Portability and Embedded Web Services for Sensor Networks

Diana Fang dcfang@cs.ucsd.edu
Kaisen Lin kaisenl@cs.ucsd.edu
Pongsakorn Teeraparpwong pteerapa@cs.ucsd.edu

ABSTRACT
In this paper, we describe various tradeoffs associated in the design of a portability layer for a diverse set of operating systems and a diverse set of hardware. We also describe our design and implementation of web services for sensor network data. We have partially implemented our work on the mPlatform sensor platform, generated data, and sent it over the serial port to a mobile client. We have also implemented a web service that can receive the data from the client over the Internet and generate and process the results.

1. INTRODUCTION
There has been much work in building sensor network operating systems, runtime environments, and hardware platforms, but two issues have made it difficult for mainstream adoption. First, porting existing operating systems and runtime environments to new hardware devices continues to be difficult. While some operating systems, such as Contiki [1], attempt to be portable, developers who choose to use it must adhere to its runtime environment. The other issue is usability at the end user. Sensor network deployments typically collect data at a basestation, and they are given to specific researchers. Efforts such as SenseWeb [2] from Microsoft Research have attempted to integrate sensor data into Web 2.0 services so that new sensor data is available for everyone to consume.

One technique for portability that has been researched is paravirtualization. Paravirtualization is a virtualization technique that improves performance by presenting a different interface to a virtual machine than the underlying hardware interface. However, this requires porting existing and very complex operating systems to a new interface. While this generally detracts from using paravirtualization, we can use that same technique to simplify writing new operating systems. The assumption is that given a properly designed unified interface, operating systems developers will not need to write hardware dependent code. This is especially important in the embedded world where there is no de facto standard such as x86. Furthermore, hardware developers and low level device driver writers need only provide a single application binary interface (ABI) for multiple operating systems.

In terms of making the sensor data accessible to everyone, easy to view, and process, Web 2.0 seems to be a good choice. Web 2.0 applications have become ubiquitous in computers. Web 2.0 applications typically draw content from other web services to provide a new service. Sharing such information on the Internet to millions of users introduces both security and scalability issues.

SOAP is a protocol that features portability and ease of use. SOAP is a protocol specification that allows for the exchange of structured information in web services in computer networks. It uses XML as its message format and uses Remote Procedure Calls (RPC) and HTTP for message negotiation and transmission. SOAP provides platform and language independence as well as versatility for the use of different transport protocols.

2. PORTABILITY IN EMBEDDED SYSTEMS
Xen [3] is a virtual machine monitor that uses paravirtualization to improve performance. It provides an abstraction for memory management, CPU, interrupts, timers, disks, and networks. It is unclear if their work is readily usable for embedded systems. We outline some differences between traditional laptop/desktop machines and embedded devices.

Realtime: Embedded systems often have hard real-time constraints. Thus multiplexing through virtualization can sometimes cause unacceptable and unpredictable delays. Arbitration and reservations for resources may be promising, but timers and interrupts are often better utilized when shared.

Cooperative: Traditional virtual machines offer protection and isolation among guests as well as with the host. However, embedded software systems are often made of cooperative tasks operating with a common goal. This implies that protection mechanisms may not be as crucial.

Diversity: Embedded platforms often have much more diversity in both their hardware capabilities and operating system environments. For example on the operating system side, TinyOS [4] provides a single threaded event driven programming style with static memory allocation, while LiteOS [5] provides a Unix like shell interface with dynamic memory. On the hardware side, there are numerous processors in use today such as TI MSP430, ARM7, and various Qualcomm processors. Even then, a diverse set of peripherals are meshed together with microcontrollers and CPUs to create a complete hardware platform. Examples of these at the low end are Telos [6], mPlatform [7], and iMote2, while at the higher end are the iPhone, Nokia phones, and Blackberry.

Resource Constraints: Embedded devices are often much more resource constrained compared to desktops particularly in computation power and memory resources. This means various data structures such as thread control blocks and message buffers must use minimal space. Although perhaps not as common, there are situations where two environments may need to be concurrently run. For example, one operating system environment might have better support for real-time support where as another is better suited for interactive usage.
2.1 Design
In our proposed design, we focus on a binary interface for dealing with the CPU, memory, I2C/SPI interfaces, ADC, Timer, GPIO control, and hardware interrupts. We choose these components because almost all sensor network applications will touch upon these areas.

The CPU is probably the most well studied device to be multiplexed. Our design will use a preemptive round-robin scheduler to enforce real-time constraints. Tasks will also always be scheduled, but they can yield their cycles to a light sleep mode to conserve energy. Deep sleep modes should never be used unless wakeup latency can be bounded such that all real-time constraints are still satisfied. It is up to the user level scheduler to determine how its slots should be used.

