Wireless sensor network administrative management

4

M. Kalochristianakis[*], *E. Varvarigos*[†]
[*]Technological Educational Institute of Crete, Crete, Greece, [†]University of Patras, Achaia, Greece

4.1 Introduction

The proliferation of wireless sensor network (WSN) technologies is challenging administration practices since the latter often lack standard solutions or established management practices. Despite the potential value of managing remote, scalable, or productive installations, there are currently very few integrated management systems that support WSN infrastructures, besides standard workstations and smart devices. Traditional network management priorities, such as fault, configuration, security, performance, accounting management, etc., need to be complemented by additional ones, such as energy conservation policing, code delivery, software updates, and topology consciousness in the case of WSNs. Most commercial infrastructure management (IM) systems use remote client or agent modules that convey network management (NM) and systems management (SM) functionality. Agents may vary in terms of operation, functionality, or presentation; they may run as user or system processes, support flexible types of installation, and they may support complex configurations and expose extensive or minimal graphical interfaces to users. IM agents typically exploit a family of management protocols by the Desktop Management Task Force (DMTF) and the Internet Engineering Task Force (IETF), including WBEM (Thompson, 1998), CIM (Tosic and Dordevic-Kajan, 1999), WMI (Microsoft technet, n.d.), SNMP (Chaparadza, 2005), and others. The overall complexity and data volume that come with such technologies is typically beyond the capabilities of WSN terminal stations and, unless management standards are established for WSNs, most IM solutions may not include them in the near future. The few available Free, Open Source (FOSS) IM systems employ customizations and extensions in order to apply integrated management for wireless networks. The rest of the chapter describes how WSNs can be managed by means of extending an open-source IM platform such as OpenRSM (Karalis et al., 2009)—that is, how administrators can implement usable, custom, high-level IM use cases for WSNs. The chapter choses TinyOS as the WSN platform but the aim is to elaborate on a general-purpose methodology that can be applied for any type of WSN technology.

TinyOS (Levis, 2006) is an open-source, component-based operating system for the construction of WSNs that has gained in popularity in recent years. It relies on systems-on-chip with communication, computation, and sensing capabilities. Other similar operating systems for WSNs are MagnetOS (The MagnetOS Operating System, 2012), MantisOS (Bhatii et al., 2004), CONTIKI (Dunkels et al., 2004), SOS (Han et al., 2005), PUSPIN (Lifton et al., 2002), and CORMOS (Yannakopoulos and Bilas, 2005). Recent surveys of networked sensor technologies can be found in Dwivedi et al. (2009) and Chatzigiannakis et al. (2007). TinyOS started as a collaboration between the University of California, Berkeley, in co-operation with Intel Research and Crossbow Technology (Crossbow technologies, 2012), and has grown to an international consortium, The TinyOS alliance (2012). The system offers libraries and tool chains for all the major families of embedded processors, and can thus build and deploy applications for various types of boards. Like any operating system, it hides the low-level details of the WSNs (Levis et al., 2005) and provides appropriate APIs for the required abstractions—that is, packet communication, routing, sensing, actuation, and storage. It is a monolithic operating system since it uses a component model at compile time and a static image at runtime. It is also completely non-blocking and supports a single stack. All I/O operations that last longer than a few hundred microseconds are asynchronous and have callbacks. To enable compiler optimizations the Network Embedded System C (NESC) language has been created in order to statically link callbacks, called events according to the philosophy of TinyOS. NESC was based on C and thus supports a syntax similar to that of C and also uses data types that correspond to basic C types. For example, in MICA and TELOS motes, the integers are 16 bits wide while for INTELMOTE2 they are 32 bits. Being non-blocking enables the system to maintain high concurrency with a single stack but it forces programmers to write complex logic by implementing many small event handlers. TinyOS is capable of supporting complex programs with low memory requirements; many applications fit within 16 KB of memory and the core OS is only 400 bytes. To support larger computations, TinyOS provides tasks that are similar to windows deferred procedure calls and interrupt handler bottom halves. Thus, a component can post a task, which the OS will schedule to run later. Tasks are non-preemptive and run in FIFO order. This simple concurrency model is sufficient for applications that focus on I/O, but has difficulty supporting ones that demand high CPU utilization. To address this, there have been several proposals for incorporating threads into TinyOS.

TinyOS supports WSN nodes that rely on specific hardware, abiding by its open licensing. Many mote architectures are supported, including EPIC, Imote, Shimmer, Kmote, and MICA, amongst others. MICAs as well as TELOS motes have been developed in UC Berkeley and became commercially available by Crossbow. WSN nodes may also rely on third-generation MICA hardware for WSNs, such as MICAZ, MICA2, or MICA2DOT, that typically support 4 KB or more data RAM, 128 or more KB of program memory, and 512 or more KB of flash memory relying on the integrated circuit CC1000 by CHIPCON. Such devices are supported by microcontrollers such as the ATMEL 8-bit ATmega 128L. The former typically relies on a multichannel transceiver that operates at 868/916 MHz and the latter on one that can utilize the band from 2.4 to 2.48 GHz that is the IEEE 802.15.4 (ZIGBEE) spectrum. Both devices may rely on battery for power source and typically use two batteries, AA size. The boards that host MICA2 or MICAZ motes with sensory circuits are MTS100, MTS101, MTS300, MTS310, MTS400, MTS420, and MDA300. Most of the motes also support expansion slots for additional sensors. MICA2DOT devices reply on multiple channel transceivers that utilize the channels 315, 433, 868, or 916 MHz. Their power source is typically lithium 3 V coin cells and they can be installed on MTS510 or MDA500 boards. TELOS is another family of motes for TinyOS that includes the TELOSA, TELOSB, and TMOTE models. The motes in this are packaged with the 16-bit SI MSP430 microcontroller, which is efficient in terms of energy consumption. The modes utilize the 2.4–2.48 GHz IEEE 802.15.4 band (ZIGBEE). The motes typically integrate light, temperature, humidity, and voltage sensors. The members of the family are differentiated in terms of resources such as memory. TELOSA support 2 KB of RAM, 128 KB of program memory, and 512 MB of flash memory. TELOSB support 10 KB of RAM, 48 KB of program memory, and 1 MB of memory flash. They rely on two AA batteries for power source or via USB ports. Other motes that may run TinyOS are TinyNode, EyesIFX, and IntelMote2, amongst others. TinyNode devices use the 16-bit SI MSP430 microcontroller and wireless transceiver at 868 MHz. They use 8 KB of RAM, 92 KB of program memory, and 512 MB memory flash, the YE1205 integrated circuit and lithium batteries. TinyNode boards support light, temperature, and humidity sensors and a small breadboard surface; they are constructed and made available by Shockfish. EyesIFX devices are developed by INFINEON during the Energy Efficient Sensor Networks (EYES) EU research program. They employ TDA5250 integrated circuits for their operation. They have a 16-bit SI MSP430 microcontroller, a wireless FSK and ASK transceiver that reaches up to 64 kbps of rate. Their power source comes from AA batteries and they also come with 10 KB of RAM, 48 KB of program memory, and 512 MB of flash memory. They support high-precision temperature and light sensors. IntelMote2 or Imote2 is an advanced WSN platform that replies the powerful PXA271 microprocessor by Intel. It uses the CC2420 integrated circuit and an IEEE 802.15.4 transceiver. It supports 256 KB SRAM, 32 KB SDRAM, and 32 MB flash memory. The mote supports different interfaces for power such as USB or Imote2 board batteries. The ITS400 family of Crossbow boards that use IntelMote2 support light sensors, temperature sensors, triple axis accelerometer, and four-channel A/D converters. The IMB400 boards, also from Crossbow, feature multimedia functionality since they support camera, microphones, sound encoding and decoding, besides passive infrared sensor radiation (PIR). IRIS devices come with the ATMEGA 1281 ATMEL 8-bit microcontroller, an IEEE 802.15.4 transceiver and also 8 KB RAM, 4 KB EEPROM, 128 KB program memory, and 512 MB flash memory. Such devices are powered by two AA batteries and are expandable through the use of MTS300, MTS310, MTS400, MTS420, MDA100, MDA300, and MDA320, just like MICA2 and MICAS motes.

