Software Architecture for GPS-enabled Wildfire Sensorboard

David M. Doolin[†], **Nicholas Sitar**[*] **and Steve Glaser**[*]

University of California, Berkeley

Civil and Environmental Engineering
University of California, Berkeley
Berkeley, CA 94720
email: doolin@ce.berkeley.edu

Abstract

Wireless sensors for conducting wildfire monitoring share many of the capabilities of other environmental sensors, collecting data such as humidity, temperature and barometric pressure. On-board GPS location finding allows rapid, remote deployment. In this poster, a scheme for developing driver and interface software for employing the Crossbow MTS420CA sensorboard is described. A high-level, generalized sensor interface is presented. Data collection algorithms implemented over implementations of this sensor interface do not require programming changes to the underlying sensor driver code.

1 Introduction

Monitoring rapidly changing environmental conditions occurring during wildfire requires deployment of large numbers of sensors into dangerous environments. The NSF Information Technology Research sponsored "Firebug" project (2003) is developing a small, inexpensive platform using wireless communication networks to support a heterogeneous array of sensors useful for detecting the initiation and monitoring the spread of wildfires. One component of the Firebug project is a environmental sensor board with GPS location capabilities ("Fireboard"). The software architecture for the fireboard is described in this poster.

1.1 Fireboard

The Fireboard, bottom and top shown in Figs. 1 and 2 respectively, is composed of an Analog Devices ADXL202JE accelarometer, an Sensirion SHT11 combined temperature and humidity sensor, an Intersema 5534AP combined barometric pressure and temperature sensor a Taos 250RD light sensor, and a LeadTek 9546 GPS unit. The driver code for the board initially combined previously written mica weatherboard code with preexisting code from a GPS unit. The code, while useful for demonstration purposes, required updating for TinyOS 1.1 and was written for synchronous operation using either the GPS unit, or all the other sensors, but not both at the same time.

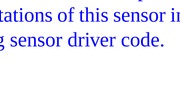

Figure 1. Bottom view of Fireboard.

Figure 2. Top view of Fireboard.

1.2 Sensor and sensorboard operation

Currently, sensor values are collected by implementing components providing access directly to ADC or UART. This has the advantage of being "close to the metal", at the expense of introducing unnecessary complexity for application programmers.

To reduce the complexity of sensor board programming, Gay et al. (2004) recently proposed formalizing a specification for implementing sensor driver code as part of a sensorboard specification. This proposal, in part, reflects current implementations of code for operating sensors on "weather boards", such as that deployed at Great Duck Island (Mainwaring et al. 2002). For example, controlling sensors would still require understanding the code and mechanisms behind the operation of I2C, UART, and ADC.

Ho (2004) proposed a MATLAB-based system for sending messages to motes pre-programmed with a "GenericSensor" application. Specifications for the active message (AM) structure are provided, including messages for command and control, route discovery, and data transmission. The advantage of this scheme is that once the software supporting the sensor hardware has been implemented, the system requires little further work on the mote software; data collection and mote control are provided through software written on the widely-familiar MATLAB platform. Disadvantages include: latency induced by having centralized control over mote behavior, slowing down the response of adaptive data collection algorithms and necessity of rewriting the application to fit changes in sensor hardware.

The Gay et al. and Ho proposals could be considered as part of a "DPI": Developer's Programming Interface. The HLSensor interface described here could be considered part of an API; an Applications Programming Interface constructed on the lower level modules, as shown in Fig. 3.

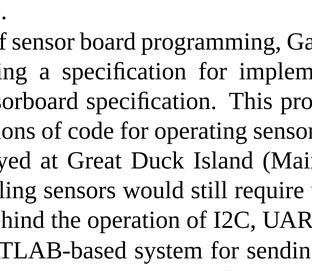

Figure 3. An API stack for TinyOS applications.

1.3 Requirements and capabilities

We needed at least the following characteristics for the driver code:

- Ability to control which sensors are working at a given time.
- Separate data collection algorithm from sensor driver code. Changing the data collection algorithm should not require any changes in the lower level sensor driver code.
- Understandable, in principle, by participants without formal training in electrical engineering /computer science.

The definition and implementation of the HLSensor interfaces form a "Programmer's API" that resides above the TinyOS core modules, but below the level of applications. Other types of applications benefit from this kind of abstraction. For example, Lynch et al. (2003) use a microcontroller to power both sensors and a computational unit for performing fast fourier transform on the platform. Typically, such transforms are used to extract a small number of lower order terms representing the dominant response. This saves a considerable amount of network traffic since the time series is not transmitted and low accuracy, higher order terms are ignored.

2 Module architecture

Separating a sensor board into constituent sensors has the following advantages:

- Reduces complexity — divide and conquer.
- Encourages incremental implementation and testing. For example, not every command in an interface need be implemented initially. As commands are incrementally implemented, the implementation can be debugged.

TinyOS and nesC use a number of naming conventions. The Sensor interface follows this example:

- Each sensor located in a directory named by: `manufacturer_modelnumber` (Figs. 4, 5).
  Example: `sensirion_sht11`.
- Each sensor board located in directory named by `manufacturer_modelnumber`.
  Example: `xbow_mts420ca`.

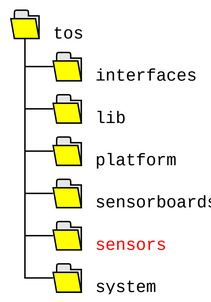

Figure 4. Structure of tos directory with sensors directory included.

