Prototyping Wireless Sensor Network Applications with BTnodes

Jan Beutel[1], Oliver Kasten[2], Friedemann Mattern[2], Kay Römer[2], Frank Siegemund[2], and Lothar Thiele[1]

[1] Computer Engineering and Networks Laboratory
Department of Information Technology and Electrical Engineering
ETH Zurich, Switzerland
{beutel, thiele}@tik.ee.ethz.ch
[2] Institute for Pervasive Computing
Department of Computer Science
ETH Zurich, Switzerland
{kasten, mattern, roemer, siegemun}@inf.ethz.ch

Abstract. We present a hardware and software platform for rapid prototyping of augmented sensor network systems, which may be temporarily connected to a backend infrastructure for data storage and user interaction, and which may also make use of actuators or devices with rich computing resources that perform complex signal processing tasks. The use of Bluetooth as the wireless networking technology provides us with a rich palette of Bluetooth-enabled commodity devices, which can be used as actuators, infrastructure gateways, or user interfaces. Our platform consists of a Bluetooth-based sensor node hardware (the BTnode), a portable operating system component, and a set of system services. This paper gives a detailed motivation of our platform and a description of the platform components. Though using Bluetooth in wireless sensor networks may seem counter-intuitive at first, we argue that the BTnode platform is indeed well suited for prototyping applications in this domain. As a proof of concept, we describe two prototype applications that have been realized using the BTnodes.

1 Introduction

Recent advances in wireless communication and micro system technology allow the construction of so-called "sensor nodes". Such sensor nodes combine means for sensing environmental parameters, processors, wireless communication capabilities, and an autonomous power supply in a single tiny device that can be fabricated in large numbers at low cost. Large and dense networks of these untethered devices can then be deployed unobtrusively in the physical environment in order to monitor a wide variety of real-world phenomena with unprecedented quality and scale while only marginally disturbing the observed physical processes [1, 9].

In other words, Wireless Sensor Networks (WSNs) provide the technological foundation for performing many "experiments" in their natural environment instead of using an artificial laboratory setting, thus eliminating many fundamental limitations of the latter. It is anticipated that a wide range of application domains can substantially benefit

from such a technological foundation. Biologists, for example, want to monitor the behavior of animals in their natural habitats; environmental research needs better means for monitoring environmental pollution; agriculture can profit from better means for observing soil quality and other parameters that influence plant growth; geologists need better support for monitoring seismic activity and its influences on the structural integrity of buildings; and of course the military is interested in monitoring activities in inaccessible areas.

These exemplary applications domains demonstrate one important property of WSNs: their inherent and close integration with the real world, with data being captured and processed automatically in the physical environment, often in real time. This is in contrast to traditional computing systems, which are typically mostly decoupled from the real world. This paradigmatic change comes with a number of important implications, also with respect to the development and deployment of WSNs.

One such implication is that sensor networks are hard to simulate, since their exact behavior very much depends on many parameters of the physical environment. Already rather subtle differences of the simulation models and the real world can make the difference between a working and a broken implementation [14]. For example, many wireless radio simulations assume a spherical communication range. However, measurements in a network of about 100 sensor nodes revealed that the radio communication range is far from spherical in practice [12]. This can lead to asymmetric or even unidirectional communication links, which can easily break algorithms that assume symmetrical links.

Hence, experimentation platforms for performing real experiments are of particular importance in the sensor network domain. As part of various research projects, a number of such sensor network platforms were developed [3, 16]. These platforms are typically optimized for energy efficiency, providing rather restricted processing capabilities and a low-power radio platform. While this leads to sensor networks with improved longevity, the chosen RF-communication technology also imposes a number of limitations. This is especially true in experimental settings, where the purpose is the development and evaluation of algorithms and prototypical application scenarios.

We have explored a different point of the design space and have developed a sensor node – the BTnode – which is optimized for programming comfort and interoperability, using a more powerful processor, more memory, and a Bluetooth radio. Our choice was motivated by the need for a functional, versatile fast prototyping platform that is certainly not the ultimate in low-power device technology but rather available today, to an extent energy aware and low cost. Additionally, we were inspired by the observation that sensor networks are almost always part of larger systems consisting of heterogeneous devices, whose collaboration requires an interoperable networking technology.

In the remainder of this paper, we present the BTnode platform, including the hardware, the operating system software, and the service infrastructure. We begin with motivating our design decisions and the use of Bluetooth as a networking technology for sensor networks. Sect. 3 describes the BTnode hardware and its operating system while Sect. 4 characterizes the system services we designed for building complex applications. As a proof of concept, we present two such applications that have been developed

using our platform in Sect. 5. Power consumption issues are described in Sect. 6. Related work and conclusions are presented in Sect. 7 and 8, respectively.

2 Motivation

Our sensor network platform uses Bluetooth as a communication means. Many features of the Bluetooth specification and its implementations make the use of Bluetooth in sensor networks inadequate at first glance. But on the other hand, Bluetooth has become an interoperable wireless networking standard which is implemented in a number of consumer devices (e.g., PDAs, laptops, mobile phones, digital cameras) or hardware extensions (e.g., USB sticks, PCMCIA cards), and is supported by all major operating systems. This interoperability with a large variety of devices makes Bluetooth valuable in the sensor network domain, in particular for prototyping of applications now while other technologies are still under development. Below we outline some of the more interesting Bluetooth features and sketch concrete applications where these come in handy.