There are several management tools, platforms, and architectures for WSN currently available, each relying on different IM approaches. The bridge of the sensors system (BOSS) (Song et al., 2005) implements a management platform that relies on the standard service discovery protocol UPnP. Since the UPnP protocol is difficult to run on every sensor, BOSS adopts the bridging approach and uses designated nodes as management intermediates that employ XML for the description of services and for the communication of data. The system offers the ability to retrieve basic information about the state of the network, the characteristics of sensor nodes, the number of nodes in the network, and the topology. Administrators may interact with notes and configure parameters such as the transmission power or the node state. The MANNA architecture (Ruiz et al., 2003) is designed to manage any WSN application by exploiting specific operation models that reflect common functionality to WSN systems. It considers three management dimensions, namely functional areas, management levels, and WSN functionalities. Functional areas include configuration, fault, performance, security, and accounting management. WSN functionality considers configuration, maintenance, sensing, processing, and communication, and management can be applied at the level of business, service, network, and network element management. MANA is flexible, independent of the adopted WSN technology, and allows all possible configurations of the managed entities. Both the BOSS and the MANA management architectures follow the example of established NM principles. Middleware solutions use additional logic layers within the firmware of motes in order to implement basic IM services. The MATE (Levis and Culler, 2002) middleware introduces lightweight virtualization that aims to overcome the diversity in boards and thus simplify the task of managing diverse sensors. MATE's objective is to reduce complexity by reducing the size of programs, making them capable to be run on MICA, RENE, and other motes. The source code is broken down into 24 instruction capsules that self-replicate through the network via ad-hoc routing and data aggregation algorithms. AGILLA (Fok and Roman, 2005) is a mobile agent middleware that facilitates the rapid deployment of adaptive applications in WSNs. It allows users to create and inject programs, called mobile agents, which can be coordinated through local tuple spaces. Mobile agents migrate across WSNs and are capable of performing application-specific tasks. COUGAR (Cougar Project, 2011) is another middleware for WSN that uses a database for scalable and flexible WSN monitoring. COUGAR issues cross-layer optimizations, such as query-layer-specific routing algorithms optimized for regular types of communication patterns. The MIRES (Eduardo et al., 2004) middleware adopts an asynchronous publish/subscribe model for the communication with WSN nodes. NEST (Network Virtual Machine for Real-Time Coordination Services, 2012) is a real-time network coordination and control middleware that abstracts, controls, and ultimately guarantees the desired behavior of large unreliable networks. It is based on operating system tasks, called micro-cells, that provide support for migration, replication, and grouping functionality. Older middleware systems are the SCADDS and the Smart Messages project (Scalable Coordination Architectures for Deeply Distributed Systems). Other management systems for WSNs include various types of implementations. SNMS (Gellersen et al., 2002) is an application-cooperative management system for WSNs that uses minimum resources to provide a query system that enables rapid, user-initiated acquisition of network state and performance data, and also an event registration system. SNMS is based on a networking stack that runs in parallel with the applications stack. LEACH (Heinzelman et al., 2000) and GAF (Xu, 2001) also fall within the category of protocol-based management platforms for WSNs. GAF exploits node redundancy and supports sleep modes for overlapping nodes, while LEACH uses dynamic, efficient clustering in order to manage sensor networks. Another category of WSN tools focuses on monitoring consoles for WSNs. WSNView (Chen et al., 2007) is a technology capable of automatically searching and displaying network facilities, collecting and analyzing network utilization, and automatically producing notifications. Other similar visualization tools are TinyDB (Madden et al., 2003) and MoteView (Touron, 2005). TinyDB uses an SQL-like syntax in order to collect data from nodes and also provides basic configuration for motes. Administrators may recover the topology of the network or create graphs of data using its versatile low-level interface that offer only minimum levels of automation since administrators need to manage network operations manually and know how to exploit its representations and operations. MoteView is a tool for NM and control for WSNs from commercial workstations; it stores the measured data in an informational system and offers a graphical user interface that presents the network topology or the measured values. MoteView may also control mote parameters such as transmission power, sampling frequency, and node identification numbers. There are also many tools that focus on battery or transmit power management—that is, sampling frequency and transmission rate configuration. Agent-Based Power Management (ABPM) uses intelligent mobile agents for power conservation when nodes reach critical battery levels. Other similar tools such as SenOS and AppSleep force WSN nodes to automatically sleep when they are not performing data measurements. Systems like Siphon, DSN RM, and WinMS are capable of managing the network traffic efficiently; Siphon takes advantage of nodes that transmit in multiple directions in order to avoid congestion and heavily shared links. DSN Resource Management (DSNRM) evaluates traffic for incoming and outgoing links and time-schedules the transmissions in order to optimize network usage.

4.2 General-purpose IM

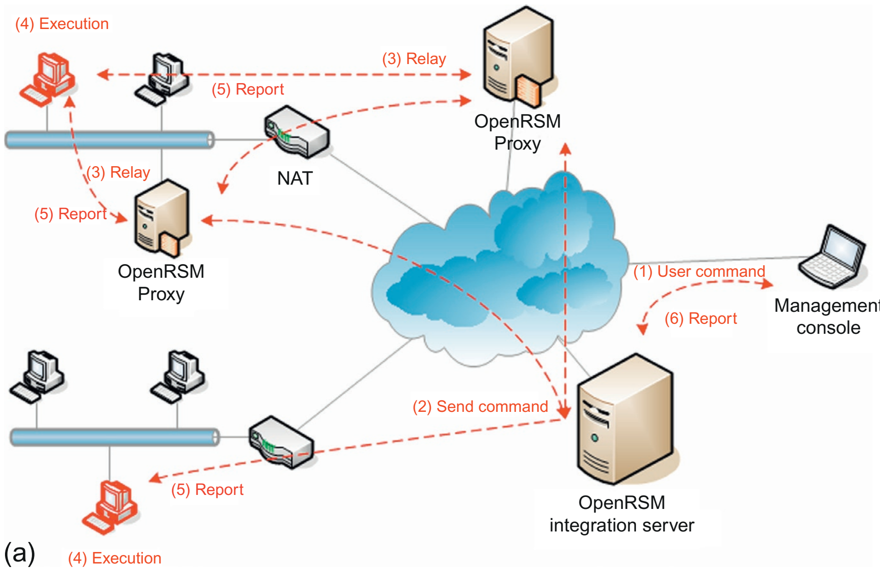
OpenRSM is a tool for the remote management of any IT infrastructure. It extends and integrates high-value FOSS projects in order to provide an integrated management platform. The goal has been to build a remote systems and NM platform capable of facilitating daily tasks. The system is designed to be fully functional yet simple, unlike most commercial management systems. IT is designed to offer information retrieval about installed assets, management of installed software, sending executable commands to stations, controlling remote desktops, wireless access points management, and integration with EGEE Grid technologies. OpenRSM was developed by integrating, enhancing, configuring, and customizing FOSS tools in order to deliver general-purpose IM. The basic services supported by OpenRSM are assets management (AM), software delivery (SD), remove desktop control (RDC), and NM. At the time the system was being created, there was no FOSS system that could claim to approach functionality similar to that of commercial Enterprise Management Systems (EMS). The capabilities of the system extend to managing any station that can be reached through standard IP connectivity in a secure manner. OpenRSM relies on local agents capable of conveying and executing management actions, on a graphical management console user interface and on an integration server that serves user requests and connections from the agents. OpenRSM depends on a management framework designed to model all the involved entities and service units necessary for IM and the description of the basic, abstract interactions between them. The design follows a layered approach; the main management entities are tasks that users can create and send to any managed workstation, and the workstations where tasks are dispatched. Services correspond to the subsystems described above—that is, AM, RDC, SD, NM, and also host discovery, remote procedure call, server tasks, access point management, task scheduling, wake on LAN, router configuration, reporting, static and dynamic entity grouping, customizable reports, and more. The hierarchy of the management entities is structured as an object-oriented tree of classes that forms the OpenRSM IM hierarchy. The design allows for modularity and extensibility and thus functionality can easily be extended to include specialized tasks and complex procedures. The top-level task object holds its identification, a mnemonic name, and execution parameters, such as priority, execution method, task dependency parameters, and more. An analogous object models the abstract managed host. The framework builds on abstract classes providing entity templates that can visually be instantiated in the user console interface. For example, users can extend any task template, fill in custom parameters, and create tasks that suit their needs. For example, in order to construct a task that shuts down a remote workstation, users need to instantiate a new task object by using the remote procedure call task class. They configure it to encapsulate a "shutdown" command for the target platform and they associate it with a managed station. The management framework takes care of the communication and execution details. Users can then use the same template to construct a second "shutdown" task for a different operating system. Both tasks will be descendants of the same framework task and inherit core functionality implemented in the task management framework.

