Poster Abstract: An Efficient Operating System Abstraction Layer for Portable Applications in the Domain of Wireless Sensor Networks

Ramon Serna Oliver, Ivan Shcherbakov, Gerhard Fohler
Chair of Real-Time Systems
TU Kaiserslautern
{serna_oliver, shcherbakov, fohler}@eit.uni-kl.de

Abstract

Portability is a major concern in developing applications for embedded devices such as Wireless Sensor Networks (WSN). Abstractions of the hardware platform which are introduced by the operating system (OS) make possible to develop code independent of the hardware, which can be reused in later deployments. However, the lack of standard APIs for the variety of OS in the domain of WSN restricts portability to those systems running the same OS.

We present on-going work on the design and development of a portable operating system abstraction layer (OSAL), which achieves a complete abstraction of the OS architecture as well as a common API across multiple OS. Portability at the application level is effectively achieved thanks to a common set of primitives which abstract the underlaying OS and its particular architecture.

We provide argumentation to highlight the efficiency of the OSAL and a general introduction to its features and design considerations. Moreover, we present a preliminary evaluation of the current implementation, which has proven to introduce minimal run-time overhead as well as negligible increase on the software footprint.

Categories and Subject Descriptors

D.4 [**Software**]: Operating Systems

General Terms

Performance, Design

Keywords

Abstraction, Embedded, Operating System, WSN, API

1 Introduction

The increasing popularity of Wireless Sensor Networks (WSN) [1] and their expansion to new application domains demand fast developments and quick integration of software and hardware components. Enabling portability of these components is a key issue which becomes essential for efficient deployments of large and complex systems.

Typical hardware elements in WSN have evolved from simple boards with minimal sensing and communication capacities to complex systems equipped with customized hardware modules and advance architectures. Operating systems (OS) play an important role in abstracting the underlaying hardware architecture, which is exposed to the applications through the OS API. New hardware components can be added to the platform (e.g. new sensor attached to an SPI bus), existing hardware can be upgraded (e.g. exchange of radio transceiver) or even the complete hardware platform may be exchanged without requiring major modifications at the application level.

Unfortunately, there is a broad number of OS targeting the domain of embedded sensor networks (e.g. [2], [3], [4], [5]), providing a common set of core functionalities and services. For a particular deployment, the decision may depend on external factors like, among others, support of the chosen platform architecture (i.e. CPU family), availability of drivers for sensors and/or communication buses, existence of a network stack or particular protocols (e.g. 802.15.4), or special characteristics of the OS (e.g. real-time scheduler).

There is, however, an implicit incompatibility between different OS due to their different APIs. Applications are design and implemented for a particular OS and their portability is reduced to those platforms supported by the OS. This issue, which is not relevant for small-scale deployments becomes relevant in complex systems where the dependency towards the particular OS API and architecture is larger.

2 OS Abstraction Layer

The definition of proper control mechanisms for the hardware platform into software frameworks arise a number of portability issues. The OS abstraction layer (OSAL) [6] is designed to address these issues and diminish the conflicts between different software and hardware platforms. In particular, it addresses the discrepancies among different OS with respect to their functional API, hardware configuration mechanisms, resource management and peripherals handling.

The OSAL is an abstraction layer set on top of the OS, which translates system primitives from the target operating system into an unified API. Thus, application builders make use of a common API and hence, portability among different platforms is reduced to the re-implementation of the OSAL.

2.1 Core Implementation

The OSAL embraces the management of hardware configurations and access to specific set-points which represent a major hook to performance trade-offs. It defines a subset of OS primitives which satisfies the basic application

Copyright is held by the author/owner(s).
SenSys'09, November 4–6, 2009, Berkeley, CA, USA.
ACM 978-1-60558-748-6

	Code size	Initialized	Non-initialized
Original	8192	10	446
OSAL	8200 (+8)	10	446

Table 1. Memory footprint for a simple application with and without OSAL (in bytes)

builder's requirements but at the same time, remain simple to match most target OS. The resemblance of the subset to a POSIX [7] OS API is motivated by the desire of reducing the learning-curve as well as the preference of a neutral reference without specific features from any particular platform.