Bluetooth Features. When compared with "traditional" communication approaches for wireless sensor networks, Bluetooth has a rather high power consumption, suffers from somewhat long connection setup times, and has a lower degree of freedom with respect to possible network topologies [17].

On the other hand, Bluetooth is a connection-oriented, wireless communication medium that assures interoperability between different devices and enables application development through a standardized interface. It offers a significantly higher bandwidth (about 1 megabit per second) compared to many low-power radios (about 50 kilobit per second). Furthermore, the standardized Host Controller Interface (HCI) provides a high level interface that requires no knowledge of the underlying baseband and media access layers and their respective processing. This abstraction offers built-in high-level link-layer functionality such as isochronous and asynchronous communication, link multiplexing, QoS, integrated audio, forward error correction, automatic packet retransmission, user authentication using link keys, and encryption. Bluetooth also offers service discovery, serial port emulation, and IP connectivity.

Infrastructure Integration. Sensor networks are typically connected to some back-end infrastructure for tasking the network, as well as for storage and evaluation of sensing results. Also, since limited resources preclude the execution of many services (e.g., for complex data processing) in the sensor network, such services might be provided by an external infrastructure possessing sufficient resources.

For these reasons it is often necessary to connect a sensor network to an external network, such as the Internet. In the Great Duck Island habitat monitoring experiment [20], for example, a number of isolated sensor network patches are linked to a base station, which in turn is connected to the Internet via a satellite link. A database system executing on the Internet is used to store and query sensing results.

To implement such solutions, typically proprietary gateway solutions are required to bridge between the sensor network and the Internet. However, if the sensor nodes

support Bluetooth, off-the-shelf devices can be used to implement such solutions. A Bluetooth-enabled laptop computer, for example, can be used to bridge between Bluetooth and WLAN. Instead of a satellite link, a mobile phone can be used as a gateway to connect a sensor network to the Internet.

User Interaction. Many Bluetooth-enabled devices can also be used for direct interaction with isolated sensor networks that are not connected to a background infrastructure. Interaction with a user might be required to specify the sensing task, or to alert the user of sensing results. A Bluetooth-enabled mobile phone, for example, can be used to alert a nearby user. By placing a phone in the vicinity of the sensor network, the sensor network can even send short text messages (SMS) to a user who is not currently present. It is advantageous to be able to use a widely spread class of devices with a well acquainted user interface for such interactions.

Actuators. In some sensor network applications, just storing measurement results for later evaluation is not sufficient but an immediate action is needed upon detecting certain phenomena – either by means of automated actuation or by notifying a user. Consider for example an animal habitat monitoring application, where it might be useful to notify a biologist or to take a picture in case a certain animal is present.

If the sensor network uses Bluetooth, we can choose among a wide variety of commercial off-the-shelf Bluetooth-enabled consumer devices to implement such actuation. For example, the sensor network can trigger a Bluetooth-enabled camera to take a picture when it suspects something interesting is happening.

Debugging. The development of algorithms and applications for sensor networks is non-trivial for various reasons. Firstly, sensor networks are highly dynamic distributed systems, where a consistent view of the global system state is typically not available to the developer. Secondly, the behavior of the sensor network is highly dependent on the physical environment, such that problems might not be easily reproducible. Thirdly, for reasons of energy efficiency, sensor network applications do often perform in-network data processing and aggregation, where raw sensor data is already processed and evaluated inside the network in order to reduce the volume of data which has to be transmitted. To verify and debug such systems, the developer often needs access to both, the raw sensor data and the aggregated output of the sensor network [8].

Currently, research groups working on the development of algorithms and applications for sensor networks typically use arrays of sensor nodes, where each node is connected to a central computer by a serial cable and a serial port multiplexer. While this is possible in small-scale laboratory settings, large-scale field experiments typically cannot be instrumented this way. On the other hand, mobile terminals can give untethered access to groups of sensor nodes in the field for in situ debugging, provided they both use the same radio technology. Here Bluetooth is an ideal candidate.

Sensor Node Heterogeneity. As mentioned earlier, sensor networks should employ in-network data processing and aggregation in order to reduce the amount of data that has to be transmitted within the network. This is desirable in order to achieve energy efficiency and to match the typically limited capacity of the communication channels.

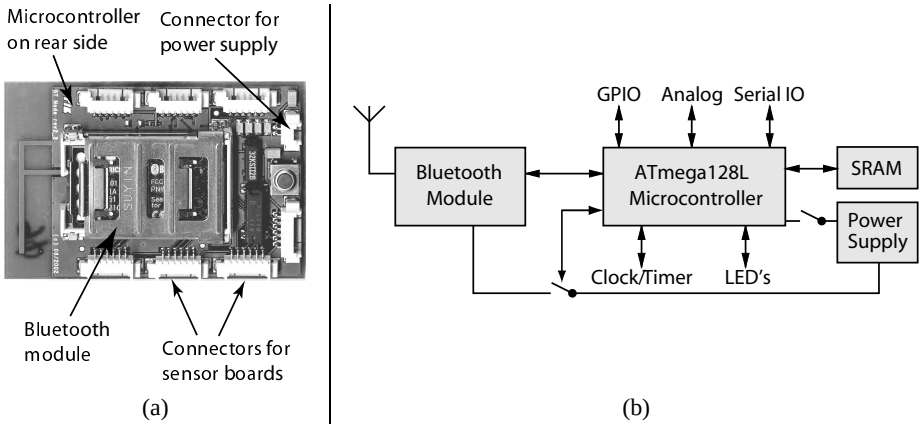

Fig. 1. (a) BTnode hardware (b) BTnode system overview.

