Toward Interoperable Publish/Subscribe Communication between Wireless Sensor Networks and Access Networks

Pruet Boonma and Junichi Suzuki
Department of Computer Science
University of Massachusetts, Boston
{pruet, jxs}@cs.umb.edu

Abstract

Traditional wireless sensor networks (WSNs) often do not consider interoperability between WSNs and access networks. To address the issue, this paper investigates interoperable publish/subscribe communication in WSNs. The proposed middleware, called TinyDDS, provides two types of interoperability, programming language interoperability and protocol interoperability, by customizing standard data types, data representation and session protocol. Evaluation results show that TinyDDS simplifies the development of publish/subscribe applications and it is implemented efficient in memory footprint and power consumption.

1. Introduction

Wireless sensor networks (WSNs) have been investigated extensively; however, most of them are designed separately from access networks, which allow human users to connect with WSNs and perform information retrieval (e.g., data collection and event detection) in WSNs. Traditional WSNs tend not to consider and enable interoperability between WSNs and access networks. As a result, it is often ad-hoc, expensive and error-prone to build a gateway, which is responsible for protocol bridging and data conversion between WSNs and access networks. Currently, gateways need to be rebuilt from scratch programming language by programming language, protocol by protocol and application by application.

In order to address the above interoperability issue, this paper investigates interoperable publish/subscribe communication with TinyDDS, which is open-source[1] and standards-based middleware for WSNs. Compliant with the Object Management Group (OMG)'s standard Data Distribution Service (DDS) specification [9], TinyDDS provides two types of interoperability: *programming language interoperability* and *protocol interoperability*.

Programming language interoperability is the ability of TinyDDS to interoperate applications written in different programming languages. TinyDDS implements a mapping of the OMG IDL (Interface Definition Language) [8] to nesC and provides a set of DDS APIs in nesC. This allows different applications to use different languages with the same DDS APIs for event subscription and publication. For example, an access network application (or enduser application) can be implemented with Java, while a WSN application is implemented with nesC. Application developers do not have to learn/use different APIs for different applications. This can significantly improve their productivity.

Protocol interoperability is the ability of TinyDDS to interoperate WSN applications and access network applications on different MAC (L2), routing (L3) and transport (L4) protocols. TinyDDS implements a session (L5) protocol, called TinyGIOP, which is a subset of the standard General Inter-ORB Protocol (GIOP) [8]. Similar to GIOP, TinyGIOP is independent from any underlying protocols. It encapsulates and transmits data formatted with TinyCDR, which is a subset of the standard Common Data Representation (CDR) [8]. CDR is the standard set of binary representations of IDL types. Taking advantage of TinyGIOP and TinyCDR, TinyDDS makes publish/subscribe communication interoperable between WSNs and access networks. This allows application developers to build and maintain gateways in a cost effective manner.

This paper describes the proposed IDL-to-nesC mapping, TinyGIOP and TinyCDR, and evaluates TinyDDS through simulations.

2 OMG DDS Standard Specification

The Data Distribution Service (DSS) is a Object Management Group (OMG) standard for topic-based pub/sub middleware.DDS provides standard interfaces for event subscription and publication in Interface Definition Language (IDL), and TinyDDS implements them with nesC. See [3] for the IDL-nesC mapping in TinyDDS.

Figure 1 shows the architecture of DDS middleware. An event sink expresses its interest to an event, or *topic*, and subscribes to its local `Subscriber` with associated

[1]TinyDDS is available at dssg.cs.umb.edu.

`SubscriberListner` and `DataReader`.

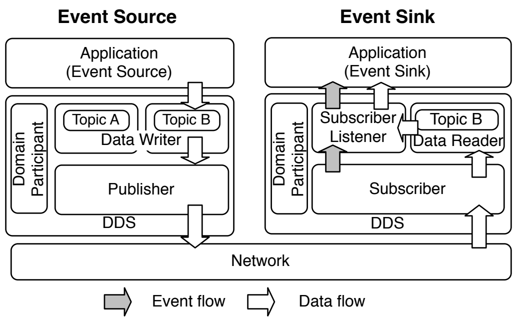

Figure 1. DDS Architecture

An event source creates an event/topic with its corresponding `DataWriter`, and the event is published by a `Publisher` to its subscribers in the network.