Minimal footprint and execution overhead is achieved with advance compilation tools and techniques. Among others, function level linking, in-line functions and extensive use of macros to provide light function wrappers for the OS native primitives. The main achievements of the OSAL are:

- unify basic data types and structures;
- unify API for primitives of same functionality;
- adjust equivalent primitives with different parameters, parameters order or parameters types;
- extend non-equivalent primitives to perform equally;
- unify return values, error codes and system constants;
- provide unified APIs for non-existing extensions;
- abstract specific OS and hardware initialization phases.

2.2 Radio API

The radio API is a subset of the OSAL which abstract internal structures and management of data packets. Its goal is to abstract platform-specific details from the user and to provide a transparent interface for dealing with packets.

As different OS use different structures for packet buffers (some of them are fixed-size, some support variable length), OSAL provides a unified preprocessor macro allowing to declare such a structure, specifying its name and the desired size. Specific OS size bounds are checked during compilation time.

Particular attention has been given to the buffer management for radio transmission and reception. For example, MANTIS OS copies incoming packets into a fixed system-maintained buffer (a packet is always received in the same place and then can be copied by the application). The OSAL enables direct access to the system buffers by means of a preprocessor macro, which avoids costly and unnecessary memory operations.

3 Evaluation

We have successfully implemented the OSAL on top of MANTIS OS and we are currently working on the FreeRTOS port (see [8]). Moreover, OSAL is currently being exploited in the development of a portable network stack for WSN [9].

The only overhead added by the current implementation of the OSAL API on top of MANTIS OS, is of 8 bytes of code for each application, which are due to the additional initialization function. Note that in cases where MANTIS did not provide required functionalities (e.g. message-oriented queues), the increase on the code size is bigger. However, this is no longer considered overhead as new functionalities were implemented.

Table 1 shows the byte size of a sample application (`blinking_led`) implemented with and without the OSAL.

4 Conclusions

Portability of application-level code is a major concern in fast and efficient developments for Wireless Sensor Networks. We present an Operating System Abstraction Layer (OSAL) to reduce the portability efforts of software components between platforms. We provide argumentation to highlight the efficiency of such abstraction layer and present preliminary figures for the current implementation on top of MANTIS OS [2]. A complete description of the OSAL API can be found in [8].

Future and on-going work includes the development of the OSAL on top of FreeRTOS [3] and the extension of the API to support additional features.

5 Acknowledgments

This work is partially financed by the European Commission under the Framework 6 IST Project ”Wirelessly Accessible Sensor Populations (WASP)”.

6 References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless sensor networks: A survey,” *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[2] “MantisOS,” 2009. [Online]. Available: http://mantis.cs.colorado.edu/

[3] “FreeRTOS,” 2009. [Online]. Available: http://www.freertos.org/

[4] “Contiki OS,” 2009. [Online]. Available: http://www.sics.se/contiki/

[5] “TinyOS,” 2009. [Online]. Available: http://www.tinyos.net/

[6] A. Schoofs, M. Aoun, P. van der Stok, J. Catalano, R. Serna Oliver, and G. Fohler, “On enabling portable and time-controlled wireless sensor network applications,” in *Proceedings of the 1st International Conference on Sensor Networks Applications, Experimentation and Logistics (SENSAPEAL09)*, Athens, Greece, September 2009.

[7] M. A. Rivas and M. G. Harbour, “Evaluation of new posix real-time operating systems services for small embedded platforms,” in *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS)*, 2003.

[8] I. Shcherbakov and R. Serna Oliver, “The WASP OS API.” [Online]. Available: http://rts-wiki.eit.uni-kl.de/WASP/index.html

[9] “EU Framework 6 IST Project ”Wirelessly Accessible Sensor Populations” (WASP),” 2006-2010. [Online]. Available: http://www.wasp-project.org/