However, many research projects came to the conclusion that the computing and memory resources of sensor nodes are often too limited to perform typical signal processing tasks (e.g., FFT, signal correlation). Hence, clustered architectures were suggested [13, 27], where the cluster-heads are equipped with more computing and memory resources. Cluster-heads then process and aggregate sensor data collected from the nodes in their cluster. In [13, 27], for example, PDAs with proprietary hardware extensions for wirelessly interfacing the sensor nodes are used for this purpose. With Bluetooth-enabled sensor nodes, off-the-shelf PDAs and laptops can be easily integrated as cluster heads without hardware modification.

3 BTnode Architecture

3.1 Hardware

The BTnode is an autonomous wireless communication and computing platform based on a Bluetooth radio module and a microcontroller. The device has no integrated sensors, since individual sensor configurations are required depending on the application. Instead, with its many general-purpose interfaces, the BTnode can be used with various peripherals, such as sensors, but also actuators, DSPs, serial devices (like GPS receivers, RFID readers, etc.) and user interface components. An interesting property of this platform is its small form factor of 6x4 cm while still maintaining a standard wireless interface.

The BTnode hardware (see Fig. 1) is built around an Atmel ATmega128L microcontroller with on-chip memory and peripherals. The microcontroller features an 8-bit RISC core delivering up to 8 MIPS at a maximum of 8 MHz. The on-chip memory consists of 128 kbytes of in-system programmable Flash memory, 4 kbytes of SRAM, and 4 kbytes of EEPROM. There are several integrated peripherals: JTAG for debugging, timers, counters, pulse-width modulation, 10-bit analog-digital converter, I2C bus, and two hardware UARTs. An external low-power SRAM adds an additional 240 kbytes of

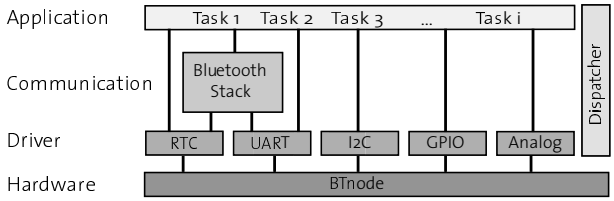

Fig. 2. A lightweight OS framework for WSN applications.

data memory to the BTnode system. A real-time clock is driven by an external quartz oscillator to support timing updates while the device is in low-power sleep mode. The system clock is generated from an external 7.3728 MHz crystal oscillator.

An Ericsson Bluetooth module is connected to one of the serial ports of the microcontroller using a detachable module carrier, and to a planar inverted F antenna (PIFA) that is integrated into the circuit board.

Four LEDs are integrated, mostly for the convenience of debugging and monitoring. One analog line is connected to the battery input and allows to monitor the battery status. Connectors that carry both power and signal lines are provided and can be used to add external peripherals, such as sensors and actuators.

3.2 Lightweight Operating System Support

The BTnode system software (see Fig. 2) is a lightweight OS written in C and assembly language. The drivers, which are available for many hardware subsystems, provide convenient APIs for application developers. The system provides coarse-grained cooperative multitasking and supports an event-based programming model.

Drivers. The drivers are designed with fixed buffer lengths that can be adjusted at compile time to meet the stringent memory requirements. Available drivers include memory, real-time clock, UART, I2C, LEDs, power modes, and AD converter. The driver for the Bluetooth radio provides a subset of the networking functionality according to the Bluetooth specification. Bluetooth link management is performed on Bluetooth's L2CAP layer. RFCOMM, a serial port emulation, provides connectivity to computer terminals and consumer devices, such as cameras and mobile phones. Dial-up connections to a modem server through a mobile GSM phone are easily established with a special-purpose function. All other GSM services (such as file sharing, phone book and calendar) can be utilized through lower-level interfaces.

Process Model. Like most embedded systems, the BTnode does not support a dynamic process model. Only a single application is present on the system at a time. At compile time, applications are linked to the system software, which comes as a library. The resulting executable is then uploaded to the BTnode's Flash memory, effectively overwriting any previous application code. After uploading, the new application starts immediately. However, the BTnode system can also be reprogrammed through the network. To do so, the application currently running on the system needs to receive the

```
 1: #include <btnode.h>
    #define THRESHOLD_EV (MAX_SYSTEM_EVENT+1)
 3: static u16 connection_id = 0;

 5: int main( int argc, char* argv[] ) {
      btn_system_init( argc, argv, /* ... */ );
      btn_bt_psm_add( 101 ); /* accept connections */
 8:   btn_disp_ev_reg( BT_CONN_EV, conn_cb, 0 );
 9:   btn_disp_run();
      return 0; /* not reached */
    }
12: void conn_cb( call_data_t call_data, cb_data_t cb_data ) {
      connection_id = (u16)(call_data & 0xFFFF);
      bt_sensor_start( BTN_SENSOR_TEMP );
15:   bt_sensor_set_threshold( BTN_SENSOR_TEMP, 30, THRESHOLD_EV );
16:   btn_disp_ev_reg( THRESHOLD_EV, sensor_cb, 0 );
    }
18: void sensor_cb( call_data_t call_data, cb_data_t cb_data ) {
      u8 error = 0;
      u8 databuf = (call_data & 0xFFFF);
      btn_bt_data_send( connection_id, databuf, sizeof(data_buffer) );
    }
```

Fig. 3. A simple BTnode program.

new executable, save it to SRAM, and then reboot the system in bootloader mode. The bootloader finally transfers the received executable to Flash memory and starts it.

