Ultra-Low-Power Sensor Nodes Featuring a Virtual Runtime Environment

Emanuele Lattanzi and Alessandro Bogliolo
DiSBeF - University of Urbino - Italy
Email: {emanuele.lattanzi, alessandro.bogliolo}@uniurb.it

Abstract—**The widespread diffusion of wireless sensor networks and wearable devices, together with the emergence of energy harvesting techniques, has motivated the development of ultra-low-power micro controller units (MCUs) which are highly energy efficient in active mode and provide a wide range of sleep states to be possibly exploited to save power during idle periods. In spite of their energy efficiency, state-of-the-art MCUs exhibit 16-bit RISC architectures clocked at up to tens of MHz and equipped with at least 16kbytes of main memory and 64kbytes of flash. This makes them suitable to run a virtual machine, bringing the benefits of a virtual runtime environment to power-constrained embedded systems. Virtual machines, however, impair the effectiveness of dynamic power management since they are seen as always-active processes by the scheduler of the operating system. This paper presents a power-manageable open-source embedded virtual environment based on a modified version of the Darjeeling VM running on top of Contiki OS. The proposed architecture has been implemented and tested on Texas Instruments' MSP430 MCUs which have been used as a testbed for the characterization of the power consumption and transitions cost of each power state.**

I. Introduction

Countless techniques have been developed in the last two decades to improve the energy efficiency of digital systems at any design stage. Beyond the design of low-power hardware components, one of the main challenges has become the run-time adaptation of the power consumption of a system to its actual workload, achieved by turning on and off the components and by changing their operation mode. The need for adapting to workload variations has induced a suitable partitioning of design concerns: on one hand, designing *power manageable* components featuring multiple operating modes with different power-performance trade-off, on the other hand, developing *dynamic power managament* (DPM) strategies capable of choosing at run time the best operating mode of each system component in order to minimize the overall energy consumption while meeting performance requirements.

The emergence of energy harvesting techniques has recently added to the complexity of the problem by opening the challenging perspective of energetically self-sufficient digital systems which take power from time-varying environmental sources. This perspective is particularly attractive in the domains of wireless sensor networks (WSN) [2] and body area networks (BANs) [1], where network nodes are usually subject to size, weight, cost, and lifetime requirements which are hardly met by traditional batteries.

A significant step towards the development of autonomous WSNs is represented by off-the-shelf ultra-low-power *microcontroller units* (MCUs) featuring energy efficient active modes and a variety of low-power inactive modes to be possibly exploited during idle periods. State-of-the-art MCUs exhibit 16-bit RISC architectures clocked at 16 MHz and equipped with at least 16 kbytes of main memory and 64kbytes of flash. These features are compatible not only with the execution of a tiny operating system (OS) [4], but even with a virtual machine (VM) running on top of it [5].

The possibility of executing a lightweight VM on an ultra-low-power MCU grants to WSNs and BANs the key benefits of virtualization, including abstraction, portability, over-the-air programmability, and bytecode efficiency. Although such benefits could facilitate the implementation of high-level DPM strategies, virtualization risks to impair the exploitability of the underlying power manageable MCU. This is due both to the limited control of the hardware platform granted to the virtual runtime environment, and to the limited visibility of the idleness of the virtual threads granted to the OS.

This paper addresses the above-mentioned issues and proposes an open-source software stack which creates a Java runtime environment on top of an ultra-low-power MCU without loosing control of its power states. The proposed software stack, based on modified versions of Contiki OS [7] and Darjeeling VM [8], was implemented and tested on a Texas Instruments' MSP430 MCU. Preliminary experimental results show that signficant power savings (reducing the average consumption of up to 3 orders of magnitude in representative scenarios) can be achieved by taking proper DPM decisions directly from the Java runtime environment.

II. Reference architecture

This section introduces the three components of the reference architecture (namely, a power manageable MCU, the Contiki OS, and the Darjeeling VM) and provides a suitable power-state model to be used in the rest of the paper.

A. Power manageable MCU

Ultra-low-power MCUs exploit idleness to switch off power-consuming components in order to save energy. Although different MCUs can differ in the number and in the names of the low-power states they provide, for our purposes we consider a generic MCU to be represented by a power state machine with four catagories of power states, called *Active*, *Standby*, *Sleep*, and *Hibernation*, characterized by the components which are turned off and by the consequent trade-off between power consumption and wake-up time.