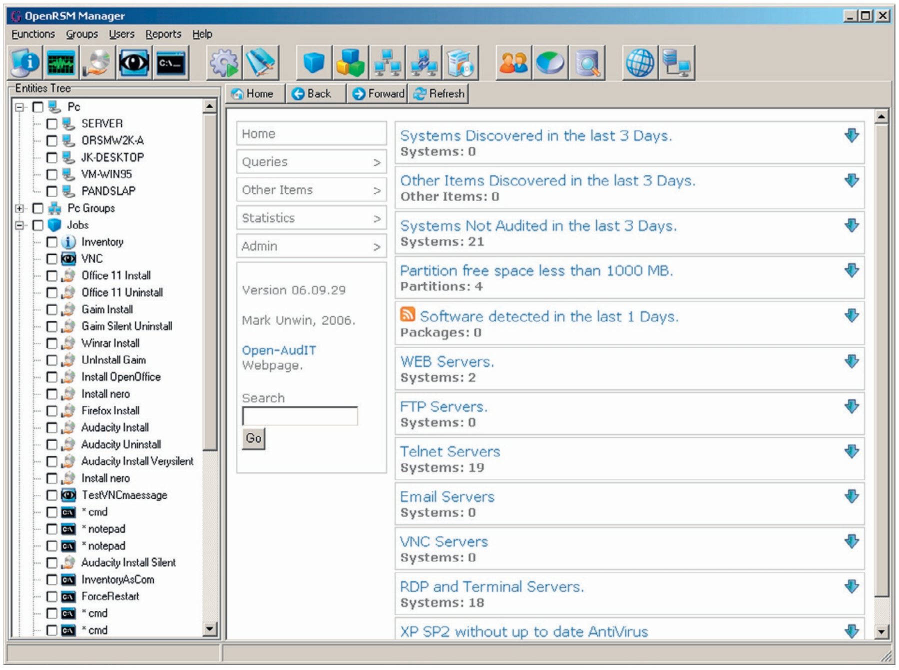
The task-handling engine rests at the core of the IM framework, since it coordinates the underlying mechanisms. Other layers include communication, task encoding/decoding, task/station verification, and security. All the previous layers have been dimensioned in terms of the agent, manager, and server functionality. For example, the communication between the server and the manager differs from that between the server and the agent; in the latter case the server needs to wake the agent asynchronously and then commence state verification handshakes, controls, and perform the actual communication. The manager–server communication is connection-oriented; however, both cases are implemented under the communications layer. The API hierarchy is made available via corresponding interfaces to all the components of the system. OpenRSM has passed stress and scalability tests and has been deployed in real conditions.

The design of the OpenRSM system considered several implementation strategies and development solutions (Hochstein et al., 2005), ranging from web-based technologies (Wren and Gutierrez, 1999), peer-to-peer technologies, and layered server-side middleware (Carey and Reilly, 2012), to more traditional client–server approaches (Lee et al., 2012). In principle, OpenRSM needed to be simple and lightweight so that it can be used by end-users who are not specialized in the use of management or asset-reporting tools. OpenRSM has also been designed for fast and automatic deployment in order to cover the needs of administrators who manage very dynamic environments. Thus, the system adopted the FOSS development model so as to exploit the dynamics of open IM projects and to gain value from integration. As mentioned above, even if there is no complete, integrated FOSS EMS, it can be observed that the related technologies have matured to the point that the FOSS community can provide all the necessary components (SourceForge hosting portal, 2012). Several FOSS projects have been examined in order to find the most appropriate FOSS management tools available for the purposes of OpenRSM. During the development of the project and the compilation of the present article, the authors have not been aware of the existence of any other integrated FOSS EMS system. The architecture has been design to be modular in order to follow the logical categorization of the entities involved and to favor integration with other IM tools. The system has thus been based on the client–server model where clients—that is, agents—model abstract manageable entities that convey administrative actions from the OpenRSM server. Agents are designed to support any operating system in order to dispatch administrative actions originating from the management console, the tool exposed to the end-users and the administrators of the service. Users describe administrative tasks in terms of management commands which are conveyed to the OpenRSM server and then scheduled to be executed at agents. The component architecture of the system is illustrated in Figure 4.1.

Tasks are entities abstracted and designed with the use of the object-oriented model. The design of the system has been based on the principle that tasks play a central role in terms of usability, design efficiency, and system scalability. Thus, they have been designed to behave as standard abstract system commands, for example AM, RDC, PRC, or as reusable user-created—that is, instantiated—objects. Tasks can be managed within OpenRSM since their creation and execution stages are decoupled. They are created by administrators at the management console. They are then submitted to the server, who checks their syntax and dependencies, schedules their execution according to user commands, prepares (wakes) the agents, and sends them task information, orchestrates interactions with back-end services, and interacts proxy modules if necessary. The OpenRSM monitors the task execution cycle, keeps logs, and produces reports, while ensuring security. The back-end services of the OpenRSM server consist of informational systems that run distinct IM server-side components. When tasks reach the execution stage, they are served by one of the subsystems incorporated in the OpenRSM system. For an AM query, the agent registers AM information about the station it resides on and submits the information AM web application, hosted by the web server of the OpenRSM system. Each server subsystem is presented in detail in the corresponding paragraph in this section. The OpenRSM project is hosted in SourceForge (The OpenRSM project, 2012). The development team consists of three to five developers. The code repository of the project is available for download and contributions.

The OpenRSM agents are the client modules residing in the workstations of the end-users. Their functionality is limited to the execution of commands sent by the server, and they do not interact with any module of the OpenRSM server system unless it is absolutely necessary. The needs for uniform logic design and security converge to this implementation; the execution of each task triggers communication with the integration server and, from there, with the appropriate server subsystem. The server subsystem controls the communication and performs all the complementary actions and database transactions. Agents are implemented by integrating subsystems corresponding to different OpenRSM functionalities: integration logic, communication with the server, system-dependent execution logic, and agent-type implementation.




OpenRSM Manager
Functions Groups Users Reports Help
Entities Tree
Home Back Forward Refresh

Pc
SERVER
ORSMW2K-A
JK-DESKTOP
VM-WIN95
PANDSLAP
Pc Groups
Jobs
Inventory
VNC
Office 11 Install
Office 11 Uninstall
Gaim Install
Gaim Silent Uninstall
Winrar Install
UnInstall Gaim
Install OpenOffice
Install nero
Firefox Install
Audacity Install
Audacity Uninstall
Audacity Install Verysilent
Install nero
TestVNCmaessage
* cmd
* notepad
* notepad
Audacity Install Silent
InventoryAsCom
ForceRestart
* cmd
* cmd

Home
Queries >
Other Items >
Statistics >
Admin >

Version 06.09.29

Mark Unwin, 2006.

Open-AudIT Webpage.

Search

Go

Systems Discovered in the last 3 Days.
Systems: 0

Other Items Discovered in the last 3 Days.
Other Items: 0

Systems Not Audited in the last 3 Days.
Systems: 21

Partition free space less than 1000 MB.
Partitions: 4

Software detected in the last 1 Days.
Packages: 0

WEB Servers.
Systems: 2

FTP Servers.
Systems: 0

Telnet Servers
Systems: 19

Email Servers
Systems: 0

VNC Servers
Systems: 0

RDP and Terminal Servers.
Systems: 18

XP SP2 without up to date AntiVirus

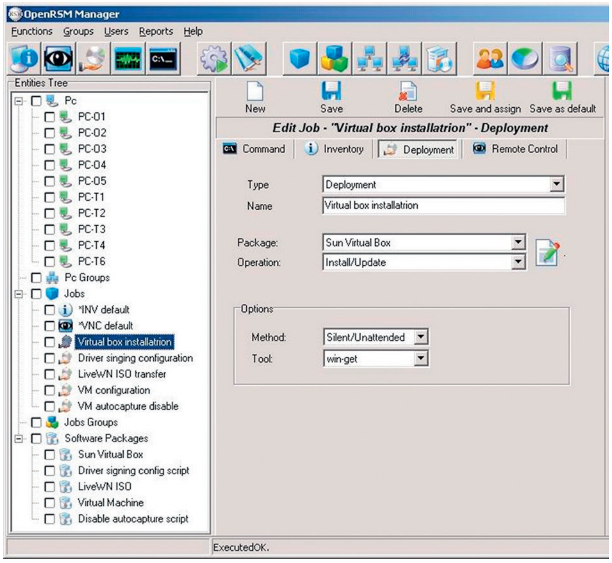
(b)

Figure 4.1 The OpenRSM component architecture and the graphical management console.

