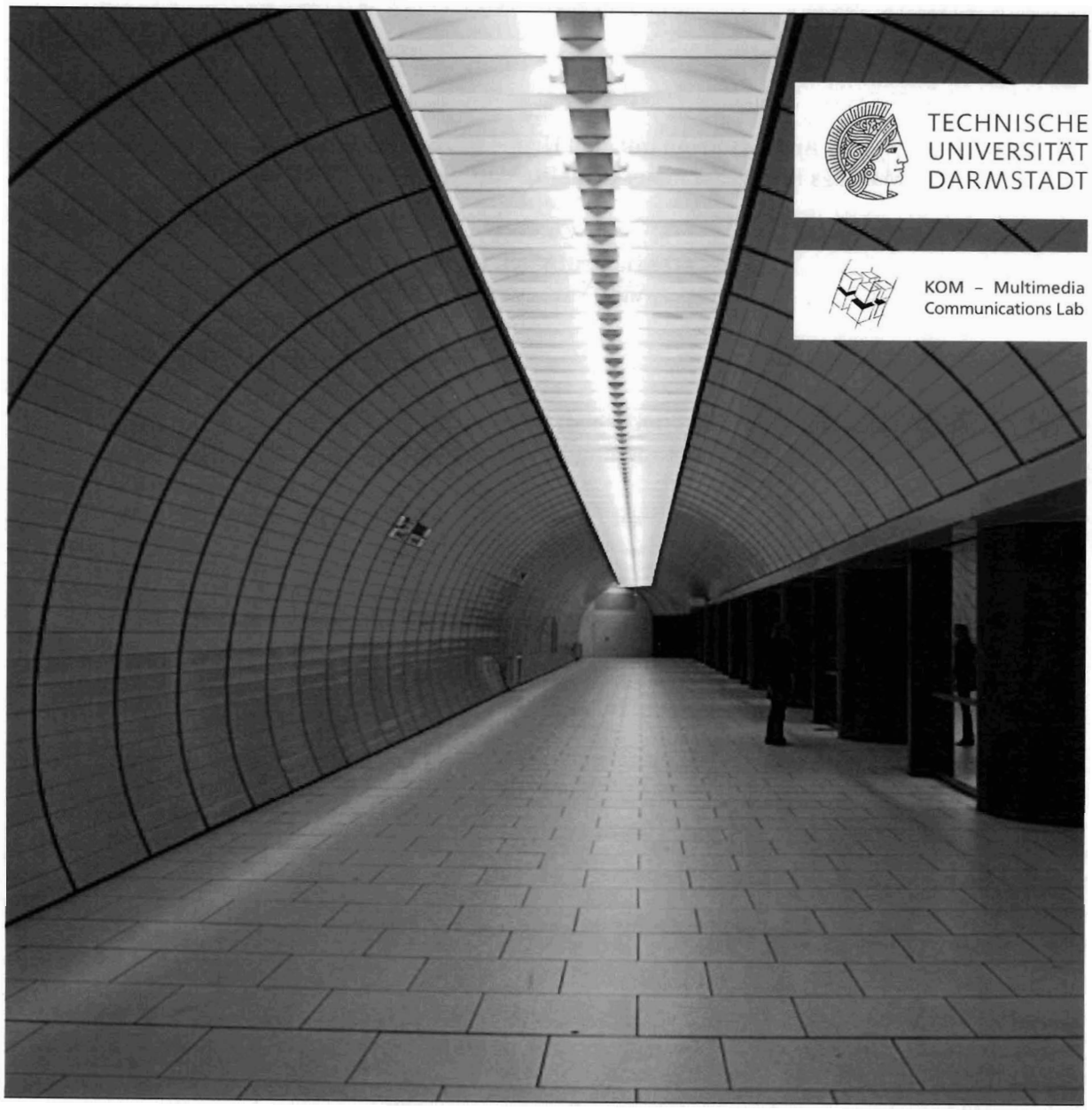
[RHG+08] *Andreas Reinhardt, Jan Hennecke, Steffen Gottwald, Matthias Kropff, Johannes Schmitt, Matthias Hollick, Ralf Steinmetz;* **Tubicles: Heterogeneous Wireless Sensor Nodes - Testbed Objectives and Assembly Instructions.** no. TR-KOM-2008-09. December 2008.

Tubicles: Heterogeneous Wireless Sensor Nodes

Testbed Objectives and Assembly Instructions
Technical Report TR-KOM-2008-09, 22 December 2008

Andreas Reinhardt, Jan Hennecke, Steffen Gottwald,
Matthias Kropff, Johannes Schmitt, Matthias Hollick, Ralf Steinmetz



Technische Universität Darmstadt
Department of Electrical Engineering and Information Technology
Department of Computer Science (Adjunct Professor)
Multimedia Communications Lab
Prof. Dr.-Ing. Ralf Steinmetz

Tubicles: Heterogeneous Wireless Sensor Nodes Testbed Objectives and Assembly Instructions

Technical Report KOM-TR-2008-09

By
Andreas Reinhardt, Jan Hennecke, Steffen Gottwald,
Matthias Kropff, Johannes Schmitt, Matthias Hollick, Ralf Steinmetz

Contact
andreas.reinhardt@kom.tu-darmstadt.de
http://www.kom.tu-darmstadt.de

First published: 25 November 2008
Last revision: 22 December 2008

For the most recent version of this report see
ftp://ftp.kom.tu-darmstadt.de/pub/TR/KOM-TR-2008-09.pdf

Front page photo source: http://www.ipernity.com

Abstract

Real sensor network deployments often exhibit characteristics that significantly differ from simulations. Various factors are known to have an impact on application behavior, including real radio characteristics, sensor readings, and timing constraints. Incomplete or stochastic models employed in current simulation utilities do not cover these aspects in their entirety, possibly leading to a limited applicability of the results. A feasible approach to address this problem is the deployment of a testbed and the evaluation of applications on real sensor node hardware.

Testbeds can provide different means of developer support, ranging from extensive logging and deployment support to pure ad hoc node deployments with barely any assistance to the user. The demand for heterogeneity in various dimensions also becomes increasingly important, and similarly considering mobile and moving nodes becomes a major issue to evaluate user-centric applications and algorithms, i.e., applications focussing on monitoring persons instead of environmental parameters. The analysis and verification process of smart applications, i.e., applications that dynamically adapt to arising requirements during runtime, poses additional constraints on the testbed.

Taking these desired features into account, we have elaborated a set of design requirements for a heterogeneous sensor network testbed suited to evaluate the behavior of smart applications in [1]. Apart from sophisticated support during deployment and experiment runtime, our testbed has been designed to provide versatility in terms of heterogeneity, portability, mobility, and application debugging support. These means prevent the testbed from being confined to a small subset of application domains.

In this technical report, we present the tubicle node platform, a heterogenous sensor node forming the basic component of our TWiNS.KOM testbed. Heterogeneous in terms of computational power, available memory, radio protocols, and sensing capabilities, the platform allows to analyze the impact of heterogeneity on applications. The support for portability allows to relocate nodes easily, an essential prerequisite to perform experiments in situ. The capability of integrating mobile sensors, and the versatile debugging and deployment support, are further assets of our platform.

This technical report presents profound details on the selected hardware and software components, and provides detailed information about the process of assembly and programming of the nodes. Binary distributions of the presented software bundles are available for download at `http://www.kom.tu-darmstadt.de/twins`. Feedback and questions are appreciated, please address your messages to Andreas.Reinhardt@KOM.tu-darmstadt.de.

Contents

1 Introduction

The emerging field of Wireless Sensor Networks (WSNs) encompasses the deployment of many small wireless nodes (motes) in selected application scenarios, targeted to take environmental readings, perform distributed processing tasks on the sampled data, and forward them over the radio to neighboring nodes or external sinks [2]. The vision of smart dust assumes these nodes are both tiny and cheap, and do hence suffer from restrictions in terms of computational power, memory storage space, and the available energy budget [3].

In recent years, a variety of node platforms has been developed, heterogeneously differing in many regards, including price, size, energy consumption, computational power, available sensors and supported operating systems. Often, sensors for solar radiation, temperature, and humidity are present on current motes. However, some platforms do also provide accelerometers or even more sophisticated sensors like webcams.

A common radio protocol for WSNs is the IEEE 802.15.4 standard [4], which allows low-power wireless communication. Sharing the 2.4 GHz frequency band with other wireless technologies, such as WiFi and Bluetooth, it is not unlikely that interference is likely to occur in environments where multiple wireless links of different kinds are present. These real-life aspects are however not covered in many simulation tools; instead, simulation models for both sensors and radio are often reduced to stochastical models for simplicity reasons, rendering the results inapplicable on real deployments in many cases. Experiments in real environments dedicatedly regard these influences, however at the cost of additional efforts that are required to set up nodes, deploy application images, and perform analyses of observed behavior.

