

A Platform for Wireless Networked Transducers

BERNARD HORAN, BILL BUSH,
JOHN NOLAN, DAVID CLEAL
SUN MICROSYSTEMS LABORATORIES

April 2007

As computers, sensors, and wireless communication have become smaller, cheaper, and more sophisticated, wireless transducer platforms have become a focus of research and commercial interest. This report describes an investigation into such platforms. It presents a new taxonomy of transducer systems, describes the construction of prototypes of a new transducer device designed for ease of application development, and discusses commercialization issues.

1. Introduction

There is much evidence of the importance of transducer platforms. For example, in 2004 fewer than 2% of all microprocessors were found in conventional computers; “the vast majority are found in embedded systems which pervade many aspects of our lives acting as both sensors and actuators” [1]. Furthermore, these platforms will become more important in coming years. Business Week, for example, heralded this type of device as one of the 21 most important technologies for the 21st century [2] (reported in [3]). Others foresee opportunities for the deployment of general purpose transducer platforms combining sensing, computation, and actuation capabilities, with the resulting ability to support many types of applications [4]. These devices are envisioned to “monitor temperatures, stress, access, vibrations, sounds, dust and air particles, and a seemingly endless parade of parameters. [They] will need to operate without physical maintenance for years at a time, operating with low power and within environmental extremes.” [5]

This report describes an undertaking to study such platforms. Section Two presents an extensive characterization of the transducer platforms that have been and are currently in use. Section Three describes an experimental platform constructed at Sun Labs based on requirements derived from the characterization of Section Two. Section Four presents an overview of the potential commercial adoption of wireless transducer platforms. Section Five summarizes the project and concludes with a discussion of the future of wireless transducer technologies.

2. A Taxonomy of Transducer Platforms

After reading much of the literature on wireless transducers we came to believe that a taxonomy was needed to describe the characteristics of the different platforms in use and under research. Other authors have presented details of their hardware and software solutions and the results of their research. In this section we present a synthesis of much of that work, with a focus on contextualizing that work within the larger body of research into transducer platforms.

As the title states, this report deals with wireless networks transducers. We emphasize the term ‘transducer’ here in contrast to other authors because we believe it is important to include actuators in our scope. Most other authors focus exclusively, and often implicitly, on sensors. We believe this focus presents a skewed view of the field, a reflection of the dominance of data acquisition applications. Hence, in this report we use the term transducer to mean a sensor or an actuator, in accordance with the following definition: “a device that converts variations in a physical quantity, such as pressure or brightness, into an electrical signal, or vice versa.” (New Oxford American Dictionary, Second Edition)

Thus, in this section we present the context for networked wireless transducer platforms. Specifically, we situate them in a hierarchy consisting of the following types¹:

- transducer platforms,
- networked transducer platforms,
- wireless networked transducer platforms, and
- battery powered wireless networked transducer platforms.

For each type of platform, we describe the characteristics of that type, present some example platforms and applications that embody those characteristics, and discuss the constraints imposed by the examples. We divide the constraints into two categories:

- inherent constraints on the technology, such as power requirements, size, and cost; and
- current constraints, that is, constraints based on currently available technology.

We also use three dimensions to classify the platforms:

- the source of energy for the platform (for example, mains, battery, renewables, scavenged);
- the presence and operation of communication between and from the platform (for example, no communication, wired, wireless); and
- the use of the platform in a control system.

Furthermore, applications that use transducers can be divided into two categories, those that simply acquire data, and those that acquire data and then affect the environment. Systems in the second category almost always include a feedback loop, since changes made to the environment are subsequently sensed. These iterative, reactive transducer systems are known as control systems, and are a rich field of study.

For example, the scientific instruments on NASA’s Mars rovers acquire data and send it back to Earth, while the operation of those instruments (such as turning them on and off) is done through real-time, onboard control systems. Based on the data, mission planners on Earth may redirect the rovers’ activities; this activity is known as planning and scheduling, and is distinct from the real-time feedback of the control systems. (Note that automated planning and scheduling, which will allow more autonomy from Earth in the future, is an active area of research; this is related to, but distinct from, real-time control.)

The following sections provide more detail of the four types of platforms identified above.

1. Other authors have made similar descriptions and classifications, for example [6].

2.1. TRANSDUCER PLATFORMS

A transducer platform typically consists of three components: transducer hardware (sensors and actuators), an embedded processor, and memory [7]. The transducer hardware interacts with the physical world, converting measurements into electronic signals in the case of a sensor, and transforming electrical stimuli into physical actions in the case of an actuator. The processor manages the transducer hardware signals and maintains the calibration between the transducer hardware and the physical world. The memory contains the software executed by the processor, and may contain a history of measurements.

Sensors frequently substitute for human observers. Example situations include environments that are hostile to humans and phenomena that require measurement over a long period (and thus prohibitive human cost). Similarly, actuators are used when humans are unable to effect proper responses to conditions. Typically this is when there are high demands for speed, accuracy, and/or consistency of response.

The characteristics of a transducer platform can be broken down into those of its constituent parts: the transducer hardware, the processor, and the memory.

Transducer hardware has been around for decades [8] and is tailored to the physical environment in which it operates. Commonly used sensors include temperature and humidity sensors and sensors to monitor the state of switches. Common actuators are relays and solenoids.

In recent years transducer technology has evolved to permit more widespread deployment. In particular, micro-electromechanical systems (MEMS), based on integrated circuit technology, have been developed that are both small and inexpensive. The first major commercial MEMS sensor, the accelerometer, is used to trigger automatic airbag release; “whereas high precision piezoelectric accelerometers cost hundred of dollars, MEMS provided sufficient precision for a few dollars” [9].

The development of MEMS, along with the commoditization of computing, has enabled vendors to augment mainstream devices such as toys and cars with control systems. It is now conceivable that such systems will become ubiquitous, embedded in everyday objects.

The processors used in small transducer platforms vary enormously, ranging from 8 bit to 32 bit architectures in a variety of form factors, and the memory available, typically composed of a combination of RAM and ROM (or flash) range from less than 5 KB to more than 16 MB of RAM, and from 100 KB to greater than 4 MB of ROM.

The overall size of a transducer platform reflects the size of its parts. Those that use small MEMS-based sensors can be as small as a cubic millimeter [3], whereas those that use large industrial actuators are as large as necessary to affect the physical environment.

Cost is an important constraint, and characteristic, of transducer platforms. Many commentators foresee the development of single-chip platforms leading to significant cost reductions (for example, [10]).

One characteristic that relates to cost is reliability. High reliability can be achieved either by manufacturing more reliable, and more expensive, platforms, or by using redundancy with cheaper, less reliable platforms [10]. Reliability is a constraint imposed by the needs of the application. For example, a home automation application requires less reliability than a medical diagnostic application.

Example transducer applications include: guaranteeing the quality of a bottle of wine by proving that it never rose above a certain temperature during shipping [11]; the remote monitoring of fluid levels and control of pumps at sewage treatment plants; and remotely reading water, gas and electric meters [12].

2.2. NETWORKED TRANSDUCER PLATFORMS

The definition of networked transducer platforms is straightforward: transducer platforms that are connected via some communication mechanism to each other and/or to a non-transducer device (or devices). The non-transducer device can be a gateway, a router, or a computer acting as a server. The communication mechanism can take the form of a network, a bus, or another form of physical connectivity. The protocol of connectivity between the devices can vary from a simple two-wire mechanisms such as 1-Wire [13] or I²C [14] to sophisticated proprietary mechanisms such as BACnet [15] or LonWorks [16], or to standards such as TCP/IP.

In this section we particularly consider platforms that use wired connectivity (wireless networked devices are considered in the next section) and platforms that obtain their power from conventional power sources. In addition to mains power, we include devices that may use the same wires to provide power and connectivity, examples of which include Power over Ethernet [17] and Powerline Ethernet [18].

In this set of platforms the RAM may be used as a temporary store for data to be communicated to other devices in the network, and the embedded processor may take on the responsibility of handling the communication between the devices in the network. Additionally, the processor may perform data processing before transmission or after reception. (For example, a networked sensor connected to a camera may perform image processing before transmitting the images.)

Networked transducer platforms offer the additional beneficial characteristic of time synchronization: the connection of multiple transducer platforms provides applications with the ability to synchronize data collection and actuation. Examples of this characteristic are described in the applications presented below.

As we discussed earlier, applications for transducer platforms can in general be divided into two categories: control systems and data acquisition applications. This is specifically true for networked transducer platforms; examples are presented below.

The feedback loop in a networked control system can take two forms: local to the device itself, or involving other devices in the network. It is generally true that the complexity of a control system increases in proportion to the number of devices it contains. Examples of networked control systems include:

- Automated lighting systems. Each lighting fixture contains an actuator controlling the lamp, and sensors are distributed in the physical environment to measure light levels and temperature and to sense activity. There are additional actuators that control window blinds and curtains. Every actuator and sensor is connected to a central server containing the control logic. The actuators must operate in parallel in a synchronized fashion in response to common control logic [19].
- Automated in-vehicle climate control. Sensors are deployed outside and inside the cabin of a vehicle at various locations to ensure a representative spread of temperature readings. Additional sensors may include a solar sensor (mounted on the dashboard), focused infrared (IR) sensors to measure the skin temperature of the driver and passenger, and a sensor to measure the acceleration of the vehicle. Actuators control the ventilation and heating (including blowers and fan direction). Every actuator and sensor is connected to a central processor that collects the temperature readings and issues commands to the actuators².
- Supervisory Control and Data Acquisition (SCADA) applications. SCADA applications are used in industrial and engineering environments to monitor and control distributed activities. A SCADA application consists of multiple Remote Terminal Units (RTUs) connected to a central server. An RTU connects to physical transducer hardware such as switches, sensors, pumps, and valves and is responsible both for reading the state of these devices and for controlling them. The central server is responsible for supervisory control of the network of RTUs. The server acquires

2. See for examples [20] and [21].

data from the RTUs and issues instructions to them. However, each RTU has a degree of autonomy: it can exert independent control over the transducer hardware to which it is connected. It also contains control logic to provide immediate feedback to its actuators based on the values it reads from its sensors. The SCADA server supervises the RTUs based on the data that it has acquired from all the RTUs it manages; hence it uses its big picture to supervise the RTUs.