A `Subscriber` on each node monitors every incoming event, and if the event matches an event sink's subscription, it will be notified to the event sink via `SubscriberListner` and `DataReader`.

3. TinyDDS Architecture

The left-hand side of the Figure 2 shows the architecture of TinyDDS running in each sensor node. With respect to TCP/IP reference model, TinyDDS operates in transport layer and work on top of any network layer (L3) implementation. TinyDDS follows Layer design pattern [4] by separating different functionalities into different layers. At the top most layer, TinyDDS provides a subset of DDS interfaces to be used by applications. An application implemented on top of DDS can disseminate events, i.e., data or control messages, to the network with associated topic and the events are captured by any subscribers, i.e., base station, who has interest on the topic of the events. The implementation of those interfaces, as described in section 2, operates on top of TinyGIOP layer. TinyGIOP encapsulates data into transportation messages and interacts with the DDS Gateway for exchanging data with DDS applications. Only the nodes, i.e., base station, that are physically connected to the DDS gateway through serial interface can exchange data with the DDS gateway. For exchanging data with the other sensor nodes in the WSN, TinyGIOP utilizes an overlay network for event routing. Different routing protocols can be used to implement the overlay network by implementing in the Overlay Event Routing Protocols (OERP) layer. This OERP layer allows application developer to choose appropriate routing protocol to suit their requirements and constraints. For example, in sensor network with very limited memory space sensor nodes, spanning-tree routing protocol may be used because it needs minimal memory space to maintain routing table. On the other hand, sensor network which try to minimize the energy consumption of memory rich sensor nodes may use DHT-based routing protocol. By using this OERP layer, TinyDDS frees developers from the limitation of routing algorithm used in network layer which generally depends on sensor node platform such as Mica Z which based on Zigbee protocol stack. The routing protocol in OERP layer utilizes low-level network layer implementation through a transport layer interface called TinyDDS L4 Adaption Layer (L4AL). L4AL allows TinyOS to operates with any network and MAC layer protocol, such as AODV and Zigbee respectively.

3.1. DDS Interfaces

In the top most layer, TinyDDS provides an API for application developers. This API provides a subset of DDS for creating topics, subscribe to events of topics and publish events for particular topics. For each interfaces, the implementation is provided so application developers do not need to implement those interfaces themselves. The implementation for the DDS interfaces is written in nesC programming language and optimized for small sensor nodes platform such as MicaZ.

```
1  typedef struct {
2    cdr_int temperature;
3    cdr_ulong time;
4  } TempData_t;
5  Publisher_t publisher;
6  Topic_t topic;
7  DataWriter_t data_writer;
8  TempData_t temp_data;
9  command result_t StdControl.start() {
10  publisher = call DomainParticipant.create_publisher();
11  topic = call DomainParticipant.create_topic("TempSensor");
12  data_writer = call Publisher.create_datawriter(
13                                        publisher , topic);
14  temp_data.temperature = TempSensor.read();
15  temp_data.time = call Time.getLow32();
16  call DataWriter.write(data_writer, serialize(data),
17                     sizeof(TempData_t));
18 }
```

Listing 1. Example of TinyDDS Application

Listing 1 shows an example of an event source application implemented on top of TinyDDS. An user-defined data type is defined at line 1-4. Then, at line 10, a *Publisher* is created. Line 11-13, a *DataWriter* is created associate with topic "TempSensor". At line 14 and 15, a sensor reading is captured from temperature sensor and also the current time is read from a local clock, both information are stored in a variable of the user-define data type. Finally, at line 16, the data in user-defined data type is serialized into byte stream and published through *DataWriter* interface.

