Appendix A: Wireless Sensor Development Platforms

Benny Lo and Guang-Zhong Yang

A.1 Introduction

The development of BSN has been facilitated by the rapid advances in *Wireless Sensor Networks* (WSNs) in recent years. BSN harnesses several allied technologies that underpin the development of pervasive sensing for healthcare, wellbeing, sports and other applications that require "ubiquitous" and "pervasive" monitoring of physical, physiological and biochemical parameters in any environment and without activity restriction and behaviour modification. Key to the development of BSN are technologies that address miniaturised biosensor design, suitable for both wearable and implantable devices, biocompatibility and materials to ensure long-term deployment, low-power wireless communication, integrated circuits and systems, power scavenging techniques from the body, autonomic sensing and standards and integration. Major technical hurdles of BSNs are related to continuous sensing and monitoring, requiring long-term stability of the sensors and low-power operation, also necessitating bio-inspired design (e.g., bio-inspired mix-signal ASIC) and power scavenging techniques ultimately for battery-less operation. For device level inter-connectivity, BSNs can be wired (e.g., interconnecting with smart fabric) or wireless (making use of common wireless sensor networks and standards.

Since the introduction of the concept of WSN, a large number of development platforms have been introduced [1, 2]. As shown in Fig. A.1, there are nearly 120 new WSN platforms introduced since 1998. The number of new platforms introduced peaked in 2004–2007 and remained relatively static 2008 onwards. The reason for the decline is largely due to the maturing of the sensing platforms

B. Lo • G.-Z. Yang, PhD (✉)
The Hamlyn Centre, Imperial College London, London SW7 2BY, UK
e-mail: g.z.yang@imperial.ac.uk

G.-Z. Yang (ed.), *Body Sensor Networks*, DOI 10.1007/978-1-4471-6374-9,

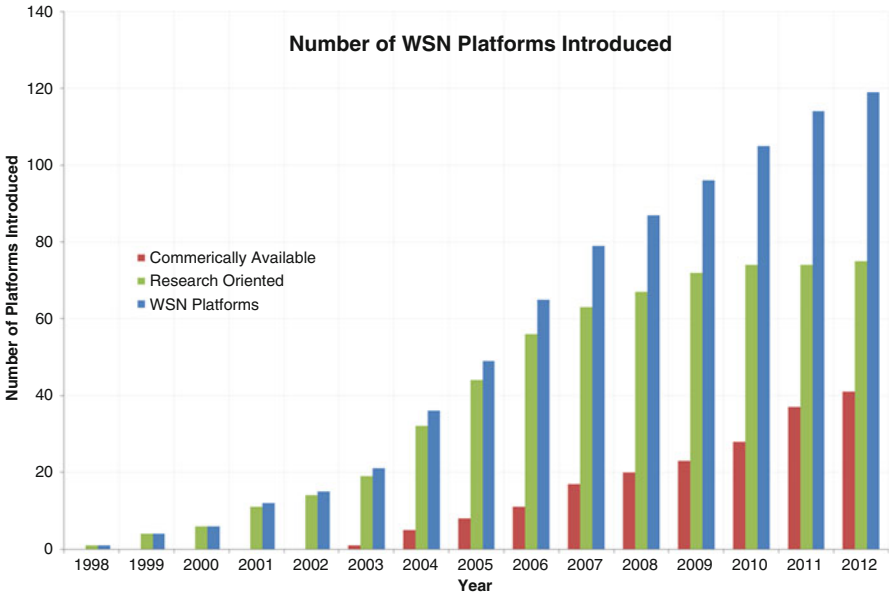

Fig. A.1 Number of new WSN hardware platforms introduced in recent years

and the increasing number of commercially available platforms. This is apparent by comparing the trend of the platforms designed for research purposes and those commercial platforms as shown in Fig. A.1.

A more detailed list of WSN platforms is provided in Table A.1. It must be pointed out that this table is by no means exhaustive and unintentional omission is likely. The general design and requirements for BSNs can be different from typical WSN applications. However, many of the WSN development platforms have been adapted or directly adopted for wearable BSN applications. For implantable applications, bespoke electronics and new ASIC designed are used due to stringent power and resource constraints. After the introduction of dedicated BSN platforms, such as the BSN node [3, 4], Pluto [5], Shimmer [6], purposely built platforms have been used in BSN research. This appendix outlines the system architecture of common WSN platforms and provides an overview of some of the main hardware components involved.