The fundamental agent module implements the communicational logic. As will be described in the following sections, this part of the agent ensures consistent and secure communication through a handshaking protocol and wake/sleep mechanisms. The remaining modules that make up the OpenRSM Agent are the subsystems that implement the task execution logic, namely the AM, NM, RDC, RPC, and agent discovery. Whenever possible, subsystems take advantage of existing software or other FOSS agent modules (e.g., the AM subsystem uses the OpenAudit agent for asset retrieval, as described earlier). The task execution subsystems are integrated with the communication logic so that all task execution stages can be monitored by the integration server. The OpenRSM system is capable of managing both Windows and *NIX systems, through corresponding agent distributions that take into account the characteristics of each platform. Each distribution includes different agent flavors that correspond to different types of usage; the agent can be executed as a background process for silent operation, as a graphical application user for verbose interaction, as a service, or as a console application.

OpenRSM is capable of supporting a centralized architecture when all components are installed on a single server or a distributed one, when each component is installed on an autonomous system. The benefits that can be derived from a distributed server topology are mainly related to customization, performance, availability, and efficiency. Since overall performance depends on the system load, subsystems that are more frequently used or subsystems that bring greater load to the system can be installed on separate server stations. In that case service availability also increases, since if a single server station fails then only a portion of the system service fails. NMS usually poses a heavy load on the overall system, and it might be preferable to set it up on an autonomous server. If the server cannot cope with the load, the database server can also be installed on a separate machine. The software repositories of the SD service can also be separated from the web server. It can be configured to provide service to a subset of the managed terminals so as to balance the overall load of the SD subsystem. Future work includes the decoupling of the two web applications, AM and NM, so that the former can be installable on a different server. A variety of different topologies are also possible. The system can be set up with many OpenRSM integration servers so as to avoid single points of failure. These server modules, each of which orchestrates the integration of subsystems, form the heart of the OpenRSM system and can be more than one per installation, so as to provide enhanced service availability. Other valuable topologies that may be useful for network traffic planning purposes make use of the OpenRSM proxy server module presented previously.

OpenRSM provides a controlling interface, illustrated in Figures 4.1 and 4.2, that can be used by the administrators to control all the subsystems and their interactions. The design has focused on synthesizing the independent functionalities provided by the subsystems in a comprehensive and effective manner, and on the provisioning of additional supervisory functionality. The OpenRSM management console provides a multilingual control environment. The console can send any system command that is supported by the operating system of the managed stations. Commands accept parameters related to CPU priority, type, and user visibility. The OpenRSM management console management is the functionality that discovers the available stations that run



Edit Job - "Virtual box installatrion" - Deployment

Type: Deployment
Name: Virtual box installatrion
Package: Sun Virtual Box
Operation: Install/Update
Method: Silent/Unattended
Tool: win-get

ExecutedOK.

Figure 4.2 The task creation tab of the OpenRSM management console is used to instantiate remote procedure calls, software management, remote desktop control, asset management tasks single or in groups.

OpenRSM agents and are therefore manageable. Discovery of OpenRSM agents can be directed towards any part of the Internet address space. Only agents configured to communicate with the specific OpenRSM server that originated the discovery packets will respond to the agent discovery. The result of the discovery process is to create active interface elements, representing corresponding managed stations. Along with core tasks (of the AM, RDC, SD, RPC type), they constitute the basic interface elements. Machines and tasks are entities that can be combined, resulting in tasks assigned to specific agent-equipped machines. They are both presented on the management tree for easy supervision. Both tasks and machines can be grouped. Groups of machines and tasks, or groups of tasks and machines, can also be combined in order to create submittable machine–task mappings. Groups are generic: a group of tasks can contain any kind of tasks and no dependencies are implemented. It is the administrator's responsibility to create a rational sequence of a group of tasks for execution. OpenRSM allows the user to create customized tasks. Following a clean installation, only core tasks exist. The core tasks include predefined AM, SD, RDC, and RPC tasks. As stated in the previous paragraph, a task must be visually combined with one or more agents. Thus, tasks can be considered as templates for submitted administration tasks. Each type of task is created using interface components used for that purpose only. For example, a delivery task must "know" the software that it has to install/uninstall etc. OpenRSM provides the interface for customized task production. The tasks created are stored and made available through the interface so that they can be reused. A user can also use the machine and task groups forms to define machine and task groups respectively. The management console also provides means of task execution supervision. Users can submit and monitor the execution of tasks in real time through the active task list. The task state is displayed along with information on the agent that executes it, related timestamps, etc. Filters can be applied to the task list, creating task–machine assignments that meet specific characteristics (e.g., owner, date, task type). One key usability feature of OpenRSM is related to its reporting functionality. This functionality can be further combined with the creation of dynamic groups of machines. The management console reporting can search across the database produced by AM tasks for machines that match specific user-defined characteristics. The objective is to enable the easy identification of workstations that share common characteristics and group them together in new machine groups, or present their selected attributes on a visual form. An example would be the retrieval of all workstations that have, for example, more physical memory than a specific value, selected by the user. The selection of attributes and the results are performed visually. The resulting workstation information can also be presented as a group of machines, called "dynamic" because of the way it is created. Dynamic groups behave as normal groups, but they also enjoy the special feature that they are associated with the database statement that created them. The query that created them may be executed at any time, in which case the group is recreated based on updated workstation information. The reporting functionality is complemented by the data explorer form, created to provide complete database supervision. The user is capable of browsing database entities and combining their contents whenever internal linking is possible. Combinations are presented in the form of reports that can be exported in various formats, such as DOC, XLS, HTML, TXT, and CSV. The management console is also capable of producing cumulative and detailed statistics about system utilization. The generated statistics can record general system usage, task distribution, workstation utilization, and user actions. General information is presented in visual charts providing summary information on the task submission rates, task error rates, and task distribution with respect to task type and submitting user. Detailed information is presented in individual reports.

The following paragraphs present the four OpenRSM use cases that correspond to fundamental EMS functionality implemented by server tiers—that is, AM, deployment and SD, NM, and RDC. The FOSS tools, informational systems, and platforms integrated into OpenRSM have been selected on the grounds of functionality, maturity, compatibility, and interoperability. AM rests at the core of IM, providing organizations with the ability to gather information about the hardware and software of their infrastructure. This service provides the necessary data for effective troubleshooting, facilitates the planning of upgrades, increases control and security, and helps in decision making and infrastructure planning. Asset management is usually realized through a silent software agent loaded on the managed system. The agent retrieves information about the system using native interfaces and presents it in a user-friendly way to administrators through the appropriate user interface. The technologies used, namely CIM/WBEM, are mature enough to provide vendor interoperability but, as mentioned in previous paragraphs, cannot be applied in WSNs. The OpenRSM platform relies on the FOSS OpenAudit (OpenAudit, 2012) AM software, a PHP/MYSQL application that relies on manual configuration and execution of audit software locally on workstations. The audit software reads information about the system and posts it using an HTTP request to the web application. The OpenRSM AM builds upon OpenAudit and enhances both automation of use and functionality. The audit software has been integrated with the OpenRSM agent module and has been ported to FOSS operating systems. Using the management console, it can be run from a remote location and, thus, physical presence is not necessary and stations can be audited remotely. The schema of the OpenAudit database is integrated within the informational system of OpenRSM so that high-level administrative functionality can be built. For instance, the dynamic groups feature takes advantage of this fact; the administrator is presented with the capability of creating groups of stations that share one or more common asset characteristics so that they can be treated in a uniform manner.

The SD functionality facilitates the management of already installed software, or of software that is to be installed on workstations within an administrative domain. Software management is time and resource consuming, in terms of experienced and specialized man hours; if it is not automated, however, administrators are required to know and manage the software of their managed stations, along with every other infrastructure asset. The functionality provided by the SD subsystem fills the gap as a high-level SD use case transparent to users. Administrators choose the software to be delivered, designate the path to be executable, or the link from where it can be downloaded; the software is then uploaded and registered within the software repository, and is subsequently delivered by the server. The designated installation file runs, and users may be required to complete the installation procedure. If silent or unattended modes of operation are chosen, a feature supported by OpenRSM, then the installation may not be interactive and users are not distracted in any way. Silent installations/uninstallations are very useful for routine administration and for operations in scale. OpenRSM uses an extended version of the Windows-Get FOSS tool, specifically enhanced in order to meet the requirements of the integrated graphical SD subsystem—that is, the support of uninstallations, broken transfers, and archive files. The SD subsystem is complementary with the AM subsystem; administrators may use the AM system in order to supervise SD and they may use the SD subsystem in order to install/uninstall desired modules. Figure 4.2 illustrates the SD task creation tab at the management console.

