453×10^{-3}	41.56×10^{-6}	will depend on the period (worst-case: 17.954×10^{-3})
$LowPowerListen$	5	0.25	1.111×10^{-6}	0.48×10^{-3}
$Route$	$Sporadic$ (Pretending 2 Hz)	100	490.278	will depend o the activation rate (would be: $2,118$)
Energy consumption of mandatory tasks				601.88

TABLE IX
SOLUTIONS FOR THE FREQUENCY OF THE COLLECTOR TASK FOR THE ADHOP CASE-STUDY.

Frequency (Phenotype)	Period (s)	Best-effort rate	Remaining Charge
23.944 Hz	0.042	9.92%	$0.008\,{}^{0}/_{00}$
41.243 Hz	0.024	9.93%	$0.130\,{}^{0}/_{00}$
41.849 Hz	0.024	9.93%	$0.013\,{}^{0}/_{00}$

the *ants* of ADHOP as best-effort tasks, as shown by the task set at Table VIII. By doing this, the basic node functionality of sensing a value (task $Sense$) and forwarding it through the radio to a sink-node (task $Forward$) where modeled as hard real-time tasks, and the functionality of forwarding other nodes' packets (and ants) when acting as a "router" was modeled as two best-effort tasks, one for monitoring the channel for arriving messages ($LowPowerListen$), and another to effectively receive the message and route it to another node ($Route$).

We set the lifetime objective for this system to 25 days. By analyzing the task set it is possible to compute the total energy consumption of hard real-time tasks for the desired lifetime to be of $601.88mAh$, thus, the initial battery charge for the system has to be greater than that. We analyzed this system using a small Panasonic CR-2 $3V$ battery with a total capacity of $850mAh$.

In order to enhance the significance of our results, we simulated larger networks (with up to 200 nodes) using the *Global Mobile Information System Simulator* (GLOMOSIM). In this setup, nodes were programmed to communicate intensively and move randomly within a simulated grid of 700 x 400 meters for 25 days, thus stimulating both the routing protocol and the power management mechanisms. GLOMOSIM was integrated to the same NSGA-II optimizer described in Section III-C, and the optimization process ran with the same parameters ($\alpha = 100, \mu = 20, \lambda = 20, g = 50$). The results of this optimization are shown in Figure 7 (simulated solutions), Table IX (best solutions, i.e., those at the Pareto front), and Figures V and V (observed parameters plotted against the collector task's frequency).

Additionally, we analyze the impact on the network performance by comparing the obtained results with the data originally published by Okazaki [24]. Figure V shows a reduction on the average energy consumed by each node on the network while Figure V shows the expected enhancement

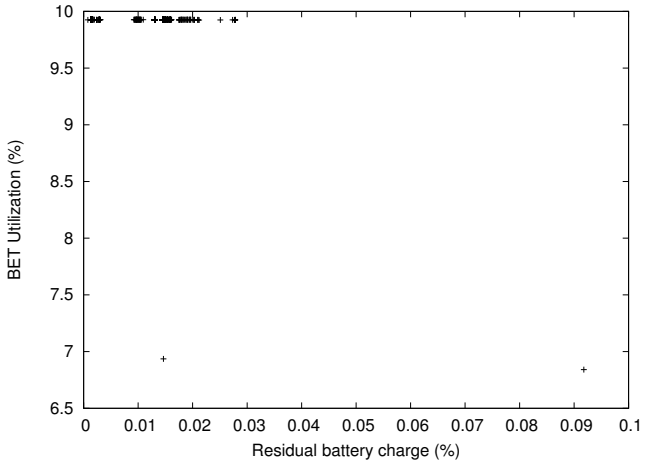


Fig. 7. All solutions for the ADHOP case-study.

on the average battery lifetime of nodes. It is important to note that all nodes in the network lived for, at least, 25 days as expected, being that the reason why the average lifetime stayed around 30 days or above.

Besides the good results from the energy consumption perspective, we observed an important decrease on the overall network quality, as shown in Figures V and V for, respectively, the "Broken routes" and "Delivery ratio" parameters. These contrast, however, with the obtained results on "Link failures" shown by Figure V, which shows that ADHOP deals well with the broken routes, allowing undelivered packets to be re-routed and finally delivered. Future work on energy-aware scheduling, which were not focus of the present work, will rely on the presently proposed accounting mechanism to enable fairer scheduling of such tasks. Initial studies have already began on flexible schedulers such as Ramanathan's (m,k)-firm scheduler [25] and Buttazzo's elastic model [15].