A.2 System Architecture

The system architectures of all WSN development platforms are relatively similar, and they mainly consist of the following components:

- Processor – the computer of the sensor node
- Wireless Communication – the wireless link between sensor nodes
- Memory – external storage for sensor readings or program images

Table A.1 Wireless sensor network development platforms

Platforms	CPU	Clock (MHz)	RAM/Flash/ EEPROM	Radio transceiver	BW (bps)	Freq. (MHz)	OS	Year	Organisation
WeC	Atmel AT90LS8535	4	512/8 K/32 K	RFM TR1000	10 k	916.5	TinyOS	1998	UC Berkeley
AWAIRS 1	Intel StrongArm SA1100	59–206	1 M/4 M	Conexant RDSSS9M	100 k	900	MicroC/OS	1999	Rockwell
Micromote	Atmel AT90S8535 Atmel AT90LS2343	4 10	512/8 K/32 K 128/2 k	RFM TR1000	10 k	916.5	TinyOS	1999	UC Berkeley
Rene 1	Atmel AT90LS8535	4	512/8 K/32 K	RFM TR1000	10 k	916.5	TinyOS	1999	UC Berkeley
Rene 2	Atmel Atmega 163	8	1 K/16 K/32 K	RFM TR1000	10 k	916.5	TinyOS	2000	UC Berkeley
uAMPS	StrongARM SA1100	206	16 M/512 K (ROM)	National LMX3162	1 M	2,450	uOS (eCOS microkernel)	2000	MIT
BT node[ab]	Atmel Atmega 128 L	8	4 K/128 K/4 K	ZV4002 BT/TI (Chipcon) CC1000	1 M 38.4 k	2,400 868	TinyOS	2001	ETH Zurich
Dot	Atmel Atmega 163	8	1 K/16 K/32 K	RFM TR1000	10 k	916.5	TinyOS	2001	UC Berkeley
RSC WINS	DEC SA1100	206	1 M/4 M	RDSSS9M	100 k	900		2001	Rockwell
SpotON	Dragonball EZ	16	2 M/2 M	RFM TR1000	10 k	916.5		2001	University of Washington/ Intel
Smart-its[b] (Lancaster)	Microchip PIC18F252	8	3 K/48 K/64 K	Radiometrix	64 k	433	Smart-its	2001	Lancaster
Smart-its (Teco)	Atmel ATMega 103 L	4	4 K/128 K	Ericsson BT	1 M	2,400	Smart-its	2001	University of Karlsruh
CENS Medusa MK2	Atmel ATMega128L Atmel AT91FR4081	40	136 K/1 M	RFM	10 k	916	Palos	2002	UCLA
iBadge	Atmel Atmega103L	6	4 K/128 K	Ericsson BT	1 M	2,400	Palos	2002	UCLA
Mica	Atmel Atmega 128 L	4	4 K/128 K/512 K	RFM TR1000	40 k	916.5	TinyOS	2002	UC Berkeley
iMote1	Zeevo ZV4002 (ARM)	12–48	64 K/512 K	Zeevo BT	720 k	2,400	TinyOS	2003	Intel
Mica2[ab] [10, 11]	Atmel Atmega 128 L	8	4 K/128 K/512 K	TI (Chipcon) CC1000	38.4 k	900	TinyOS	2003	UC Berkeley/ Crossbow
Mica2Dot[ab]	Atmel Atmega 128 L	4	4 K/128 K/512 K	TI (Chipcon) CC1000	38.4 k	900	TinyOS	2003	UC Berkeley/ Crossbow
Mantis Nymph	Atmel Atmega128L	4	4 K/128 K/512 K	TI (Chipcon) CC1000	38.4 k	900	Mantis	2003	U Colorado
RFRAIN	TI (Chipcon) CC1010	3–24	2 K/32 K	TI (Chipcon) CC1010	76.8 k	0.3–1,000	RFRAIN Libraries	2003	MIT

(continued)

Table A.1 (continued)