Programming Model. The system is geared towards the processing of (typically externally triggered) events, such as sensor readings, or the reception of data packets on the radio. To this end, BTnode applications follow an event-based programming model, common to embedded systems and GUI programming toolkits, where the system schedules application tasks on the occurrence of events. Internally, those tasks (or event handlers) are implemented as C functions.

In the BTnode system, events model state changes. A set of predefined event types is used by the drivers to indicate critical system conditions, such as expired timeouts or data being ready for reading on an I/O device. Applications can define their own event types, which are represented by 8-bit integer values. Individual events can carry type-specific parameters for evaluation in the application task.

A central component of the system is the dispatcher, where applications register event/event-handler-function pairs. After the dispatcher has been started, it also accepts individual events from drivers and application tasks and stores them in a FIFO queue. While the event queue is not empty, the dispatcher starts the corresponding handler (i.e., the previously registered function) for the next event in the queue, passing the events individual parameters.

Events are processed in the order they are received by the dispatcher. Only one event handler can be active at a time and every event handler is always completely executed before the next is scheduled. So every event handler depends on the previous event handler to terminate in time. However, long tasks can be broken into smaller pieces and can be triggered subsequently by the event mechanism (e.g., by application-specific events).

Fig. 3 shows a typical BTnode program, which is waiting for an incoming Bluetooth connection. Once the connection is established, it repeatedly transmits sensor data exceeding a given threshold. During initialization, the program registers a handler for connection events (line 8). It then passes control to the dispatcher (line 9), which enters sleep mode until events occur that need processing. Once the connection is established, the corresponding handler function `conn_cb` is called by the dispatcher (line 12). The program initializes the sensors with a threshold value (line 15) and registers the event handler `sensor_cb` for handling this event (line 16). On the occurrence of the sensor event, the associated data is sent over the connection (line 18 ff).

Portability. The whole system software is designed for portability and is available for different emulation environments (x86 and iPAQ Linux, Cygwin, and Mac OS X) apart from the embedded platform itself. Emulation simplifies application building and speeds up debugging since developers can rely on the sophisticated debugging tools available on desktop systems. Also, the time for uploading the sensor-node application to the embedded target can be saved. Furthermore, different platforms, such as an iPAQ running Linux, can be used as cluster heads, reusing much of the software written for the embedded nodes and making use of the resources of the larger host platform for interfacing, extended computation or storage.

4 System Services

Besides the core system software, the BTnode platform also provides means to integrate sensor nodes into a surrounding communication infrastructure, to exchange and collaboratively process the data of locally distributed sensors, and to outsource computations to nearby devices with more sophisticated resources. These functions constitute the core of the BTnode system services.

4.1 Ad Hoc Infrastructure Access

As noted in Sect. 2, sensor networks often need access to a background infrastructure. However, due to their deployment in remote, inaccessible, or undeveloped regions, fixed access to a background infrastructure via stationary gateways usually cannot be assumed. We therefore provide *mobile ad hoc infrastructure access* by means of mobile gateways. The core idea of this approach is depicted in Fig. 4: when handheld devices – such as Bluetooth-enabled mobile phones or PDAs – are placed in the vicinity of the sensor network, BTnodes can utilize these mobile devices to access background services in an ad hoc fashion. Such mobile gateways provide access to an infrastructure via GSM or WLAN, for example.

The BTnode system provides services for utilizing nearby mobile phones as mobile infrastructure gateways. An RFCOMM connection is set up between a BTnode and the nearby phone. AT commands manage and control GSM data connections to a background infrastructure server, which in turn must provide a regular modem or GSM gateway itself. Alternatively, a BTnode can embed sensory data into an SMS message

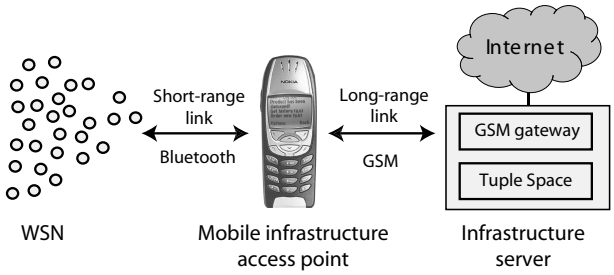

Fig. 4. Ad hoc infrastructure access with handheld devices as mobile gateways.

and transfer it over the public phone network. Depending on the application, both approaches have their advantages and disadvantages. While building up a GSM data connection is more suitable for sending larger amounts of data, sending data in an SMS message is asynchronous and does not require connection establishment with the communication partner. The infrastructure provides software for running a GSM gateway and for processing of incoming messages (see Fig. 4).