In Active mode the CPU is running and the unit is able to execute tasks without incurring any delay. In Standby mode the CPU is not powered, but the clock system is running and the unit is able to self wake-up by means of timer interrupts. In Sleep mode both the CPU and the clock system are turned off and the unit wakes up only upon external interrupts. In Hibernation even the memory system is turned off, so that there is no data retention. Wake-up can be triggered only by external interrupts and it entails a complete reboot of the CPU.

B. Contiki OS

Contiki is an open source real-time OS specifically designed for sensor networks and networked embedded systems [7]. The key features of Contiki OS are: portability, multi-tasking, memory efficiency, and event-driven organization. Each process in Contiki can schedule its own wake-up and go to sleep without loosing the capability of reacting to external events.

The basic power management mechanism in Contiki exploits the Standby state of the MCU by setting a timer interrupt (namely, `clock`) which periodically turns on the CPU in order to check for elapsed wake-up times. The period of the timer interrupt is a constant (namely, `INTERVAL`) defined at compile time and intialized once and for all during the boot (for Texas Instruments' MSP430 MCUs the period is set at 10ms). The `INTERVAL` determines the time resolution of the events managed by the OS. Although the CPU can react to asynchronous external interrupts, the OS reacts as if they were aligned with the last timer interrupt.

The inherent event-driven structure of Contiki provides a mean for minimizing the energy overhead caused by periodic wake-up. This is done by making the interrupt handler aware of the next time at which a process has to be resumed in order to go back to sleep without invoking the scheduler in case of premature wake-up. The only exception is represented by processes waiting for external events, which are resumed whenever an interrupt arrives, including timer interrupts.

C. Darjeeling VM

Darjeeling is a VM designed for extremely limited devices, specifically targeting wireless sensor networks [8]. Its main advantage stems from the capability of supporting a substantial subset of the Java libraries while running on 8-bit and 16-bit MCUs with at least 10kbytes of RAM. The size of the bytecode is significanty reduced by means of an off-line tool, called *infuser*, which transforms the Java bytecode into a custom bytecode and performs static linking of groups of classes. It is also worth mentioning that the Darjeeling VM provides a garbage collector and supports multi threading.

The VM executes on Contiki as a process which runs together with the OS proto-threads implementing the *NETSTACK* communication protocol. The VM scheduler implements a round-robin policy in which each thread is allowed to execute for up to a fixed number of bytecodes before releasing the CPU. Whenever a thread is suspended, the VM waits for the next timer interrupt (possibly yielding resources) before resuming the execution of the next running thread.

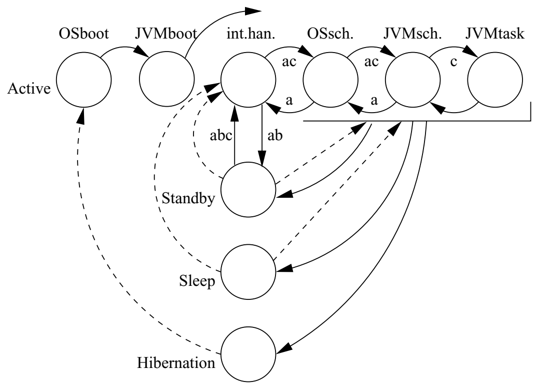

Fig. 1. Power state diagram of a generic power manageable MCU running the Darjeeling VM on top of Contiki OS.

D. Power state diagram

Figure 1 shows the power-state diagram of a system running Contiky OS and Darjeeling VM on top of a MCU which provides the four power states introduced in Section II-A. The Active mode is split into several states to represent the macro-steps required at wake-up to resume the execution of a task on the runtime virtual environment. Dashed arcs represent external interrupts, while solid arcs represent self-events. According to the definition of the low-power states, self wake-up is enabled only from Standby, while external interrupts are required to wake up from Sleep and Hibernation. It is also worth noticing that a complete boot is required when exiting from Hibernation, while data retention allows execution to resume directly when exiting from Standby and Sleep modes. Transitions represented on the right-hand side of the graph denote the possibility of entering low-power states directly from a code segment and resuming execution from the same point at wake-up at any level of the software stack.