Platforms	CPU	Clock (MHz)	RAM/Flash/ EEPROM	Radio transceiver	BW (bps)	Freq. (MHz)	OS	Year	Organisation
U3	Microchip PIC18F452	0.031–8	1 K/32 K/256	CDC-TR-02B	100 k	315	Pavenet	2003	U Tokyo
AquisGrain	Atmel Atmega128L	4	4 K/128 K/512 K	TI (Chipcon) CC2420	250 k	2,400	–	2004	Philips Research
BEAN	TI MSP430F149	8	2 K/60 K/512 K	TI (Chipcon) CC1000	76.8 k	0.3–1,000	YATOS	2004	Universidade Federal de Minas Gerais
BSN node v2[b] [3, 4]	TI MSP430F149	8	2 K/60 K/512 K	TI (Chipcon) CC2420	250 k	2,400	TinyOS	2004	Imperial College London/ Sensixa
CIT sensor node	Microchip PIC16F877	20	368/8 K	Nordic nRF903	76.8 k	868	TinyOS	2004	Cork Institute of Technology
DSYS25[ab]	Atmel Atmega 128	4	4 K/128 K	Nordic nRF2401	1 M	2,400	TinyOS	2004	UCC
eXtreme Scale Mote (XSM)	Atmel ATmega128L	8	4 K/128 K	TI (Chipcon) CC1000	76.8 k	433	TinyOS	2004	Ohio State U and Crossbow
Fleck 1 Fleck 2	Atmel Atmega 128	4	4 K/128 K/512 K	Nordic nRF903	76.8 k	902–928	TinyOS	2004	CSIRO
MicaZ[ab]	Atmel Atmega 128 L	8	4 K/128 K	TI (Chipcon) CC2420	250 k	2,400	TinyOS	2004	Crossbow
MITes[b]	Nordic nRF24E1	16	512/4 K	Nordic nRF24E1	1 M	2,400	–	2004	MIT
Parasitic	Silicon C8051F311	25	1.2 k/16 k	BlueRadios BR-C11A	1 M	2,400		2004	MIT
Particle2/29[a]	Microchip PIC 18 F6720	20	4K128K/512 K	RFM TR1001	125 k	868.35	Smart-its	2004	Lancaster/University of Karlsruhe
Pluto[b] [5]	TI MSP430F149	8	4 K/60 K/512 K	TI (Chipcon) CC2420	250 k	2,400	TinyOS	2004	Harvard
ProSpeckz[b]	Cypress CY8C2764	12	256/16 K	TI (Chipcon) CC2420	250 k	2,400	Speckle net	2004	University of Edinburgh
Spec	8-bit AVR-like RISC core	4–8	3 K	FSK Transmitter	100 k		TinyOS	2004	UC Berkeley
Telos[ab]	TI MSP430F149	8	2 K/60 K/512 K	TI (Chipcon) CC2420	250 k	2,400	TinyOS	2004	UC Berkeley/ Sentilla (Moteiv)

Ember RF module[a]	Atmel Atmega 128 L	8	4 K/128 K	Ember 2420	250 k	2,400	EmberNet	2005	Ember
EnOcean TCM120	Microchip PIC18F452	10	1.5 K/32 K/256	Infineon TDA 5200	120 k	868	TinyOS	2005	Helmut Schmidt University
eyesIFXv2[b]	TI MSP430F1611	8	10 K/48 K	Infineon TDA5250	64 k	868	TinyOS	2005	TU Berlin
Hogthrob	Atmel Atmega 128 L Xilinx Spartan 3 XC3S400	8 48	4 K/128 K 56 K	Nordic nRF2401	1 M	2,400		2005	Technical University of Denmark
iMote2	Intel PXA 271	13–104	256 K/32 M	TI (Chipcon) CC2420	250 k	2,400	TinyOS	2005	Intel
MediMesh[b]	TI MSP430F1611	8	10 K/48 K	TI (Chipcon) CC2420	250 k	2,400	TinyOS	2005	Chinese University of Hong Kong
Mulle[a]	Renesas M16C/62P	10	31 K/384 K/2 M	Mitsumi WML C46AHR	2.178 M	2,400	Mulle	2005	Eistec
RISE	TI (Chipcon) CC1010 EM	3–24	2 K/32 K	TI (Chipcon) CC1010 EM	76.8 k	0.3–1,000	TinyOS	2005	UC Riverside
ScatterWeb ESB	TI MSP430F149	8	2 K/60 K	RFM TR1001	115.2 k	868	TinyOS/Contiki	2005	Freie Universität Berlin
Stack[b]	Silicon C8051F206	25	1 K/8 K	RFM TR1000	115.2 k	916		2005	MIT
Solar biscuit	Microchip PIC 18LF452	7.3728	1 K/32 K	TI (Chipcon) CC1000	76.8 k	315		2005	The University of Tokyo
Tmote sky/TelosB[ab] [12]	TI MSP430F1611	8	10 K/48 K/1 M	TI (Chipcon) CC2420	250 k	2,400	TinyOS	2005	UC Berkeley/ Sentilla (Moteiv)
XYZ sensor node	OKI ML67Q500x (ARM/THUMB)	1.8–57.6	4 K/256 K/512 K	TI (Chipcon) CC2420	250 k	2,400	SOS	2005	Yale
Ant[ab]	TI MSP430F1232	8	256/8 K	Nordic nRF24AP1	1 M	2,400	Ant	2006	Dynastream Innovation Inc.
BT node rev3[ab] [13]	Ateml ATmega128L	8	4 K/128 K/4 K	ZV4002 BT/TI (Chipcon) CC1000	1 M 38.4 k	2,400 868	BTnut TinyOS	2006	ETH Zurich/Art of Technology
Cookies	Analog Device ADuC841 Xilinx XC3S200 Spartan3	16 48	4 K/62 K 234 K	ConnectBlue OEMSPA13i	1 M	2,400		2006	Universidad Politécnica de Madrid
DSRPN	TI OMAP5912 TMS320C55x	192 200	250 K 128 K			340		2006	Chinese Academy of Science