Memory should be able to be dynamically allocated so that unused memory can be reclaimed, but operating systems with more stringent constraints or with statically allocated memory models should pre-allocate memory and hold it. Hardware platforms with virtual memory will need an interface to pin pages down for real-time deadlines. Operating systems that want to use virtual memory will need to write their own memory manager if it is not supported in hardware.

Serial interfaces such as I2C and SPI, and even GPIO lines are typically arbitrated rather than virtualized because of state on those devices. Virtualization of these devices can also cause unbounded delays. We choose not to provide a high level abstraction because various sensors each may have unique interfaces for controlling the hardware. We feel it should be the operating system's responsibility to provide higher level interfaces. TinyOS2, for example, provides telescoping abstractions to allow various levels of device interaction.

Timers and hardware interrupts, unlike serial interfaces, should be virtualized because they are often used for very short amounts of time in an asynchronous manner. However, handlers for these services have to be kept very short. While it is possible to have fast handlers built into the portability layer that services the event and then schedules a task, this could have poor real-time consequences. Some solutions instead check the handler code at runtime before accepting it.

3. WEB SERVICES
3.1 Background
Web services are a standardized way of communicating and exchanging data between heterogeneous systems. The concept of remote data access is not new, but the main goal of web services is to allow remote access in a loosely coupled fashion. DCOM, IIOP and Java/RMI [8] are previous attempts that require the client and server to be tightly integrated in terms of platform implementation and binary data formats. Web services, on the other hand, only require that the client and server understand the messages they exchange. They do not need a particular component technology or object-calling convention like previous technologies do. As a result, with web services, one application can invoke the methods of another application regardless of their systems or programming languages. The core standards of web services include:

- Extensible Markup Language (XML) – The data passing to or from a web service must be encoded in the XML format. XML stands for extensible markup language, and is well-known for its expressiveness and platform-independence.

- Simple Object Access Protocol (SOAP) – SOAP is a standard that defines the format of XML messages that are passed between the client and server. SOAP specifies what XML elements must be used, in what order, and what data types they can contain. The SOAP message consists of three main parts: a SOAP envelope, an optional SOAP header and a SOAP body. The SOAP header and body are enclosed in the SOAP envelope. The web service invocation involves two messages: request and response. For the request message, the SOAP body must specify the name of the method the client wants to call and its arguments. For the response message, the body will contain the return value of the method. Figure 1 shows a simple SOAP request/response message in XML format.

- Web Service Description Language (WSDL) – WSDL is an XML document that describes the web service itself. In particular, it is a schema that describes the data types that a web service will receive and send, the type of message format, and the web service’s endpoints (URL). That is, it describes the methods that the web service provides for clients to use.

The web service messages are typically sent and received over a well-known transport like HTTP. Also, the flexibility of XML allows the web service to be evolved and extended over time. One key extension is Microsoft’s Web Service Enhancements (WSE). A broad array of standards extended from the three core standards includes many security features such as Digital signatures, encryption, and authentication, which are end-to-end message-level security features.

4. IMPLEMENTATION
Our implementation focuses on the design of the interaction between the sensor network and web services. Our
implementation includes our sensor nodes, the web service, and the web service clients. We use the mPlatform as our sensor nodes and then built a web service that stores the data collected from the sensor nodes. Figure 2 describes the overall web service architecture. The data can then be displayed on a website or on a mobile client.

### 4.1 mPlatform

We were not able to implement our portability layer due to time constraints, but we did implement our web service using live sensor data from an mPlatform node. The mPlatform is a stackable hardware platform from Microsoft Research that supports WiFi access, compact flash storage, light and temperature sensing, and 802.15.4 radio communication. It has processing capabilities from both TI MSP430 microcontrollers as well ARM7 processors. Multiple boards communicate with each other through a shared bus that is controlled by a Xilinx CPLD. Serial communication to a PC is handled by the ARM board. It was designed as a prototyping hardware platform because components can be swapped in and out depending on the requirements. In our work, we wrote ADC code to periodically sample a light sensor. The light data was then sent to the radio for transmission back to a basestation, where it was then handed off to the ARM processor for serial communication onto a PC.

### 4.2 Web Services

We implemented a web service running on a web server which connects to an SQL Server database. The database can be located either on the same server or a different one. The technology we used to implement the web service is C# ASP.NET using Visual Studio 2008. The web service provides functions to store data, fetch data of a particular device and fetch device information. In the future, the service can be extended to match the specification of a particular application domain. Table 1 summarizes the interface of the service. Furthermore, the web service contains a basic notification function which can send an email to notify users.