Sensor networks or parts thereof can be temporarily disconnected from the background infrastructure. Our service infrastructure provides a centralized tuple space [5, 7] in order to enable cooperation across such temporarily isolated network patches. When disconnected, nodes buffer sensory data locally and synchronize with the tuple space when they regain connectivity. As infrastructure communication takes place via mobile gateways the delay for relaying the information over the centralized server can become relatively large in the order of tens of seconds.

4.2 Collaborative Processing of Sensory Data

A typical sensor node has only limited processing power, can perceive only a small local subset of its physical environment, and might not have all necessary sensors to solve a complex sensing task on its own. For these reasons, sensor nodes usually need to cooperate with others. The basic system concept that enables such kind of inter-node interaction in the BTnode system services is a distributed tuple space.

The distributed tuple space serves as a shared data structure for a set of sensor nodes which enables them to exchange and process data collaboratively. The main advantage of the tuple space approach is that nodes become able to share their resources, such as sensors and memory, with other nodes. For example, it is possible for nodes to operate on remote sensors as if they were locally attached, and to store data tuples at other nodes with free memory resources.

This distributed tuple space implementation does not require access to the centralized background infrastructure mentioned in the previous section. Instead, it is implemented as a data structure covering multiple sensor nodes. BTnodes read local sensors, embed sensory data into tuples, and write it into the distributed tuple space. Other nodes can then access these tuples and fuse the corresponding sensory data with local measurements. The corresponding result is usually embedded into a tuple that is again written into the space. Our implementation also offers the possibility to register callbacks

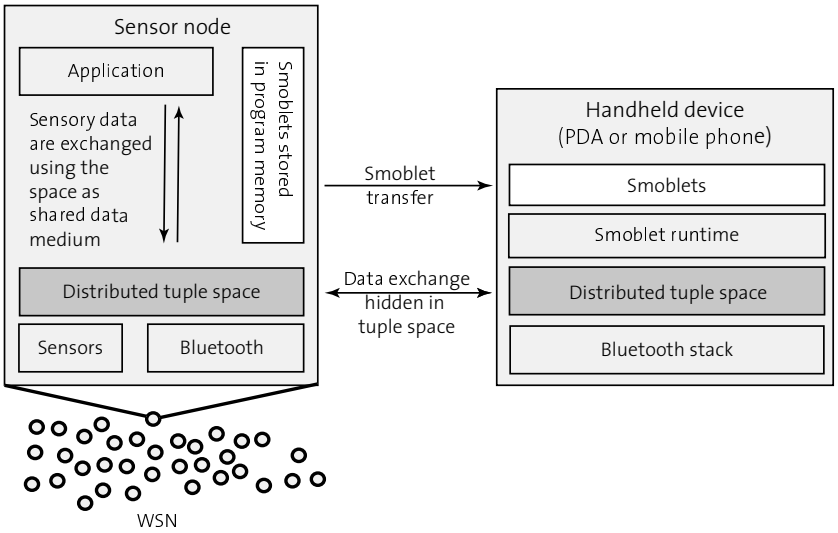

Fig. 5. Ad hoc outsourcing of computations to handheld devices with Smoblets.

on certain sensor types, which is very handy for sensor fusion algorithms. For example, a callback could be registered at a remote sensor for a temperature tuple, which would notify the local node each time a tuple is generated. The temperature data from the callback could then be fused with the local node's own temperature readings.

The distributed tuple space also makes it easier for an application programmer to design sensor fusion algorithms because they can operate on a broader common data basis. From the programmer's viewpoint, the tuple space is a grouping concept based on a node's current situation or on application-specific information. Therefore, sensor nodes that share their data in a distributed tuple space appear like a single node with many sensors and resources. This simplifies the system view for the application programmer.

4.3 Outsourcing Computations

In Sect. 2 we mentioned that some sensor network applications require the use of devices with more computing and memory resources than sensor nodes in order to perform, for example, complex signal processing tasks. Both the use of Bluetooth as the BTnodes' communication technology and the portability of the BTnode software makes it easy to integrate commodity devices such as Bluetooth-enabled PDAs into the sensor network, for example as cluster heads.

Additionally, for outsourcing computations spontaneously to nearby mobile Bluetooth-enabled devices, the BTnode system provides the so-called *Smoblet* service. Smoblets are precompiled Java classes, similar to applets, that are stored in the program memory of BTnodes and transferred to nearby mobile devices, where they are executed to process input from the sensor network. To enable this kind of cooperation, the mobile device joins the distributed tuple space of the sensor node that originally provided the

Smoblet code. As the actual origin of sensory data and the actual location of resources becomes to some degree transparent through the tuple space, it becomes irrelevant on which device (handheld or sensor node) the code is actually executed. By transferring code to a nearby handheld device with more resources and by providing access to the same data basis through the distributed tuple space, sensor nodes can save considerable amounts of energy [11, 27].

A Smoblet can also contain Java code for a user interface, which – when transmitted to the mobile device – allows the user to adapt certain settings of a sensor node or to control an application. The transmission of code can be initiated by a sensor node itself based on its current context, or by a user by means of a simple user interface on the handheld.