(continued)

Table A.1 (continued)

Platforms	CPU	Clock (MHz)	RAM/Flash/ EEPROM	Radio transceiver	BW (bps)	Freq. (MHz)	OS	Year	Organisation
e-Watch[b] [14]	Philips LPC2106 ARM7TDMI	60	64 K/128 K	SMARTM Bluetooth	1 M	2,400		2006	Carnegie Mellon University
IECAS	Silicon C8051F121	100	84 K/128 K	TI (Chipcon) CC2420	250 k	2,400		2006	Institute of Electronics Chinese Academy of Science
LEAP	Intel PXA255	400	128 M/64 M/ 192 M	TI (Chipcon) CC2420 Atheros 5006XS	250 k 20 M	2,400 2,437	LEAP software framework	2006	UCLA
Nano-Qplus	Atmel ATmegal28	8	4 K/128 K	TI (Chipcon) CC2420	250 k	2,400	Nano-Qplus OS	2006	ETRI Korea
SENTIO	Atmel Atmega 128L	8	4 K/128 K	TI (Chipcon) CC2420	250 k	2,400		2006	Mid Sweden University
Shimmer[b] [6]	TI MSP430F1611	8	10 K/48 K/2G	TI (Chipcon) CC2420 Mitsumi WML-C46N	250 k 2.178 M	2,400 2,400	TinyOS	2006	Intel
Tinynode 584[a]	TI MSP430F1611	8	10 K/48 K/512 K	Semtech XE1205	152 k	868/902	TinyOS	2006	EPFL/Tinynode
TUTWSN[b]	Microchip PIC18LF4620	10	4 K/64 K/1 K	Nordic nRF905	50 k	433/868/ 915		2006	Tampere University of Technology
Tyndall Mote[b]	Atmel Atmega 128L	8	4 K/128 K/32 K	Nordic nRF2401	1 M	2,400	TinyOS	2006	Tyndall
ubER-Badge[b] [15]	TI MSP430F149	8	4 K/60 K/256 M	TI (Chipcon) CC1010	76.8 k	0.3–1,000		2006	MIT
uPart0140ilmt[a]	Microchip rfPIC16F675	4	64/1 K	Microchip rfPIC16F675	19.2 k	868	Smart-it	2006	University of Karlsruh
ZN1	Renesas H8S/2218	4–24	128 K/128 K (ROM)	TI (Chipcon) CC2420	250 k	2,400		2006	Hitachi
AVRraven[a]	Atmel AtMega1284p Atmel ATmega3290p	20	128 K/16 K/256 K	Atmel AT86RF230	250 k	2,400	Atmel Studio	2008	Atmel
FireFly	Atmel ATmega1281	16	8 K/128 K/2G	TI (Chipcon) CC2420	250 k	2,400	Nano-RK	2007	Carnegie Mellon University
Fleck 3	Atmel Atmega128L	8	4 K/128 K/1 M	Nordic nRF905	50 k	433/868/ 915	TinyOS	2007	CSIRO