Conversely, examples of data acquisition applications for this set of platforms include:

- Monitoring of the infirm. Elite Care is a residential community in which each individual's home and person are monitored. There are sensors on walls, ceilings, beds, lights, and other appliances. Each resident wears an IR/RF ID tag. Information about the resident's health and well-being is captured as well as movement. The application alerts authorized parties when health and well-being parameters are exceeded, and is also used to regulate and monitor environmental conditions³. One could imagine this application being extended to monitor the therapies that a resident is receiving (for example, tablets, drips, gases) in addition to the monitoring of observable signs (for example, pulse, breathing, blood pressure, temperature, weight) [22].
- Enhanced office collaboration. Within an office setting sensors may be used to determine the presence or availability of an office occupier [23]. The data from motion and audio sensors, in addition to a telephone sensor (to determine if the handset is in use) can be combined to detect when a user is present and/or available. This information can then be used in office collaboration tools to assist with remote worker collaboration.
- High energy physics data gathering. The Stanford Linear Accelerator [24] uses a multi-layer detector that observes the result of particle collision experiments, and data detection must be synchronized between the layers. This is an example where data collection requirements are beyond human capabilities.

Other data acquisition applications fall into the category of "distributed sensing", when the precise location of a signal of interest in a monitored region is not known a priori. These applications employ multiple sensors to locate phenomena, and perform better signal detection than systems with single sensors (due to the likely proximity of one of the sensors to the phenomenon). They also provide robustness in the face of environmental obstacles [25]. Examples of distributed sensing include the following:

- Oceanographic sensing. A network of acoustic sensors (hydrophones) on the ocean bottom, known as the Sound Surveillance System (SOSUS), was deployed at strategic locations to detect and track quiet Soviet submarines during the Cold War. SOSUS is now used by the National Oceanographic and Atmospheric Administration for monitoring seismic and animal activity in the ocean [3].
- Sensor fabrics. Textiles can be used to host self-organizing networks of sensors. Chips woven into large textile surfaces, such as carpets, can be used to monitor buildings and provide directions in an emergency. The chips are able to sense temperature, vibration, pressure and motion. Each chip in the fabric communicates with its four immediate neighbours by fine, electrically conductive threads [26].

The inherent constraints of this type of platform result from their lack of mobility. They are only useful for observing or effecting static phenomena. Installations are inflexible and can thus be costly to rewire.

Current constraints for these platforms are twofold. Firstly, the size of the platforms tends to be large, especially for SCADA applications. Secondly, many existing technologies used in these platforms are proprietary, particularly those related to programming languages and networking protocols. However, in recent years there has been a move towards industry standards, such as Linux and Microsoft Windows.

3. However, this requires 30 miles of wiring and over 300 relays per house in addition to 20 control boxes and 18 networked computers.

2.3. WIRELESS NETWORKED TRANSDUCER PLATFORMS

A subset of networked transducer platforms include an extra capability: wireless (radio) communication between devices and/or between a device and a central computer. This subset is in general, however, dependent on a mains-provided energy source.

For this platform type the embedded processor may take on the additional burden of managing wireless communication protocols. These protocols are more complex than those of their tethered counterparts due to the need to deal with reduced reliability of communication and increased unpredictability of latency and bandwidth. Indeed, in some platforms wireless communication is handled by a separate processor located physically close to the radio, often on the same chip⁴.

There are several physical wireless interfaces that are available for wireless networked transducer platforms. These include IEEE 802.11 (WLAN), IEEE 802.15.1 (Bluetooth), IEEE 802.16 (WiMax), IEEE 802.20, Ultra Wide Band (UWB) and IEEE 802.15.4. The last of these, 802.15.4, is the basis for the ZigBee protocol [27]. However, it is important to distinguish the two: 802.15.4 is a specification of the basic over-the-air interface and accompanying Media Access Control (MAC) protocol (plus basic security functionality), whereas the ZigBee protocol defines higher layers of the ISO stack, such as network, security, and application software [10].

The selection of a wireless interface depends on many factors. In many cases the most significant factor is power consumption; other factors include range, reliability, and cost.

The characteristics of this type of platform are primarily determined by the chosen wireless interface. Range, throughput, latency, and jitter are all dependent on the interface chosen (in addition to environmental conditions).

The key advantages of this type of platform are simple installation and flexible communication, due to the lack of need for signal cabling. In industrial control settings, for example, wireless networks can be installed for a fraction of the cost of wired devices (one study estimated that each physical wire in a commercial workplace costs \$800.00 to install⁵), and can provide increased flexibility, with high density sensing and deployments in unsafe areas (such as inside waterways, or in high-temperature oil refineries) that may be impossible to instrument with standard wired approaches [29]. The following two examples highlight the benefits.

1) Motorola Homesight [30] provides a coordinated system of wired and wireless cameras, wireless door and window sensors, and other devices that work together to monitor a home environment. The wireless devices, provided by Xanboo [31], use Xanboo's proprietary AFM II wireless protocol, which Motorola claims can be integrated with other wireless networking protocols such as ZigBee. The power for most devices is provided by domestic mains supply, although some of the smaller devices, such as the door and window sensor, operate on AAA batteries. The devices can be placed anywhere in the home and can be moved according to the needs of the user. No cabling is required, other than that required for power. The system can be adjusted to users' needs over time, and can be deployed by a non-technical user who is not willing or able to install cabling.

2) Heating, ventilating, and air-conditioning (HVAC) installations are beginning to use wireless communications between networked devices⁶. Wireless communication provides three benefits to HVAC installers: reduced installation costs, flexibility of installation, and the ability to retrofit older buildings. The reduced installation costs enable more sensing and

4. The ChipCon CC2430 includes a RF transceiver core, 128 KB flash memory, and a high performance 8051 microcontroller core, all on the same die.

5. Elsewhere there are reports that the elimination of wiring lowers installation costs by 80% and installation time by 90% [28].

monitoring points to be installed than in a wired environment and thus increase the effectiveness and granularity of control. Installation flexibility is improved because installers are more easily able to overcome obstacles that would hamper wired installations; for example, installation problems with existing marble, concrete, or asphalt flooring can be easily overcome using wireless devices. Furthermore, the process of installation is itself less intrusive, meaning that devices can be installed during normal business hours in an office or retail store. By eliminating wires, devices can be easily moved to respond to customer requests and building modifications.

Most of the examples in the literature of this type of platform are concerned with data acquisition. This is probably due to the nature of wireless communications: increased latency, reduced reliability, and reduced bandwidth compared to wired networks. These three factors have an impact on the real-time characteristics of control system applications where timeliness (and hence latency) can be critical, but are less significant factors for applications that have no direct feedback-loop, such as data acquisition.

The inherent constraint on this type of platform is the radio. It is implausible in the foreseeable future that wireless connectivity will be as reliable as wired connectivity. And current radio technology imposes specific constraints on this type of platform. Constraints on range, latency, and bandwidth can be expected to improve over the coming years⁷, but at present these constraints mean that not all potential wireless transducer applications can be realized. Furthermore, the confusion of standards for wireless protocols create difficulties for many developers and installers, requiring commitments to technologies that may be evanescent.

Radio communication is lossy and has variable latency, which means that control applications implemented on this type of platform have to be tolerant of missing and delayed data. Currently, loss rates can be greater than 50% when multiple nodes in a network are transmitting [29]. It is doubtful that this condition can be significantly improved in the foreseeable future, although re-transmission can alleviate the problem at the expense of increased and unpredictable latency [29]. In addition, these platforms have lower bandwidth than their tethered counterparts [25].

However, some of these issues will be addressed by the development of the Zigbee standard mentioned above. This protocol sits on top of the existing IEEE 802.15.4 protocol to provide self-organization and self-healing mechanisms for networks whose nodes automatically establish and maintain connectivity among themselves [34].

There are also examples of distributed data processing using these platforms. For example, US government research projects have developed distributed algorithms for audio sensing and target tracking (such as those discussed at the Netted Sensors Community Workshop in October 2005 [76]).

2.4. BATTERY POWERED WIRELESS NETWORKED TRANSDUCER PLATFORMS

A subset of wireless networked transducer platforms are those that operate without mains power. Generally these are battery powered, but they may obtain their power from renewable energy sources such as solar cells or wind turbines. The absence of mains power allows these platforms to be completely untethered, thus enabling them to be used in applications in which the transducers themselves, the human operators, or the phenomena associated with the transducer hardware, are mobile [7].

The wireless communication protocols available to this type of platform are identical to those used by wireless transducers generally. The 802.15.4 standard is specifically targeted at small devices that have little power. It is optimized

6. For example, Advance Transformer, a division of Philips Lighting, has based a series of wireless lighting products on Ember's EmberNet hardware and software [32].

7. According to [33] wireless bandwidth increased by a factor of 25 between 1997 and 2002.

for short range (less than 90m) and low data rates. The 802.15.1 standard, popular for personal area networking (such as between mobile phones and laptops, PDAs, or printers), can be used. However, it has been shown to interfere with other wireless standards that use the same frequency, such as 802.11. A new version of the specification is intended to address this issue, as well as introducing a new mechanism to reduce the power consumption. The 802.11 standard itself can be used, but, because of its energy requirements, it is more suited to larger devices, such as laptops and PDAs.

As with the others, this platform type divides into two subtypes, those that are focused on data acquisition and those that embody control systems. The work on control systems has been quite limited [29].

The “Mote” platform dominates data acquisition research. It comes from the University of California at Berkeley (UCB) in the USA [35], [36], later commercially exploited by Crossbow Technology Inc. [37]. However, there are and have been several similar efforts elsewhere. These include the SoapBox [38], the Pushpin Computing System [39], TecO Smart-Its [40], BTNode Smart-Its [41] and the Lancaster Smart-Its [42], [43]. The devices share similar characteristics:

- program memory: between 8 and 60 KB;
- RAM: between 0.5 and 4 KB;
- non-volatile storage: between 32 and 512 KB;
- supply voltage: between 2.7 and 6 volts;
- size: all less than 0.5 cm thick, with a surface area that ranges from less than a half a square centimeter to just over two square centimeters; and
- transducer hardware: the devices support a variety of sensors, including 3D accelerometers, light sensors, magnetic sensors, temperature sensors, IR proximity sensors, and acoustic and pressure sensors. However, none of them offer any actuators other than an LED or a sounder/buzzer.

In addition, two research projects have examined much smaller platforms. The focus of the “Smart Dust” project [44], [45], [46] and the “Speckled Computing” project [47] has been to construct a device that is 1 cubic millimeter or less in size. Just like larger platforms, the device contains a sensor, a microprocessor, and memory.