VI. CONCLUSION

In this paper we presented a software implementation of an energy consumption accounter for battery-operated embedded systems. We modeled the accounter and implemented it in a simulation environment and in a real platform [21]. In order to lower the processing overhead imposed by our approach we extracted runtime parameters of energy consumption and execution time of a given application and submitted it to a genetic optimizer (NSGA-II) to look for good solutions for the

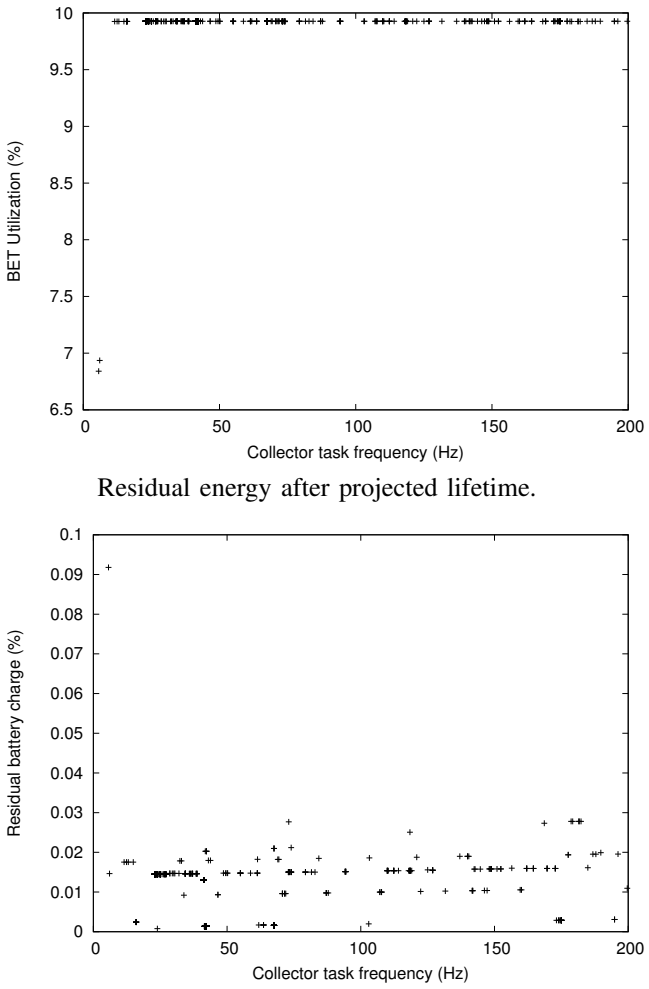


Fig. 8. Optimization objectives plotted against variations on the frequency of the collector task for the ADHOP case-study.

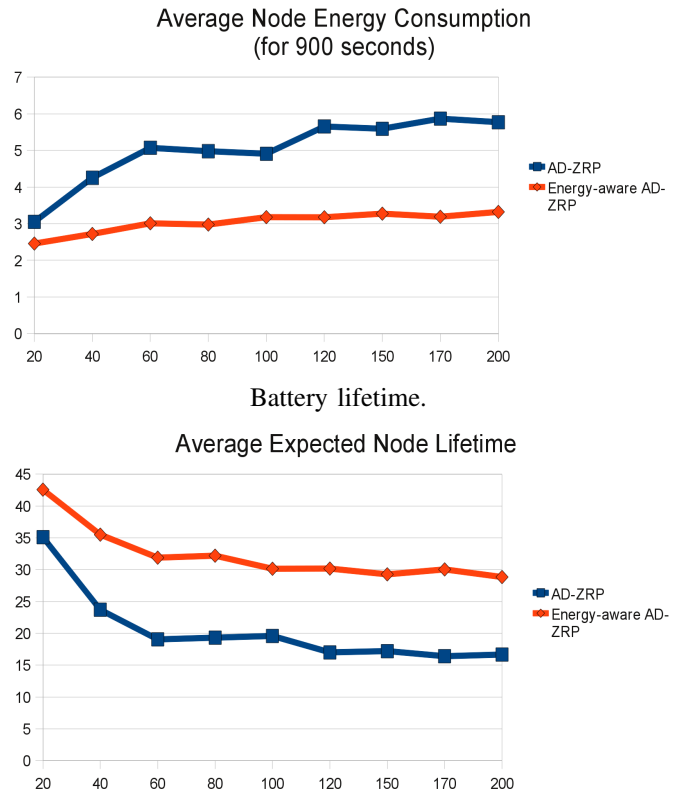


Fig. 9. Average energy-related parameters for the simulated ADHOP setup.

period at which battery-related information should be updated in order to maximize system utilization while minimizing residual energy and guaranteeing a pre-defined system lifetime (mission duration). A case study on an IP-based network running over IEEE 802.15.4 sensing nodes showed promising results of the application of the proposed approach to real applications.