1.1 Motivation

The evaluation of simulated algorithms on real sensor node hardware is essential in the process of application development, ensuring that systems perform their dedicated task at all times. Operating on real underlying systems also allows to determine how applications handle enforced resource constraints, real sensor readings and radio channels. While on-the-fly deployments suffice to evaluate applications on a small number of nodes, they expose a number of downsides that reduce their applicability in many cases. Primarily, the time required to distribute new application images to all participating nodes, set them up in the desired topology, and finally perform an analysis of the measured characteristics log files, is a time-consuming process.

Testbeds generally alleviate this situation, as they present a ready-made deployment of nodes with versatile application developer support in different regards. Application images can be deployed easily, experiments can be controlled and supervised during runtime, and measured results and log files are processed and presented to the user after an experiment for detailed analysis of the application behavior. Especially in cases where simulated behavior and measurements on real hardware do not match, the debugging capabilities of a testbed are very useful tools to determine the origin of the discrepancy. By logging application behavior and forwarding this data, the user can synoptically view encountered events and errors and derive improvements to the application from the observations.

1.2 Contribution

In the course of conducting research on WSNs, the Ubiquitous Communications Group at the Multimedia Communications Lab (KOM) of Technische Universität Darmstadt, has decided to set up the TWiNS.KOM testbed, a Testbed for a Wireless Network of Sensors. Towards achieving this goal, we have determined a set of design considerations for successful testbed operation in [1]. Following these objectives, we have developed a heterogeneous node platform, the tubicle. Supporting multiple dimensions of heterogeneity (computational power, radio protocols, sensor devices, available memory), and the possibility of simple relocation due to its portable design, the platform can be used to perform a broad range of experiments. Covering WiFi, Bluetooth, and the IEEE 802.15.4 radio standard, a variety of sensing devices operating on any of these protocols can be integrated with the nodes easily.

The node platform has been designed with special regard to allow user-centric operations, where users carrying sensor nodes, such as mobile phones or internet tablets, may also be part of the network. This is significantly different from common sensor network approaches where fixed autonomous nodes with pre-defined energy budgets comprise the main elements of the sensor network, as mobile devices can rather be assumed to be perpetually recharged by their users, while constant availability within the network cannot be guaranteed.

Our testbed comprises twenty tubicles, and offers a convenient set of support features for application developers. It allows versatile experimental analysis and evaluation by providing extensive deployment support in terms of simple

application deployment and node status analysis. The inherent debugging capabilities allow to analyze application behavior in detail and provide helpful clues to the developer to determine the origin of encountered errors. We provide a detailed assembly instruction for the tubicle platform, with all technical details required to build a tubicle from its components. Special attention is also paid to provide a step-by-step manual for compiling the operating system from scratch. Stable revisions of our current system images are also available for download on the TWiNS.KOM webpage, found at `http://www.kom.tu-darmstadt.de/twins`.

1.3 Outline

We present a detailed overview of the tubicle in Chapter 2, with details on the employed hardware devices, their interconnections and the resulting functionalities offered by the node platform. Subsequently, we provide step-by-step manuals on how to assemble and modify the hardware in preparation for their operation in the tubicle in Section 3. Optional steps are marked as such, allowing to selectively disregard them during the construction process. The process of setting the system up from scratch is presented in Section 4. Steps range from the basic bootstrapping process to compiling dedicated device drivers, e.g. for the integrated webcam or any peripherals connected to the external USB sockets. We summarize this document in Section 5.

2 The Tubicle

The first revision of the tubicle is illustrated in Fig. 2.1, made up from a 40cm (16inch) acrylic glass tube with a set of integrated hardware devices. The components are placed on a Plexiglas carrier board located in the center of the tube. Vacant sites on the carrier board allow to extend the tubicle by further peripheral devices, such as additional sensors, more sophisticated user interface elements, or audio equipment.

The tube exposes an outer diameter of 100mm (3.9inch) at a thickness of 3mm (0.12inch). Top and socket parts are both manufactured from solid aluminum, with diameters of 100mm (3.9inch) for the top part, and 140mm (5.5inch) for the socket part, respectively. A cut-out in the socket seats a printed circuit board (PCB) with power inlet and an Ethernet jack, while two USB jacks are located in the top to allow connecting additional peripherals. The enclosure provides basic protection against splashing water, and prevents the integrated components from accidental damage. A set of LEDs in the socket is used to illuminate the acrylic glass tube and the carrier board, thus indicating operation of the tubicle.

Only the connection to the power supply wall plug (or alternatively, a storage battery) is mandatory to operate the tubicle, although the rare case of recovering from an erroneous write operation to the kernel flash memory (thus rendering the tubicle unable to boot) requires an additional connection to the wired Ethernet. By the use of a Power-over-Ethernet (PoE) splitter, such as the D-Link DWL-P200, the required cabling can be reduced to a single PoE connection.

Apart from the custom housing, all components are available off-the-shelf, allowing the gentle reader to construct a tubicle without further requirements. The housing however is an optional feature to increase portability and address the protection of the integrated components; it is not necessarily required to operate the tubicle.

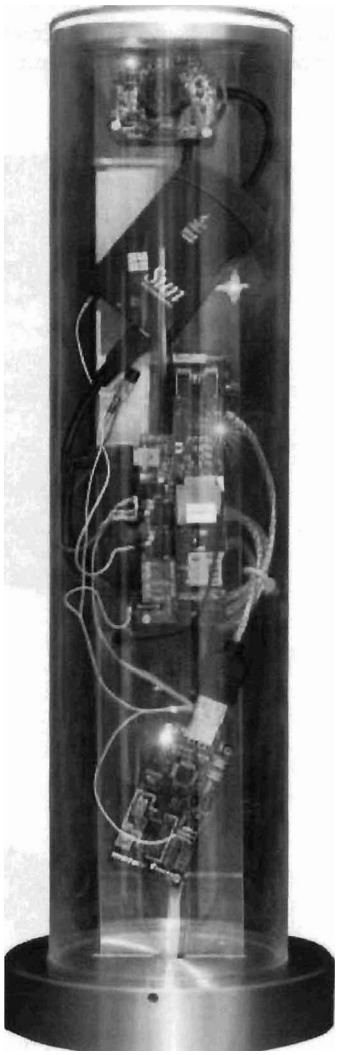


Figure 2.1: The first tubicle prototype

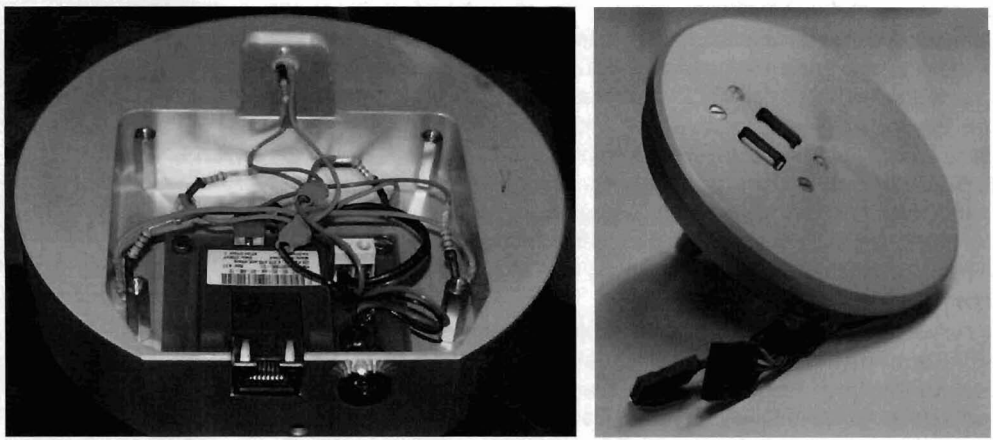
2.1 System Overview

Three commonly known sensor node platforms have been integrated within the tubicle. The low-power tmote sky sensor node hereby represents the least powerful platform with tightest resource constraints. In contrast, the Gumstix embodies the most powerful device integrated, with a multiple of the tmote's memory and processing power. Its performance is comparable to the one offered by current smartphones. The SunSPOT platform represents an intermediate system that allows to run algorithms that would exceed the computational capabilities of the tmote sky, while still being more resource-constrained than the Gumstix.

