Tiny Web Services for Sensor Device Interoperability

Bodhi Priyantha, Aman Kansal, Michel Goraczko, and Feng Zhao
Microsoft Research, One Microsoft Way, Redmond, WA.
{bodhip,kansal,michelg,zhao}@microsoft.com

Abstract

There are many scenarios where interoperability is required for sensor devices. We demonstrate one approach to achieve interoperability: using web services. Hosting a web service challenges the battery-life, bandwidth, and processing power constraints of low power sensor nodes. We demonstrate a lightweight implementation on MSP430 based sensor nodes with 802.15.4 radios. The implementation allows standards compliant web service clients to use the sensors but minimizes code size and energy at the sensor nodes. It allows sensor nodes to enter sleep modes. We prototype an example application for a home sensor network along with two types of sensor nodes required for it. We also show how our system enables sensor nodes to be used easily from applications written in high level languages using existing development tools.

1. Introduction

Interoperability of sensor nodes is important in several scenarios and has significant advantages. Consider a typical home for instance. It may have a security system with perimeter intrusion sensors on doors and windows, and motion detection sensors in some indoor areas. A home typically also has fire sensors wired directly to the fire-station. It contains a thermostat for indoor temperature sensing, connected to a cooling or heating system, and smoke sensors that may not be connected to a network. It may also have a hobbyist's weather station or other sensors. Note that each sensor is being used by only one application. If all these sensors were accessible from a common interface, not only could we continue to run the existing applications but also use these sensors for additional applications. For instance, the perimeter intrusion and motion sensors could be used in an activity monitoring application for home automation, these sensors along with the thermostat could be used in a home energy control application, and the activity monitoring could also be used for an assisted living application (Fig 1). Also, adding a few additional application specific sensors to such an existing

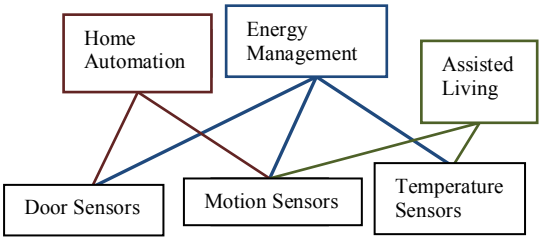

Figure 1. Interoperable sensor devices.

infrastructure is much cheaper than adding an entire sensor suite for each new application. To enable this vision, we need interoperability at the network and application layers.

Network layer interoperability can be achieved using IP. It works over many PHY layers and is a commonly used standard. Optimizations to IP energy and bandwidth usage are feasible using protocols such as the proposed 6LowPAN [1]. We assume that the IP overheads are acceptable for the PHY layer used, such as 802.15.4.

The next key challenge is **interoperability at the application layer**. An application developer needs to understand the control messages expected by a sensor, the parameter values needed by it, and the type of data produced. Even when the application developer may know the sensor semantics, the actual formats for messages sent and data values communicated are sensor specific. Developers are required to read detailed documentation, and develop custom programs to generate the relevant bit patterns and sequences of packet exchanges. For instance, the message formats required by X10, Insteon, Echelon, Zwave and HomePlug sensors are different for communicating the same information. One approach to achieve interoperability is to force each sensor vendor to adopt a new common specification such as Zigbee or Wibree. A second approach is to use existing web service standards in a lightweight manner. Each approach has its pros and cons. We demonstrate the second

approach, providing a sample point that may facilitate the debate between the two approaches.

2. Interoperability Using Web Services

In this approach, a sensor node reports its interface using the web service description language (WSDL) [2] and applications that wish to use the sensor can send it the messages specified. This has several advantages. If the application developer knows the semantics of the sensor (e.g. that a temperature sensor in a thermostat provides indoor air temperature) then automated tools can be used to parse the WSDL specification and generate method calls in an easy to use high level language (e.g. Java, C#). The data types of the parameters passed and data values received are well understood. For instance, a programmer could use Visual Studio or NetBeans IDE to automatically parse the WSDL specification and create a Java object that provides the device messages as method calls with typed arguments. The actual format and sequence of packets sent to the node is automatically generated by Visual Studio or NetBeans according to the node's WSDL specification. Using a device from a new manufacturer is easy as now the high level language object providing the device control messages as method calls is automatically generated. If a vendor's sensor has additional features these may simply appear as additional available method calls.

2.1. Challenges and Design

The web service approach has several challenges for use on low power sensor nodes. First, if the sensor node is battery powered and expected to operate for a long duration, it must enter *sleep modes* while typical web service hosts are assumed to be always on. Second, a battery powered node will use a low powered radio which has a *low data rate*. The total amount of data sent cannot be very large to meet battery life and latency constraints. Third, low power nodes must use constrained processors and memory, supporting only *limited complexity message processing*.

In our prototype we address the above challenges by judiciously selecting the web service components to be implemented on low power devices, but ensure that the device WSDL specification is standards compliant. We make simplifying assumptions in reducing the code complexity and data overheads for servicing the messages by leveraging the fact that all messages received are in accordance with the device WSDL specification and need only simple responses specified by the device itself. This allows a low power radio and processor to be used. The requirement for sleep is addressed by using Web Services Eventing [3]. The sensors expose their key methods as web service events so that the device can enter a sleep state after the method has been called. It may then wake up when the actual event (e.g. based on a sensor value) occurs and send the response as an event.

2.2. Application Example: Energy Control

As an illustration, we developed two new MSP430 based nodes for homes: (1) a power sensor-actuator that may be attached to wall sockets for measuring power drawn and turning the socket on/off, and (2) an 802.15.4 enabled thermostat, both with a web service interface. These are used in an energy control application written in a high level language.

A gateway node is also built that can connect the sensor nodes to Internet-based sensing services such as MSR SenseWeb [4] or other remote applications such as security monitoring or assisted living help-lines. The gateway node, not being power constrained, can implement sophisticated security and other web service features not implemented on the device.

3. Demonstration

We demonstrate:

1. Lightweight web services hosted on tiny devices (sensor device consisting of an MSP430 processor with 48k of ROM and 10k of RAM, an 802.15.4 radio, powered by AA batteries). The web service messages are carried over HTTP and TCP/IP.
2. Direct use of sensor WSDL specifications in existing application development tools, Visual Studio and NetBeans IDE, using high level languages.
3. Support for sleep mode at sensors that host the web service, using events.
4. Prototype devices: an 802.15.4 thermostat and smart-power-sockets (sense power draw, allow remote turn on/off) that host a web service interface. These are used in an example application.

References

[1] G. Montenegro et al, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", IETF RFC 4944. http://www.ietf.org/rfc/rfc4944.txt

[2] E. Chiristensen et al, "Web Services Description Language, W3C Note," http://www.w3.org/TR/wsdl.

[3] D. Box et al, "Web Services Eventing (WS-Eventing)," http://www.w3.org/Submission/WS-Eventing/

[4] A. Kansal et al, "SenseWeb: An Infrastructure for Shared Sensing," *IEEE Multimedia*. Vol. 14, No. 4, pp. 8-13, 2007.