<table>
<thead>
<tr>
<th>Byte Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-2</td>
<td>Start bytes (0x0A, 0x0B, 0x0C)</td>
</tr>
<tr>
<td>3-4</td>
<td>Device ID</td>
</tr>
<tr>
<td>6-11</td>
<td>Timestamp</td>
</tr>
<tr>
<td>12-13</td>
<td>Data1</td>
</tr>
<tr>
<td>14-15</td>
<td>Data2</td>
</tr>
<tr>
<td>16-18</td>
<td>Stop bytes (0x0D, 0x0E, 0x0F)</td>
</tr>
</tbody>
</table>

**Table 2: Data Byte Order for the Sensor Node**

Figure 2 shows the overall architecture of the web service and its clients. User devices such as smart phones or laptops can have a client application that receives data from the sensor and calls the web service via the Internet to store and/or display data. The web server can also get the collected data from the web service and display it on a website.

### 4.3 Clients

To connect to the web service, we have implemented two different clients: one for mobile devices and one for the web. The goal was to make the information widely available and easy to understand for the user. The mobile device performs two different functions. It first receives data from the sensor nodes and sends it to the web service for storage and processing. It is also able to retrieve previously stored data from the web service. The web client allows you to access all the data from the web service from the convenience of any computer that has a web browser and an Internet connection. The web client also provides a login for a username and password for security. Both clients were implemented in Visual Studio 2008 using the .Net framework in C#.

#### 4.3.1 Mobile Client

The first functionality of the mobile client is to retrieve data from the sensor nodes and send it to the web service. In our setup, we had one sensor node act as the base station of the other nodes. This node is connected to a simulated mobile device on a PC via USB. We map it to a COM port and data is received on the mobile device in bytes. The byte order is given in Table 2. Whenever there is data waiting on this connection, an event handler is called to parse the data into the appropriate fields. It is then sent to the web service by calling the SOAP function PlaceDataOnService. The client also notifies the web service to send an alert email message when the data is above or below a given threshold.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int PlaceDataOnService(int devID, int data, DateTime ts)</td>
<td></td>
</tr>
<tr>
<td>List&lt;DataRecord&gt; GetDataRecordFromDeviceID(int id)</td>
<td></td>
</tr>
<tr>
<td>List&lt;Device&gt; GetDeviceList()</td>
<td></td>
</tr>
</tbody>
</table>
The mobile client is also able to retrieve data from the web service by calling other SOAP functions. If the mobile client is able to connect to the Internet it can also open a web page designed for mobile devices to view and examine the data stored on the web service.

4.3.2 Web Client
We implemented an ASP.NET web client which connects to the web service to get data and display it on a web site. AJAX technology was also used to make the website more interactive to users. A web server can be located on the same savior, same network or even a separate network as that of web service. Users can conveniently use normal web browser such IE or firefox to view the web page containing collected data from the sensor. User can select the device that he is interested. The data of that device will be displayed in a table as well as a graph. A basic user access control was implemented to ensure security and privacy of data. An example of useful scenario is that a doctor can easily monitor status of patient remotely by viewing collected data of medical sensors attached to a patient as long as the doctor has an Internet access. Figure 5 shows a sample captured screen of the web site.

5. DISCUSSION
The mobile client provides portability in that data can be sent and retrieved from any location and on any device that runs the client as long as there is Internet access. The user is also able to configure the appropriate range of values collected from various sensor devices that are deemed normal. This allows the user to receive alert emails when the values are out of range permitting the user to react to the alerts appropriately. However, sending and retrieving large amounts of data will be costly in terms of power consumption of the mobile client. This is a tradeoff between the throughput of data transfer and power conservation. This tradeoff can be further studied in future research.

One of the advantages of the web services is extensibility. The web services can be easily extended so it covers necessary functions for a particular application domain. For example, in a health application, the web service can provide a function for general statistics without disclosing individual data.

In terms of the mPlatform, while it is good for prototyping deployments, stacking multiple boards quickly adds to the physical size of the platform. Commercial deployments will still probably benefit from custom sensor hardware for cost and performance.

6. CONCLUSION
We have focused on a design for data collection in sensor networks that emphasizes portability and usability. We have proposed a design for a portable interface for dealing with the various hardware components of an embedded device. We have demonstrated a working build for our web service system for sensor network data. In particular, we have implemented the
infrastructure in terms of server applications and client applications needed to facilitate the transfer of data over the Internet. The various sensor nodes relay their data, and in our case, light sensor data, to a base node connected to a mobile device via a serial interface. The mobile device then sends this information to the web service using the SOAP protocol for processing and storage. The device is also able to query the web service for previously accumulated data. A web page can also be used to view the data. The whole architecture is able to be run with any mobile device and sensor network.

7. REFERENCES