

# Ringling the Doorbell—Configuring Sensor Networks

**Bernard Horan, David Anderson, James “Bo” Begole, Gabriel Montenegro,  
Stéphane Perret, Randall Smith, Robert Tow, Bruno Zoppis**

Sun Microsystems Laboratories, Europe  
180, avenue de l'Europe  
38334 Saint Ismier cedex- France  
+33.4.76.18.80.00    bernard.horan@sun.com

## **ABSTRACT**

Much work on Wireless Sensor Networks (WSNs) is focused on the functionality afforded by the devices and the means by which they communicate. This demonstration focuses on the programming model and deployment mechanisms for a WSN. We present a visual environment in which the services provided by sensors can be assembled and configured to fulfil the needs of an example scenario: a visitor ringing the doorbell of someone's home.

## **Keywords**

Wireless sensor networks, transducers, service-based sensor networks.

## **INTRODUCTION**

This demonstration provides a view of the Ant eater project at Sun Microsystems Research Laboratories. This project is exploring networks of sensors and actuators (collectively ‘transducers’) from the perspective of identifying the necessary middleware required to compose, program, and deploy a network of nodes. The approach that we are taking characterises the transducers as providing services within a service-oriented architecture.

The Ant eater project grew from the emergence of new technologies in the area of wireless sensor networks, particularly in the direction of low-cost, low-power integrated devices. We developed several scenarios to help us motivate further research and explore ways in which the project could implement various programming models. These scenarios included elderly care, home automation, and industrial control. For the purposes of a self-contained demonstration, the scenario explored here is that of a “smart doorbell” (inspired by the 2001 Scientific American article “The Semantic Web” [1]).

## **SCENARIO**

You're working in the garden, listening to your radio. A courier visits your home to make a delivery and presses the button located near your front door. This causes a lamp near the button to illuminate—indicating to the courier that the button has indeed been pressed. The radio mutes the

current programme and broadcasts the sound of the doorbell. If you don't have a radio, the garden sprinkler comes on, or the water fountain spurts, or your mobile phone alerts you, or ...

The scenario demonstrates the need for an open, extensible system. This wouldn't be possible with a closed, proprietary system because there is no way the existence of your devices (or their capabilities) could be known in advance. Hence the devices must be highly usable, configurable and installable.

Security is often a crucial concern in sensor network applications, and in this simple scenario we want to ensure that you only hear your own doorbell (and not your neighbour's). “Logical” location is also an important consideration, i.e., this is my front porch light, and it is near my doorbell, etc.

Lastly, to be feasible, all the components in this network need to be an cheap and available from your local hardware/Do-It-Yourself store.

## **DEMONSTRATION**

The demonstration includes several simple physical devices with embedded networked sensors and actuators. We considered many devices that could be used to notify the resident that a visitor is at the door, such as a digital radio, garden water fountain, drapery or window control, television, bell, intercom or loudspeaker, lighting, and telephones (mobile and fixed). Using some of these devices, the demonstration will cover three steps: (1) device installation, wherein a new device is securely added to the network and its capabilities are advertised; (2) using a visual workspace to configure (or “program”) the collection of devices; and (3) their use, i.e. ringing the doorbell.

The visual workspace is a Java™ application in which a visual Java object takes the role of a physical device. The visual manifestations of the devices can be configured and the services provided by devices can be composed.

## **ARCHITECTURE**

Each physical device is represented on the workspace by a Java object acting as its proxy. A proxy provides a visual

*LEAVE BLANK THE LAST 2.5 cm (1”) OF THE LEFT  
COLUMN ON THE FIRST PAGE FOR THE  
COPYRIGHT NOTICE.*

---

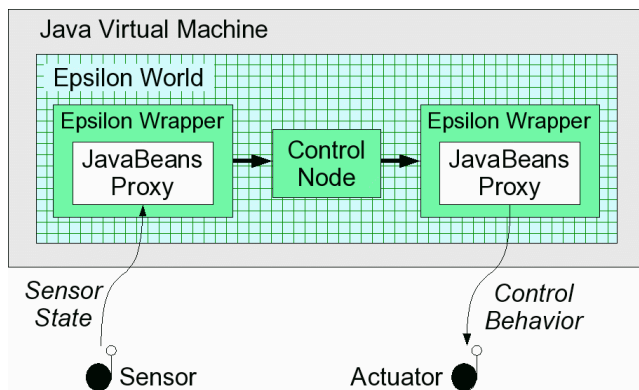
™ Java is a registered trademark of Sun Microsystems, Inc. in the United States and other countries.

representation of the state of its counterpart device: sensed value for sensors, on/off/level state for actuators, and other properties of the devices and/or services. A proxy also provides access methods to allow other objects to interact with the device.