NM systems are essential tools for remote SM and are capable of managing active network elements using the SNMP protocol. The OpenRSM remote management system is integrated with the NINO FOSS NMS tool and the OpenNMS platform, both of which are full-featured NM systems that utilizes SNMP and WMI technologies for the provisioning of real-time monitoring information for stations and network active elements. The systems have been integrated so that changes in the OpenRSM management console are instantly reflected in the web interface of the monitoring system and vice versa. The NM systems typically support features such as network discovery using various methods, events (that is, traps), monitoring presets and groups, various presentation methods (web interface device browser, reports, applet graphs), various utilities, such as MIB browser, snmpwalk, service response meter (HTTP, FTP, POP), and other useful features.

OpenRSM integrates the TightVNC remote desktop control package to deliver the graphical remote control service. The management console is capable of starting the

TightVNC server at a managed station or at a group of managed stations by sending an appropriate RDC task to the TightVNC viewer in the host where the console is being run. Taking advantage of the features of the underlying FOSS tools, the remote desktop control request is started after the server has been started at the agent, that is the remote desktop control task has been delivered since the agent calls back the administration station. Thus, no synchronization failures may occur, since the server is guaranteed to have started when the remote control client (viewer) initiates the connection. Synchronization also enhances security, since the server wakes when the agent has been informed of a new connection request. The server sleeps again after a specific and configurable amount of time. Besides the above, the remote control software has been configured to ask users of the managed stations for permission whenever a connection is to open, in order to avoid unwanted remote access and to proxy connections in cases of isolated networks.

4.3 Managing WSNs

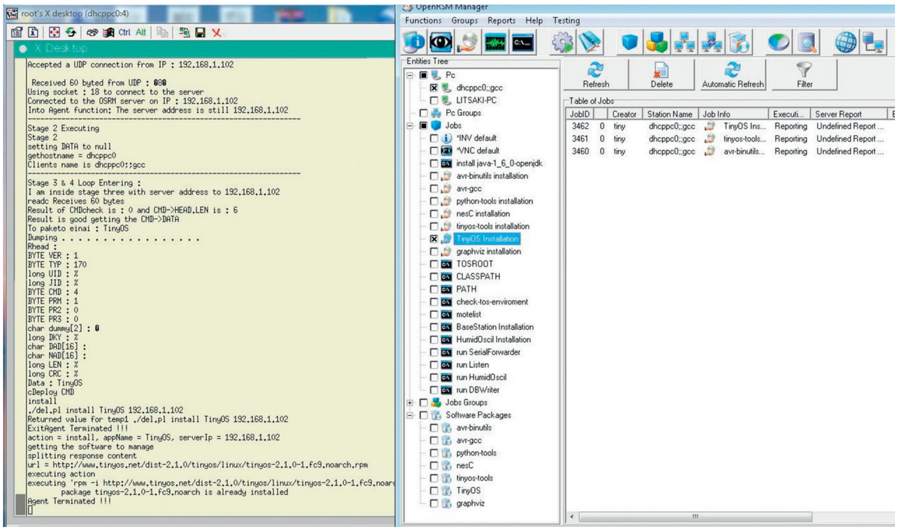
Considering the diversity of the related IT systems, it is difficult to picture an IM system that would inherently support any type of WSN. However, a general-purpose remote management system, such as OpenRSM, can be customized in order to exploit the management services offered by any WSN platform. Our goal has been to exploit the general-purpose nature of OpenRSM in order to offer full support for remote management for TinyOS-based WSNs by constructing a finite number of tasks so that the latter is installed on an FOSS or commercial platform, the environment is checked and configured properly, sensing applications are installed on motes, readings and measurements are stored, the operation of WSNs is monitored, and motes are configured in real time. The above tasks were defined as OpenRSM framework entities and made available at the management console. In the next paragraphs, we describe how customized tasks can be constructed in order to remotely create any sensing and configuration scenario and how we can retrieve measured data in the OpenRSM database. For the purposes of our case study we used CrossBow TELOSB motes, which are typically equipped with sensors that measure battery voltage, humidity, luminosity, and temperature. The WSN uses a designated node as the intermediate between the managed station and the sensing motes.

4.3.1 Installing WSNs

The remote installation of the TinyOS 2.1 platform presupposes the installation of a number of components, such as the JAVA programming platform, native compilers, such as the ATMEL AVR Tools (Atmel AVR8 microcontrollers, 2012) or the IT MSP430 tools (MSP430 16-bit Ultra-Low Power MCUs, 2012), the TinyOS tool chains, the TinyOS source code, and the GraphVIZ tool. The environment can then be configured by running appropriate shell scripts and by setting environment variables. The installation procedure can be implemented using either SD tasks or remote procedure ones that call native package managers such as APT, YUM, ZYPPER, or download and execute the necessary binaries. Whenever possible, features of the underlying operating system can be used, such as package management software. For the installation of tools and tool chains, commands such as

> *zypper --non-interactive --no-gpg-checks in -f --auto-agree-with-licenses java-1_6_0-openjdk*
> *apt-get install sun-java6-bin sun-java6-jre sun-java6-jdk openjdk-6-jre -y –force-yes*

can be encapsulated in RPC tasks and then be sent to managed hosts. Such commands can largely be sent by the management console. Alternatively, creating SD tasks would include providing the URL pointing to JAVA, creating a software package for it at the management console, and then associating it with an appropriate task. This is a typical procedure for SD tasks that are created graphically and may encapsulate binaries, archive files or binary images. They can be configured to support any unattended installation or uninstallation method provided by the installer of the software. The administrator will know of the software they are using, since it will be registered with the system and made visible on the entity tree of the management console. Another alternative would be to send a WGET task to the agent, followed by an execution command that utilizes the software downloaded by WGET or a shell script that would contain the aforementioned commands. Analogous tasks that install TinyOS, NESC, deputy, TinyOS-tools, and the TinyOS source tree are illustrated in Figure 4.3. The window on the left is the agent terminal window as presented by the LINUX window manager via a remote desktop connection, also opened with an OpenRSM RDC connection task. On the right, there is a management console which has sent a "TinyOS installation" SD task. Figure 4.3 also presents



```
Accepted a UDP connection from IP : 192.168.1.102

 Received 60 byted from UDP : 080
Using socket : 18 to connect to the server
Connected to the OSRM server on IP : 192.168.1.102
Into Agent function: The server address is still 192.168.1.102
-----------------------------------------------------------------
Stage 2 Executing
Stage 2
setting DATA to null
gethostname = dhcppc0
Clients name is dhcppc0;;gcc
-----------------------------------------------------------------
Stage 3 & 4 Loop Entering :
I am inside stage three with server address to 192.168.1.102
readc Receives 60 bytes
Result of CMDcheck is : 0 and CMD->HEAD.LEN is : 6
Result is good getting the CMD->DATA
To paketo einai : TinyOS
Dumping . . . . . . . . . . . . . . . . . .
Rhead :
BYTE VER : 1
BYTE TYP : 170
long UID : %
long JID : %
BYTE CMD : 4
BYTE PRM : 1
BYTE PR2 : 0
BYTE PR3 : 0
char dummy[2] : 0
long DKY : %
char DAD[16] :
char NAD[16] :
long LEN : %
long CRC : %
Data : TinyOS
cDeploy CMD
install
./del.pl install TinyOS 192.168.1.102
Returned value for temp1 ./del.pl install TinyOS 192.168.1.102
ExitAgent Terminated !!!
action = install, appName = TinyOS, serverIp = 192.168.1.102
getting the software to manage
splitting response content
url = http://www.tinyos.net/dist-2.1.0/tinyos/linux/tinyos-2.1.0-1.fc9.noarch.rpm
executing action
executing 'rpm -i http://www.tinyos.net/dist-2.1.0/tinyos/linux/tinyos-2.1.0-1.fc9.noar
        package tinyos-2.1.0-1.fc9.noarch is already installed
Agent Terminated !!!
```

JobID		Creator	Station Name	Job Info	Executi...	Server Report
3462	0	tiny	dhcppc0;;gcc	TinyOS Ins...	Reporting	Undefined Report ...
3461	0	tiny	dhcppc0;;gcc	tinyos-tools...	Reporting	Undefined Report ...
3460	0	tiny	dhcppc0;;gcc	avr-binutils...	Reporting	Undefined Report ...

Figure 4.3 The execution of a software installation task designed to install TinyOS on remote stations.

tasks that install the JAVA platform and the Atmel AVR tools—that is, avr-binutils, avr-gcc, avr-libc, avarice, avr-gdb, avrdude. The TI MSP430 tools are installed in a similar way, namely the basic toolset, python tools, binutils, GCC, LIBC, JTAG, GDB for MSP430. The registered packages, used by SD tasks, are presented in the bottom branch of the tree. The final step in the installation of TinyOS is the installation of the GraphVIZ tool and MAKE. Note that tasks can be grouped so that they are sent with a single click and they can be configured to be executed sequentially in the same thread, taking advantage of task configuration properties. Thus, the software is executed after the download and not in parallel with it, since the two tasks are executed by the main management console thread and not by forked ones. Administrators may monitor the execution logs via the logging console of the management interface of OpenRSM, which conveys the output of the agents at the management console.