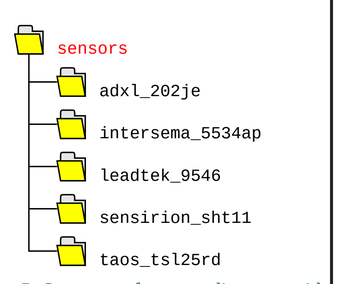

Figure 5. Structure of sensors directory with sensors used for Crossbow MTS420CA sensor board.

The actual files making the sensor work are divided into a driver file, then everything else:

- *_driver.nc is the configuration for sensor "*". This provides a "first line of defense" when driver code needs to be modified, and should consist only of nesC system code.
- Every other file in the directory supports the driver, and may consist of a mixture of nesC system code and hardware specific code.
- CPU specific code (Atmel, TI, etc) should not be located in this directory.

Constructing unique names for sensor boards eliminates any ambiguity about precisely which piece of hardware is being used. A unique name reduces the potential for blunders caused by incorrect include paths, where `sensorboard.h` may be included more than once, or from the wrong location. An example of this is compiling the micawbdot into an application, then using the fireboard, which will return data of some unknown sort.

2.1 High-level Sensor interface

```
interface HLSensor {

  command result_t powerOn(uint8_t power_state);
  event   result_t powerOnDone();
  command result_t init();
  command result_t powerOff();
  event   result_t powerOffDone();
  command result_t setSamplingInterval(//
                   uint16_t interval);
  command result_t getSamplingInterval(//
                   uint16_t interval);
  command result_t startSampling();
  command result_t stopSampling();
  command result_t sampleOnce();
  event   result_t dataReady(void * userdata);
  command result_t loadProgram(uint8_t * program);
  event   result_t error(uint16_t error_code);
}
```

The HLSensor interface is "heavier" than most of the interfaces defined in the TinyOS core. Interfaces in the core are more general, designed for flexibility. The HLSensor interface trades some flexibility for ease-of-programming at the application level. Interfaces in the TinyOS core could be considered an API for library extension development, constituting a "developer's programming interface" (DPI). In contrast, the HLSensor interface provides an application's programmer interface (API). Implementations of HLSensor allow application programmers to experiment with distributed data collection algorithms, such as feedback-controlled adaptive algorithms, without having to modify any driver code. One advantage of HLSensor is that a general sensorboard component may be written that provides multiple sensors. The sensorboard component then packages the requisite data into a custom data structure which is then tucked into an active message for radio transmission.

Sampling is controlled by 6 commands which reflect the expectations and vocabulary of domain experts: [get,set]SamplingInterval, [start,stop]Sampling, sampleOnce and dataReady. These commands do exactly what the names suggest. The commands [start,stop]Sampling are useful for continuous, asynchronous sampling where the implementation controls the timing of the data collection. This capability is useful for adaptive sampling. sampleOnce is useful for synchronous data collection controlled, for example, by a sensorboard component.

The dataReady event provides a void pointer cast to a sensor-specific data structure. This differs from the Gay et al. (2004) proposal for a lower level sensor interface, where the dataReady event returns raw values from hardware components such as the ADC. This and the Gay et al. specifications complement each other in the sense that HLSensor may be implemented on top of the Gay et al. design for processing and calibrating hardware readings on the mote to allow distibuted/autonomous control over data collection.

Power, init and error Arguments to the powerOn command allow control of sensors with multiple power states. The init command provides software initialization of a sensor component, independent of the power[On/Off] command. The error event passes an sensor-specific error argument, defined in the header file for each sensor.

Runtime sensor programming is performed using the loadProgram command. All sensors are assumed *logically* programmable,

- If there is hardware support for programming, loading a program causes sensor/implementation dependent behavior.
- If there is no hardware programming support, then loading a program is a no-op.

Note that the program could just as easily be used for controlling driver software state controlling sensor hardware operations.

3 Summary

- Separating the Fireboard sensors into modules allowed driver code for each sensor to be upgraded independently.
- Using an implementation of the HLSensor interface is much easier than modifying drivers.
- The HLSensor interface is not a panacea, but given there is no "one size fits all" the current implementation works well.
- In light of recently proposed sensor abstractions (Gay et al. 2004; Ho 2004), more work needs to be done to define the interaction between the TinyOS core and higher levels interfaces such as HLSensor.

References

Chen, M. M., C. Majidi, D. M. Doolin, S. Glaser, and N. Sitar (2003, June 17-18).Design and construction of a wildfire instrumentation system using networked sensors.Poster — Network Embedded Systems Technology (NEST) Retreat, Oakland, CA.

Gay, D., P. Levis, and J. Polastre (2004).Standard TinyOS sensorboard interface. (Unpublished) Distributed with TinyOS source, 4 Feb., 2004.

Ho, H. (2004).Generic sensor platform for networked sensor nodes. (Unpublished) Distributed with TinyOS source, 4 Feb., 2004.

Lynch, J. P., A. Sundararajan, K. Law, A. S. Kiremidjian, T. Kenny, and E. Carryer (2003).Embedment of structural monitoring algorithms in a wireless sensing unit.*Structural Engineering and Mechanics 15*, 285–297.

Mainwaring, A., J. Polastre, R. Szewczyk, D. Culler, and J. Anderson (2002, September).Wireless sensor networks for habitat monitoring.In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pp. 88–97. ACM Press.

[†]Post-doctoral researcher, Civil and Environmental Engineering, University of California, Berkeley

[*]Professor, Civil and Environmental Engineering, University of California, Berkeley