The proxies are written in Java and follow the JavaBeans specification [3]—this allows runtime discovery and invocation of the properties, event types and functionality of a device. The JavaBeans specification is also supported by several Integrated Development Environments (IDEs) such as BeanBox, BeanBuilder, and NetBeans, allowing developers to access devices using common development tools.

The application behaviour executes in one or more Java virtual machines in which the proxies reside. The goal of this proxy architecture is to provide a uniform abstraction that can be applied to small transducers on a variety of platforms (e.g. Crossbow Motes running TinyOS [5]) as well as being appropriate for the integration of legacy devices, such as light switches and the like. Although there are some applications where the additional latency of proxy communication could introduce problems, we believe that for a large percentage of sensor applications, the benefits of the proxy architecture—such as giving developers access to their customary tools—outweigh these disadvantages.

One consequence of the proxy architecture is that interaction between the devices is indirect and thus requires extra care to ensure the overall security of the system despite the intermediaries involved. The underlying security architecture and mechanisms are understood [2], although we, as well as others, have ongoing work aimed at further optimizations and extensions.

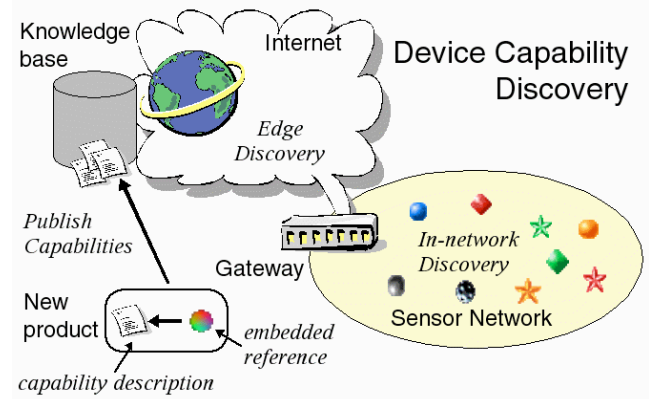


**Figure 1. Architecture of a simple control system developed in EpsilonWorld**

### Visual Workspace

The visual workspace, called *EpsilonWorld*, is a Java application that provides a wrapper for the JavaBeans proxies. Using this workspace, developers can create dataflow-style applications by employing a visual metaphor that resembles a “wiring diagram.” This facilitates the creation of applications without requiring knowledge of a general-purpose programming language. If

more complex behaviour is still required, the general-purpose language is available to create new nodes with complex behaviour (e.g., data filters, connections to web services, etc.). Figure 1 shows an example of using the device proxies within EpsilonWorld to create a simple control system.



**Figure 2. Capability registration and device discovery**

EpsilonWorld is specifically tailored to the development of transducer applications and provides sensor-application functionality beyond that provided by standard IDEs. For example, EpsilonWorld provides developers with a means of browsing available devices according to the capabilities they offer. The capabilities of a device are expressed as services in a similar fashion to that specified by the OWL-S architecture [4]. Services are classified in a taxonomy and presented as a directed graph from which the user can select a particular device that offers a service. In Figure 2 we provide an example of a new product being added to a sensor network. The new device contains an embedded reference (e.g., a Uniform Resource Identifier) to a complete description of the capabilities and services it offers.

### SUMMARY

This demonstration illustrates how to configure a wireless network of transducers to meet the needs of an example scenario (a “smart doorbell”).

### REFERENCES

1. Tim Berners-Lee, James Hendler, Ora Lassila, The Semantic Web, *Scientific American*, May 2001. (<http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&ref=sciam>)
2. M. Burnside, D. Clarke, T. Mills, A. Maywah, S. Devadas, and R. Rivest, "Proxy-Based Security Protocols in Networked Mobile Devices," *Proceedings of the 17th ACM Symposium on Applied Computing (Security Track)*, Mar 2002, pp. 265–272.
3. G. Hamilton (ed.), JavaBeans, , available at <http://java.sun.com/products/javabeans/docs/spec.html>.
4. OWL-S, <http://www.daml.org/services/owl-s/>.
5. TinyOS, <http://webs.cs.berkeley.edu/tos/>