On going studies are focusing on deeper exploration of the proposed adjustable energy accounter by deploying fairer scheduling mechanisms to reduce the impact on system quality. This effort aims at (1) enhancing system quality by using flexible task scheduling schemes, such as *(m,k)-firm* [25] or the elastic model [15], to be put in place of the current egoist approach of preventing tasks' execution; and (2) alleviating the impact on the network quality by using network-wide battery charge information as a parameter for the pheromone generation function of ADHOP.

ACKNOWLEDGMENT

Authors would like to thank the support given by Alexandre Massayuki Okazaki and Rafael Luiz Cancian on the integration of the work herein with, respectively, the ADHOP routing protocol and the ADESD's optimization tool.

REFERENCES

[1] PanasonicCorporation, *Manganese dioxide lithium batteries(CR series) datasheet*, Japan, 2006.

[2] T. Mundra and A. Kumar, "Micro power battery state-of-charge monitor," *Consumer Electronics, IEEE Transactions on*, vol. 54, no. 2, pp. 623 – 630, May 2008.

[3] M. Penella and M. Gasulla, "Runtime extension of low-power wireless sensor nodes using hybrid-storage units," *Instrumentation and Measurement, IEEE Transactions on*, vol. 59, no. 4, pp. 857 –865, Apr. 2010.

[4] D. Lohmann, W. Schroder-Preikschat, and O. Spinczyk, "Functional and non-functional properties in a family of embedded operating systems," in *Object-Oriented Real-Time Dependable Systems, 2005. WORDS 2005. 10th IEEE International Workshop on*, feb 2005, pp. 413 – 420.

[5] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in *The Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, Los Angeles, California, Apr. 2005.

[6] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han, "Mantis os: an embedded multithreaded operating system for wireless micro sensor platforms," *Mob. Netw. Appl.*, vol. 10, pp. 563–579, August 2005. [Online]. Available: http://dx.doi.org/10.1145/1160162.1160178

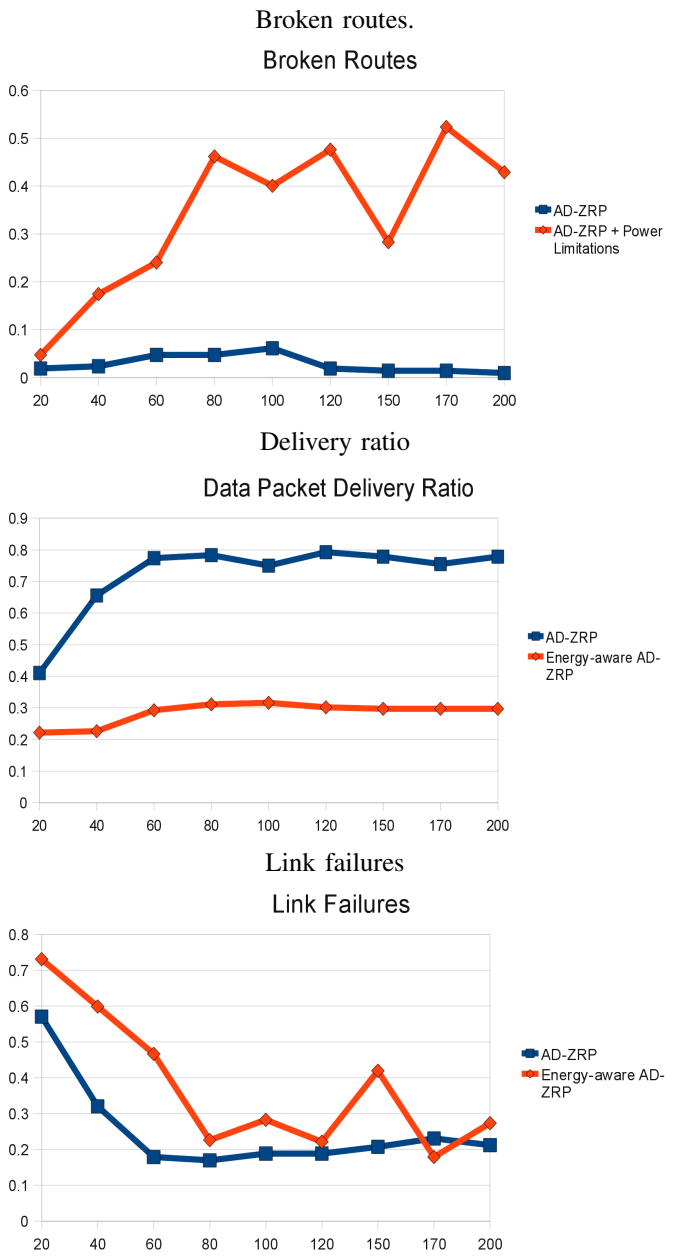


Fig. 10. Network quality impact for the ADHOPcase-study.