3.2 Overlay Event Routing Protocols

This OERP layer provides an overlay network over sensor network's physical ad-hoc networks. The overlay network is used for transporting published events, i.e. sensor data, to all nodes who subscribes to the events. The published event is routed to each subscribers according to the routing protocols deployed within OERP layer. Application developers can specify the deployed routing protocols to suit their need. The OERP layer encapsulates the

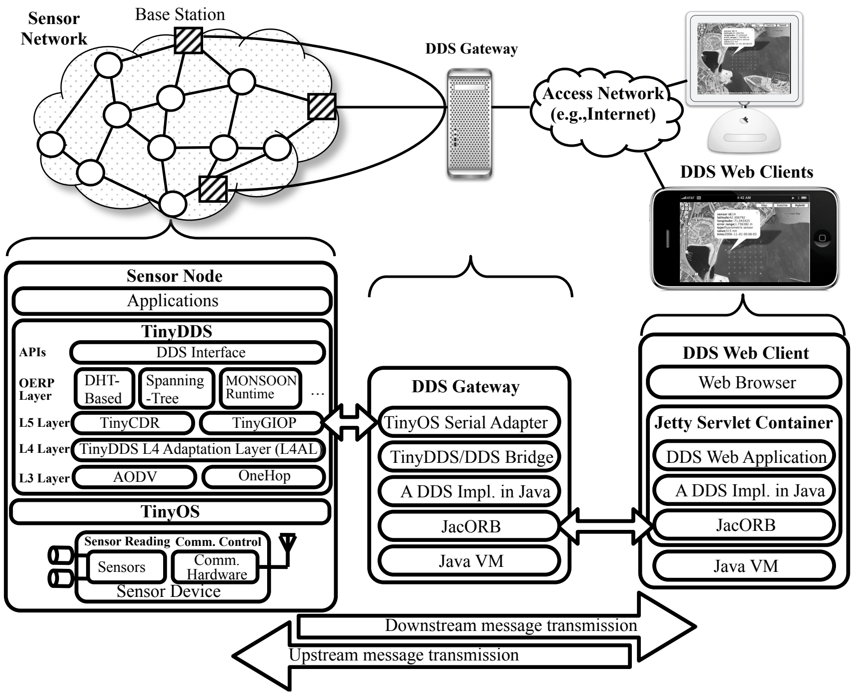

Figure 2. Architectural Components in TinyDDS

overlay network algorithm and implementation from DDS interfaces and the lower level physical network. Routing protocols in OERP layer work with lower-level network protocol through the TinyGIOP. In the other words, routing protocol is a pluggable component of TinyDDS which can be deployed/configured to meet application developers' requirements. For example, application developers who want to reduce the cost and size of sensor nodes by using small-memory sensor nodes may choose to use spanning-tree routing protocol which will use very small memory space. The routing protocols used in this OERP framework are developed by library developers and can be used in any TinyDDS-based applications.

3.3. TinyCDR

TinyCDR is a subset of the Common Data Representation (CDR) [8] which allow TinyDDS applications to directly exchange data with DDS applications. CDR is the format for exchanging data in DDS standardized by OMG. CDR enables different parties, i.e., sensor nodes and client applications, which utilizes different programming languages, such as nesC or Java, to be able to exchange data. CDR defines standard data type with specific size and endian which have to be followed by each parties in order to guarantee seamlessly data exchanging. Table 3 shows the mapping of primitive data type between CDR version 1.3, TinyCDR and nesC native data type. Listing 1 shows an example how the data type defined in TinyCDR is used in an application.

CDR Type	TinyCDR Type	nesC Type
char	cdr_char	uint8_t
wchar	N/A	N/A
octet	cdr_octet	uint8_t
short	cdr_short	int16_t
unsigned short	cdr_ushort	uint16_t
long	cdr_long	int32_t
unsigned long	cdr_ulong	uint32_t
long long	cdr_longlong	int64_t
unsigned long long	cdr_ulonglong	uint64_t
float	cdr_float	float
double	cdr_double	double
long doublel	cdr_longdouble	double
boolean	cdr_boolean	uint8_t

Figure 3. Primitive Data Type Mapping between CDR, TinyCDR and TinyOS

In the table, TinyCDR does not support *wchar* (wide character, i.e., Unicode characters) because it is not used in WSN environment. Beside primitive data types, TinyCDR also supports CDR constructed types such as *struct*, *union* and *array*. TinyCDR serializes constructed data structure into an octet stream which is compatible with CDR octet stream. Therefore, TinyDDS applications can exchange data formatted in TinyCDR directly with DDS applications using CDR data format.