4.3.2 Running applications

After installation, the WSN environment needs to be configured so that applications can be compiled and deployed. The configuration of the environment is achieved by setting appropriate environment variables; this can be accomplished via the execution of tasks or scripts that set the TinyOS root and home directories, the CLASAPTH for JAVA and NESC, the display variable, the paths to the rules for MAKE, and any additional parameters. Additional tasks can be used to run system utilities that check and return the state of the environment, such as tos-check-env and tos-install-jni. An example set of RPC commands that can be sent either as RPC tasks or as an SD script may include the configuration of the directory where the rules for MAKE are located—that is, "export MAKERULES=$TOSROOT/support/make/Makerules." After the configuration of TinyOS, RPC or SD tasks can be used to encapsulate standard deployment commands that deploy applications on motes.

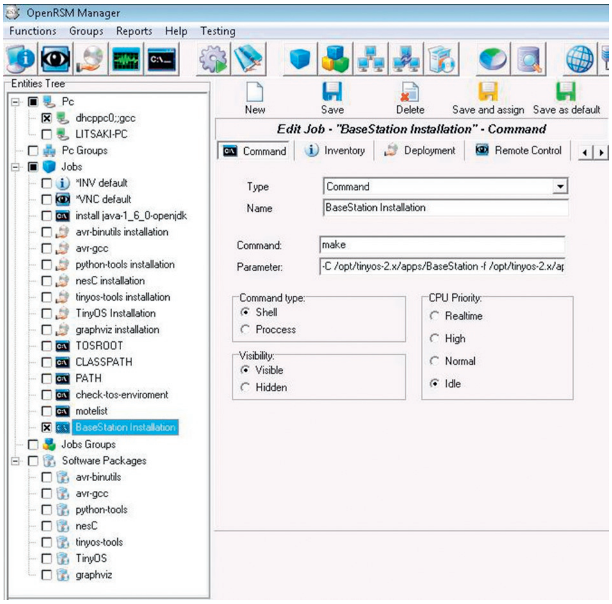
TinyOS provides toolboxes of applications that offer pieces of WSN IM functionality. The fundamental implementation of a gateway between a serial port and the WSN is the BaseStation application. When receiving packets from serial ports this applications transmits data towards the network and, vice versa, when receiving packets from a network it transmits them to the serial port. In order to forward traffic from the serial port towards network TCP sockets, the TinyOS platform provides the SerialForwarder. This functionality enables any tool such as IM, NM, or data analysis to access WSN measurements. Visualization for data is provided by Oscilloscope applications. The BaseStation application can be deployed at a designated mote in order for it to collect data from the WSN. Oscilloscope applications can also be deployed at the rest of the motes, in order to take measurements and forward them to the BaseStation. Such commands can be the following:

```
make -C apps/BaseStation -f apps/BaseStation/Makefile telosb install.0
java net.tinyos.sf.SerialForwarder -comm serial@/dev/ttyUSB0:telosb
java -cp support/sdk/java/tinyos.jar: apps/Oscilloscope/java/oscilloscope.jar Oscilloscope
```

The SerialForwarder application is usually executed at the managed station in order to read data from the serial port where the mote that runs the BaseStation application is connected and to forward them over network connections. If remote desktop functionality is desired at the managed host, the oscilloscope graphical application can also be executed in order to graphically present the received data. Figure 4.4 presents the configuration of an RPC command for the remote installation of the BaseStation application at the OpenRSM console. Retrieving measurements for battery voltage, temperature, luminosity, or humidity entails the deployment of the respective measuring applications—that is, VoltageOscil, TempOscil, LightOscil, or HumidOscil—and the execution of the respective client applications at the OpenRSM agent. The aforementioned applications are variations of Oscilloscope provided by TinyOS and use appropriate NESC components for data retrieval. For the measurement of luminosity, the HamamatsuS1087ParC driver is configured, and SensirionSht11C for temperature and humidity. For each task that deploys a sensing application, an oscilloscope task presents the measurements in the managed station. Commands such as:

make -C apps/TempOscil -f apps/TempOscil/Makefile telosb install
java -cp support/sdk/java/tinyos.jar:apps/TempOscil/java/oscilloscope.jar TempOscil

can be used to implement such tasks. In order to concentrate the functionality described above in a single tab, a general-purpose TinyOS management task instantiation panel has been developed for OpenRSM. The panel delivers graphical WSN task creation. The panel includes tasks for the installation of TinyOS, for uninstallation, for the discovery of motes, and for the deployment of applications. Figure 4.5 presents



Pc
[✓] dhcppc0.;gcc
LITSAKI-PC
Pc Groups
Jobs
*INV default
*VNC default
install java-1_6_0-openjdk
avr-binutils installation
avr-gcc
python-tools installation
nesC installation
tinyos-tools installation
TinyOS Installation
graphviz installation
TOSROOT
CLASSPATH
PATH
check-tos-enviroment
motelist
[✓] BaseStation Installation
Jobs Groups
Software Packages
avr-binutils
avr-gcc
python-tools
nesC
tinyos-tools
TinyOS
graphviz

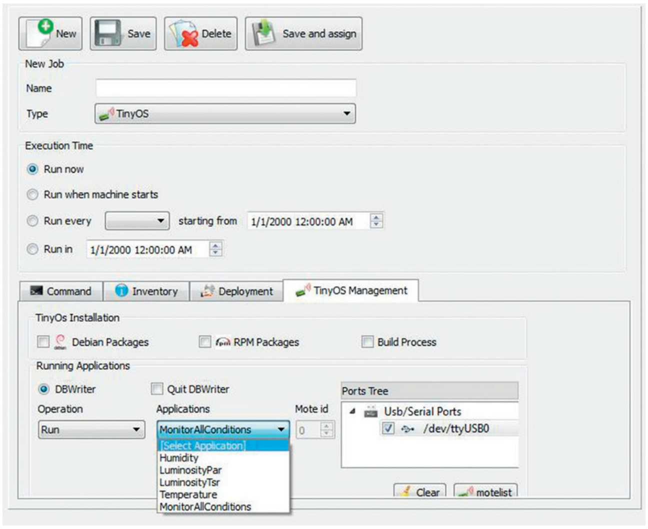
Edit Job - "BaseStation Installation" - Command

Type: Command
Name: BaseStation Installation

Command: make
Parameter: -C /opt/tinyos-2.x/apps/BaseStation -f /opt/tinyos-2.x/ap

Command type: [✓ Shell] Proccess
Visibility: [✓ Visible] Hidden
CPU Priority: Realtime High Normal [✓ Idle]

Figure 4.4 An RPC task can be configured to remotely run the BaseStation and SerialForwarder applications from the OpenRSM console.



New Job
Name:
Type: TinyOS

Execution Time
[✓ Run now]
Run when machine starts
Run every starting from 1/1/2000 12:00:00 AM
Run in 1/1/2000 12:00:00 AM

TinyOs Installation
☐ Debian Packages ☐ RPM Packages ☐ Build Process

Running Applications
[✓ DBWriter] ☐ Quit DBWriter

Operation: Run
Applications: MonitorAllConditions
[Select Application]
Humidity
LuminosityPar
LuminosityTsr
Temperature
MonitorAllConditions
Mote id: 0

Ports Tree
Usb/Serial Ports
☑ /dev/ttyUSB0

Figure 4.5 The management console supports a task creation tab for WSN tasks.