GWNode [16]	Microchip PIC18LF8722	40	128 K/64 K/1 K	BiM 1	10 k	173		2007	Southampton
iSense[ab]	Jennic JN5148	4–32	128 K/512 K	802.15.4	250 k	2,400	iSense	2007	coalesenses
JN5121[ab]	OpenRSIC1000	16	96 K/64 K	IEEE802.15.4	250 k	2,400	JenNet/Jenie	2007	Jennic
mPlatform[b] [17]	TI MSP430F1611 OKI ML67Q5003 XC2C512 CoolRunner-II CPLD	8 60 32–200	10 K/48 K 32 K/512 K	TI (Chipcon) CC2420	250 k	2,400	mPlatform	2007	Microsoft
NeoMote[a] (EcoWizard)	Atmel ATMega 128L	8	4 K/128 K	TI (Chipcon) CC2420	250 k	2,400		2007	MEMSIC (Crossbow)/ Sumitomo Precision
NWSP[b]	Altera EP2C20F256C7N	500	26 K	National LMX9830	1 M	2,400	eCos	2007	Nokia
Power meter WSN	TI (Chipcon) CC1010	3–24	2 K/32 K	TI (Chipcon) CC1010	76.8 k	0.3–1,000		2007	University of New South Wales Sydney
S-Mote	TI (Chipcon) CC2430	32	8 K/128 K	TI (Chipcon) CC2430	250 k	2,400	RETOS	2007	Yonsei University Korea
Sun SPOT[ab] [18]	ARM920T	180	512 K/4 M	TI (Chipcon) CC2420	250 k	2,400	Java J2ME	2007	Orcale (Sun)
Tmote mini[ab]	TI MSP430F1611	8	10 K/48 K/1024 K	TI (Chipcon) CC2420	250 k	2,400	TinyOS	2007	Sentilla (Moteiv)
WeBee3[b]	TI (Chipcon) CC2431	16	8 K/128 K	TI (Chipcon) CC2430/ CC2431	250 k	2,400		2007	Lucerne University of Applied Sciences
ZigBit[a]	Atmel ATmega1281V	16	8 K/128 K	Atmel AT86RF230RF	250 k	2,400	ZigBit Development Kit/TinyOS	2007	MeshNetics
Arduino BT[ab]	Atmel ATmega328	16	2 K/32 K/1 K	Blue giga WT11i	3 M	2,400	Arduino IDE/Java	2008	Arduino
BSN node v3[ab] [19]	TI MSP430F1611	8	10 K/48 K/4 M	TI (Chipcon) CC2420	250 k	2,400	TinyOS/BSNOS	2008	Imperial College London/ Sensixa

(continued)

Table A.1 (continued)

Platforms	CPU	Clock (MHz)	RAM/Flash/ EEPROM	Radio transceiver	BW (bps)	Freq. (MHz)	OS	Year	Organisation
Dalian WSN	LPC2138	30	32 K/512 K	TI (Chipcon) CC2420	250 k	2,400		2008	Dalian University of Technology
EPIC mote	TI MSP430F1611	8	10 K/48 K/512 K	TI (Chipcon) CC2420	250 k	2,400	TinyOS	2008	UC Berkeley
MASN	Ember EM250 (XAP2b)	12	5 K/128 K	Ember EM250	250 k	2,400	TinyOS	2008	Rochester Institute of Technology
Shimmer 1.3[ab]	TI MSP430F1611	8	10 K/48 K/2G	TI (Chipcon) CC2420 Roving Networks RN-42 (Bluetooth)	250 k 3 M	2,400 2,400	TinyOS	2008	Shimmer
WBSN[b]	TI MSP430F1611	8	10 K/48 K/124 K	TI (Chipcon) CC2420	250 k	2,400	TinyOS	2008	Queen Mary's University of London
FemtoNode	FemtoJava	56	0.5 K	TI (Chipcon) CC2420	250 k	2,400	API-Wireless	2009	Universidade Federal do Rio Grande do Sul
iCubes	Silicon C8051F320	25	2 K/16 K	TI (Chipcon) CC2500	500 k	2,400	Over The Air Programmed (OTAP)	2009	Technical University of Crete
Kmote-B[a]	TI MSP430F1611	8	10 K/48 K/512 K	TI (Chipcon) CC2420	250 k	2,400	TinyOS	2009	INETCH Co./ TinyOS Mall
PowWow – CAIRN	TI MSP430F1612	8	5 K/55 K	TI (Chipcon) CC2420	250 k	2,400	PowWow	2009	Inria
SenseNode[a]	TI MSP430F1611	8	10 K/48 K/1024 K	TI (Chipcon) CC2420	250 k	2,400	TinyOS	2009	Genet
Sensium[ab]	Toumaz Sensium TZ1030	1	64 k	Toumaz Sensium TZ1030	50 k	928			Toumaz
Shimmer 2[ab]	TI MSP430F1611	8	10 K/48 K/2G	TI (Chipcon) CC2420 Roving networks RN-42 (Bluetooth)	250 k 3 M	2,400 2,400	TinyOS	2009	Shimmer
T-node[a]	Atmel ATMega 128L	8	4 K/128 K	TI (Chipcon) CC1000	38.4 k	868	SOWNet	2009	SOWNet Technologies
Tempo 3.1[b] [20]	TI MSP430F1611	8	10 K/48 K	Roving Networks RN-41 (Bluetooth)	1 M	2,400	TEMPOS	2009	University of Virginia