3.4. TinyGIOP

TinyGIOP defines message format use for exchanging between TinyDDS/DDS applications, based on General Inter-ORB Protocol (GIOP) version 1.3 [8]. GIOP is an abstract protocol for communicating between object request brokers (ORBs). There are several concrete implementation based on GIOP such as Internet Inter-ORB Protocol (IIOP) , an implementation of GIOP over TCP/IP, and HyperText Inter-ORB Protocol (HTIOP), an implementation of GIOP over HTTP. GIOP consists of three components; CDR, Interoperable Object Reference (IOR), and a set of message types. In TinyDDS, the CDR part of GIOP is addressed by TinyCDR while the message type is addressed by TinyGIOP. Given the limited resources of sensor nodes, IOR is not supported by TinyDDS.

TinyGIOP supports three message types; *Request*, *Reply* and *CancelRequest*. When a TinyDDS application wants to communicate with the other TinyDDS application, for example, for subscribing to a topic, it sends out *Request* message. The message will be serialized and passed to lower level, i.e. L4, or to DDS Gateway for delivering to DDS applications. *Reply* message is used for answering the request, e.g., when a TinyDDS application publish an event subscribed by another TinyDDS application, the publisher sends out *Reply* message to the subscriber. *CancelRequest* is used for withdraw request sending out earlier. Contrast with GIOP, TinyGIOP does not support object location message formats because there is no notion of object in TinyDDS, as discussed before, TinyDDS does not support GIOP's IOR.

3.5 TinyDDS L4 Adaptation Layer

To access to low level physical network, the routing protocols in OERP make use of low level physical network through a network abstract layer called TinyDDS L4 Adaptation Layer (L4AL). This L4AL utilize Bridge design pattern to separate the real low level physical network implementation from the higher level overlay network. Thus, TinyDDS can be portable among different sensor platform. In particular, L4AL provides an interface to access physical network functions such as how to get the list of neighboring nodes, how to get the link quality to each neighboring node and also how to send/receive data to/from particular nodes in the network. These functions are used by the routing protocols on the OERP layer and implemented by the Network Layer implementation. Internally, L4AL contains a set of tables that maintains the information of network, such as neighbor list and link quality, and a set of event queues. There are two types of event queues, incoming queues and outgoing queues. The events submitted from OERP for sending out to physical network is put to the end of outgoing queue while the events collected from physical network are put to the end of incoming queue, waiting to be processed by the routing protocol in OERP.

3.6. Application Development with TinyDDS

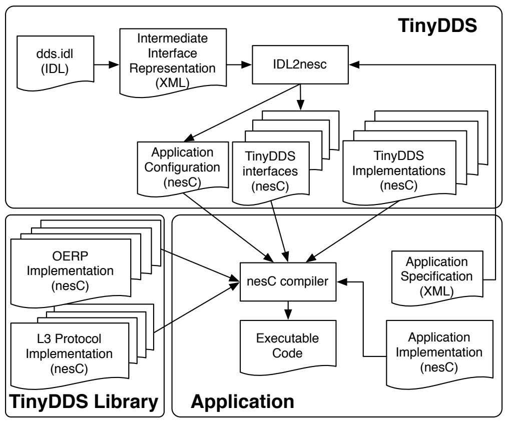
TinyDDS
dds.idl (IDL)
Intermediate Interface Representation (XML)
IDL2nesc
Application Configuration (nesC)
TinyDDS interfaces (nesC)
TinyDDS Implementations (nesC)
OERP Implementation (nesC)
L3 Protocol Implementation (nesC)
TinyDDS Library
nesC compiler
Executable Code
Application Specification (XML)
Application Implementation (nesC)
Application

Figure 4. Application Development Model

Figure 4 shows the development model of an TinyDDS application. There are three main components of the development model, TinyDDS middleware, TinyDDS Library and the application. The TinyDDS middleware comprises of two parts, the DDS interfaces definition and the TinyDDS implementation of the interfaces. The DDS interfaces definition is directly generated from the *dds.idl*, which is the official DDS interfaces definition in IDL format from OMG. The *dds.idl* is first converted into XML format. Then, *IDL2nesc* converts the DDS interfaces definition from XML format to *TinyDDS interfaces* and *Application Configuration*. The Application Configuration follows Facade design pattern [5] and describes how to connect each interfaces and implementation together. IDL2nesc also uses an *Application Specification*, written in XML, in order to generates appropriate Application Configuration, for example, Application Specification specifies which routing protocol will be used in OERP layer, then Application Configuration connects the implementation of the routing protocol into OERP interface.