Apart from the sensor node platforms, additional devices are integrated to interconnect all nodes as well as to provide deployment support. Inside the socket, shown in Fig. 2.2(a), an XPort has been integrated, transferring data received over the Ethernet to a serial console of the Gumstix platform, as well as forwarding reset requests to any of the three devices. The Gumstix offers USB host capabilities, allowing to attach further peripherals. As only a single connector is present, the tubicle makes use of a 7-port USB hub mounted on the back side of the carrier board. At the current stage, both SunSPOT and tmote sky are attached to the hub, as well as an additional Bluetooth dongle, a webcam, and the two USB jacks in the top part, which can be seen in Fig. 2.2(b). Another jack is left available inside the tubicle to allow attaching further devices at later stages. Also, the audio connectors of the Gumstix base board are currently unconnected, but allow connection of both speakers and microphones.

An overview of the devices and interconnections is given in Fig. 2.3, additionally indicating the supported radio protocols. Four different types of connections are indicated in the figure: Reset signals are generated by the XPort and

forwarded to each connected sensor node platform. As the SunSPOT requires a reset impulse during programming, a logical OR gate has been added to the signal to allow both Gumstix and XPort to trigger the signal. A serial connection is present between XPort and the Gumstix board. It allows both remote reprogramming of the entire Gumstix platform as well as management in emergency cases when no WiFi access to the Gumstix is possible. The USB hub connects all integrated peripherals to the Gumstix, and additionally supplies power to all components.



(a) Socket (b) Top

Figure 2.2: Top and socket parts of the tubicle

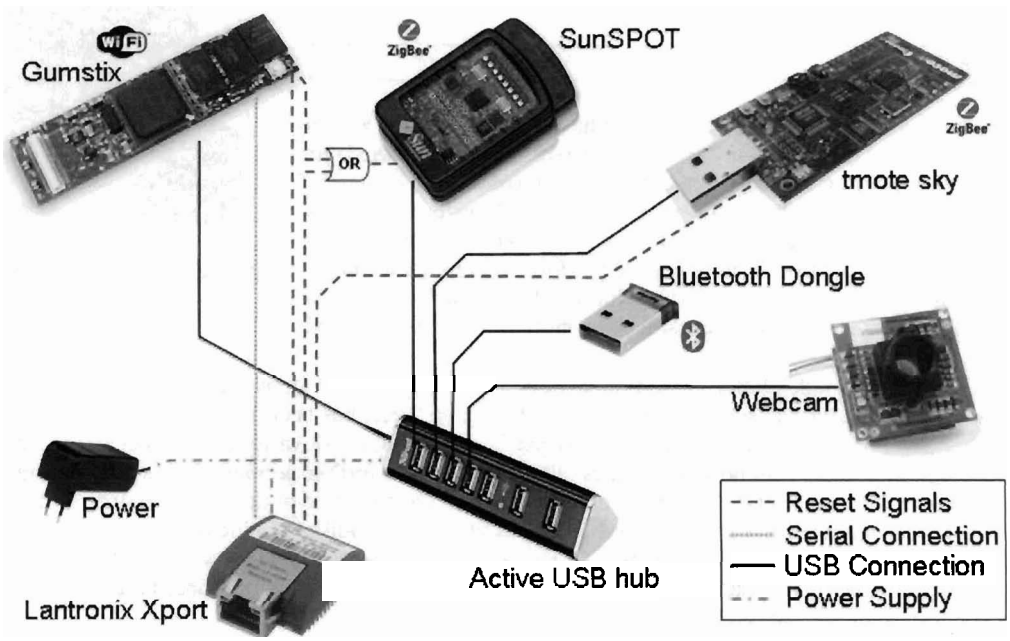


Figure 2.3: Internal structure of the tubicle

2.2 Hardware

We introduce the integrated components in the following subsections, summarizing their functions, requirements, and their task within the tubicle. The required modifications to the devices are not discussed in detail in these sections; we refer the reader to Chapter 3 for according details.

2.2.1 Gumstix Verdex

The Gumstix Verdex XL6P board is an embedded system based on the PXA270 XScale processor, running at a clock frequency of 600MHz [5]. It is fitted with 128MB of on-board RAM and a 32MB flash chip to store the Linux operating system. An integrated USB Full Speed (v1.1) host controller allows to control peripheral devices, and up to 98 general purpose I/O (GPIO) pins are available to interface external devices. The device features three levels of power management and three logic level serial ports at a board size of only 80mm x 20mm. Hirose extension headers are present on the boards to connect additional extension boards.

The audiostix2 and netwifimicroSD extension boards extend the core system by a set of functionalities. While the former adds audio input and output jacks, a micro USB connector, and soldering pads for both serial and GPIO pins, the netwifimicroSD board provides an 802.11b/g compatible wireless networking adapter, wired 10/100baseT ethernet, and a microSD memory card slot, which allows to extend the storage capacity and thus increase the space available to the operating system.

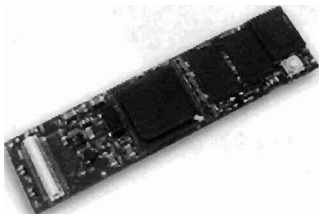


Figure 2.4: Gumstix Verdex

2.2.2 SunSPOT

Figure 2.5: SunSPOT

The SunSPOT is a sensor node manufactured by Sun Microsystems. Less powerful than the presented Gumstix platform, its ARM9 CPU is still operated at a clock frequency of 180MHz [6]. Fitted with 512kB of RAM and 4MB for application code storage, the node offers less resources than the Gumstix, but also consumes less energy during operation. In contrast to the Linux operating system run on the Gumstix, SunSPOTs natively run the SquawkVM Java virtual machine, hence only allowing the execution of Java applications.

Radio communication is handled by the on-board CC2420 radio transceiver, an IEEE 802.15.4 compliant 2.4GHz radio module [7]. The implemented radio stack is provides an implementation of 6LoWPAN [8], an energy-aware IPv6 stack for embedded systems.

The sensor board shipped with SunSPOT nodes provides a set of sensors, comprising two momentary switches, temperature and light sensors, and a 3-axis accelerometer. Eight tricolor LEDs can be used to indicate the node status to the user, and six analog inputs, and five general purpose I/O pins are available to attach external devices.

2.2.3 tmote sky

Moteiv's tmote sky platform, technically equal to Crossbow's TelosB, is a node platform based on a low-power 16-bit Texas Instruments MSP430 microcontroller, operated at 8MHz clock frequency. It offers 10kB of RAM and 48kB of code flash, rendering it incompatible with Linux-based operating systems or even complex Java virtual machines. To still run applications, special operating systems for embedded devices are necessary, such as TinyOS [9], or the Contiki operating system [10].

Figure 2.6: tmote sky

Similar to the SunSPOT, the tmote sky employs the CC2420 transceiver for radio transmissions over the IEEE 802.15.4 standard. Communication with SunSPOTs is possible by either implementing 6LoWPAN on the tmote sky, or alternatively modifying the SunSPOT radio stack to support inter-node communication. The platform additionally offers 1MB of external serial flash memory, and sensors for temperature, humidity, and solar radiation. Two extension headers allow to attach peripheral devices, and provide six analog inputs, up to four GPIO pins, and logic level I^2C and serial bus interfaces, as well as an interrupt input to wake the node up from low-power sleep modes, and a reset pin.

2.2.4 XPort

Figure 2.7: XPort Direct+

A dedicated Ethernet to serial converter device is present within the tubicle to allow for bootstrapping in emergency cases, i.e. when neither platform is in an operable state to fulfill its determined operations. The Lantronix XPort device comprises a small web server with telnet capabilities [11] that allows to remotely execute commands on the Gumstix platform, or even remotely reprogram it when the system boot scripts are damaged or the system is irreversibly stalled. The XPort also offers three dedicated GPIO pins, which have been connected to the reset signal lines of the employed node platforms. This wiring allows to remotely reset stalled nodes or selectively deactivate nodes to keep them from returning to their normal mode of operation. As the XPort has been designed to run on a lower operating voltage and requires external components, a dedicated PCB has been designed, and is presented in Sec. 3.2.6.

2.2.5 Bluetooth Dongle

To allow interaction between Bluetooth-based devices and sensors, a Hama Nano Bluetooth dongle has been integrated. The compact USB device supports Bluetooth Version 2.0 +EDR, allowing for data rates of up to 3Mbit/s. Operating in Bluetooth Class 2, the adapter is designed to achieve transmission ranges of up to 40 meters.

Attached to the Gumstix platform, applications can access the Bluetooth dongle over the USB bus, allowing interoperability with Bluetooth-based sensors, such as calendar and address book applications on mobile phones.

Figure 2.8: Hama Nano

2.2.6 Webcam

Figure 2.9: Labtec Webcam Pro

Both still images and video streams from a node's surroundings can be captured by the integrated USB webcam. We have selected the **Labtec Webcam Pro** in particular because of its known interoperability with Linux operating systems and the low price. The housing of the webcam has been removed to allow fitting it inside the tubicle, also revealing Logitech as the real manufacturer of the hardware.