Fig. 5 gives an overview of the Smoblet concept. The BTnode system services provide a set of libraries that allow Java programs to operate on data in a distributed tuple space and to transfer Java code over a Bluetooth connection. An application programmer generates the Smoblet code using these libraries on a host machine and transforms the Java classes (or JAR archive files) into C source files. The programmer can then compile and link these files into the application for the sensor node. When the resulting program file is transferred to the microcontroller of the sensor node, the Smoblets are automatically stored into the program memory of that node. BTnodes themselves cannot execute Java code, the program memory serves merely as data storage for the Smoblets. When a handheld device now enters the range of the sensor node, the Smoblet code can be transferred over a local Bluetooth connection. Thereby, the handheld joins the distributed tuple space of the sensor node and executes the Smoblet code, which can now operate on the shared data structure provided by the distributed tuple space.

5 Applications

BTnodes have been used as a prototyping platform for a number of applications [2, 11, 21, 25, 26]. Below we present two additional applications and discuss how they make use of the various features of our platform. The first application is a sensor network for tracking mobile objects using a remote-controlled toy car as a sample object. The second application is about monitoring the state of products during transportation on a truck.

5.1 Object Tracking

The purpose of this project is to explore the use of Smart Dust for tracking the movements of mobile entities in the open field (e.g., vehicles, animals). Smart Dust [10] are next generation sensor nodes, which provide sensing, computing, and wireless communication capabilities on a single device as tiny as a few cubic millimeters. The focus here is on support mechanisms such as node localization and time synchronization that are suitable for large networks of Smart Dust.

Since Smart Dust is not yet available, we built a prototype system based on BTnodes in order to evaluate the algorithms we developed. We used a remote-controlled toy car as a sample target. A number of BTnodes are randomly deployed in the area of interest

and can change their location after deployment. The nodes use attached sensors to detect the presence of the car and send respective notifications to a base station. The base station fuses these notifications in order to estimate the current location of the car. A graphical user interface displays the track and allows to control various aspects of the system. The sensor data fusion requires that all nodes share a common reference system both in time and space, which necessitates mechanisms for node localization and time synchronization. The base station consists of a Linux laptop computer equipped with a Bluetooth radio. A detailed description of the system can be found in [23].

The hardware abstraction provided by the BTnode operating system software allowed us to develop and debug the BTnode software on a Linux platform. Moreover, the base station software executing on the Linux laptop uses the same operating system software as the BTnodes, which makes it particularly easy to port the base station software to another BTnode, which could then send off tracking results to a remote user via a mobile phone.

5.2 Product Monitoring

The goal of this application is to monitor the state of goods during transport such that the owner of a product can query the product's state and is notified whenever it is damaged. This notification should be instantaneous, containing information about when, where, and why a product has been damaged. To simplify prototyping, the application does not rely on extra stationary hardware installed inside vehicles. In our prototype, BTnodes are attached to ordinary products. The nodes themselves are equipped with acceleration and temperature sensors to determine the product's current state and to derive the reason why a product has been damaged. When a product is damaged, the attached BTnode tries to notify the owner of the product.

Background infrastructure access takes place using the Bluetooth-enabled mobile phone of the vehicle's driver. Whenever it is in range of the products, they use it as a mobile access point for communicating with a background infrastructure server (see Sect. 4). Between consecutive infrastructure accesses, BTnodes buffer data and synchronize their state with a background infrastructure service whenever access becomes possible. Additionally, the driver's mobile phone is used as location sensor. When a product is damaged, the corresponding BTnode obtains the current cell id and location area code from the mobile phone. The information about where an event occurred is then embedded into the message sent to the background infrastructure service.

The user interaction through the mobile phone consists of a WAP (Wireless Application Protocol) interface for querying the status of a product and an SMS message to notify the owner by a short text message whenever the product has been damaged. It is implemented as a Servlet that generates WAP pages providing information about the product's current state. User inputs can also be passed on directly to the product when a mobile access point enters the range of the attached BTnode.

6 Energy Consumption

One major criticism of Bluetooth and its use in sensor networks is its high energy consumption. However, as measurements in [18] confirm, the energy consumption per

transmitted bit is competitive with dedicated low-power radios. The authors conclude, that Bluetooth is a viable solution energy-wise if the radio is only switched on for short burst transfers and stays turned off otherwise. Note that this fits many sensor network applications quite well, where network activity is rather bursty and short, triggered by rare activities in the environment (e.g., a product being damaged in the product monitoring application). There are two important strategies to switch off the radio frequently: duty-cycling and wake-up radio.

With *duty cycling*, the radio is switched on in a periodical pattern, trading off network latency for energy efficiency. Consider for example the product monitoring application described above. We can estimate the average battery lifetime of a BTnode as follows: A 10 % duty cycle for Bluetooth yields a quite acceptable average power consumption of 6.5 mW and a battery lifetime in the order of weeks on a standard 840 mAh Li-ion battery. Here, 12 mW, 160 mW, and 0.5 mW power consumption were assumed for sensing, communication, and idle modes, respectively.

For *wake-up radio* [24], a second low-power radio is attached to a BTnode. While Bluetooth is normally switched off, the low-power radio handles low-bandwidth data and network management traffic. Bursts of high bandwidth traffic are negotiated on the low-power channel. Both the receiver and sender then switch on their Bluetooth radios, before sending the actual data via Bluetooth. This strategy can save a significant amount of power if the high-power radio is only occasionally needed. We envision a future dual-radio platform, which would allow the implementation of such a strategy.