WeBee3G	TI (Chipcon) CC2430	16	8 K/128 K	TI (Chipcon) CC2430	250 k	2,400		2009	Lucerne University of Applied Sciences
Egs	Atmel SAM3U	96	52 K/256 K	TI (Chipcon) CC2520 WML-C46	250 k 2.178 M	2,400 2,400		2010	Harbin Institute of Technology
ENS[b]	TI MSP430F2410	16	4 K/56 K	TI (Chipcon) CC2420	250 k	2,400	ENS Network software	2010	University of Edinburgh
G-node[a]	TI MSP430F2418	16	8 K/116 K/1 M	TI (Chipcon) CC1101	1.2 k	868/915	SOWNet	2010	SOWNet Technologies
Indriya CS-03A14[a]	Atmel ATMega 128L	8	4 K/128 K	XBee	250 k	2,400	TinyOS	2010	Indrion
RecoNode	Xilinx Virtex-4 FX20	450	128 M/	TI (Chipcon) CC2520	250 k	2,400	RTOS/TinyOS	2010	University of Denver
Shimmer 2R[ab]	TI MSP430F1611	8	10 K/48 K/2G	TI (Chipcon) CC2420 Roving Networks RN-42 (Bluetooth)	250 k 3 M	2,400 2,400	TinyOS	2010	Shimmer
VE209-ST VEmesh[a]	TI MSP430	16	512B/8 K	Semtech SX1211/SX1231	50 k	902		2010	Virtual Extension
XBee-PRO ZB[ab]	Freescale HCS08	50.33	2 K/32 K	ZigBee	250 k	2,400	XBee SDK	2010	Digi International
Z1[a]	TI MSP430F2617	16	8 K/92 K	TI (Chipcon) CC2420	250 k	2,400	TinyOS/ContikiOS	2010	Zolertia
AS-XM1000[a]	TI MSP430F2618	16	8 K/116 K/1024 K	TI (Chipcon) CC2420	250 k	2,400	TinyOS/ContikiOS	2011	Advanticsys
Indriya_ DP_01A11[a]	TI MSP430F2618	16	8 K/116 K	TI (Chipcon) CC2520	250 k	2,400	TinyOS	2011	Indrion
Indriya_ DP_03A20[a]	Atmel ATMega 128L	8	4 K/128 K	GainSpan GS1011 M	1 M	2,400	TinyOS	2011	Indrion
IRIS[a]	Atmel ATmega1281	16	8 K/128 K/512 K	Atmel AT86RF230	250 k	2,400	Mote Works/Mote Runner	2011	MEMSIC (Crossbow)
Libelium Wasp mote[ab]	Atmel ATMega1281	8	8 K/128 K/2GB	XBee module	250/230/ 230 k	2,400/900/ 848	Libelium (Over the Air) OTA	2011	Libelium
LOTUS[a]	NXP LPC1758	100	64 K/512 k/64 M	Atmel AT86RF231	250 k	2,400	Free RTOS/ TinyOS/Mote Runner	2011	MEMSIC (Crossbow)
MTM-CM5000-MSP[a] MTM-CM3000-MSP[a] MTM-CM3300-MSP[a] MTM-CM4000-MSP[a]	TI MSP430F1611	8	10 K/48 K/1024 K	TI (Chipcon) CC2420	250 k	2,400	TinyOS/ContikiOS	2011	Advanticsys
Preon32[a]	ARM Cortex-M3	72	64 K/256 K	Atmel AT86RF231	2 M	2,400	Java	2011	Virtenio
WiSMote Dev[a]	TI MSP430F5437	8	16 K/256 K/1 M	TI (Chipcon) CC2520	250 k	2,400	Contiki	2011	Arago Systems