2.4.1. Battery Powered Wireless Networked Data Acquisition Systems

Most research is focused on data acquisition and Wireless Sensor Networks (WSNs), epitomized by research into environmental monitoring and battlefield surveillance. In general, the research has focused on “monitoring space, monitoring things; and monitoring the interaction of things with each other and the encompassing space” [9]. Hence the focus has been sensors, radios, and low cost, resulting in simple software platforms.

Many of the data acquisition applications using this set of platforms are still in the research phase, and are dominated by research into WSNs prompted by the DARPA “Networked Embedded Software Technology” (NEST) initiative [48]. For example, Motes have been used to evaluate WSN algorithms, to perform environmental monitoring, and to track physical objects in three dimensional space.

A WSN comprises a large number of battery powered wireless networked transducer nodes, densely deployed inside the phenomenon under observation or close to it. The key feature of a WSN is that the location of the nodes need not be pre-determined, thus allowing random deployment, for example in inaccessible terrain or disaster relief operations. In consequence, WSN protocols and algorithms must possess self-organizing capabilities. Research applications in this area have focused mainly on routing algorithms for the network, examining issues such as mesh networking and multi-hop mechanisms. Another unique feature of WSNs is the cooperative effort of its nodes. Instead of sending raw data to nodes responsible for data fusion, sensor nodes use their processing abilities to locally carry out data reduction, and transmit only the required and partially processed data [49]. Several research groups have experimented with conserving power by controlling sampling rates and compressing or prioritizing data before transmission [50].

The wireless telemetry data provided by a WSN offers benefits over standalone dataloggers and wired instrumentation. A WSN also minimizes site intrusion and alterations to the environment necessitated by cable installation. In addition, it is no longer necessary for researchers to visit study areas once the sensors have been deployed, as the data can be transmitted from the study area to a remote collection point [51].

Example data acquisition applications include environmental monitoring, smart spaces, medical applications, and precision agriculture [52], [53], [25]. In the paragraphs below we briefly describe a few examples.

1) A project at UC Berkeley has demonstrated how mote networks can assess structural damage from earthquakes, making eventual re-entry of large buildings quicker, less expensive, and more effective. It showed how a hundred motes could be installed in an afternoon, with sensors collecting vibration data and measuring how building elements moved relative to each other, thus revealing the degree of damage at each location after mock earthquakes [54].

2) The Great Duck Island experiment [51] is an example of a large scale WSN collecting data from the environment, in this case, the habits and habitat of Leach's Storm Petrel. In 2002, a 43 node WSN was deployed on the island. Each node hosted up to five sensor devices monitoring the island's habitat. The experiment collected data on light levels, temperature, and relative humidity.

3) Motes have been used to help with frost control in the San Bernabe vineyard in California, USA. Thirteen motes were deployed to monitor temperatures in the 5,115 hectare property. The motes sense the temperature and transmit the data back to sprinkler controllers. The sprinklers are activated before the onset of frost to prevent damage to the grapes. The motes have replaced human monitors using handheld thermometers, which takes longer and costs more [55].

4) Wireless sensors have been used in construction. Sensors from Xsily are added to cement as it is poured to form concrete pilings. After curing, the concrete is driven into the ground using hydraulic hammers. The sensors read the characteristics of the waveform created by the impact of the hammer and reflected by the soil surrounding the piling. The reading is transmitted to a central computer and analyzed to determine the characteristics of the soil and the load-carrying capacity of the piling. This analysis is then used to determine the number of pilings necessary and thus the amount of cement required, frequently resulting in reductions of both⁸ [56].

2.4.2. Battery Powered Wireless Networked Control Systems

Research groups that focus on wireless networked transducer control systems use platforms that are much larger than those used in pure data acquisition research. Products are available from many vendors⁹ that offer a range of hardware characteristics, ranging from devices similar to a Mote to ones closer to a PDA. However, typically these devices have relatively more compute power, larger memory and more I/O connectivity than their sensor-oriented cousins. These devices are seen in autonomous robot experiments and also in domestic commercial products such as the Roomba [58]. Some researchers use a mundane and easily available platform such as an HP iPaq [59]. Note that few, if any, of these platforms have embedded transducer hardware. Even the iPaq only has audio transducers¹⁰. However, the platforms are notable in terms of I/O connectivity, offering a variety of protocols such as I²C, 1-Wire, etc.

8. The Xsily sensor comes with a 30-minute warranty that starts running with the first hammer blow.

9. For a comparison of just those devices that are able to run Java, see [57]

10. The iPaq's characteristics are significantly different from the sensor-based platforms. The smallest of the devices has 32MB RAM, 16MB ROM and a much faster processor. However, in consequence the power requirements are much higher.

Research on closed-loop applications is focused on robotics. Nonetheless, although there are many examples of battery powered transducer platforms in this field, there are few examples using wireless networks.

The Robocup challenge [60] is one example. The goal of the challenge is to develop a team of fully autonomous humanoid robots that can beat the human world champion team in football (soccer). The research effort is split between AI and robotics, with many competitors using the Sony AIBO [61] as their fundamental platform.

2.4.3. Battery Powered Wireless Networked Transducer Platform Constraints

The general constraints for this type of platform apply to both data acquisition and control systems. The constraints of wireless communication are identical to those described earlier.

The significant unique constraint for this type of platforms is low power consumption, required by limited battery power [49].

It is unlikely that current energy constraints will be greatly reduced in the short term. However, in the 1990s the introduction of Lithium Ion (Li-ion) technology saw battery capacity increase at 3-10% per year [33], [62]. New electrode materials are expected to increase the capacity of Li-ion batteries dramatically, so that by 2011 performance will be approximately twice current levels [62]. Nonetheless, unlike Moore's law, battery capacity has only doubled every 35 years during the last century.

There has been some progress in energy scavenging [63]. Recently, fuel cells have been developed that "scavenge" energy by drawing off the ambient vibration energy generated by industrial machines. Also, low levels of light can be gathered to help tiny wireless sensor devices run much longer, sometimes several years before a new or refreshed battery is required. Note that devices that scavenge power from the environment do not have the hard limit on power that battery powered devices do, but their dependence on the environment makes their operation potentially less predictable.

The amount of energy required for on-device data processing is in general much less than that required for wireless data communication. For example, [49] reports that "the energy cost of transmitting 1 KB a distance of 100m is approximately the same as for executing 3 million instructions by a 100 million instructions per second processor." One way of reducing energy requirements, then, is to communicate less by performing local computation before transmitting data.

Unfortunately, communication and power constraints fight against each other. The problems of bandwidth and latency can be overcome by increasing the energy used in wireless communication. On the other hand, energy resource constraints can be overcome by doing the opposite, and reducing the amount and frequency of data transmission. These constraints will be traded off for the foreseeable future, with a general favoring of on-platform computation and data reduction over data transmission [7].

Another constraint on these devices is security. Implementing effective security mechanisms such as cryptography, replication, and redundancy in tiny devices at an affordable cost can be extremely difficult and complex.

There are other constraints that are worth noting that are particular to the Mote [64]:

- the MAC layer does not guarantee reliable delivery;
- packet loss can be as high as 20%¹¹;

11. The consequence of this is that more energy is required to transmit the signal: guaranteed delivery would reduce the energy used because the signal would only require sending once.

- there is minimal support for concurrency control;
- there is no buffering mechanism for packet scheduling;
- there is no remote passive wake-up; and
- there is drift in software timers.

2.5. PLATFORM SUMMARY

We have described four types of transducer platforms:

1. At the most general are platforms that simply consist of transducer hardware plus a microprocessor. The microprocessor processes data received from sensors and can additionally control actuators. This type of platform is used in appliances and larger devices in the home (for example, washers and microwaves) and in industry (sensors on motors, actuators on valves).
2. Networked transducer platforms are just that: collections of transducer platforms networked together. They are used in applications that require synchrony or large data aggregation. These applications have centralized architectures.
3. Wireless networked transducer platforms are like the second type of platform, but use wireless networking. They are used in many industrial and building settings. Radio communication avoids extra cabling and provides increased flexibility. The use of wireless communication introduces problems of latency and reliability.
4. The most specialized of the platforms are battery powered wireless networked transducers. They have serious power constraints, and are often designed to be smaller than other platforms. They are almost always used in data acquisition applications.

Many of the platforms that are used for data acquisition applications are designed with an emphasis on the networking aspects of the platform, such as focusing on routing protocols that are efficient for an ad hoc network of low power devices. This includes mesh-networking, multi-hop and self-adaptive networking.

Of the battery powered wireless networked data acquisition platforms, the UCB Mote is by far the most popular. However, it suffers from problems reported in the literature. The limited storage capacity of the motes makes it difficult to implement complex algorithms [50] that could be used to assist with reducing network bandwidth requirements (and hence power requirements). In addition, the specialized version of C used on the Mote—NesC—is not easy to program with, especially when compared to higher-level languages such as Java [63]. Anecdotal evidence also suggests that the exception handling behavior of the Mote is poor as, like C, there are no explicit exception mechanisms. Furthermore, there is little to guide the application developer in debugging or profiling an application executing on the device

Configuration and deployment are important issues that most of these platforms fail to address, in particular the configuration of individual devices, the configuration of the network, and the deployment and update of software on the devices. Almost all devices use C or assembler as the development language.

We believe these devices pose many problems for software development:

1. the lack of portability of code between devices;
2. the inability to update or modify the software in the devices after deployment;
3. the problems inherent in low level C-like languages;
4. unproductive development tools that are difficult to use;
5. too many low level concerns made manifest; and
6. a lack of understanding of how hardware works on the part of most high level software developers. As a result, these devices are inaccessible to many developers.

We believe that the important factors in a battery powered wireless transducer are how it is used, deployed, programmed, and interfaced. Security and networking characteristics are secondary. (They involve packaging and computational overhead; the latter issue can be addressed by custom silicon, for example.)

Note that the Mote supports reconfiguration, but not reprogramming. It has “the ability to reprogram [itself] dynamically with the new configuration parameters such as sensitivity. This eliminates the need to download the application code on all the motes each time the configuration is modified” [64]. The configuration parameters are being downloaded, not the application itself¹². In many existing battery powered wireless transducer platforms it is impossible to download the application to the device over the air, leading to obvious deployment problems. For example, “When we deployed 70 motes on the field for the first time, it took us an hour to collect the motes and reprogram them manually” [64].