Additionally, newer Bluetooth hardware has considerably improved power characteristics compared to the modules used for the current generation of BTnodes, and is expected to reduce power consumption in communication mode by a factor of 2-4. However, even with the current hardware, energy consumption is low enough to actually build prototype applications that run for hours or days, enabling us to gain practical experience through experimentation.

7 Related Work

The classical approach to Wireless Sensor Network devices aims at low power and highly integrated node hardware. One of its most prominent representatives is the UC Berkeley Mote [16] that has been commercialized and is widely used by researchers all over the world. Other research groups have developed similar platforms, all based on a low power microcontroller and a custom radio front-end chip [3, 16, 22]. Common to these architectures is that all protocol processing (baseband and MAC) is done on the host microcontroller, which implies a meticulous design process, knowledge about RT systems, and locks up many resources on the host CPU. Another drawback of these platforms is that they can only communicate with devices equipped with the same radio and protocol processing capabilities. In contrast to that, the BTnodes can interact with any Bluetooth-enabled device without the need to integrate further hardware and software.

Other prototyping systems based on FPGAs, StrongARM processors, and similar hardware components [22] are geared towards protocol and communication development only. Other researches have tried to set up a development platform that contains

as many subsystems as possible (e.g., the I-Badge system [6]), inducing considerable system complexity and also interfacing constraints.

Most of the available hardware platforms only provide a thin hardware abstraction layer, on top of which applications are executing. One particular exception is TinyOS [16], a component-oriented operating system designed for the Berkeley Motes. Similar to our approach it provides asynchronous events as a basic programming abstraction. System services (such as multi-hop routing) as well as applications are implemented as a set of components with well-defined interfaces. A simple declarative language is used to connect these components in order to form a complete application. TinyOS applications are programmed in a custom-made C dialect, requiring special development tools. This is in contrast to our platform, which uses standard C and an unmodified GNU tool chain.

Various projects aim to provide a service infrastructure or middleware which supports the development of complex sensor network applications. TinyDB [15] interprets the sensor network as a distributed database being constantly filled with sensory data such that a simple SQL-like query language can be used to query the sensor network. SensorWare [4] uses an agent-like approach to program sensor networks for complex tasks. A scripting language is used to define small programs which are then distributed to the nodes of the network. Later on, executing scripts can migrate from node to node along with their state information. DSWare [19] uses a real-time variant of an event notification system, which allows to subscribe to specific events generated by sensor nodes. In contrast, we decided to follow an approach based on tuple spaces. We believe that this is a useful abstraction for many sensor applications that depend on fusing information from many nearby sensor nodes. Firstly, a tuple space provides a natural place for collecting and fusing this information. Secondly, since physical phenomena are often local, cooperation in sensor networks is also often local. Hence, local tuple spaces can be efficiently implemented by using broadcast communication over very few hops.

8 Conclusion and Future Work

We have presented the BTnode platform for prototyping sensor network applications. It consists of a Bluetooth-based sensor node, a lightweight operating system with support for several hardware platforms, and a set of system services to support the development of complex sensor network applications. As a proof-of-concept, we presented two complex applications which we developed using this platform.

We motivated our platform by the need for an environment to quickly prototype sensor network applications using commercial off the shelf devices in conjunction with sensor nodes. Additionally, we observed that sensor networks are often part of larger systems consisting of heterogeneous devices. Collaboration of the sensor nodes with these devices requires a common and widely supported networking technology, such as Bluetooth.

Up to now, about 200 BTnodes have been manufactured, which are used by a number of research projects across Europe. As part of the Swiss National Competence Center for Mobile Information and Communication Systems (NCCR-MICS) [29], we intend to develop the BTnode platform into a versatile and widely available prototyping

platform for sensor network applications. A BTnode system-software kit consisting of a build environment (avr-gcc cross compiler and standard libraries), source code, debugging support, demo examples, and documentation has been assembled and is available to developers along with a support mailing list and software repository [28].

We are currently considering augmenting the next generation of BTnodes with an additional ultra low-power radio, which is developed by one of our partner institutions within NCCR-MICS. Among other things, this will allow us to build more energy-efficient sensor networks based on Bluetooth by using a wake-up strategy similar to the one described in [24] – thus combining the energy efficiency of ultra low power radios with the interoperability of Bluetooth.

9 Acknowledgments

The work presented in this paper was conducted as part of the Smart-Its project and the NCCR-MICS. The Smart-Its project is funded by the European Union under contract IST-2000-25428, and by the Swiss Federal Office for Education and Science (BBW 00.0281). NCCR-MICS, the National Competence Center in Research on Mobile Information and Communication Systems, is supported by the Swiss National Science Foundation under grant number 5005-67322.