The webcam operates at a resolution of 640x480 pixels at 30 frames per second, and a built-in microphone offers capabilities to also sample environmental sounds. As with most current webcams, only manual focusing is available.

The Gumstix platform provides sufficient computational power to process captured data, allowing to use the webcam as a sensor device. Versatile functions, including motion and activity area detection thus render possible.

2.2.7 USB Hub

A USB hub has been integrated to attach the presented USB devices to the Gumstix board. We have selected the **Trust HU-5870V** because it fits inside the tubicle well due to its triangular shape and offers seven ports, leaving room for prospective extensions to the nodes.

Two of these seven connectors are employed to attach the tmote sky and SunSPOT devices, and another two connect the Bluetooth dongle and the webcam, leaving three ports available for other devices.

Two USB sockets have been mounted in the top part and were connected to the available ports of the hub, allowing to easily connect new devices by the user. Finally, a single slot is available inside the platform to permanently attach further devices.

Figure 2.10: Trust HU-5870V

2.3 Software

While the SunSPOT features a dedicated Java virtual machine, and tmote sky application images also include the operating system, the Gumstix is a fully fledged embedded system with support for various operating systems. We give a

short overview of our setup in this section, but refer the interested reader to Chapter 4 for configuration and installation details. Similarly, our node management application and deployment support system is introduced in this section, supporting developers in application development and offering sophisticated logging options.

2.3.1 The Operating System

We have based our application image for the Gumstix devices on the Ångström-2007.1 distribution of the OpenEmbedded operating system, a Linux distribution specifically designed for embedded devices. Due to the limited size of the on-board flash chip, a microSD memory card has been used to increase the space for the root file system.

Besides installing a complete build tool chain, we have also cross-compiled the kernel modules required for the attached USB peripherals. This includes the rxtx and toscomm libraries for the serial communication with tmote sky and SunSPOT, the bluez library for Bluetooth communications, and the gspca module for webcam support.

We have included the jamvm Java virtual machine as an execution environment for developed applications, and added the SunSPOT and TinyOS SDKs to support data transfer and reprogramming of the attached node platforms. Applications to interface the webcam (w3cam and xawtv) and the Bluetooth dongle (the bluez utility suite) have also been added, as well as a dedicated set of developed scripts to easily deploy application images on the attached nodes.

2.3.2 Tubicle Management System

To aid developers in application deployment and analysis, a server-based solution to control the tubicles has been developed and integrated. Our Tubicle Management System (TMS) keeps track of all available nodes within the network, and indicates their current status and location on an overview page. We have implemented an experiment scheduler, which automates time slot selection for experiments and displays the current experiment queue. New jobs can easily be added to the queue and are scheduled to time slots where the requested number of nodes is available. Additional constraints, such as a preferred experiment execution time during night hours (where less cross traffic is present in the 2.4GHz band) are also considered in the node selection process. A connected database stores all finished experiments and the corresponding log files, and thus allows for both deferred experiment analysis and repetitions on demand.

2.3.3 Application Deployment

A dedicated script on the tubicle allows the Gumstix to receive and install new application images for all integrated platforms. As in the case of tmote sky and SunSPOT, only a single application can be run at a time, this operation must be triggered by TMS at the scheduled time slot. The OpenEmbedded distribution supports concurrent execution of applications on the system, assuming they do not require exclusive access to resources. TMS hence spawns new processes for each application, which either terminate normally, or will be forced to exit at the pre-defined end of the experiment runtime.

In some cases, symbols must be set in application images, such as the RF transmission power, or the local node address of tmote sky nodes. To alleviate the process of setting these values independently for each node, the corresponding SDKs were installed on the Gumstix, which can either set these values to pre-defined values, or values defined for a given experiment in the TMS input form.

To successfully execute a Gumstix application, a manifest file needs to be present in the application archive, defining the call sequence to the executable file, and any further setup commands or parameters. The process handle of the shell encapsulating the user applications is returned to the TMS server on successful application start, and saved within TMS to allow manual termination of the application at any time.

2.3.4 Debugging Capabilities

The possibility to perform node debugging is important in sensor networks, as many exceptional behavior schemes only occur when nodes are distributed and real radio communication is taking place. Therefore, a set of debugging capabilities have been integrated with the platform, allowing to support developers in many regards.

First of all, the data logged from the nodes' serial ports can be retrieved and compared. A synoptic view of status and debug messages of all participating nodes allows to monitor node interactions and replay the sequence of occurred events. Nodes that do not actively participate in experiments are also configured to monitor radio traffic by default, allowing to passively inspect the traffic on the radio, providing support in both determining erroneously disregarded packets or even falsely transmitted ones.

3 Assembly Instructions

This section describes all steps necessary to construct a tubicle from scratch. All required components are listed, and a detailed description on the assembly steps is provided.

3.1 Bill of Materials

In Table 3.1, all required materials and corresponding numbers to are listed. Prices are not quoted as they are subject to change. However, as of October 2008, the overall cost for a tubicle is around 700 Euros (about 1000$ US).

Amount	Part	Further specifications
1	Gumstix Verdex XL6P mainboard	
1	Gumstix audiostix2 and netwifiSD modules	Choose the module corresponding to your country
1	Gumstix screws and spacers set	
1	USB hub	Trust HU-5870V model, or any other 7-port hub
1	tmote sky with sensors	Or due to the lack of tmote sky availability, a TelosB
1	SunSPOT sensor node	Including the sensor board
1	Lantronix Xport Direct+	
1	2GB MicroSD card	Many 4GB SDHC cards do not work in Gumstix boards
1	Bluetooth Dongle	
1	Web Camera	Supported by Linux, e.g. the Labtec Webcam Pro

Table 3.1: List of required components

Amount	Component	Distributor	Part No
1	LF33CDT Voltage Regulator	Farnell	1087187
1	SMD Resistor 10k 1206	Farnell	9335765
1	PCB Terminal Block 2pin 5mm grid	Farnell	151789
1	SMD Capacitor 2u2 0805	Farnell	9527702
2	SMD Capacitor 100n 0805	Farnell	1362552
1	Secondary Plug 2.1mm	Farnell	1453757
1	Header/socket 0.1"	Farnell	9728910
2	NPN Transistor	Farnell	1467880
2	Resistor 4k7 0309	Farnell	9338829
1	1N4148 diode	Farnell	9565124
2	USB connector male A	Farnell	1308875
4	Pin header 2row 0.1"	Reichelt	SL 2X36G
1	Pin header 1row 0.1"	Reichelt	SL 1X36G 2,54
1	Secondary Connector 2.1mm	Reichelt	HEBL M21
2	Reset and serial connectors	Reichelt	PS 25/3W BR
1	USB connectors female mountable	Reichelt	AK 674/2
2	USB extension cord 0.3m	Reichelt	AK 669-0,3
2	USB cable with Mini-B connector	Reichelt	AK 673-A

Table 3.2: List of required discrete electronic components

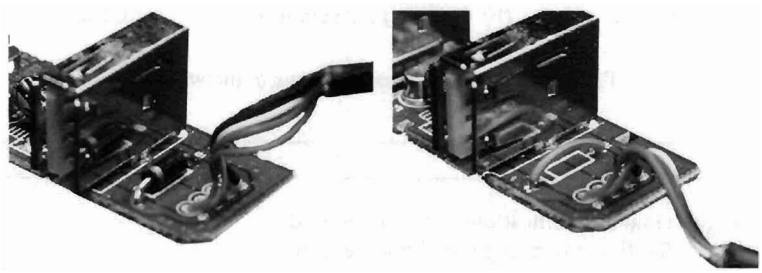
Besides some parts that are expected to be present in most electrical workshops (such as heatshrink and cable straps), a set of further discrete components are required to successfully complete a tubicle. Details about these components, their amounts, distributors and order numbers are listed in Table 3.2. Optional extensions, including mounting material, LEDs, and the aluminum socket and top parts, are not included in the table.

3.2 Hardware Modifications

This section provides detailed descriptions about all required modifications that need to be performed on the integrated devices. Keep in mind that basic mechanical and soldering skills are a prerequisite for some of the steps.

3.2.1 Trust USB Hub

In a first step, the triangular front part of the USB hub must be removed, as modifications to the PCB are necessary for operation. When the cover has been removed, take the PCB out of the enclosing case. First, the diode next to the USB input connection needs to be removed as it disallows powering the host device. Replacing the diode by a simple wire allows to supply the Gumstix via its USB port and thus unnecessitates a separate power supply cable. This first step is indicated in Fig. 3.1.