[7] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, Tampa, Florida, USA, Nov. 2004. [Online]. Available: http://www.sics.se/~adam/dunkels04contiki.pdf

[8] A. J. Hoeller, L. F. Wanner, and A. A. Fröhlich, "A Hierarchical Approach For Power Management on Mobile Embedded Systems," in *5th IFIP Working Conference on Distributed and Parallel Embedded Systems*, Braga, Portugal, Oct. 2006, pp. 265–274. [Online]. Available: http://www.lisha.ufsc.br/pub/dipes2006.pdf

[9] E. Seo, S. Kim, S. Park, and J. Lee, "Dynamic alteration schemes of real-time schedules for i/o device energy efficiency," *ACM Trans. on Embedded Computing Systems*, vol. 10, pp. 23:1–23:32, January 2011. [Online]. Available: http://doi.acm.org/10.1145/1880050.1880059

[10] T. Yang, Y. K. Toh, and L. Xie, "Run-time monitoring of energy consumption in wireless sensor networks," in *Control and Automation, 2007. ICCA 2007. IEEE International Conference on*, Jun. 2007, pp. 1360 –1365.

[11] A. Weissel and S. Kellner, "Energy-aware reconfiguration of sensor nodes," in *Proceedings of the First GI/ITG Workshop on Non-Functional Properties of Embedded Systems*, ser. NFPES'2006, Nuremberg, Germany, mar 2006, pp. 69–75. [Online]. Available: http://www4.informatik.uni-erlangen.de/~weissel/Publications/Papers/NFPES06.pdf

[12] J. Beutel, O. Kasten, F. Mattern, K. Römer, F. Siegemund, and L. Thiele, "Prototyping wireless sensor network applications with btnodes," in *Wireless Sensor Networks*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004, vol. 2920, pp. 323–338.

[13] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He, "Software-based on-line energy estimation for sensor nodes," in *Proceedings of the Fourth Workshop on Embedded Networked Sensors (Emnets IV)*, Cork, Ireland, jun 2007. [Online]. Available: http://www.sics.se/~adam/dunkels07softwarebased.pdf

[14] T. Chantem, X. S. Hu, and M. Lemmon, "Generalized elastic scheduling for real-time tasks," *Computers, IEEE Transactions on*, vol. 58, no. 4, pp. 480 –495, apr 2009.

[15] G. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, dec 1998, pp. 286 –295.

[16] J. Eker, P. Hagander, and K.-E. Årzén, "A feedback scheduler for real-time controller tasks," *Control Engineering Practice*, vol. 8, no. 12, pp. 1369 – 1378, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/B6V2H-41KP512-4/2/94350c118741c6eee56838fab60681ca

[17] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén, "Feedback–feedforward scheduling of control tasks," *Real-Time Systems*, vol. 23, pp. 25–53, 2002, 10.1023/A:1015394302429. [Online]. Available: http://dx.doi.org/10.1023/A:1015394302429

[18] E. Bini and M. Di Natale, "Optimal task rate selection in fixed priority systems," in *Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International*, dec 2005, pp. 11 pp. –409.

[19] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182 –197, Apr. 2002.

[20] A. A. Fröhlich, *Application-Oriented Operating Systems*. Sankt Augustin: GMD - Forschungszentrum Informationstechnik, 2001. [Online]. Available: http://www.lisha.ufsc.br/pub/aoos.pdf

[21] LISHA, "Epos project website," Internet, feb 2011. [Online]. Available: http://epos.lisha.ufsc.br

[22] K. Mens, C. V. Lopes, B. Tekinerdogan, and G. Kiczales, "Aspect-oriented programming workshop report," in *Proceedings of the Workshops on Object-Oriented Technology*, ser. ECOOP '97. London, UK: Springer-Verlag, 1998, pp. 483–496. [Online]. Available: http://portal.acm.org/citation.cfm?id=646777.706037

[23] A. A. Fröhlich and W. Schröder-Preikschat, "Scenario Adapters: Efficiently Adapting Components," in *4th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, USA, Jul. 2000. [Online]. Available: http://www.lisha.ufsc.br/pub/sci2000.pdf

[24] A. M. Okazaki and A. A. Fröhlich, "Ad-zrp: And-based routing algorithm for dynamic wireless sensor networks," in *Proceedings of the 18th International Conference on Telecommunications*, ser. ICT'2011, IEEE Communications Society. Ayia Napa, Cyprus: IEEE, may 2011, to appear.

[25] P. Ramanathan, "Overload management in real-time control applications using (m, k)-firm guarantee," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 10, no. 6, pp. 549 –559, Jun. 1999.