3. Sun SPOT: An Experimental Platform

Sun Labs initiated a project to build a platform that would address the problems outlined in the previous section. The resulting device is called a Sun SPOT (Small Programmable Object Technology). The goals of the project were:

The goals of the project were:

- *To construct a general purpose device that could be used both for control systems and for data acquisition applications.* We believed that other researchers’ focus on data acquisition was unwarranted and that an effective general purpose device could be constructed.
- *To construct a system that would facilitate the development of complex wireless networked transducer applications.* Analysis of prevailing market offerings and experimentation by the Anteaters team led us to believe that the development of applications with sufficient complexity to satisfy commercial needs was being hampered on current platforms by the efficacy and productivity of the available development tools.
- *To provide more memory and processing capabilities than those available on many other wireless networked transducer devices.* The development stage of an application often requires more memory and processing power than its eventual deployment. The processing and memory constraints of contemporary platforms presented obstacles to developers.
- *To support dynamic, over-the-air (OTA) deployment of software.* OTA deployment is the only practical solution to the problem of changing software rapidly in the context of widespread physical distribution of potentially hundreds or thousands of devices.

The goals led to the following requirements:

- a small battery powered platform that was capable of actuation and sensing;
- flexibility in terms of software and hardware configuration;
- powerful processing capabilities near the transducer hardware to perform signal analysis and control;
- relatively large memory;
- software libraries for supporting all aspects of wireless transducer applications (ranging from drivers for hardware to APIs for communication networks); and
- a set of easy to use and productive development tools for creating and deploying wireless networked transducer applications.

12. To address this problem, the team developing the mote have devised Maté, a virtual machine specifically for sensor networks [65]. They conclude “virtual machines are a promising way to provide protective hardware abstractions to application code in sensor networks, fulfilling the traditional role of an operating system.”

The typical battery powered WSN constraints of low power, small size, and wireless communications also applied, along with the ability to alter the hardware configuration to use alternate sensors or transducers.

3.1. COMPONENTS

The project divided itself naturally into two parts: a software platform, including a software development kit (SDK); and a hardware platform.

3.1.1. Software Platform

The main goal in choosing a software platform for the project was to overcome the software development problems identified above: the lack of code portability, the inability to dynamically modify software, the problems inherent in low level C-like languages, unproductive and hard to use development tools, too many low level concerns facing the developer, and a lack of hardware expertise among software developers.

Choosing the Java Language

Our first, highest level choice was to use the Java language.

Java eliminates or streamlines many of the low level tasks of traditional development languages such as C, increasing developer productivity by as much as four times [66]. Developers can use standard Java development tools with which they are already familiar. Integrated Development Environments (IDEs) such as NetBeans, and other supporting tools such as ant, are available to assist Java developers with initial development and later deployment.

The portability of Java means that it is easier for the developer to create software on one platform and deploy it to another. Portability also makes it simpler to migrate applications between platforms and enable developers to build new wireless sensor devices using off-the-shelf hardware components.

The use of Java also means that devices can be provisioned with their software over the air dynamically. Java was designed to be dynamically provisioned over a network (originally into a browser), and has specifically been used to provision mobile phones and other wireless devices [67].

The architecture of a JVM affords a good abstraction of low level hardware details, providing object-oriented encapsulation and information hiding mechanisms. The use of a JVM also promotes a safe execution environment by avoiding the use of pointers and providing a secure memory mechanism that can protect vital areas of a device from accidental or purposeful corruption.

Java has a high degree of acceptability within the development community and is frequently used to introduce university students to high level and object oriented programming. We identified the early adopters for the Sun SPOT as being similar to those who already develop applications in this domain: university research groups, post graduate students, and industrial research institutions. All of these groups are familiar with Java, so that the use of Java made the Sun SPOT quickly accessible to a large audience.

Choosing a version of Java: CLDC

Having chosen Java, we then had to select a specific version. The versions are distinguished based on the resources they require and the libraries of functionality they support, and include J2EE, J2SE, J2ME CDC, and J2ME CLDC.

One configuration of Java 2 Micro Edition (J2ME [68]), known as the Connected Limited Device Configuration (CLDC), was particularly suitable for this size of platform. It was designed specifically for small, resource limited devices.

Choosing a CLDC implementation

In terms of specific implementations, we considered the ubiquitous Kilobyte Virtual Machine (KVM). It was designed to operate with as little as 160KB total memory on 25MHz 16 or 32 bit RISC or CISC processors. It was written in C and is freely available for several operating systems. The KVM is the reference implementation for CLDC.

We also considered, and ultimately chose, an existing implementation developed within Sun Labs in the Squawk project [69], [70]. The Squawk VM implements a JVM in a very small footprint. The virtual machine architecture adopted by the Squawk project provides several attractive features for the Sun SPOT, compared to KVM:

- As much functionality as possible is written in Java, thus enabling the rapid development of the JVM as well as simplifying the task of porting it to other hardware platforms.
- Squawk provides the ability to execute applications directly on the CPU without an underlying operating system, saving overhead and improving performance.
- Multiple applications can be run in one Squawk JVM.
- Squawk requires an extremely small amount of RAM for the Java stack: approximately 32 KB, plus a similar amount for the C stack.
- It has a dynamic library loading mechanism and an alternate bytecode set that produces small libraries, resulting in a complete CLDC implementation that fits in about 480KB of non-volatile storage. It is nonetheless a fully functional JVM, with threads and garbage collection, and is fully compatible with Java applications developed using standard J2ME tools.
- Squawk had been in development for two years and was reasonably stable.

Continuing the “Java-all-the-way-down” philosophy of the Squawk VM, we also decided to attempt to write as much as possible of the low level hardware driver code in Java. This approach enabled rapid development of the driver code and also dynamic deployment and loading (addressing the goal of transducer hardware reconfiguration).

System-level programming features such as access to buses and bit manipulation of specific hardware registers were provided via standard Java mechanisms. In other words, all hardware access was via Java libraries and method calls, without any recourse to special language features or additional generated bytecodes. See the Appendix for an example of how LEDs can be controlled at the highest level in Java, compared to similar functionality provided by NesC.

3.1.2. Hardware Platform

We found no existing wireless transducer platform that would suit our needs. A small team of the project’s researchers constructed the Sun SPOT from commercially-available off-the-shelf components, using single-chip solutions where available and appropriate.

In order to meet the requirement above for flexible hardware, we designed hardware that was stackable via a serial connector. Transducer hardware could be attached to ancillary printed circuit boards (PCBs), thus providing an easy mechanism to modify a hardware configuration.

The hardware challenge was to design and build a low cost, low power device that provided significant processing and memory capabilities. Use of the Squawk JVM led us to a memory profile of a minimum of 128KB RAM and 512KB ROM. Furthermore, we required sufficient processing power for non-trivial data processing (for example, signal filtering algorithms, radio stacks, and sensor processing).

After intensive investigation we decided to base the hardware platform around an ARM7TDMI core that would give us sufficient processing power. This core was used by many manufacturers as the basis for their single chip solutions, allowing us flexibility in the choice of an actual product. The ARM7 also had relatively low power requirements for a device of its class. Furthermore, the ARM7 provided us an upgrade path through the ARM9 series.

Finally, we decided to use a complete 802.15.4 radio chip and communicate with it via a serial peripheral interface (SPI). This would ease real-time requirements on the main processor for the processing of radio signals, and several vendors sold these components.

3.2. PROOF OF CONCEPT

We built a software-only proof of concept based on the design choices described above. Within five days a working version of the Squawk JVM was shown running on an ARM software emulator. The proof of concept version demonstrated that the memory profile was achievable, and that the cost of porting Squawk to the ARM was minimal in the simplest approach.

3.3. INITIAL DEVELOPMENT PHASE

The initial development phase was divided into two main efforts: the design and fabrication of the hardware, including the selection of components and the layouts of the board; and the development of the software, including the porting of the Squawk JVM to the ARM7 platform and the development of tools for creating and deploying software onto that platform.

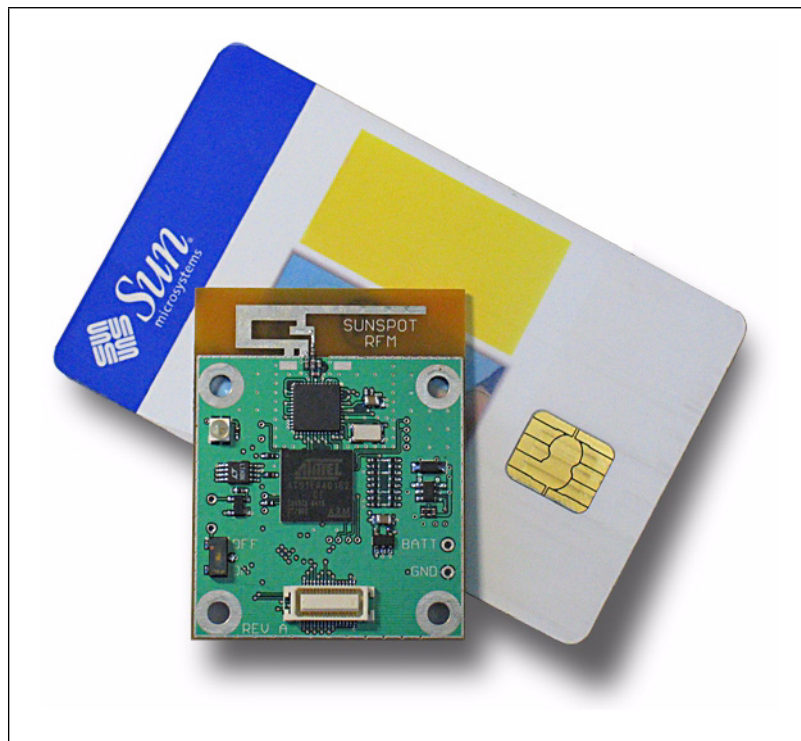


FIGURE 1. A Revision A Sun SPOT

The first hardware design for the Sun SPOT, Revision A, was a modularized layout using the following main components:

- AT91FR40162 ARM7 from Atmel. This device had a memory profile of 256KB of RAM and 2MB of ROM. The ROM was far larger than we required, but there was not at the time a similar device with the RAM size needed. The disadvantage of this device was that it required an external clock and had no hardware support for SPI interfacing.
- A silicon oscillator for the clock. We decided to use a silicon clock instead of a crystal, as this consumed much less power. We used an LTC6903 oscillator from Linear Technology as it was SPI controllable, giving us the potential to vary the main processor clock speed.
- Either the ChipCon 2420 or the Freescale MC13192 radio chip. We used the ChipCon later in the development phase. Both products provided the same functionality and were controlled via SPI interfaces, but with different protocols.
- A strip antenna for the radio device.
- A 30-pin connector on top and beneath the main Sun SPOT PCB, which enabled stacking of sensor boards and test boards.
- Power control devices allowing the device to operate from a battery supply between 1.5V and 3.2V.