The second components is the *TinyDDS Library*. TinyDDS Library consists of pluggable components, namely, application-level and middleware-level pluggable components. The application-level pluggable components provides a set of services which can be used by application, such as data aggregation and event detection. The middleware-level pluggable components provide the services inside the middleware, for example, routing protocols in OERP layer. Library developer develops these functionality in the TinyDDS Library and the TinyDDS Library can be used in any application on any hardware platforms which support TinyOS.

The third components is the application. Every application implemented on TinyDDS consists of two parts, the

Application Specification which is used by the IDL2nesc compiler and the *Application Implementation*. The Application Implementation is developed by application developer and perform a certain task such as data collection and event detection.

The nesC compiler combines the Application Configuration, TinyDDS Interfaces, TinyDDS Implementations, Application Implementations and the implementation from TinyDDS Library into target executable code.

4 DDS Gateway

Figure 2 shows that TinyDDS uses TinyGIOP to communicate with the DDS gateway in order to exchange data with DDS applications. The DDS gateway is an Java application that interact with TinyDDS running in sensor nodes through serial port using TinyOS' serial adapter Java class. The DDS gateway uses JacORB [1] and a Java implementation of DDS [2] to communicate with another DDS application. A TinyDDS/DDS bridge operates on top of DDS implementation and communicate with TinyGIOP to exchange data between TinyDDS and DDS. In particular, when a message is pushed from TinyGIOP in a sensor node to DDS gateway, the TinyDDS/DDS bridge translate message into DDS format, i.e., encapsulate with GIOP header, and send out to DDS network. This is called **downstream** message transmissions because the message is sent from source (sensor nodes) to sink (client applications). On the other hand, when the DDS gateway receives messages destinate to the sensor network from the DDS network, TinyDDS/DDS bridge translates the message into TinyDDS format, i.e., encapsulate with TinyGIOP header, and injects the message into sensor nodes through serial interface. This is called **upstream** message transmission.

5 DDS Web Clients

On the right-hand side of the Figure 2, a DDS web client is shown connected to DDS gateway. Currently, a DDS web client using Google Map is implemented as a Java servlet application running in Jetty web server. The DDS web client is able to operate on ordinary desktop computers or mobile devices such as Apple's iPhone. By using JacORB, the DDS web client can communicate with DDS gateway in order to subscribe to data published from sensor networks and show the result on the Google map. Figure 5 and 6 show examples of web interface running on a desktop computer and a iPhone respectively. In the Figures, the small dots show location of each sensor node, bright dots represent sensor node which report data.

6. Evaluation

This section evaluates TinyDDS in terms of memory footprint, power consumption and lines of application code. One time event subscription and subsequent event publications are implemented in TinyDDS and simulated on PowerTOSSIM [10] with the MICA2 power consumption model. The duration of each simulation is 120 seconds, and 25 nodes transmit sensor data to the base station every 2 seconds. TinyDDS is compared with Surge, a simple data collection application bundled in TinyOS. Similar to Surge, TinyDDS uses a spanning tree-based protocol as its OERP.

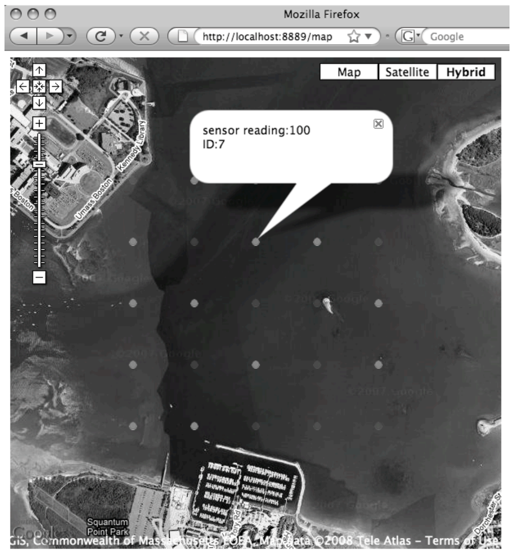

Figure 5. A DDS Web Client on a Desktop Computer

Figure 6. A DDS Web Client on an iPhone

Table 7 shows the memory footprint of TinyDDS and Surge. Without running applications, TinyDDS consumes 36.1 kB in ROM and 3.4 kB in RAM. With an application, memory footprint slightly increases to 37.6 kB in ROM and 3.4 kB in RAM. However, the difference of memory footprint is very small between TinyDDS and Surge. TinyDDS is implemented lightweight, and it can operate in resource-limited nodes such as MICA2.