(a) Original state (b) Diode replaced

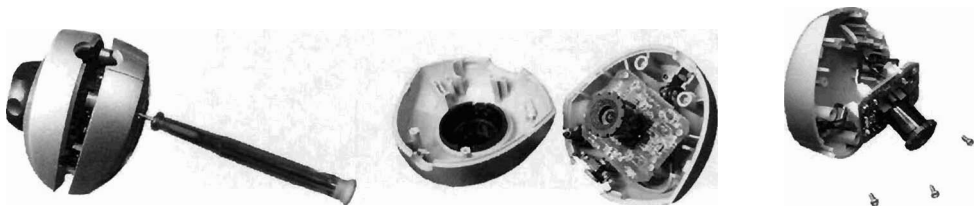
Figure 3.1: Required modifications to the USB hub

Subsequently, the USB cable needs to be removed as it is way too long and also is fitted with a male A type connector, while the Gumstix would require it to expose a mini B type plug. Hence, completely desolder the USB cable, and replace it by a cable with mini B connector, which has been cut down to a length of about 30 centimetres (12inch). Make sure to maintain the color scheme when connecting the shortened replacement cable.

Eventually, close the case again, and put the new USB cable through the opening that held the original USB cable. Depending on the diameter of the conneted cable, it might be necessary to provide some pull relief.

3.2.2 Webcam

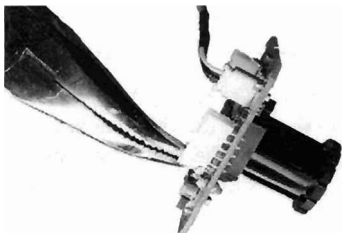
Start the disassembly process of the webcam by removing the screws on the back side of the case. Then carefully pry the case open using a small screwdriver or similar utility. All screws that affix the PCB to the case must be removed in a third step, and the PCB extracted from the housing. These steps are shown in Fig. 3.2.



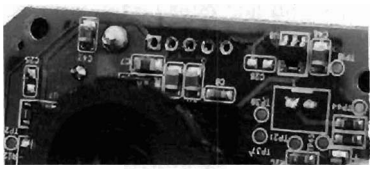
(a) Opening the case (b) Removed front part of case (c) Removed back part

Figure 3.2: Removing the case of the webcam

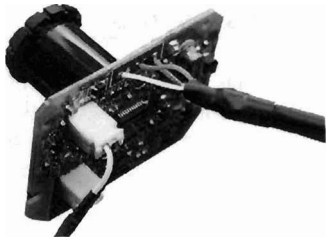
Now remove the plastic socket piece carefully, so only the pin connectors remain on the PCB. After removing these pins, and wiping the remaining solder lead bits, shorten the USB cable to about 15cm (6inch), and directly solder the shortened cable to the solder eyes on the PCB. Hereby, make sure to maintain the proper color order (white, n/c, green, black, red), shown in Fig. 3.3.



(a) Removing the connector



(b) Cleaning the solder eyes



(c) Connecting the shortened cable

Figure 3.3: Required modifications to the webcam

3.2.3 tmote sky

The tmote sky does not require major modifications to the integrated circuits or SMD components. To allow mounting it withing the tubicle more easily, the battery compartment must be removed. This is done easily by desoldering the two pins next to the USB connector.

In a second step, two-row standard 0.1" pin headers need to be soldered to the extension header pins. When no further extensibility is required, the six-pin header (which comprises the reset signal) can be solely populated, while the additional ten-pin connector allows for further extensions at later stages. The resulting board with both connectors populated is depicted in Fig. 3.4.

Figure 3.4: tmote sky with extension headers

3.2.4 Gumstix Verdex

The Gumstix Audiostix2 extension board does not natively forward reset pins, nor does it feature a serial console. Hence, both must be connected to the board to enable remotely resetting the node as well as remote deployment of new OS images and emergency node bootstrapping. To fulfill these requirements, the following steps are necessary.

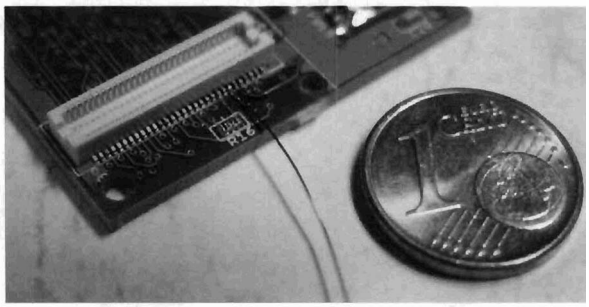


Figure 3.5: Attaching the reset wire to the Hirose connector

Pin 37 on the 60-pin Hirose connector present on the audiostix2 base board is connected to the reset signal of the Gumstix platform. As it is not connected to any other component on the base board, the only way of interfacing external signals to this pin lies in soldering a thin copper wire to the Hirose connector. This is shown in detail in Fig. 3.5.

To additionally allow the recovery console connect to the Gumstix, a serial port needs to be connected as well. Herefore, connect the RXD, TXD, and GND pins of the FFUART interface to a connector. To simplify construction, we have employed a 0.1" connector simply affixed to the PCB using sticky tape. In addition to the reset signals and the serial port, a GPIO line also needs to be forwarded to the connector, in order to reset the SunSPOT platform during programming. We have arbitrarily chosen the LDD15 pin on the base board, which can be accessed via `/proc/gpio/gpio73`. The resulting connector attached to the PCB and providing the serial, reset, and GPIO connection, is shown in Fig. 3.6.

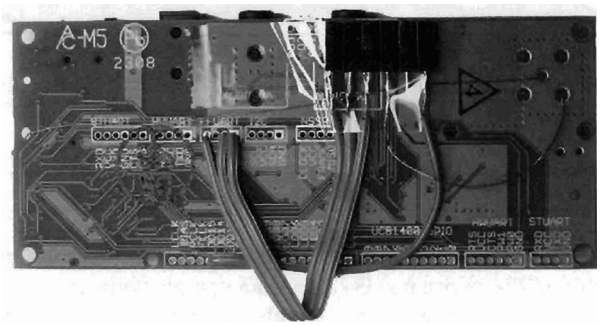
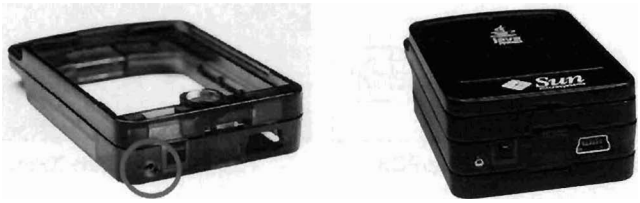


Figure 3.6: Required modifications to the Gumstix

3.2.5 SunSPOT

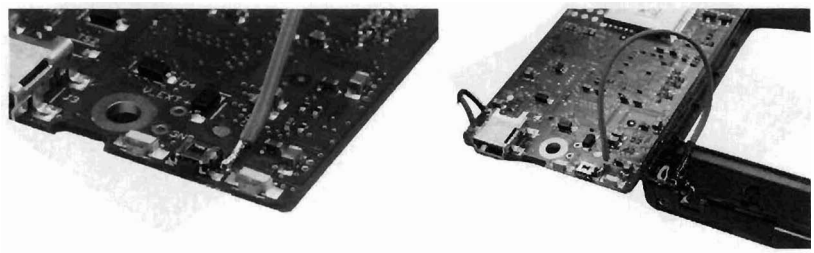
To allow remote reprogramming of the SunSPOT, its reset signal must be triggered to the Gumstix. As the required signal is present on the SunSPOT reset button already, a simple connection to an external connector is required. Therefore, open the SunSPOT device by removing the screw and taking apart all contained components. The main board, featuring the CPU, is attached to the plastic frame by molten plastic pins, which can however be easily separated using a sharp knife.



(a) After drilling the hole (b) With the connector fitted

Figure 3.7: Required case modifications of the SunSPOT

When the main board is exposed, drill a hole next to the reset button and affix a simple 0.1" jack connector using two-component adhesive. Hereby, be careful to maintain sufficient distance between the connector jack and the diode on the main board. Additionally, provide insulation by enclosing the bare metal with some heatshrink. These steps are shown in detail in Fig. 3.7



(a) Connecting to the reset button (b) The external reset input

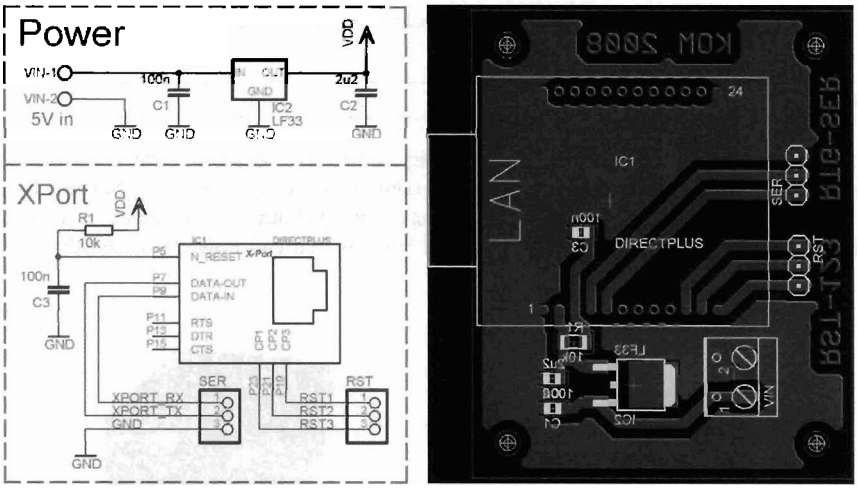
Figure 3.8: Required PCB modifications of the SunSPOT

Both solder pads of the reset button pointing inward the main board PCB expose the reset signal, while the connections towards the rim of the PCB are at ground level. Hence, make sure to solder a wire from the newly populated connector

to one of the inward pads. A detailed photo of this step is shown in Fig. 3.8. Subsequently, reassemble all components and fit them together again with the screw.