Two additional pieces of hardware were built for the Revision A Sun SPOT:

- Test board. This board provided a cradle to hold a Sun SPOT. The board was used to connect a Sun SPOT to a host computer for the purposes of downloading software and testing the hardware via inspection points. Additionally, the board could be used as a wireless base station when a Sun SPOT was docked in its cradle, allowing the host computer to interrogate other Sun SPOTs communicating wirelessly with the docked Sun SPOT. The test board was connected to the host computer via a serial or USB cable.
- Sensor board. An initial demonstration sensor board contained a plethora of transducer devices, allowing testing of many types of applications. The transducers included a 3D accelerometer, a light sensor, a temperature sensor, two tri-color LEDs, two push-button switches, and eight general purpose I/O connections.

In addition, the Revision A Sun SPOT included other sophisticated features such as a Lithium-ion battery, a USB connection to a host computer, and the ability to turn off the processor (but not the rest of the board) when it was inactive.

The hardware design, implementation, and testing took place in parallel with the development of the software. Consequently, the software was developed using the Atmel ARM7 evaluation hardware with hand-made radio devices connected to the evaluation board.

To make the application developer's experience as efficient as possible, a suite of tools were developed that enabled the developer to use a standard Java IDE and deploy a fully developed application onto a SPOT docked in the test board in a single operation.

The Revision A software consisted of:

- A fully ported Squawk JVM running on the AT91FR40162. After much optimization and close collaboration with the Squawk team the memory profile fell to a minimum of 64k RAM, 380KB ROM.
- A bootloader resident on the SPOT whose functions were to upload the VM, the libraries, the applications, and the bootloader itself. The bootloader was RAM resident at start up to enable this self-upgrade facility. The bootloader also provided dynamic loading of the VM and libraries, taking advantage of the extended ROM space to enable "paging" between the two 1MB flash memory areas in ROM.
- A low level VM subset and minimal library in C. As most of the Squawk JVM is implemented in Java, only a small amount of the platform was written in C.
- A set of Java tools on the host computer for converting and uploading Java applications to a Sun SPOT when one is docked in the test board.

- A basic debug facility. The Squawk JVM at this stage did not have a fully compliant standard debugger interface (Java Debug Wire Protocol -- JDWP) implementation, and certainly not one that would run within the memory profile of the Sun SPOT. The basic debug facility provided a gdb-style command line debugging experience.
- A C implementation of the SPI protocol. As the ARM AT91FR40162 does not have hardware SPI it was necessary to implement SPI in software. An initial version was implemented in Java, but there was a bootstrap issue requiring the silicon oscillator to be configured to 64Mhz early in the boot sequence via the SPI. Consequently the SPI implementation was rewritten in C as a low level routine and used by the bootstrap as well as the JVM.

In addition to the standard J2ME libraries provided by the original Squawk implementation we also created libraries for the following features:

- 802.15.4 MAC Layer. This simple, minimal implementation was fully implemented in Java and conformed to the 802.15.4 specification [71] for wireless communications between devices.
- Simple CLDC Generic Connection Framework classes for basic application developer network interfacing (that is, socket-like connections). This simplified mechanism provided a “good enough” implementation without the complexities of a full networking layer.
- Transducer Interface Libraries. These were also completely written in Java. They provided software interfaces with the transducers and some portions of the ARM subsystems, such as Timer-Counters and GPIO. The libraries included SPI, ADC, and UART interfacing as well as access to higher level devices such as LEDs and accelerometers.

We believed it important that the tools we developed for the Sun SPOT be easily integrated with existing tools, so that the learning curve for developers was as gentle as possible. To that end, we wished to be able to provide developers with scripts that could be used with ant [72], and with a debugger compatible with the Java Debug Interface, ensuring that it could be used either from the command line or from existing IDEs.

3.3.1. Delivery and Usage

The initial development phase took place between October 2004 and February 2005, at which point we demonstrated an application development system including battery powered wireless devices. Our initial demonstrations were of radio functionality and performance tests, supplemented with several examples in which the sensors were used to control LEDs on the demonstration sensor board. The system allowed Java developers to develop and deploy basic applications onto the Sun SPOT via the test board, and use them in wireless applications. We had produced a prototype wireless transducer platform that met our requirements.

During the development and delivery of the Revision A Sun SPOT several additional requirements and refinements were generated:

- An improved 802.15.4 network layer that would include broadcast protocols, signal strength measurement and power output control.
- Introduction of networking protocols suitable for low-power wireless personal area networks (LoWPANs¹³).
- Over-the-air deployment of applications (removing the test board from the deployment mechanism).
- Java libraries for J2SE that mirror the Sun SPOT network layers that would enable single API usage across both the Sun SPOT platform and other computers. This would allow developers to create software that would execute on either a J2SE or a Squawk JVM without modification.

13.A LoWPAN is a simple low cost communication network that allows wireless connectivity in applications with limited power and relaxed throughput requirements. A LoWPAN typically includes devices that work together to connect the physical environment to real-world applications [73].

Sun SPOT: An Experimental Platform

- Various hardware bugs needing correction via layout and component changes, including removing a redundant on/off switch on the Sun SPOT and altering values of some components for better stability.
- Improvements to the test board, including a PIC for monitoring power consumption, removal of FETs for LEDs, additional test points, and simpler USB and serial connection logic.
- Design and delivery of an AA battery board (at this stage we used hand-made battery sources).
- An improved sensor board including a faster light sensor, engineering fixes for tristate components, and replacement of parts with cheaper alternatives.

In March 2005, the project moved into an ongoing iterative delivery process to “customers” external to the original development team. These customers were still within Sun Labs, but were users of the system rather than the originators and system developers.

Between March and the end of April 2005, we produced a second revision of the Sun SPOT, Revision B, to meet the

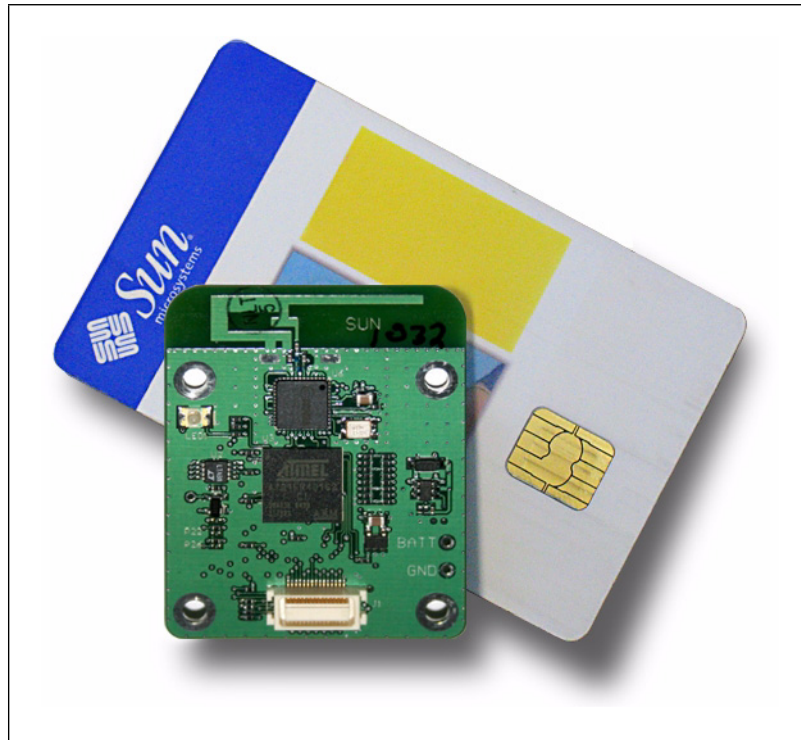


FIGURE 2. A Revision B Sun SPOT

requirements described above. Additionally, ongoing support and features were provided to a small group of Sun Labs developers who were creating demonstrations for the Sun Labs Open House at the end of April. Demonstrations were successfully shown including a remotely controlled robot, remote light sensing of a laser light and a hand gesture recognition application.

3.4. THE REVISION E SUN SPOT

Later in 2005, interest in the Sun SPOT both within and outside Sun Labs increased as a wide range of researchers and developers recognized the value to them of the unique set of features that the Sun SPOT provides. A decision was made to move cautiously toward the development of a product for external sale. The intention was to develop a product to sell in limited quantity at first, with the emphasis still on research and development. To this end, this prototype product has been developed within Sun Labs.

The Revision E Sun SPOT took advantage of the experience of the Revision B Sun SPOT and was designed to improve upon it in a number of ways:

- to take advantage of the greater power provided by the ARM9 processor series;
- to use the experience of the Revision B SunSPOT to achieve lower power consumption, in particular to enable quiescent applications to survive for months on a single battery charge;
- to increase available memory (both volatile and non-volatile);
- to achieve these benefits without increasing the physical size of the device;
- to implement a new sensor board with greater support for controlling external devices and with more sensory feedback;
- to make the development tools more complete, more robust, and better documented as befits a semi-commercial product.

The hardware design for Revision E of the SPOT uses the following main components:

- AT91RM9200 ARM9 from Atmel. This device contains the processor we required, no memory, but several useful peripherals including support for USB host and device and SPI.
- Atmega88/V microcontroller from Atmel. This device runs at very low power, and is responsible for controlling power to the main processor and other peripherals. The ARM9 processor can ask the Atmega for a wake up at some scheduled future point, thus allowing power to be removed from the ARM9, the radio and the sensor board. In deep sleep mode the device consumes less than 40 μ A.
- The ChipCon 2420 radio chip.
- A strip antenna for the radio device.
- A 30-pin connector on top and beneath the main Sun SPOT PCB, which enables stacking of sensor boards and test boards.
- Power control devices allowing the device to operate from an integrated rechargeable Li-ion battery.

The Revision E Sun SPOT also features a new demonstration sensor board similar to the Revision B Sun SPOT. Additions included a set of eight 24 bit RGB LEDs, five general purpose I/O pins and four high current output pins.