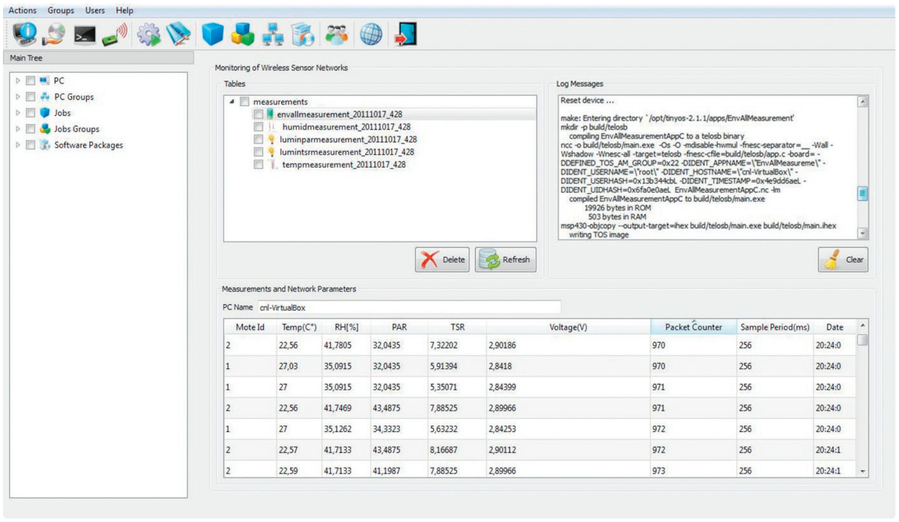
this panel; users define the name of the task, the execution time, and the type. They then instantiate it by saving it and then correlating it with managed TinyOS hosts. If the task includes the deployment of applications, users may select which of the available ones will be deployed and the target mote. This procedure can be used in order to implement any remote sensing scenario supported by the underlying technology.

4.3.3 Concentrating measurements

OpenRSM has been enriched with an application for TinyOS, DBWriter, that interfaces with the SerialForwarder application at the managed host in order to receive measured data as sent by motes and write them in the OpenRSM database. The application is configurable with the appropriate transformations in order to convert the data to the appropriate measurement systems. Figure 4.6 illustrates the measured data as presented in the management console. The left panel in the database view presents the navigation tree of the OpenRSM system and the right panel illustrates the functionality for measurement presentation. Users can select a sensing dataset from each mote that corresponds to a table in the database of OpenRSM. They are then presented with the data and meta-information that includes moteIds, packet counters, sampling frequency, date, etc. They can also view the logs of the system.

As mentioned in the previous paragraphs, motes utilize sensor circuits that sample physical quantities from motes through applications such as TempMeasurement, HumidMeasurement, LuminParMeasurement, and LuminTsrMeasurement. The EnvAllMeasurement application is used to recover data from the network besides the total number of packets received by each node, sampling frequency, and battery levels. The code of the application is presented in Table 4.1.

Each sensor uses analog-to-digital components that produce numeric output that can directly be converted to a metric system. The SHT11 sensor belongs to the Sensirion


Actions
Groups
Users
Help
Main Tree
PC
PC Groups
Jobs
Jobs Groups
Software Packages
Monitoring of Wireless Sensor Networks
Tables
measurements
envallmeasurement_20111017_428
humidmeasurement_20111017_428
luminparmeasurement_20111017_428
lumintsrmeasurement_20111017_428
tempmeasurement_20111017_428
Delete
Refresh
Log Messages
Clear
Measurements and Network Parameters

```
Reset device ...

make: Entering directory `/opt/tinyos-2.1.1/apps/EnvAllMeasurement'
mkdir -p build/telosb
    compiling EnvAllMeasurementAppC to a telosb binary
ncc -o build/telosb/main.exe  -Os -O -mdisable-hwmul -fnesc-separator=__ -Wall -
Wshadow -Wnesc-all -target=telosb -fnesc-cfile=build/telosb/app.c -board= -
DDEFINED_TOS_AM_GROUP=0x22 -DIDENT_APPNAME=\"EnvAllMeasureme\" -
DIDENT_USERNAME=\"root\" -DIDENT_HOSTNAME=\"cnl-VirtualBox\" -
DIDENT_USERHASH=0x13b344cbL -DIDENT_TIMESTAMP=0x4e9dd6aeL -
DIDENT_UIDHASH=0x6fa0e0aeL  EnvAllMeasurementAppC.nc -lm
    compiled EnvAllMeasurementAppC to build/telosb/main.exe
          19926 bytes in ROM
           503 bytes in RAM
msp430-objcopy --output-target=ihex build/telosb/main.exe build/telosb/main.ihex
    writing TOS image
```

PC Name cnl-VirtualBox

Mote Id	Temp(C°)	RH[%]	PAR	TSR	Voltage(V)	Packet Counter	Sample Period(ms)	Date
2	22,56	41,7805	32,0435	7,32202	2,90186	970	256	20:24:0
1	27,03	35,0915	32,0435	5,91394	2,8418	970	256	20:24:0
1	27	35,0915	32,0435	5,35071	2,84399	971	256	20:24:0
2	22,56	41,7469	43,4875	7,88525	2,89966	971	256	20:24:0
1	27	35,1262	34,3323	5,63232	2,84253	972	256	20:24:0
2	22,57	41,7133	43,4875	8,16687	2,90112	972	256	20:24:1
2	22,59	41,7133	41,1987	7,88525	2,89966	973	256	20:24:1

Figure 4.6 Measurements are sent by the DBWriter application to OpenRSM server and can be displayed at management console in real time.

AG family of SHT1x temperature/humidity surface mount. SHT11 integrates analog measuring devices and signal processing logic to provide calibrated output. Moisture is measured by a capacitive sensor and the temperature of a band-gap (proportional to absolute temperature) sensor. Measurements are then converted to 14-bit digital words which are conveyable to TinyOS via the serial interface. Converting the digital value to relative humidity (RH) units is achieved using the following formula:

$$\mathrm{RH}_{\mathrm{linear}} = c_1 + c_2 \times \mathrm{SO}_{\mathrm{RH}} + c_3 \times \mathrm{SO}_{\mathrm{RH}}^2$$

where $\mathrm{SO}_{\mathrm{RH}}$ represents the 12-bit measured data (Sensirion Output), $c_1 = -4$, $c_2 = 0.0405$, $c_3 = -2.8 \times 10^{-6}$. The respective conversion formula for temperature measurements is the following:

$$\mathrm{Temp} = c_1 + c_2 \times \mathrm{SO}$$

Light sensors for TinyOS motes typically use photo-diodes manufactured by Corporation. Hamamatsu S1087 photo-diodes detect photosynthetically active radiation (PAR) and S1087-01 photo-diodes detect all the visible spectrum, including infrared (TSR). Photosynthetic active radiation can be defined as the electromagnetic spectrum of visible light, namely wavelengths from 400 to 700 nm that helps process plant growth via photosynthesis. Both analog sensors TSR and PAR convert measurements to 12-bit digital words of length using 1.5-V signal pulses. LEDs generate current I along a resistance 100 kΩ. The output of S1087 or S1087-01 can be converted into units for brightness, LUX, using the following formulas:

Table 4.1 **The application EnvAllMeasurement concentrates data from all nodes of the WSN**

EnvAllMeasurementAppC.nc

```
components new HamamatsuS1087ParC() as Sensor;
components new HamamatsuS10871TsrC() as Sensor1;
components new SensirionSht11C() as Sensor2;
components new DemoSensorC() as Sensor3;
EnvAllMeasurementC.ReadPARLumin -> Sensor.Read;
EnvAllMeasurementC.ReadTSRLumin -> Sensor1.Read;
EnvAllMeasurementC.ReadExtTemp -> Sensor2.Temperature;
EnvAllMeasurementC.ReadHumid -> Sensor2.Humidity;
EnvAllMeasurementC.ReadVoltage -> Sensor3.Read;
```

EnvAllMeasurementC.nc

```
interface Read<uint16_t> as ReadPARLumin;
interface Read<uint16_t> as ReadTSRLumin;
interface Read<uint16_t> as ReadExtTemp;
interface Read<uint16_t> as ReadHumid;
interface Read<uint16_t> as ReadVoltage;
call. ReadPARLumin.read();
call. ReadTSRLumin.read();
call ReadExtTemp.read()
call ReadHumid.read();
call ReadVoltage.read();
event void ReadExtTemp.readDone(error_t result, uint16_t data){
 if (result != SUCCESS) {
data = 0xffff;
report_problem();
 }
 local.readingTemp = data;
}
```

EnvAllMeasurement.h

```
typedef nx_struct envAllMeasure {
 nx_uint16_t version;              /* Version of the interval. */
 nx_uint16_t interval;             /* Samping period. */
 nx_uint16_t id;                   /* Mote id of sending mote. */
 nx_uint16_t count;                /* Number of readings */
 nx_uint16_t readingTemp;          /*Var for temp*/
 nx_uint16_t readingHumid;         /*Var for humitity*/
 nx_uint16_t readingPARLumin;      /*Var for Par Luminosity*/
 nx_uint16_t readingTSRLumin;      /*Var for Tsr Luminosity */
 nx_uint16_t readingVolt;          /*Var for voltage*/
} envAllMeasure_t;
```

$$\text{LUX} = 0.625 \times 1\text{e}6 \times I \times 1000 \text{ for photo} - \text{diode S1087}$$

$$\text{LUX} = 0.769 \times 1\text{e}6 \times I \times 1000 \text{ for photo} - \text{diode S1087} - 01$$

where I is defined as:

$$I = \frac{AD_{\text{output}} \times \left(\frac{1.5}{4096} \right)}{10{,}000}$$

The microcontroller MPS30 has internal sensors such as temperature and voltage mentioned in the description of the devices category TELOS. The analog measurements of the sensor voltage are converted into digital words of length 12 bits and conversion to physical units is carried by the following formula:

$$VCC = 2 \times \left(\frac{AD_{\text{output}}}{4096} \right) \times V_{\text{ref}}$$

where $V_{\text{ref}} = 1.5\,\text{V}$.