3.2.6 XPort

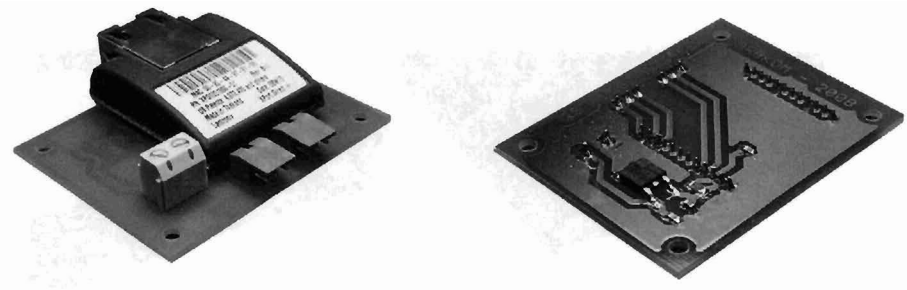
To allow sending reset signals to the devices and remotely accessing the Gumstix over its serial port, an XPort Direct+ has been integrated with the tubicle. The device however does not operate without some external circuitry, hence a PCB has been designed to provide some space for these parts, and also allow for more convenient mounting.



(a) Schematic of the XPort PCB (b) Layout of the XPort PCB

Figure 3.9: The PCB design for the XPort Direct+

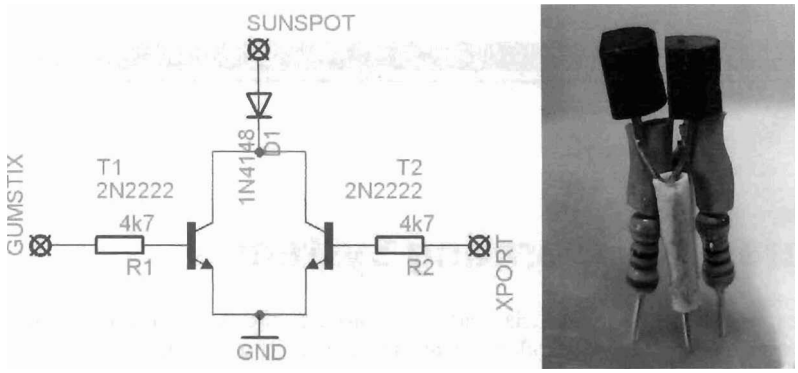
Schematic and PCB layout of the board are presented in Fig. 3.9, and also available for download on the tubicle website. It is recommended to check the output voltage of the linear voltage regulator before soldering the XPort onto the PCB. It should not exceed the nominal voltage of 3.3 volts. According photos of the populated board are shown in Fig. 3.10



(a) Top view (b) Bottom view

Figure 3.10: The populated PCB for the XPort Direct+

The logical OR gate that connects both XPort and Gumstix to the reset pin of the SunSPOT has not been assembled on a dedicated PCB, but instead corresponding pins of the discrete components were simply connected. Both circuit diagram and realization of the OR gate are shown in Fig. 3.11. No additional power supply is necessary, as the SunSPOT internally provides a pull-up resistor to the reset pin.



(a) Schematic (b) Realization

Figure 3.11: The logical OR gate

3.2.7 Remaining Interconnections

Before operation is possible, the last step comprises interconnecting all components. Interconnections basically follow the structure depicted in Fig. 2.3, and will be explained in more detail in this section.

To cater for the power supply, cut the USB hub's original power adapter cable in two pieces, leaving around 20cm (8inch) of cable to the plug, and the remaining length on the adapter side. Solder the DC connector onto the adapter side end, and optionally fill the plug with hot glue to increase its mechanical stability. On the other side, connect the plug to the USB hub and attach the wires into the terminals on the XPort PCB. Similarly, connect the LED cabling (if you chose to integrate LEDs and the corresponding resistors into the socket) to the terminal on the PCB.

Two further connectors are present on the XPort PCB; let's start with the one that features the reset pins for all three integrated node platforms (marked RST) and needs to be connected accordingly. The SunSPOT reset pin needs to be connected to one resistor of the discrete OR gate described in the last section. The Gumstix reset is connected by directly attaching the reset pin to the connector that features the connection to pin 37, while the tmote sky reset pin is located on pin 6 of the 6-pin extension header. Finally, connect the LDD15 pin to the second resistor of the discrete OR gate, and the common emitter of both transistors to ground level, to also allow the Gumstix to reset the SunSPOT. After connecting the reset pins, the Gumstix must be connected to the XPort via the serial connection. Therefore, solder the RXD, TXD, and GND pins from the XPort PCB to the 0.1" pin header that connects to the Gumstix board.

Eventually, plug all devices with USB connectivity into the USB hub and enjoy your brand new tubicle.

4 Software and Operating System

The operating system deployed on the Gumstix platform is based on the OpenEmbedded (OE) Linux distribution[1]. To assist in the installation process, we present all necessary steps to create a fully functional system in Sections 4.2 to 4.5. Alternatively, new tubicles can be set up easily without configuring a dedicated cross-compilation host by following the six steps presented in Sec. 4.1.

4.1 Step-by-Step Setup of a Tubicle

In preparation for setting up a new tubicle, make sure to download the current SD card image (tubicle_sdhc_backup.tar.gz) from the testbed website. Under a Linux operating system, perform the following steps:

1. Prepare the SD card by formatting it using the ext2 file system
2. Extract the downloaded archive to the root directory of the SD card
3. Run sudo chown -R 0:0 *<mount point of SDHC card>* to fix the file ownerships
4. Make sure to note down the WiFi adapter's MAC address of the Gumstix, and enter it in newGum/files/mappings if not already there
5. Insert the just prepared SD card into the Gumstix and boot the system
6. Login via the serial console or SSH (use root as user name, and gumstix as the corresponding password), and execute /media/card/newGum/flashGum

By completing these steps, the tubicle has been configured properly and should be ready to operate. Applications can either be installed manually (as shown in Sec. 4.5) or using the Tubicle Management System.

4.2 Setting up a Development System (on Debian/Ubuntu)

On most current (as of November 2008) distributions of both Debian and Ubuntu Linux, the /bin/sh shell is linked to /bin/dash. However, dash is known to cause corruption in OE files, resulting in build images that fail to boot. Running sudo dpkg-reconfigure dash and changing /bin/sh to link to /bin/bash resolves this issue.

To fully perform the build process, a set of additional packages are required on the development system. Make sure to install them in the package manager by executing the commands:

```
apt-get install texinfo libncurses5-dev subversion help2man diffstat texi2html cvs gawk
apt-get install python-psyco python-dev python-pysqlite2 gnome-terminal
```

Subsequently, it is essential to create necessary Gumstix development directories and download the current version of the Linux distribution by issuing the following sequence of commands (substituting *[username]* in the second command by your user name):

```
sudo mkdir -p /usr/share/sources
sudo chown -R [username]:users /usr/share/sources/
mkdir ~/gumstix/
cd ~/gumstix
svn co https://gumstix.svn.sourceforge.net/svnroot/gumstix/trunk gumstix-oe
```

Now, a set of required environmental variables and paths for the bash environment need to be set. This can be accomplished by executing the following commands:

[1] http://www.openembedded.org

```
echo 'source ~/gumstix/gumstix-oe/extras/profile' >> ~/.bashrc
source ~/gumstix/gumstix-oe/extras/profile
echo 'export PATH=$PATH:~/gumstix/gumstix-oe/tmp/cross/bin' >> ~/.bashrc
export PATH=$PATH:~/gumstix/gumstix-oe/tmp/cross/bin
```

After completing the last step, a full Gumstix OpenEmbedded development host has been set up. A minor set of changes need to be integrated to adapt the distribution to the tubicles; as the tubicle image requires a slighly modified root file system and kernel, create a new directory called ~/gumstix-oe/user.collection. Due to a higher priority number, this directory overrides default recipes. Its contents are found in the user.collection archive, which can be downloaded from the testbed website. After downloading, extract the contents into ~/gumstix-oe/user.collection.