References

1. I.F. Akyildiz et al. Wireless sensor networks: A survey. *Computer Networks*, 38(4):393–422, Mar. 2002.
2. S. Antifakos, F. Michahelles, and B. Schiele. Proactive instructions for furniture assembly. In *Proc. 4th Int'l Conf. Ubiquitous Computing (UbiComp 2002)*, pages 351–356. Springer, Berlin, Sept. 2002.
3. M. Beigl and H. Gellersen. Smart-Its: An Embedded Platform for Smart Objects. In *Proc. Smart Objects Conference (SOC 2003)*, Grenoble, France, May 2003.
4. A. Boulis, C.C. Han, and M.B. Srivastava. Design and implementation of a framework for programmable and efficient sensor networks. In *Proc. 1st ACM/USENIX Conf. Mobile Systems, Applications, and Services (MobiSys 2003)*, pages 187–200. ACM Press, New York, May 2003.
5. N. Carriero and D. Gelernter. Linda in Context. *Communications of the ACM*, 32(4):444–458, April 1989.
6. A. Chen et al. A support infrastructure for the smart kindergarten. *IEEE Pervasive Computing*, 1(2):49–57, 2002.
7. N. Davies et al. Limbo: A tuple space based platform for adaptive mobile applications. In *Proc. 6th Int'l Conf. Open Distributed Processing/Distributed Platforms (ICODP/ICDP '97)*, pages 291–302, May 1997.
8. J. Elson et al. EmStar: An Environment for Developing Embedded Systems Software. Technical Report 0009, Center for Embedded Networked Sensing (CENS), 2003.
9. D. Estrin et al. Connecting the physical world with pervasive networks. *IEEE Pervasive Computing*, 1(1):59–69, 2002.
10. B. Warneke et al. Smart Dust: Communicating with a Cubic-Millimeter Computer. *IEEE Computer*, 34(1):44–51, Jan. 2001.

11. C. Plessl et al. Reconfigurable hardware in wearable computing nodes. In *Proc. Int. Symp. on Wearable Computers (ISWC'02)*, pages 215–222. IEEE, Oct. 2002.
12. D. Ganesan et al. Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks. Technical Report CSD-TR 02-0013, UCLA, Feb. 2002.
13. H. Wang et al. Target classification and localization in habit monitoring. In *Proc. 2003 Int'l Conf. Acoustics, Speech, and Signal Processing (ICASSP 2003)*, volume 4, pages 844–847. IEEE, Piscataway, NJ, Apr. 2003.
14. J. Heidemann et al. Effects of Detail in Wireless Network Simulation. In *Proc. SCS Multiconference on Distributed Simulation 2001*, Jan. 2001.
15. S. R. Madden et al. TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks. In *Proc. 5th Symp. Operating Systems Design and Implementation (OSDI 2002)*, Boston, USA, Dec. 2002.
16. J. Hill et al. System architecture directions for networked sensors. In *Proc. 9th Int'l Conf. Architectural Support Programming Languages and Operating Systems (ASPLOS-IX)*, pages 93–104. ACM Press, New York, Nov. 2000.
17. O. Kasten and M. Langheinrich. First Experiences with Bluetooth in the Smart-Its Distributed Sensor Network. In *Workshop on Ubiquitous Computing and Communications, PACT 01*, Barcelona, Spain, October 2001.
18. M. Leopold, M.B. Dydensborg, and P. Bonnet. Bluetooth and Sensor Networks: A Reality Check. In *Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys 2003)*. ACM Press, New York, Nov. 2003.
19. S. Li, S. H. Son, and J. A. Stankovic. Event Detection Services Using Data Service Middleware in Distributed Sensor Networks. In *IPSN 2003*, Palo Alto, USA, April 2003.
20. A. Mainwaring et al. Wireless sensor networks for habitat monitoring. In *Proc. 1st ACM Int'l Workshop Wireless Sensor Networks and Applications (WSNA 2002)*, pages 88–97. ACM Press, New York, Sept. 2002.
21. F. Michahelles and B. Schiele. Better rescue through sensors. In *Proc. 1st Int'l Workshop Ubiquitous Computing for Cognitive Aids (at UbiComp 2002)*, Sept. 2002.
22. J.M. Rabaey et al. PicoRadio Supports Ad Hoc Ultra-Low Power Wireless Networking. *IEEE Computer*, 33(7):42–48, July 2000.
23. K. Römer. Tracking Real-World Phenomena with Smart Dust. In *EWSN 2004*, Berlin, Germany, January 2004.
24. E. Shih, P. Bahl, and M. Sinclair. Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices. In *Proc. 6th ACM/IEEE Ann. Int'l Conf. Mobile Computing and Networking (MobiCom 2001)*, pages 160–171. ACM Press, New York, Sept. 2002.
25. F. Siegemund. Spontaneous interaction in ubiquitous computing settings using mobile phones and short text messages. In *Proc. Workshop Supporting Spontaneous Interaction in Ubiquitous Computing Settings (at Ubicomp 2002)*, Sept. 2002.
26. F. Siegemund and C. Flörkemeier. Interaction in Pervasive Computing Settings using Bluetooth-enabled Active Tags and Passive RFID Technology together with Mobile Phones. In *Proc. 1st IEEE Int'l Conf. Pervasive Computing and Communications (PerCom 2003)*, pages 378–387. IEEE CS Press, Los Alamitos, CA, Mar. 2003.
27. H. Wang, D. Estrin, and L. Girod. Preprocessing in a Tiered Sensor Network for Habitat Monitoring. Submitted for publication, September 2002.
28. BTnodes - A Distributed Environment for Prototyping Ad Hoc Networks. http://www.btnode.ethz.ch.
29. NCCR-MICS: Swiss National Competence Center on Mobile Information and Communication Systems. http://www.mics.org.