The Revision E Sun SPOT inherited the entire software environment from the Revision B SPOT. A number of extra elements were added:

- Native support for USB, allowing the Li-ion battery to recharge from the host computer;
- A JDWP implementation that allows remote debugging of Sun SPOTs from mainstream IDEs over the air;
- Support for the migration of executing code between Sun SPOTs via the Squawk JVM isolate mechanism (which allows encapsulated state and processing to be moved between Squawk JVMs)[74];
- Improvements to the Network Layer. The LoWPAN network layers have been extended to provide a primitive mesh network using the Ad hoc On Demand Distance Vector (AODV) routing protocol. Host side processes appear as simple nodes in a mesh network; with their own (fake) IEEE addresses.
- Support for the deployment of standard libraries by users, so that applications can be modified and redeployed quickly;

- Improved sleep management. The Sun SPOT puts itself into deep sleep mode whenever it detects that no processing has been required for more than two seconds. This behavior is transparent to application developers: if all application threads are in sleep or wait states and no device driver vetoes sleeping then the VM removes power to most parts of the device, including the CPU itself. Wake-up is controlled by an external low-powered timer.

3.4.1. Productization

To make the Sun SPOT into a viable product for broader use, we undertook the following enhancements:

- the addition of a translucent plastic case to hold a Sun SPOT with sensor board and battery;
- the development of manufacturing process, including automatic tests and installation of initial software;
- enhancements to documentation, API design, and usability of development tools.

The initial release of the Revision E version of the Sun SPOTs was beta tested in the fall of 2006, followed by a public offering for sale in North America in the Spring of 2007. The standard kit includes three Sun SPOTs. Two of these are equipped with batteries and sensor boards for remote operation, and one is a Sun SPOT intended for use as a base station. The Sun SPOT used as a base station remains permanently attached to a host computer and exists as a conduit for communication between host applications and the remote Sun SPOTs. The kit also includes an installation CD containing the SDK and examples. More details are available at <http://www.sunspotworld.com>.

3.4.2. Future work

Planned improvements for the Sun SPOT include the following items:

- Improvements in the Interrupt Mechanism. The original interrupt system relied upon the backward branching tests in the JVM to poll for intervening interrupts. Although this mechanism was shown to be valid and sufficient for radio communications and hardware integration, it was too unpredictable for the real-time regularity necessary for strict time sampling systems. A true interrupt mechanism using a second JVM with separate stack and memory management policies was implemented and shown to be practical. Further work is necessary to reduce the latency and to test this solution under stress conditions.
- Improvements to OS-style services. The isolate mechanism allows multiple independent applications to execute on a single SPOT. However, the current libraries for controlling the hardware do not mediate between multiple users of the hardware, and so there is no mechanism for these multiple applications to avoid treading on each others toes.
- Reductions in size and cost.

3.5. RESULTS

We have demonstrated the Sun SPOT to many potential customers and have observed it in use by the University of Essex as a fundamental component of their “Pervasive Computing and Ambient Intelligence” postgraduate course in Computer Science [75]. A Masters dissertation evaluating the use of the Sun SPOT [76] determined that the device was rated highly (a 90% approval rating) by students.

We have also received feedback from early beta testers of the Revision E Sun SPOT, who score the device highly in terms of ease of use, performance, and functionality. Overall, anecdotal evidence confirms that the Sun SPOT is easy to use and a highly productive platform for experimentation.

Based on these experiences we believe that we have met the three original goals that we identified in the previous section.

- Ease of creation: developers are able to build wireless applications in Java that use the simple demonstration sensor board for IO and that use the radio to communicate between Sun SPOT devices. (For example, in Section 7 on page 25 we briefly compare the code for the Sun SPOT against that for a Mote.) Developers are also able to integrate new hardware by reusing existing libraries and developing their own for their particular requirements¹⁴. Furthermore, the ease of use and productivity of the Sun SPOT is achieved by integration with existing development tools.
- More memory and processing power: the Sun SPOT is a mid-level device that provides significant capabilities to promote exploratory programming and enables more on-device computation to reduce network traffic.
- Over-the-air deployment: the Sun SPOT provides developers with two mechanisms for OTA deployment. The first, more conventional mechanism enables a developer to deploy a Java application to a Sun SPOT wirelessly via a base station connected to a computer. The second, more experimental mechanism involves the use of Squawk Isolates mentioned above. When using this mechanism, a developer suspends an application that is executing on a Sun SPOT, then transfers its state and behavior to another Sun SPOT, where the application is resumed.

The Sun SPOT addresses common shortcomings of battery powered WSN platforms identified in section two in the following ways.

inability to modify or update behavior easily	various mechanisms whereby behavior may be changed dynamically
low level languages	the Java language
manifestation of too many low level concerns	timing, threading, and other low level concerns are abstracted by the Java language
requirement that software developers understand low level hardware	software libraries provide abstractions to all of the hardware platform

Although not cited as a major concern in the applications described in section two, some control systems require real-time, or close to real-time, behavior. The current Java implementation available for the Sun SPOT was not designed to provide real-time behavior. However, other Java implementations such as Real Time Java [77] or Safety Critical Java [78] could be implemented on the current hardware to address this requirement¹⁵.

4. The Commercialization of Wireless Sensor Systems

This section discusses the major issues in commercializing wireless sensor networks and systems. Note that the discussion is focused on *sensor* networks and systems. Actuators play a role in a few applications (specifically HVAC and building automation), but they are an afterthought (or not a thought) in wireless network engineering and deployment. Wireless sensor platforms have been evolving technically for some time. The challenge now is to achieve widespread deployment, which can only happen through commercialization.

Successful commercialization in general consists of finding a large group of customers and selling them a useful product that will satisfy their needs for some time. Key to making this happen for WSNs are:

14. The University of Essex course is using external actuators and sensors via the I²C protocol.

15. Real-time Java is based on the Connected Device Configuration of J2ME, whereas Safety Critical Java is based on the CLDC.

1. identifying people (or organizations) with a need for a wireless sensor product,
2. determining which of those people are willing and able to spend the necessary money,
3. selling them the right sensor systems, and
4. supporting the installation and ongoing use of the systems.

Much current WSN rhetoric is visionary in tone, appropriate for getting research or business funding, but largely irrelevant or unhelpful in recruiting potential customers. And much WSN work is aimed at exploring product ideas or enhancing system properties, rather than meeting direct customer needs.

The remainder of this section presents introductory material on WSN markets, user needs, and resulting issues. Much of the material comes from the Netted Sensors Community Workshop, October 2005 [79]. More extensive market analysis is available for a fee from various market research companies (such as www.onworld.com and www.abiresearch.com).

4.1. MARKETS

David Culler of UC Berkeley (via Mark Goodman of Crossbow) has identified three meta-classes of WSN applications, with concomitant examples [*Crossbow Technology* in 79]:

- monitoring things and spaces: agriculture and habitat, HVAC, and condition based monitoring of structures;
- tagging and tracking: smart RFID; and
- ubiquitous computing: health care, context aware and non-verbal computing, and smart furniture.

Monitoring is the most evolutionary class of applications. It has the potential to provide the most immediate and obvious value. Tagging and tracking can be seen as the next generation of RFID. Organizations will generally need substantial experience with the current generation before the next generation will be widely deployed. Ubiquitous computing is still nascent from the point of view of applications and customer demand.

These meta-classes can be related to specific markets. From most to least ready for commercialization, and including a rough estimate of market size, there are (according to Mark Goodman):

- environmental and agriculture monitoring (small);
- industrial monitoring (medium);
- defense and security (small; both monitoring and tracking);
- building automation (large; both monitoring and tracking);
- asset tracking (very large);
- automotive (medium; both monitoring and ubiquitous computing); and
- consumer (very large; all classes).

To demonstrate the utility and value of WSNs, commercialization efforts should be focused on the most ready markets. In fact, Intel, which has a commercialization effort under way, is exploring deployments in manufacturing, distribution, retail, construction, agriculture, environment, and health and life sciences.

4.2. CUSTOMER REQUIREMENTS

Kris Pister of UC Berkeley and Dust Networks has identified some specific WSN customer requirements (first and foremost, reliability in general) [*From Smart Dust to Reliable Networks* in 79]:

- hardware reliability: resistance to temperature extremes, humidity, shock, and aging;
- software reliability: replace the problematic TinyOS;
- RF reliability in the face of interference and variability;
- low installation and ownership costs: greater than five year battery life, no network configuration, no network management; and
- no customer need for expertise in embedded software, RF, or networking.

Ralph Kling, through Intel's prototype deployments, believes that key issues in commercialization revolve around adapting WSNs to the customer's specific situation, specifically dealing with packaging requirements; device installation and customization; proper data formats; and interaction with existing software and user interfaces.

He notes that applications inside structures are trickier to deploy because of RF issues related to FCC regulation and interference. He also notes that battery life is a simple maintenance issue, along with others; in an industrial or public sector deployment other maintenance issues can be at least as significant.

He believes that the important issues affecting adoption are:

- packaging;
- system integration;
- sensor I/O integration; and
- security, for health and defense applications.

He also says that successful deployment requires crucial domain specific expertise with respect to the deployment environment, specifically involving packaging, power, and RF issues. He believes that successful deployments will involve partnerships with organizations and individuals who are not WSN experts but who are experts in the deployment environment.

4.3. COMMERCIALIZATION ISSUES

The most important factor in commercializing WSNs is converting them from a technology to a solution. Geoffrey Moore addresses this issue in his book "Crossing the Chasm" [80]. The chasm in the title is the distance between early deployments of a technology and adoption by a large body of users. Characteristics that attract early adopters—new, exciting, relatively untried, publicity worthy—actually repel most users, who want stability, minimal risk, and no surprises. Most factory managers, for example, don't want to spend their time learning and debugging a new sensor technology, and most building managers don't want to learn and debug a new HVAC or lighting technology.

Widespread adoption of a new technology must fundamentally not involve much (perceived) risk, and must have obvious benefits. And note that adoption is not just about technology, but is at least as much about the supplying organizations. A factory or building manager will want reasonable assurances that the supplier will be around for a substantial fraction of the equipment's reasonable life expectancy. Thus the key to WSN commercialization is getting enough real world experience to be able to offer low risk, high benefit solutions from solid suppliers. This is especially true for most of the WSN markets listed above. Industrial monitoring, building automation, retail, and automotive are all relatively large, mature, and risk-averse industries. The outlier is the defense market. The Department of Defense sponsored much of the early WSN research, and has a vision of a new, sensor-rich battlefield. But they are far from the heart of the overall market.

Also, other than defense, perhaps environmental monitoring, and speculative ubiquitous computing, WSN technology is in the near term largely about cost savings rather than paradigm shifting new applications. Deployment decisions will be focused on return on investment.

4.4. SUMMARY

One can identify four types of activity leading to the successful commercialization of WSNs: basic platform work, the development of experimental applications, trial deployments, and real deployments with paying customers. Most WSN activity to date has been of the first three types.