The default Bitbake configuration file, bitbake.conf, is located in the org.openembedded.snapshot/conf subdirectory, and contains configuration information for all bundles contained in the collection. It needs the following modification: Search for the line starting with GNOME_TERMCMDRUN and replace the terminating ${SHELLRCCMD} by ${SHELLCMDS}.

To speed up compilation on SMP (multi-core processor) machines, set PARALLEL_MAKE and BB_NUMBER_THREADS in ~/gumstix/gumstix-oe/build/conf/site.conf according to your system.

At last, the Bitbake executable shipped with the distribution must be run to build a toolchain, application programs, the kernel image and root file system for the Gumstix platform. This process takes quite some time and can be triggered by executing the following commands:

```
cd ~/gumstix/gumstix-oe/
bitbake gumstix-tubicle-image
```

Additional information regarding both the operating system and the underlying hardware can be found on the Gumstix webpage[2]. Pre-built Linux images and software bundles can be found at the feeds section of the Gumstix page[3]. Instead of the commonly employed make application, OpenEmbedded uses bitbake as its build tool. It hence requires recipes (with an .bb extension) instead of makefiles. Recipes are bundled in collections, where each collection provides a priority number allowing to override settings in other recipe files with smaller priorities. The Gumstix distribution of OE, as used for the tubicles, includes the following collections:

- gumstix-oe/org.openembedded.snapshot, which provides basic features and hence has the lowest priority
- gumstix-oe/com.gumstix.collection, with an extended set of applications
- gumstix-oe/user.collection, which contains tubicle-specific changes by the user/developer

4.3 Customizing OpenEmbedded

This section provides some more detailed descriptions of which changes have been made to the user.collection recipe. It includes both modifications to the kernel as well as the root file system (rootfs) prior to installation, and a guideline to application installation and removal during normal operation.

Configuring and Building the Kernel Image

To make modifications to the kernel configurations, it is recommended to use the menuconfig user interface. Although the kernel comes nicely configured for tubicles, developers might want to add additional modules for hardware that is connected to the external USB ports, or change some other kernel parameters. To set the kernel configuration up accordingly, run the following commands:

```
cd ~/gumstix/gumstix-oe/
bitbake gumstix-kernel -c menuconfig
```

Make sure to also save the new configuration (the menuconfig user interface offers a convenient function to save the kernel configuration) to a location within the user.collection subdirectory. In this document, we assume the configuration file has been placed in ~/gumstix/gumstix-oe/user.collection/packages/linux/gumstix-kernel-2.6.21/gumstix-custom-verdex/defconfig. Exact path names may vary with the used kernel version. To now compile the kernel and the corresponding modules, simply execute bitbake gumstix-kernel -c rebuild. The resulting kernel image can then be found in ~/gumstix/gumstix-oe/tmp/deploy/glibc/images/gumstix-custom-verdex/.

[2] http://www.gumstix.net

[3] http://www.gumstix.net/feeds

Building the Root File System (rootfs)

Before generating the root file system, all packages needed by gumstix-tubicle-image need to be created prior to assembling them to the rootfs image. Once compiled, issuing the following command will place the full rootfs image into ~/gumstix/gumstix-oe/tmp/deploy/glibc/images/gumstix-custom-verdex/:

```
bitbake task-base-gumstix -c rebuild && bitbake gumstix-tubicle-image -c rebuild
```

4.3.1 Installing and Uninstalling Packages on the Gumstix

OpenEmbedded uses the Itsy package management system, which is based on packages in the ipkg format. The use of Itsy requires some prerequisites to be fulfilled:

- After connecting to the tubicle via SSH, make sure to change to the destination directory /media/card before triggering software installations. Some ipkg packages will not link properly otherwise.
- The installation directory must not contain any symbolic links.

To install packages on the gumstix, perform the following steps, substituting <*package*> for the package name:

```
cd /media/card
ipkg -d mmc install <package>
ipkg-link mount /media/card
```

Similarly, remove packages from the system is done as follows:

```
ipkg-link remove <package>
ipkg remove <package>
```

When using these methods, only the specified packages are removed, but not the corresponding dependencies. If these should be removed as well, use ipkg remove -recursive <package>.

4.3.2 Cross-Compiling Applications and Kernel Modules for the Gumstix

Although native compilation on the Gumstix is possible, it does generally exhibit far lower performance than compiling on a dedicated cross-compilation system. Hence, after ensuring that all required tools are available, simply provide the target platform during configuration on the host system (./configure –host=arm-angstrom-linux-gnueabi) to create executables that can be run on the Gumstix.

Cross-Compiling the Webcam Kernel Module

A set of three steps are required to install the kernel module for the employed Labtec Webcam Pro. As support for this device (and many other webcams) is present within the gspca driver, it is used in the following steps. Compiling other drivers for different webcam models can however be performed in a similar manner.

To install the gspca module, make sure to get the latest release from its website[4] and extract the contents into any directory of your choice. In a next step, edit the contained Makefile and change both the KERNELDIR and KERNEL_VERSION to the current values, as determined in Section 4.3. In a next step, invoke the compiler with the following command:

```
make ARCH=arm CROSS=arm-angstrom-linux-gnueabi CC=arm-angstrom-linux-gnueabi-gcc \
LD=arm-angstrom-linux-gnueabi-ld
```

Finally, copy the resulting gspca.ko file to the /lib/modules/*kernel_version*/kernel/drivers/media/video subdirectory on the Gumstix.

[4] http://mxhaard.free.fr/spca50x/Download/

Grabbing Webcam Images and Conversion to PGM

In the current Gumstix feed (R318 as of November 2008), there is no pre-compiled binary for any utility that allows to grab pictures from the webcam and subsequently save them to disk. Hence, we need to bitbake the w3cam tool and ImageMagick to perform this task. Therefore, execute bitbake w3cam imagemagick tiff on the host system, and copy all resulting files from the gumstix-oe/tmp/deploy/ subdirectory to the /media/card subdirectory on the Gumstix.

Now, log into the Gumstix via SSH and install the newly created packages by running the following commands (substituting the version numbers for the ones just generated). The last call triggers the webcam to capture a still image and save it to filename.pgm for testing purposes. You may copy it to your local hard disk to view it with any image processor tool.

```
cd /media/card
ipkg -d mmc install w3cam_0.7.2-r0_armv5te.ipk
ipkg -d mmc install libtiff3_3.7.2-r3_armv5te.ipk
ipkg -d mmc install imagemagick_6.3.5-10-r1_armv5te.ipk
ipkg -d mmc install imagemagick-dev_6.3.5-10-r1_armv5te.ipk
ipkg-link mount /media/card
vidcat -f ppm | convert - filename.pgm
```

4.4 Interfacing the SunSPOT and the tmote sky

To connect to the SunSPOT, the SPOTMangager application must be downloaded from Sun's website[5] and installed on the development system. Then simply copy the /sdk folder to /media/card/opt/SunSPOT/sdk on the Gumstix, a directory referred to as sunspot.home from now on. Then create the /home/root/.sunspot.properties file on the Gumstix and insert the following lines:

```
sunspot.home=/media/card/opt/SunSPOT/sdk
sunspot.lib=${sunspot.home}/lib
spot.library.name=transducerlib
```

Accessing Serial Connections in Java

To connect to the SunSPOT, Java uses JNI calls to access the serial connection. As direct communication is not supported by the kernel, the rxtx library must be cross-compiled to run on the Gumstix. A set of minor modifications are essential here, hence extract its source code, located in the SunSPOT/sdk/external-src directory and copy both the serial-port-name patch and the select-retry patch, found on the testbed website, to the same location. You might have to edit rxtx-2.1-7r2-select-retry.patch and change define UTS_RELEASE "2.6.21" to the appropriate kernel version of the Gumstix.