According to the behaviour described in Section II-B, Contiki exploits the Standby state whenever all its running processes are waiting for scheduled timers or external events, but in order to keep control of the elapsed time it sets a periodic timer interrupt which wakes up the CPU every 10ms. Upon wake-up the interrupt handler evaluates if there are running processes which need to resume execution. If this is the case the control is passed to the scheduler, otherwise the CPU is turned off again soon. As mentioned in Section II-C, the Darjeeling VM running on Contiki is a process which needs to resume at each timer interrupt in order to check for the status of its threads. If there are no threads ready to resume, the process is suspended until next timer interrupt.

Labels a, b, and c in Figure 1 denote the transitions taken upon a timer interrupt in case of: a) VM with no tasks to resume, b) system with no processes to resume, and c) virtual task to be resumed. Case c) is the only one which makes the CPU worth to be woken up, while cases a) and b) are nothing but an overhead to be periodically paid while in Standby.

Figure 2.a provides a simplified version of the power state diagram where a) and b) are represented as self-loops of the Standby state, while the overhead of the boot is implicitly associated with the wake-up transition from Hibernation,

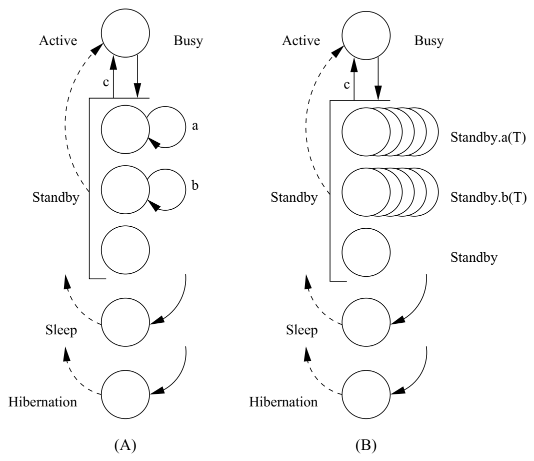

Fig. 2. Abstract representations of the state diagram of Figure 1 obtained by: a) implicitly representing transient states as arcs with non-null transition time and energy b), removing periodic self-loops and accounting for the corresponding cost into the power consumption of the corresponding states.

rather than being explicitly represented by the transient states of Figure 1. On the other hand, Standby mode is split into three different states in Figure 2.a to stress the difference between pure Standby without periodic wake-up, and intermittent Standby with type a) or type b) timer interrupts.

A further abstraction is provided in Figure 2.b, where self loops a) and b) have disappeared and their power overhead has been directly accounted for in the average power consumption of the corresponding Standby states. Since such an overhead depends on the period of the timer interrupt (denoted by T), power states Standby.a and Standby.b represent families of power states the average power consumption of which depends on the value of T. Because of the default settings of Contiki for the target MCU and of the interaction with the Darjeeling VM, the only low power state which is actually exploited is Standby.a with $T = 10ms$.

III. Proposed architecture

This section outlines the changes made to the reference architecture in order to make it able to fully exploit the power states of Figure 2.b.

A. Avoiding periodic wake-up

Period wake-up from Standby is used in the reference architecture to maintain time awarness. The `INTERVAL` between periodic timer interrupts is also the time resolution in Contiki.

In principle, periodic wake-up could be avoided in a power-managed system as long as an oracle exists to wake up the system right in time to execute useful tasks. The scheduler of the OS has the capability of acting as an omniscient oracle for the self-events scheduled by its processes. Similarly, the scheduler of the VM can act as an oracle for the self-events scheduled by its threads. In the reference architecture, however, neither the OS nor the VM expoits these prediction capabilities, ultimately impairing the energy efficiency of Standby mode. Two changes were made to overcome this limitation.

First, the scheduler of the Darjeeling VM was modified in order to take the time of the next scheduled task (as returned by function `dj_vm_schedule()`) and to use it to set an OS timer and suspend the entire process (by means of `PROCESS_WAIT_EVENT_UNTIL()`). This makes the OS aware of the idleness of the VM, so that premature periodic wake-up can be effectively filtered out by the interrupt handler. Referring to the power state diagram of Figure 2.b, this enables the exploitation of state Standby.b in place of Standby.a.