4.3.4 Conclusions and future trends

The previous sections illustrated how OpenRSM can be used in order to provide remote IM for WSNs based on the TinyOS platform. OpenRSM can be used to create analogous tasks for any type of WSN that offers high-level tools or utilities and, since it has been stressed for scaled operation, it can support effective remote management cases. Current work focuses on productive installations of distant WSN infrastructures, on the integration of functionality for more WSN systems, and on the enrichment of the functionality OpenRSM offers. Organizations that need to minimize IM costs must be capable of testing the solutions described above by using an OpenRSM according to their needs. They may autonomously install the system or use a service dedicated for this purpose. The system must also not be limited for WSNs that rely on TinyOS; since OpenRSM is general purpose, it must be extended to include task creation forms for the best-known WSNs and functionality such as automatic installation of the WSN platform, the deployment of applications, and the concentration of measurements. The system can also be complemented with interesting functionality that increases automation, such as support for task execution in response to measurements. Users will be allowed to define alerts or actions that will be executed as standard tasks in case measurements reach thresholds, also defined by users. Interesting as well as challenging would be to include AM for WSN motes, whenever the underlying WSN platform offers identification applications or data based on the active mote and its types. Last but not least, the system will have to be extended to web-based technologies such as HTML5, AJAX, and middleware.

References

Atmel AVR8 microcontrollers. http://www.atmel.com/products/avr/, October 2012.

Bhatii, S., et al., 2004. Mantis OS: an embedded multithreaded operating system for wireless micro sensor platforms. Mobile Netw. Appl. 10 (4), 563–579.

Carey, K., Reilly, F., 2012. Integrating CIM/WBEM with the Java enterprise model. http://www.dmtf.org/education/academicalliance/, October.

Chaparadza, R., 2005. On designing SNMP based monitoring systems supporting ubiquitous access and real-time visualization of traffic flow in the network, using low cost tools. In: 2005 13th IEEE International Conference on Networks, 2005. Jointly held with the 2005 IEEE 7th Malaysia International Conference on Communication, vol. 2, pp. 16–18.

Chatzigiannakis, I., Mylonas, G., Nikoletseas, S., 2007. 50 ways to build your application: a survey of middleware and systems for Wireless Sensor Networks. In: IEEE Conference on Emerging Technologies and Factory Automation, 2007, ETFA, 25–28 September, pp. 466–473.

Chen, J., Lu, H., Lee, M., 2007. WSNView system for wireless sensor network management. In: 11th IASTED International Conference on Internet and Multimedia Systems and Applications, pp. 126–131.

Cougar Project. http://www.cs.cornell.edu/database/cougar, August 2011.

Crossbow technologies. http://www.xbow.com/, October 2012.

Dunkels, A., Gronvall, B., Voigt, T., 2004. Contiki – a lightweight and flexible operating system for tiny networked sensors. In: 29th Annual IEEE International Conference on Local, Computer Networks, pp. 455–462.

Dwivedi, A.K., Tiwari, M.K., Vyas, O.P., 2009. Operating systems for tiny networked sensors: a survey. Int. J. Recent Trends Eng. 1 (2), 152–157.

Eduardo, S., Germano, G., Glauco, V., 2004. A message-oriented middleware for sensor networks. In: Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing, pp. 127–134.

Fok, C., Roman, G., 2005. Mobile agent middleware for sensor networks: an application case study. In: Proceedings of the 4th International Conference on Information Processing in Sensor Networks. pp. 382–387.

Gellersen, H.W., Schmidt, A., Beigl, M., 2002. Multi-sensor context-awareness in mobile devices and smart artefacts. Mobile Netw. Appl. 5, 341–351.

Han, C.C., Kumar, R., Shea, R., Kohler, E., Srivastava, M., 2005. SOS – a dynamic operating system for sensor networks. In: Proceedings of the Third International Conference on Mobile Systems, Applications, and Services (Mobisys).

Heinzelman, W.R., Chandrakasan, A., Balakrishan, H., 2000. Energy efficient communication protocol for wireless microsensors networks. In: Proceedings of the Hawaii International Conference on System Sciences.

Hochstein, A., Zarnekow, R., Brenner, W., 2005. Evaluation of service-oriented IT management in practice. In: Proceedings of the International Conference on Services Systems and Services Management, vol. 1. pp. 80–84.

Karalis, Y., Kalochristianakis, M., Kokkinos, P., Varvarigos, E., 2009. OpenRSM: a lightweight open source remote management tool. Int. J. Netw. Manag. 19 (3), 237–252.

Lee, S., Choi, M., Yoo, S., Hong, J., Cho, H., Ahn, C., Jung, S., 2012. Design of a wbem-based management system for ubiquitous computing servers. http://www.dmtf.org/education/academicalliance/, July 2015.

Levis, P.A., 2006. TinyOS: an open operating system for wireless sensor networks. In: Invited Seminar, Proceedings of the 7th International Conference on Mobile Data Management, MDM'06.

Levis, P., Culler, D., 2002. Mate: a tiny virtual machine for sensor networks. In: Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 100–111.

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., Culler, D., 2005. TinyOS: an operating system for sensor networks export. In: Ambient Intelligence, pp. 115–148.

Lifton, J., Seetharam, D., Broxton, M., Paradiso, J., 2002. Pushpin computing system overview: a platform for distributed, embedded, ubiquitous sensor network. Proceedings of the 1st International Conference on Pervasive Computing, vol. 2414, pp. 139–151.

Madden, S., Hellerstein, J., Hong, W., 2003. TinyDB: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst. 30, 122–173.

Windows management instrumentation and simple network management protocol, Microsoft technet. http://technet.microsoft.com/en-us/library/bb742612.aspx, July 2015.

MSP430 16-bit Ultra-Low Power MCUs. http://focus.ti.com/mcu/docs/mcuprodoverview.tsp?sectionId=95&tabId=140&familyId=342, October 2012.

A Network Virtual Machine for Real-Time Coordination Services. http://www.cs.virginia.edu/wsn/nest.html, October 2012.

OpenAudit. http://sourceforge.net/projects/openaudit/, October 2012.

Ruiz, L.B., Nogueira, J.M.S., Loureiro, A.A., 2003. MANNA: a management architecture for wireless sensor network. IEEE Communications Magazine, vol. 41, pp. 116–125.

Scalable Coordination Architectures for Deeply Distributed Systems. http://www.isi.edu/div7/scadds, July 2015.

Song, H., Kim, D., Lee, K., Sung, J., 2005. UPnP-based sensor network management architecture and implementation. In: Second International Conference on Mobile Computing and Ubiquitous Networking (ICMU 2005).

The list of available projects in the SourceForge hosting portal. http://sourceforge.net/software-map/index.php, October 2012.

The MagnetOS Operating System. http://www.cs.cornell.edu/people/egs/magnetos/, October 2012.

The OpenRSM project at sourceforge. http://sourceforge.net/projects/openrsm/, October 2012.

The TinyOS alliance. http://www.tinyos.net/scoop/special/tinyos_alliance, October 2012.

Thompson, J.P., 1998. Web-based enterprise management architecture. IEEE Communications Magazine 36 (3), 80–86.

Tosic, V., Dordevic-Kajan, S., 1999. The Common Information Model (CIM) standard – an analysis of features and open issues. In: 4th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services, vol. 2, pp. 677–680. http://dx.doi.org/10.1109/TELSKS.1999.806301.

Touron. Crossbow: moteview interface, Crossbow, 2005. http://www.xbow.com/Technology/UserInterface.aspx.

Wren, M., Gutierrez, J., 1999. Agent and web-based technologies in network management. In: Proceedings of the Global Telecommunications Conference (GLOBECOM), vol. 3, pp. 1877–1881.

Xu, Y., 2001. Geography-informed energy conservation for ad hoc routing. In: Mobicom'01, pp. 203–212.

Yannakopoulos, J., Bilas, A., 2005. Cormos: a communication-oriented runtime system for sensor networks. In: 2nd European Workshop on Wireless Sensor Networks, February.