Afterwards, apply the patches and initiate the compilation process by running the following commands on the cross-compilation host:

```
patch -i rxtx-2.1-7r2-select-retry.patch -p 1
patch -i rxtx-2.1-7r2-addSunSpot_SerialPortName.patch -p 1
./configure --host=arm-angstrom-linux-gnueabi
make arm-angstrom-linux-gnueabi/librxtxSerial.la

arm-angstrom-linux-gnueabi-gcc -shared \
arm-angstrom-linux-gnueabi/.libs/fuserImp.o \
arm-angstrom-linux-gnueabi/.libs/SerialImp.o \
-lpthread -Wl,-soname -Wl,librxtxSerial-2.1-7.so \
-o arm-angstrom-linux-gnueabi/.libs/librxtxSerial-2.1-7.so
```

Finally, copy the newly generated RXTXcomm.jar and arm-angstrom-linux-gnueabi/.libs/librxtxSerial.so files to the library directory on the Gumstix, located at *sunspot.home*/sdk/libs

Install Apache Ant on the Gumstix

Apache Ant is required to download new application suites to the attached SunSPOT device. Hence, download the current distribution from its webpage[6] and extract it on the Gumstix into the /media/card/opt/ant directory. Next, add /media/card/opt/ant/bin to your PATH environment variable by appending the location to your local profile:

[5] http://www.sunspotworld.com/SPOTManager

[6] http://ant.apache.org/bindownload.cgi

```
echo 'export PATH=\$PATH:/media/card/opt/ant/bin/' > /etc/profile
```

Installing the Java Virtual Machine

As the current (as of November 2008) R318 release of OpenEmbedded does not provide a Java Virtual Machine, while the older R316 release does, load the corresponding package from the R316 repository by executing the following commands on the Gumstix:

```
wget http://gumstix.net/feeds/archive/316M/glibc/ipk/armv5te/jamvm_1.5.0-r0_armv5te.ipk
ipkg -d mmc install jamvm_1.5.0-r0_armv5te.ipk
ipkg-link mount /media/card
```

Communication with the SunSPOT

To upload new applications to the SunSPOT, which is connected to the USB port /dev/ttyACM0, copy the image.suite file of the desired application into the sunspot.home directory on the Gumstix and execute:

```
ant flashapp -Dport=/dev/ttyACM0
```

The tmote sky Serial Forwarder

TinyOS provides a set of tools that allow forwarding data received from the tmote sky. Prior to installation on the Gumstix, download the latest revision of TinyOS from the website[7] and extract the archives on the development host in the /opt directory. Then compile the applications by executing the the following commands:

```
cd /opt/tinyos-2.1.0/support/sdk/c
./bootstrap
./configure --host=arm-angstrom-linux-gnueabi
make
```

Finally, copy the created executables prettylisten, seriallisten, serialsend, sf, sfsend, sflisten to the Gumstix, preferably into the /media/card/opt/tmote/tinyosTools directory, and test the connectivity by running:

```
sf 9003 /dev/ttyUSB0 115200 &
prettylisten localhost 9003
```

4.5 General Information

After introducing the sequence to set a tubicle system up from scratch in the preceding sections, we have developed a set of scripts to automate frequently performed steps, as well as collected some general information about the software installation on the tubicle.

Directory Structure on the SD Memory Card

The SD memory card (or SDHC in case of capacities of four gigabytes and more), plays a significant role for setting up the tubicles. All software used is preinstalled there, and briefly presented in the following:

- newGum/ contains setup-scripts, tubicle-specific configuration files and the current Gumstix kernel image

  flashGum herein is a shell script to update the Gumstix firmware without further interaction

  initGum is a script which is run after every system boot-up to setup the tubicle and the attached platforms

- opt/ contains a variety of required utilities

  remoteDeploy is a client script to remotely install applications on the Gumstix, SunSPOT, or tmote sky

- The remaining directories /etc, /lib, /usr contain pre-installed packages

[7] http://www.tinyos.net

Sensor Node Application Deployment

To deploy applications on the Gumstix or any of the attached motes, you only need the remoteDeploy.sh script, available from the testbed website, and SSH/SCP installed on your machine. To eliminate the need to enter a password, it is recommended to provide a copy of the SSH private key in ~/.ssh/id_rsa_gumstix for automated authorization. The usage instructions of the script are as follows:

```
Usage: remoteDeploy.sh FILE TUBICLELIST|LISTFILE
    FILE has to be *.exe for tmote sky applications
                   *.suite for SunSPOT application suites
                   *.tar.gz for zipped tar archives including a "runner.sh" manifest script
                   newGum.tar.gz for kernel and rootfs updates
    TUBICLELIST is a space seperated list of IP addresses
    LISTFILE is a space/tab/newline seperated file of IP addresses
    example: ./remoteDeploy.sh foo.exe "10.0.0.11 10.0.0.12 10.0.0.13"
```

Flashing the Gumstix over Serial Line or Xport

In very rare situations, e.g. when the kernel image has become corrupted during operation, the Gumstix will not boot any longer, hence the corresponding image cannot easily be flashed from the console. In this situation, the XPort device can be used to deploy the image over the connected serial line. On the tubicle website, you can find a kermit script which flashes the Gumstix over either a direct serial line, or an Xport, if it is installed in a tubicle. To successfully recover your tubicle, perform the following steps:

- Download tubicle_sdhc_backup.tar.gz and gumstix_recover_xport_rs232
- Extract newGum/images/rootfs.jffs2 and uImage.bin from the archive
- Change line 2 and 3 in the script file according to the desired interface, i.e. whether you want to use the XPort or a serial line
- Run kermit gumstix_recover_xport_rs232

The transmission of the entire kernel image takes up to 20 minutes, so it is recommended to stay patient to avoid persistent damage to the bootloader. Recovering from an overwritten bootloader is only possible by the use of JTAG.

5 Summary

In this technical report, we have briefly introduced our motivation to design and construct the tubicle platform. Being heterogeneous in various dimensions, tubicles allow us to run practical experiments in both sensor network foundations and higher-level applications on real hardware. Applications can be run on any of the integrated platforms, and by integrating three hardware platforms, it is even possible to evaluate several applications at the same time.

We have presented detailed step-by-step instructions for building a tubicle from scratch, including a detailed bill of materials, and many photographs to support the user in performing the necessary soldering steps. Our design relies on integrating the hardware into a tube, however various further implementation options can be thought of.

At the current stage, twenty tubicles are deployed in the rooms of the Multimedia Communications Lab of Technische Universität Darmstadt, and used as both the versatile TWiNS.KOM sensor network testbed, as well as a supplier for context-aware communications, where plentiful information is required to decide on a user's current context [12].

Acknowledgements

Special gratitude is dedicated to Frank Jöst and the mechanic's workshop of the department of Electrical Engineering and Information Technology at Technische Universität Darmstadt, especially Walter Creter, who offered help in word and deed, and provided prototypical designs whenever needed. Photographs of our first tubicle have been kindly provided by Hans-Jürgen Weber.

This work has been partly supported by the German Research Foundation (DFG) within the Research Training Group 1362 "Cooperative, Adaptive and Responsive Monitoring in Mixed Mode Environments". Partial support was also given by the Hessian Ministry of Higher Education, Research, and the Arts (HMWK) within the LOEWE CASED initiative. This work would not have been possible without the sponsorship of the Adolf-Messer Foundation.

References

[1] A. Reinhardt, M. Kropff, M. Hollick, and R. Steinmetz, "Designing a Sensor Network Testbed for Smart Heterogeneous Applications," in Proceedings of the Third IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp), 2008.

[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A Survey on Sensor Networks," IEEE Communications Magazine, vol. 40, pp. 102–114, 2002.

[3] B. Warneke, M. Last, B. Liebowitz, and K. S. J. Pister, "Smart Dust: Communicating with a Cubic-Millimeter Computer," Computer, vol. 34, no. 1, pp. 44–51, 2001.

[4] IEEE Std, "802.15.4 Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-rate Wireless Personal Area Networks (LR-WPANs)," Online: http://www.ieee802.org/15/pub/TG4.html, 2006.

[5] Gumstix Inc., "Gumstix - Way Small Computing," Online: http://www.gumstix.com, 2008.

[6] Sun Microsystems Inc., "Project SunSPOT - Sun Small Programmable Object Technology," Online: http://www.sunspotworld.com, 2008.

[7] Texas Instruments Inc., "CC2420: 2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver (Rev. B)," Online: http://www.ti.com/lit/gpn/cc2420, 2007.

[8] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," RFC 4944 (Proposed Standard), Online: http://www.ietf.org/rfc/rfc4944.txt, 2007.

[9] University of California, Berkeley, "TinyOS – An Open-Source Operating System Designed for Wireless Embedded Sensor Networks," Online: http://www.tinyos.net, 2008.

[10] Swedish Institute of Computer Science, "Contiki – A Memory-Efficient Operating System for Embedded Smart Objects," Online: http://www.sics.se/contiki, 2008.

[11] Lantronix Inc., "Lantronix XPort DirectX+ Embedded Device Server," Online: http://www.lantronix.com, 2007.

[12] J. Schmitt, M. Hollick, and R. Steinmetz, "Der Assistent im Hintergrund: Adaptives Kommunikationsmanagement durch Lernen vom Nutzer," PIK II/2007 - Current Trends in Network and Service Management, 2007.