A second change was implemented in Contiki to make it able to dynamically adjust the `INTERVAL` of the timer interrupt. An integer slow-down coefficient was used to this purpose in order to maintain compatibility with the 10ms time resolution of the OS. The interrupt handler was modified accordingly by applying the same coefficient to the timer-interrupt counter to be compared with the time of the next scheduled process. Referring to the diagram of Figure 2.b, the modified version of Contiki provides control of parameter T of Standby states a) and b). Moreover, it makes it possible for the OS to set the timer interrupt before entering the Standby mode in order to wake up the processor right in time to execute the next scheduled event, thus avoiding any useless periodic wake-up (state Standby in Figure 2.b).

The only drawback of slowing down the timer interrupt is the loss of accuracy in the perceived arrival time of external interrupts. In fact, although the MCU is able to react to asynchronous external interrupts regardless of the timer when in Standby mode, timer interrupts are required to update the system clock. This problem will be addressed in Section III-C.

B. Hibernation

The main problem with Hibernation is the lack of data retention, which requires the heap of the VM to be saved in flash and restored at wake-up together with a few external variables, including the base address of the heap. In order to allow hibernation to be possibly triggered by a high-level task running on a thread, the context of the running thread has to be saved in the heap without waiting for a context switch. Moreover, a flag has to be added to the `thread` data structure to recognize the thread which triggered hibernation, while control bytes have to be stored in flash to make sure that the heap is fresh. The control bytes are then used by the `load_machine()` function to decide whether to resume execution from the point of hibernation or to restart the VM process from the `main`. Hibernation process is implemented as a native method, while wake-up from hibernation is directly implemented in the `main` of the VM.

It is worth noticing that the OS is not hibernated, so that it is rebooted at wake-up and the clock needs to be restored in order to make it coherent with the timers of the scheduled self events. Timing issues are discussed in the next subsection.

C. Dealing with time

The low-power states described so far pose timing issues which get worse when moving from Standby to Hibernation. In pure Standby mode, in fact, the timer interrupt is used only to implement right-in-time wake-up, so that it doesn't provide any information about the actual time at which external

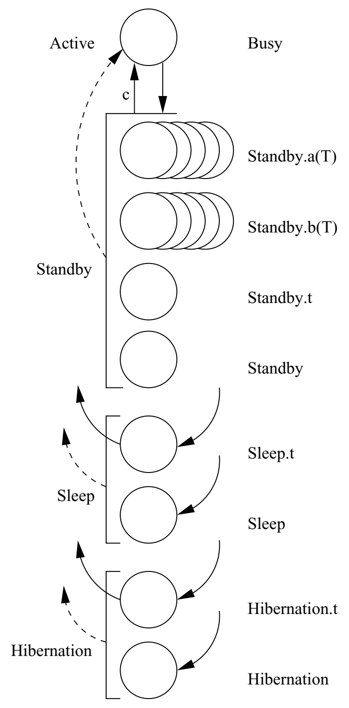

Fig. 3. State diagram of the power states made available by the proposed architecture.

interrupts occurrs. In Sleep mode the clock system is switched off, so that the MCU is unable to schedule its own wake-up, which can be triggered only by external events which do not provide any information about the time elapsed while the MCU was sleeping. Finally, in Hibernation the RAM is switched off and the OS is rebooted at wake-up, so that even the time at which hibernation was triggered is lost unless it is stored in flash and restored at wake-up.

All these issues can be addressed by equipping the system with an external real time clock (RTC) optimized for low-power applications. The RTC can be used by the MCU both to schedule wake-up calls from the deepest low-power states, and to update the system clock at wake-up. No RTC is required if the MCU is used only to implement bare reactive applications, such as sensor nodes used either to count external events or to give the alarm when specific conditions are detected.

D. Modified power state diagram

The changes outlined in the previous subsections enable the full exploitation of the low-power states depicted in Figure 1 and introduce three additional states which correspond to the Standby, Sleep, and Hibernation modes with external RTC. The new low-power states are denoted by suffix ".t" in Figure 3. With respect to the corresponding original states the new ones not only provide more accurate timing information, but they also grant to the MCU the capability of scheduling its own wake-up from Sleep and Hibernation. This possibility, denoted by the solid arcs exiting from states Sleep.t and Hibernation.t, allows the MCU to exploit the corresponding low power modes even if there are future self events scheduled by the running processes/threads.