The technical barriers to entry in the WSN platform market are not high. The basic technology is fundamentally a commodity platform. Successful competitors will probably be differentiated by knowledge of the customer's business and expertise in the customer's deployment environment.

We believe the key to success (large scale deployments) in most potential WSN markets will be in making WSNs non-disruptive. As Geoff Moore points out in *Crossing the Chasm*, disruptive technology gets funding and PR, but non-disruptive technology gets customers. In most markets the customers will be large organizations. These customers will generally want low risk solutions, obvious advantages in return on investment, and long term customer support.

5. Conclusions

This report presented an overview of transducer platforms with a focus on wireless sensor networks, identified their shortcomings, and described a project to produce an alternative platform that overcome these shortcomings. It also presented an overview of the commercial activity necessary to make such a platform widely deployed.

People talk about sensor platforms imprecisely, meaning everything from any system with sensors to specific battery powered wireless sensor devices using particular networking protocols. This report presents a concentric taxonomy: transducer devices, networked transducer devices, wirelessly networked transducer devices, and battery powered wirelessly networked transducer devices. The taxonomy is significant because each type of device has very different characteristics, and generalizations about one type of device may very well not apply to another.

The Sun SPOT device aims to bring new ease of development to decentralized, multipurpose, battery powered, wireless transducer applications. The key enabler of ease of development is the Java platform. Java is already in widespread use in the software community and is used on hardware platforms ranging from large-scale enterprise servers to mobile phones. The benefits of Java are threefold:

- As a high level language it provides features of good software engineering practice such as modularity, late binding, and polymorphism.
- The version of Java (CLDC) that is deployed on the Sun SPOT hardware is well defined, having been through the Java Community Process (JSR 139).
- Java can perform over-the-air provisioning. This capability is already employed in the mobile phone industry to provide mobile phone users with the ability to download new Java applications, such as games, to their handsets. This capability gives the Sun SPOT flexibility and agility not seen in other wireless transducer platforms.

The most important goal in commercializing wireless networked transducers (necessary for their widespread deployment) will be converting them from a technology to a solution. Solutions in the near term will largely be about cost savings rather than paradigm shifting new applications. Deployment decisions will be focused on return on investment. Furthermore, widespread adoption of a new technology must fundamentally not involve much perceived risk, and must have obvious benefits. Thus the key to commercialization will be getting real world experience with transducer systems and offering low risk, high benefit solutions from solid suppliers.

The Sun SPOT design, with its emphasis on modern software engineering practice, flexible application deployment, and reasonably robust computational power, can perhaps provide some technical guidance on the road to commercialization.

6. Acknowledgements

We thank our colleagues Ron Goldman, Doug Simon and Mario Wolczko for their constructive reviews of this report. During its lifetime this project has been known as “Anteater” (2003—2004), “SunSpecks” (2004—2005), and “Sun SPOT” (2005—present). We thank present and past members of the project for their significant contributions to the ideas presented in this document: Robert Alkire, David Anderson, Bo Begole, Samita Chakrabarti, Cristina Cifuentes, John Daniels, Alex Garthwaite, Ron Goldman, Vipul Gupta, Roger Meike, Gabriel Montengro, Lisa Pavey, Stephane Perret, Arshan Poursohi, Adin Scannell, Nik Shaylor, David Simmons, Doug Simon, Randall Smith, Rob Tow, Stephen Uhler, Christian Wagner, Derek White.

7. Appendix: A comparison of Sun SPOT and NesC program complexity

This appendix presents the comparative complexity of programs written for the Sun SPOT platform and the NesC platform. A real example is beyond the scope of this appendix. In this example we consider a simple program to blink an LED on the device.

An example implementation of an LED blinker was obtained from the NesC project's website. It involves two source code components, shown in listings 1 and 2. The simplest possible implementation for the Sun SPOT¹⁶ is shown in listing 3.

The Sun SPOT listing is clearly simpler. However, it is only fair to point out that the NesC implementation drives the timing from a processor timer counter, rather than from the `Thread.sleep()` mechanism built into Java, and is therefore likely to blink at more precise intervals. While this is unlikely to matter for a simple LED blinker, accurate timing can be critical for real world problems. Listing 4 shows an alternative Sun SPOT listing in which an LED is blinked according to triggers from a timer counter. While a little more complex, it also provides a good illustration of the ease of manipulating the hardware from Java.

As well as supporting simpler code, the Sun SPOT is also more approachable because it only uses a standard Java programming model, whereas NesC adds its own specific constructs to the normal C programming model. This makes the learning curve for novices using NesC substantially steeper. Even desktop developers without hardware experience will find themselves productive with the Sun SPOT almost immediately, using their favorite IDE to browse the SPOT libraries, and applying directly their previous Java experience to dealing with the thread library and other standard Java components.

Listing 1. *NesC blinker listing one*

```
/*
 * Copyright (c) 2000-2002 The Regents of the University of California.
 * and Copyright (c) 2002 Intel Corporation
 * All rights reserved.
 */
configuration Blink {
}
implementation {
  components Main, BlinkM, ClockC, LedsC;
  Main.StdControl -> BlinkM.StdControl;
  BlinkM.Clock -> ClockC;
  BlinkM.Leds -> LedsC;
}
```

16.The listing shown is for the Sun SPOT libraries towards the end of the 2005. The current implementation has slightly different boilerplate due to the recent adoption of the MIDlet standard, but the central example would be much the same.

Listing 2. *NesC blinker listing two*

```
/*
 * Copyright (c) 2000-2002 The Regents of the University of California.
 * and Copyright (c) 2002 Intel Corporation
 * All rights reserved.
 *
 * Implementation for Blink application. Toggle the red LED when the
 * clock fires.
 */
module BlinkM {
  provides interface StdControl;
  uses interface Clock;
  uses interface Leds;
}
implementation {
  bool state; /* the state of the red LED (on or off) */

  /* Initialize the component.
   * @return Always returns SUCCESS */
  command result_t StdControl.init() {
    state = FALSE;
    call Leds.init();
    return SUCCESS;
  }

  /* Start things up. This just sets the rate for the clock component.
   * @return Always returns SUCCESS */
  command result_t StdControl.start() {
    return call Clock.setRate(TOS_I1PS, TOS_S1PS);
  }

  /* Halt execution of the application.
   * This just disables the clock component.
   * @return Always returns SUCCESS */
  command result_t StdControl.stop() {
    return call Clock.setRate(TOS_I0PS, TOS_S0PS);
  }

  /* Toggle the red LED in response to the Clock.fire event.
   * @return Always returns SUCCESS */
  event result_t Clock.fire() {
    state = !state;
    if (state) {
      call Leds.redOn();
    } else {
      call Leds.redOff();
    }

    return SUCCESS;
  }
}
```

Listing 3. *Simple blinker on Sun SPOT*

```
/*
 * Copyright 2004 Sun Microsystems, Inc. All Rights Reserved.
 *
 * This software is the proprietary information of Sun Microsystems, Inc.
 * Use is subject to license terms.
 *
 * This is a part of the Squawk JVM.
 */
package squawk.application;

import com.sun.squawk.peripheral.ILED;

public class Startup {
    public static void main(String[] args) throws IOException {
        boolean ledState = false;
        ILED led = Spot.getInstance().getRedLed();
        while(true) {
            led.setOn(ledState = !ledState);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) { };
        }
    }
}
```

Listing 4. *Timercounter-driven blinker on Sun SPOT*

```
/*
 * Copyright 2004 Sun Microsystems, Inc. All Rights Reserved.
 *
 * This software is the proprietary information of Sun Microsystems, Inc.
 * Use is subject to license terms.
 *
 * This is a part of the Squawk JVM.
 */
package squawk.application;

import com.sun.squawk.peripheral.ILed;
import com.sun.squawk.peripheral.spot.*;

public class Startup implements TimerCounterBits {
    public static void main(String[] args) throws IOException {
        boolean ledState = false;
        ILED led = Spot.getInstance().getRedLed();
        IAT91_TC timer = Spot.getInstance().getAT91_TC(0); // Get a Timer Counter
        int cnt = 1000 * 1872; // number of clock counts for 1000 msecs
        timer.configure(TC_CAPT | TC_CPCTRG | TC_CLKS_MCK32); // enable RC compare
        timer.setRegC(cnt);
        timer.enableAndReset();
        while(true) {
            timer.enableIrq(TC_CPCS); // Enable RC Compare interrupt 2
            timer.waitForIrq(); // Wait for interrupt
            timer.status(); // Clear interrupt pending flag
            led.setOn(ledState = !ledState);
        }
    }
}
```

8. References

1. *Runes Newsletter*, 2 September 2005.
2. *Business Week*, pp. 78-167, Aug. 30, 1999.
3. Chong, C. and Kumar, S. P.; "Sensor networks: Evolution, opportunities, and challenges," *Proceedings of The IEEE*, vol. 91, no. 8, pp. 1247--1256, Aug. 2003.
4. Estrin, D., Culler, D., Pister, K. and Sukhatme, G.; "Connecting the physical world with pervasive networks," *IEEE Pervasive Computing*, 1, January 2002.
5. Couper, C. and Murphy, M.; "Emerging wireless: Who's on first?," *Communication News*, December 2004.
6. *Runes D3.1 requirements analysis report*, Feb. 2005.
7. Tilak, S., Abu-Ghazaleh, N. and Heinzelman, W.; "A Taxonomy of Wireless Microsensor Network Models," *ACM Mobile Computing and Communications Review (MC2R)*, 2002.
8. Allan, R.; "Wireless Sensors Land Anywhere And Everywhere," *Electronic Design*, 21st July 2005.
9. Culler, D., Estrin, D., Srivastava, M.; "Guest Editors' Introduction: Overview of Sensor Networks," *IEEE Computer*, vol. 37, no. 8, pp. 41-49, August, 2004.
10. *Runes D1.1 Review Document: Existing Architectures and Components*, December 2004.
11. Takahashi, D.; "Little sensors, big bucks," *Electronic Business*, January 2005.
12. David, M.; "Electronica Looks Ahead To MEMS And Wireless Sensors," *Electronic Design*, 29th November 2004.
13. <http://www.maxim-ic.com/1-Wire.cfm>.
14. *The I²C-bus Specification, Version 2.1, January 2000*, <http://www.semiconductors.philips.com/acrobat/literature/9398/39340011.pdf>.
15. <http://www.bacnet.org/>.
16. <http://www.echelon.com/>.
17. *IEEE Std 802.3af™—2003*, <http://standards.ieee.org/getieee802/download/802.3af-2003.pdf>.
18. <http://www.homeplug.org/>.
19. http://lighting.lbl.gov/IBECs/it_overview.html
20. Schweber, B.; "Extra sensor(y) perception: automotive sensors," *EDN*, 30th September 2004, <http://www.edn.com/article/CA454659.html>
21. Pearce, J.; "Saab Automatic Climate Control," *Swedish Wrench*, http://www.swedishwrench.com/saab_ACC_part_1.htm
22. Ornstein, B.; "CARE Technology Smart Home System for the Elderly," *NIST Pervasive Computing 2001*, May 2001, Gaithersburg, Maryland.
23. Begole, J.; "Lilsys: Sensing Unavailability," Technical Note in *Proceedings of CSCW 2004*, Nov. 2004.
24. <http://www.slac.stanford.edu/>
25. Estrin, D., Girod, L., Pottie, G. and Srivastava, M.; "Instrumenting the world with wireless sensor networks," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, (Salt Lake City, UT), pp. 2033.
26. Blau, J.; "Is that a Computer Chip in your Carpet?," *PCWorld* May 05, 2003.
27. <http://www.zigbee.org>.
28. Fahrion, M.; "What Venture Capitalists Won't Tell You About Wireless", *The Industrial Wireless Book*, <http://wireless.industrial-networking.com/articles/articledisplay.asp?id=234>.
29. Deshpande, A., Guestrin, C., Madden, S.; "Resource-Aware Wireless Sensor-Actuator Networks," *IEEE Data Engineering Bulletin*, 28(1), pp.40-47, 2005.
30. http://broadband.motorola.com/consumers/home_monitoring.asp