Table 8 shows the average and standard deviation (SD) of power consumption by 25 nodes. Without a subscriber, TinyDDS transmits no data; thus, its power consumption remains small. In contrast, Surge always transmits data to the base station; it consumes much more power than TinyDDS. With a subscriber, TinyDDS consumes a comparable

Memory Footprint (kB)		TinyDDS	Surge
Without an Application	ROM	36.132	N/A
	RAM	3.360	N/A
With an Application	ROM	37.572	37.430
	RAM	3.394	1.929

Figure 7. Memory Footprint

amount of power compared with Surge. TinyDDS is implemented power efficient.

Power Consumption (mW)		TinyDDS	Surge
Without a Subscriber	Average	189.59	37430
	SD	61.24	76.03
With a Subscriber	Average	3900.9	3924.97
	SD	52.55	76.03

Figure 8. Power Consumption

With TinyDDS, only 60 lines of nesC code is required to implement the same application as Surge. Surge is implemented with 300 lines of nesC code. TinyDDS effectively simplifies the development of WSN applications.

7. Related Work

There exist many research and commercial implementations of the DDS specification. However, to the best of the authors' knowledge, no DDS implementations exist for WSNs. TinyDDS is the first implementation of DDS and IDL-to-nesC mapping for WSNs.

There are several research efforts that focus on the interoperability between WSNs and access networks [6, 7, 11, 12]. [6, 11, 12] propose interoperable middleware/frameworks, and [7] proposes a protocol bridging between WSNs and access networks. However, all of these related work do not provide programming language interoperability. They also do not provide protocol interoperability as TinyDDS does by introducing an interoperable session (L5) protocol.

8. Conclusion

Traditional wireless sensor networks (WSNs) often do not consider interoperability between WSNs and access networks. To address the issue, this paper investigates interoperable publish/subscribe communication in WSNs. The proposed middleware, called TinyDDS, provides two types of interoperability, programming language interoperability and protocol interoperability, by customizing standard data types, data representation and session protocol. Evaluation results show that TinyDDS simplifies the development of publish/subscribe applications and it is implemented efficient in memory footprint and power consumption.

References

[1] JacORB website, 2008. http://www.jacorb.org/.

[2] F. Allaoui, A. Yehdih, and D. Donsez. Open-source java-based OMG DDS implementation, 2005. http://www-adele.imag.fr/users/Didier.Donsez/dev/dds/readme.html.

[3] P. Boonma and J. Suzuki. Middleware support for pluggable non-functional properties in wireless sensor networks. In *Proc. of IEEE Workshop on Methodologies for Non-functional Properties in Services Computing*, 2008.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal. *Pattern-Oriented Software Architecture - A System of Patterns.* Wiley and Sons, 1996.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Professional, 1995.

[6] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin. Emstar: a software environment for developing and deploying wireless sensor networks. In *Proc. of USENIX Technical Conf.*, 2004.

[7] A. Marchiori and Q. Han. A foundation for interoperable sensor networks with internet bridging. In *Proc. of ACM Workshop on Embedded Networked Sensors*, 2008.

[8] Object Management Group. Common Object Request Broker Architecture (CORBA) specification, version 3.1; part 2: CORBA interoperability, 2007. http://www.omg.org/spec/CORBA/3.1/.

[9] Object Management Group. Data Distribution Service (DDS) for real-time systems, v1.2, 2007. http://www.omg.org/technology/documents/formal/data_distribution.htm.

[10] V. Shnayder, M. Hempstead, B. rong Chen, G. W. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proc. of ACM Int'l Conf. on Embedded Networked Sensor Systems*, 2004.

[11] J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, and M. Welsh. Hourglass: An infrastructure for connecting sensor networks and applications. Technical report, Harvard University, 2004.

[12] P. Spiess, H. Vogt, and H. Jütting. Integrating sensor networks with business processes. In *Proc. of ACM Real-World Sensor Networks Workshop*, 2006.