State name	**Power [μW]**	**WUt [ms]**	**WUe [μJ]**	**SDt [ms]**	**SDe [μJ]**
Active	6600μW	n.a.	n.a.	n.a.	n.a.
Standby.a(T)	$4.5 + 153.72/T$	23.41	153.57	–	–
Standby.b(T)	$4.5 + 0.33/T$	23.41	153.57	–	–
Standby.t	$4.5 + 0.3$	23.41	153.57	–	–
Standby	4.5	23.41	153.57	–	–
Sleep.t	*1.5+0.3*	23.41	153.57	–	–
Sleep	*1.5*	23.41	153.57	–	–
Hibernation.t	*0.1+0.3*	560	2312.8	78.8	606.76
Hibernation	*0.1*	560	2312.8	78.8	606.76

TABLE I
CHARACTERIZATION RESULTS.

To make it possible to develop advanced DPM algorithms while working on top of the virtual machine, a `PowerManager` class has been created which exports methods for adjusting the `INTERVAL` of the timer interrupt and for triggering transitions to any low-power state, possibly specifying the wake-up time and deciding whether to use the external RTC or not.

IV. PRELIMINARY EXPERIMENTAL RESULTS

The solution proposed in Section III was implemented targeting Texas Instruments' MSP430 16-bit MCUs, which offer ultra-low-power active mode and various low-power inactive modes (LPMs) with sub-μA supply current and very fast wake-up times. A prototype board is currently under development based on the new MSP430F5418a mixed-signal MCU, equipped with an external sub-μA RTC (namely, NXP PCF2123). The target MCU features LPMs which can be easily mapped on the low-power states of Figure 3: Standby corresponds to LPM3, where both self wake-up and data retention are guaranteed; Sleep corresponds to LPM4, where the content of main memory is preserved, but the clock system is turned off; Hibernation corresponds to LPM4.5, which does not provide RAM retention.

Since the target board was not yet available, the power states were characterized using a testbed based on MSP430F2618, which does not support LPM4.5. Hence, transitions to and from Hibernation were emulated by making a copy of the heap in flash and by resetting the MCU. Synthetic benchmarks were developed in Java (using the API provided by the `PowerManager` class introduced in Section III-D) and run on top of the modified version of the Darjeeling VM. Each benchmark triggered periodic transitions to and from the inactive state under characterization in order to produce a periodic current waveform that was sampled by a National Instruments DAQ Board PCI 6024E. The sampled waveforms were then automatically analyzed using an in-house tool capable of isolating all the occurrences of a given pattern and making statistically-sound measurements on it.

Table I reports the results obtained with the MSP430F2618 MCU powered at 3V and clocked at 16MHz, using 4Kbyte for the heap of the VM. For each inactive state 5 parameters are reported: power consumption (Power), wake-up time (WUt), wake-up energy (WUe), shut-down time (SDt), and shut-down energy (SDe). Since the proposed architecture makes power management available on top of the virtual machine, wake-up costs include the time and energy spent in all the steps required to resume the execution of the running thread. Missing entries in Table I refer to transition times lower than 0.01ms and transition energies lower than 0.01μJ. Numbers reported in Italic are taken from the datasheet rather than from real measurements. The power consumption of states Standby.a and Standby.b is expressed as a function of the `INTERVAL` (denoted by T and expressed in seconds) used to trigger periodic timer interrupts.

It is worth noticing that there is a sizeable difference between the average power consumption of the four Standby states. For instance, with an `INTERVAL` of 100ms, the MCU consumes 1541.7μW in Standby.a, 7.8μW in Standby.b, 4.8μW in Standby.t, and 4.5μW in pure Standby mode. Moreover, the power consumption of the external RTC doesn't impair the energy efficiency of Sleep.t and Hibernation.t modes, their power consumption (of 1.8μW and 0.4μW, respectively) being significantly lower than that of the lowest Standby state.

Using for comparison the same MCU running standard releases of Contiki OS and Darjeeling VM, the only inactive state available would have been Standby.a with $T = 10$ms. Since the default `INTERVAL` of Contiki is lower than the time spent to resume the execution of a virtual thread (23.41ms), power management would have been totally ineffective without the proposed changes.

In order to have a practial idea of the energy efficiency offered by the power states made available by the proposed architecture, consider, for instance, the case of a sensor node periodically executing a monitoring task which keeps the CPU busy for 1 second. Figure 4 provides the average power consumption of the MCU as a function of the monitoring period, plotted in a log-log graph. Each curve refers to a specific power state and reports the average power consumption obtained by spending all the idle time in that state, taking into account transition costs. For Standby.a and Standby.b an `INTERVAL` of 100ms was used. Arrows show how the corresponding curves would change by reducing the `INTERVAL`. Sleep and Hibernation states without external RTC are not reported in the graph since they cannot be used in this case because thet do not support self wake-up.