References

31. <http://www.xanboo.com>
32. Lecklider, T.; "Wireless Sensor Networks," *Evaluation Engineering*, July 2004.
33. Feng, J., Koushanfar, F. and Potkonjak, M.; "System-Architectures for Sensor Networks: Issues, Alternatives, and Directions," *2002 IEEE International Conference on Computer Design (ICCD'02)*, special session on Sensor Networks, Freiburg, Germany, Step 2002. Pg.112-121.
34. Furrer, S.: "Innovation Matters—Wireless sensor networking," <http://domino.research.ibm.com/comm/research.nsf/pages/r.communications.innovation2.html>.
35. Warneke, B., Last, L. M., Liebowitz, B. and Pister, K.S.J.; "Smart Dust: Communicating with a Cubic-Millimeter Computer," *IEEE Computer*, January 2001, pp. 44-51.
36. Warneke, B., Atwood, B. and Pister, K.S.J.; "Smart Dust Mote Forerunners," *Proceedings of the Fourteenth Annual International Conference on Microelectromechanical Systems (MEMS 2001)*, Interlaken, Switzerland, January 21-25, 2001, pp. 357-360.
37. <http://www.xbow.com>
38. Tuulari, E. and Ylisaukko-oja, A.; "SoapBox: A Platform for Ubiquitous Computing Research and Applications," *Lecture Notes in Computer Science 2414: Pervasive Computing*, Zürich, Switzerland, August 26-28, 2002. F. Mattern and M. Naghshineh. (eds.). Springer (2002), pp.125 - 138
39. <http://web.media.mit.edu/~lifton/Pushpin/hardware.html>
40. Beigl, M., Zimmer, T., Krohn, A., Decker, C. and Robinson, P.; "Smart-Its—Communication and Sensing Technology for UbiComp Environments," Technical Report ISSN 1432-7864 2003/2, http://www.teco.edu/%7Emichael/publication/tech_r1.pdf.
41. <http://www.inf.ethz.ch/vs/res/proj/smartits/btnode.html>
42. Schmidt, A.; "Ubiquitous Computing—Computing in Context," PhD Thesis, Computing Department, Lancaster University, UK, November 2002.
43. <http://ubicomp.lancs.ac.uk/twiki>
44. <http://robotics.eecs.berkeley.edu/~pister/SmartDust/>
45. Doherty, L., Warneke, B.A., Boser, B.E. and Pister, K.S.J.; "Energy and Performance Considerations for Smart Dust," *International Journal of Parallel Distributed Systems and Networks*, vol. 4, no. 3, 2001, pp. 121-133.
46. Kahn, J. M., Katz, R. H. and Pister, K. S. J.; "Next century challenges: Mobile networking for 'smart dust'," in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Network Monitoring (MobiCom 99)*, pages 271--278, Seattle, Washington, Aug. 1999.
47. <http://www.specknet.org/>
48. http://www.darpa.mil/ixo/nest_vision.asp.
49. Akyildiz, I. F., Su, W., Sankarasubramaniam, Y. and Cayirci, E.; "Wireless Sensor Networks: a survey," *Computer Networks*, Volume 38, Issue 4, 15 March 2002, Pages 393-422.
50. Martinez, K., Hart, J.K. and Ong, R.; "Environmental Sensor Networks," *Computer*, vol. 37, no. 8, pp. 50-56, August, 2004.
51. Szewczyk, R., Mainwaring, A., Polastre, J., Anderson, J. and Culler, D.; "An analysis of a large scale habitat monitoring application," in *SenSys'04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, ACM Press, 2004, pp. 214-226.
52. Estrin, D., Govindan, R. and Heidemann, J. (eds); "Special Issue on Embedding the Internet," *Communications of the ACM*, vol. 43, no. 5, May 2000.
53. Badrinath, B.R., Srivastava, M., Mills, K., Scholtz, J. and Sollins, K. (eds); "Special Issue on Smart Spaces and Environments," *IEEE Personal Communications*, Oct. 2000.
54. "Instrumenting the World," http://www.intel.com/research/exploratory/instrument_world.htm.

References

55. "Transforming BP Processes Using Sensory Networks," August 2003, BP, http://www.worldinternetcenter.com/Other_Events/Working_Groups/2003/sibf/Transforming%20BP%20Processes%20Using%20Sensory%20Networks.pdf.
56. Cox, J.; "Wireless sensor networks grabbing greater attention," *Network World*, 27th September 2004.
57. <http://jstik.systronix.com/compare.htm>
58. <http://www.irobot.com>
59. McClain, J. T., Wimpey, B. J., Barnhard, D. H. and Potter, W.D.; "Distributed robotic target acquisition using Bluetooth communication," *ACM Southeast Regional Conference 2004*: 291-296
60. <http://www.robocup.org/>
61. <http://www.sony.net/Products/aibo/>
62. Bradbury, D.; "Seize the power", *Guardian newspaper*, 5 January 2006.
63. Hemingway, B., Brunette, W., Anderl, T., and Borriello, G.; "The Flock: Mote Sensors Sing in Undergraduate Curriculum," *IEEE Computer*, August 2004, 72-78.
64. He, T., Krishnamurthy, S., Stankovic, J., Abdelzaher, T., Luo, L., Stoleru, R., Yan, T., Gu, L., Hui, J. and Krogh, B.; "An energy-efficient surveillance system using wireless sensor networks," In *2nd International Conference on Mobile Systems, Applications, and Services (MobiSys04)*, Boston, MA, June 2004.
65. Levis, P., Culler, D.; "MatE: A Tiny Virtual Machine for Sensor Networks," *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.
66. Wells, R.; "Java offers Increased Productivity," <http://www.wellscs.com/robert/java/productivity.htm>
67. <http://java.sun.com/products/cldc/>
68. <http://java.sun.com/javame/index.jsp>
69. Shaylor, N., Simon, D. N., and Bush, W. R.; "A java virtual machine architecture for very small devices," in *Proceedings of the 2003 ACM SIGPLAN Conference on Language, Compiler, and Tool For Embedded Systems* (San Diego, California, USA, June 11 - 13, 2003). LCTES '03. ACM Press, New York, NY, 34-41.
70. Simon, D., Cifuentes, C., Cleal, D., Daniels, J., and White, D.; "Java™ on the bare metal of wireless sensor devices: the squawk Java virtual machine," in *Proceedings of the 2nd international Conference on Virtual Execution Environments* (Ottawa, Ontario, Canada, June 14 - 16, 2006). VEE '06. ACM Press, New York, NY, 78-88.
71. "Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)," <http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf>.
72. <http://ant.apache.org/>
73. Kushalnagar, N., Montenegro, G.; "6LoWPAN: Overview, Assumptions, Problem Statement and Goals," <http://www.ietf.org/internet-drafts/draft-ietf-6lowpan-problem-01.txt>
74. <http://jcp.org/en/jsr/detail?id=121>
75. <http://www2.essex.ac.uk/courses/result.asp?coursecode=CC464&level=G&period=SP&yearofcourse=06>
76. Wagner, C., "Design and Implementation of a Java Based Toolkit For Wireless Based Pervasive Computing," CC406 Project Report, University of Essex Department of Computer Science, 2005.
77. <https://rtsj.dev.java.net/>
78. <http://www.jcp.org/en/jsr/detail?id=302>
79. "Netted Sensors Community Workshop," <http://www.twistedtiger.com/nettedsensors/agenda/>.
80. Moore, G.A.; *Crossing the Chasm*, Collins Business Essentials, 2002.

9. About the Authors

Bernard Horan is a senior staff engineer at Sun Microsystems Laboratories, where he is currently a member of the Advanced Search Technologies team. Bernard was one of the original members of the Anteatr project involved in the research of existing transducer platforms. He was also responsible for the development of materials to support the Pervasive Computing course at the University of Essex.
<http://research.sun.com/people/mybio.php?uid=37721>.

Bill Bush was the principal investigator of the Squawk project. He has worked on or led various other Java projects for small platforms, including the KVM, the Secure KVM, the next generation Java Card, Java-based enterprise clients on small devices, and Safety Critical Java. He began investigating the use of Java on sensor devices in 2003 and presented preliminary results at the Sun Sensor Summit in February of 2004. He is currently developing tools that facilitate the production of high assurance software.

John Nolan was the principal investigator for the Sun Specks and Sun SPOT projects until September 2005. John created the initial prototypes of the Sun SPOT and lead the team through the first two releases. Currently, John is a partner in a consultancy specializing in ensuring its clients deliver systems rapidly and in a sustainable manner. John is an ACM Distinguished Engineer.

David Cleal is an independent developer, currently working on the SPOT project with Sun Microsystems Laboratories. He is currently focused on agile development with dispersed and distributed development teams.

10. Trademarks

Sun, Sun Microsystems, the Sun Logo, Java, J2ME, J2SE, PersonalJava, JDK, Java Card, iPlanet, and Java Community Process are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.