This simple experiment clearly shows the enhanced energy efficiency provided by the deepest low-power states in case of long idle periods, which are typical of sensor node applications. Moreover, it demonstrates that all the power modes are worth being made available, since none of them outperforms the others in all workload conditions.

V. Conclusion

The convergence between the performance of ultra-low-power MCUs and the requirements of tiny virtual machines makes it possible to build energy efficient sensor nodes providing a virtual runtime environment. Virtualization, however, impairs the effectiveness of dynamic power management since it hides to the OS the idleness of virtual threads. The virtual machine is viewed by the OS as a process which can never be suspended for a period longer than the time resolution of

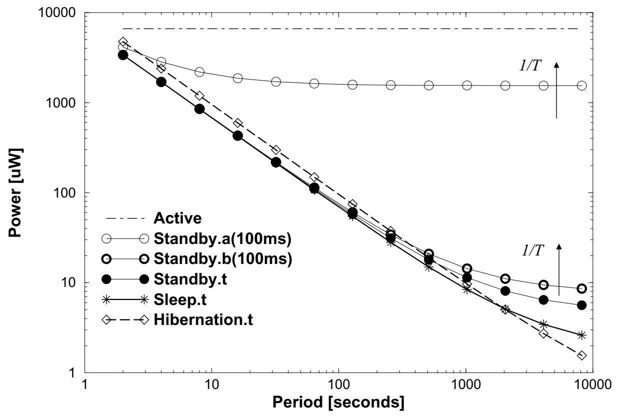

Fig. 4. Average power consumption of the MCU used to execute a periodic monitoring task which keeps the CPU busy for 1 second.

the system, so that idleness cannot be exploited to switch the MCU to inactive low-power modes.

This paper has presented an open-source power-manageable virtual environment for ultra-low-power tiny systems based on modified versions of Contiki OS and Darjeeling VM. The proposed architecture makes it possible to fully exploit the low-power modes of the MCU while working on top of a Java runtime environment.

Preliminary experimental results have been reported, while a prototype sensor node equipped with a Texas Instruments' MSP430F5418a MCU is currently under development to enable the deployment of full-fledged power manageable WSNs featuring a Java runtime environment.

Acknowledgment

The authors would like to thank Andrea Seraghiti, Massimo Zandri and NeuNet Cultural Association. The research leading to these results has received funding from European Commission's Seventh Framework Programme (FP7) under grant agreement No. 257418, project ULOOP (User-centric Wireless Local Loop).

References

[1] M Chen, S. Gonzalez, A. Vasilakos, H Cao, and V C. Leung, "Body Area Networks: A Survey", *Mobile Networks and Applications*, Vol. 16, No. 2, pp. 171-193, 2011.
[2] L. M. Oliveira and J. J. Rodrigues, "Wireless Sensor Networks: a Survey on Environmental Monitoring", *Journal of Communications*, Vol. 6, No. 2, pp. 143-151, 2011.
[3] E. Lattanzi and A. Bogliolo, "WSN Design for Unlimited Lifetime", In Yen Kheng Tan (Ed.), *Sustainable Energy Harvesting Technologies: Past, Present and Future*, InTech, 2011.
[4] M. O. Farooq and T. Kunz, "Operating Systems for Wireless Sensor Networks: A Survey", *Sensors*, Vol. 11, No. 6, pp. 5900-5930, 2011.
[5] F. Aslam, *Challenges and Solutions in the Design of a Java Virtual Machine for Resource Constrained Microcontrollers*, Ph.D. Thesis, University of Freiburg, 2011.
[6] V. Sharma, U. Mukherji, V. Joseph, and S. Gupta, "Optimal energy management policies for energy harvesting sensor nodes", *IEEE Trans. on Wireless Communications*, Vol. 9, No. 4, pp. 1326-1336, 2010.
[7] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors", in *Proc. of the IEEE COnf. on Local Computer Networks*, pp. 455-462, 2004.
[8] N. Brouwers, P. Corke, and K. Langendoen. "Darjeeling, a Java compatible virtual machine for microcontrollers", in *Proc. of the ACM/IFIP/USENIX Middleware Conference Companion*, pp. 18-23, 2008.