(continued)

Table A.1 (continued)

Platforms	CPU	Clock (MHz)	RAM/Flash/ EEPROM	Radio transceiver	BW (bps)	Freq. (MHz)	OS	Year	Organisation
CoSeN/MantaroBlocks[a]	Atmel ATmega32A4	32	4 K/32 K/1 K	Atmel AT86RF231	250 k	2,400	Atmel Studio	2012	UMBC/Mantaro
FireFly3	Atmel ATmega128RFA1	16	16 K/128 K/4 K	Atmel ATmega128RFA1	250 k	2,400	Nano-RK	2012	Carnegie Mellon University
M12 module[a]	Freescale MC13224v (ARM7)	26	96 K/128 K	Freescale MC13224v (802.15.4)	250 k	2,400	Contiki OS	2012	RedWire
panStamp[a]	Atmel Atmega328p	8	2 K/32 K/1 K	TI (Chipcon) CC1101	1.2 k	868/915	panStamp	2012	panStamp
WiSMote mini[a]	Atmel ATmega128RFA2	16	16 K/128 K/4 K	Atmel ATmega128RFA2	2 M	2,400	Contiki	2012	Arago Systems

[a]Commercially available platforms
[b]Designed or used for BSN application

- Sensor Interface – interface with sensors and other devices
- Power Supply – the power source of the sensor node

A.2.1 Processor

Most WSN platforms are based on COTS (*Commercial Off-The-Shelf*) components, and the development of WSN depends extensively on the rapid advances of microprocessors. The majority of the WSN platforms use 8-bit or 16-bit RISC processors, such as the Atmel ATmega and *Texas Instruments*' (TI) MSP430 microcontrollers, due to their low power design, integrated multi-sensor interfaces and widely available developing tools. More recently, some platforms have started to use the ARM Cortex processors due to an increase in computational power of the sensor nodes.

A.2.2 Wireless Communication

Wireless communication is the most power-demanding component of WSNs. This often accounts for more than 50 % of the overall power budget of a sensor node [7]. Parallel to the development of micro-power radio transceivers, such as the Pico radio [8], existing research in WSNs has been focused on developing energy-efficient protocols and routing strategies. The three main components of wireless communication include the radio transceiver, antenna, and communication protocols.

A.2.2.1 Radio Transceiver

Among different radio transceivers, the TI (Chipcon) CC2420 is the most popular of the IEEE 802.15.4 chipsets used in the WSN platforms. Since the introduction of *SoC* (System on a Chip) by integrating both microcontroller and radio transceiver onto a single chip (e.g., Nordic nRF24E1 and ATmega128FEA1), new WSN platforms are increasingly using bespoke SoC chipsets to minimise the footprint and reduce the overall size of the sensor node.

A.2.2.2 Antenna

For WSN and BSN, antenna design is important, although the majority of the current WSN platforms use COTS antennas. Among these, the ceramic antennas are most commonly used in WSN platforms due to their small footprints and consistent signal quality for mass production. For BSNs, particularly for implantable applications, miniaturised antenna design remains a key research topic, as mentioned in earlier chapters of this book.

A.2.2.3 Communication Protocol

Depending on the hardware and the operating systems used, early WSN platforms mainly relied on proprietary communication protocols. The introduction of the IEEE 802.15.4 standard enables the standardisation of communication between WSN platforms. Many recent WSN platforms have adopted this standard as the basis for their wireless communication protocol. Examples of this include Telos, MicaZ, Pluto, iMote2, Tmote sky, XYZ node, ProSpeckz and the BSN node.

Due to the broad range of WSN applications, the 802.15.4 standard defines only the MAC and physical layers of the communication protocol. This enables the design of application-specific protocols. For example, Zigbee is built based on the 802.15.4 standard and is designed to ease the inter-operation between different devices. Zigbee specifies all the protocol layers required for forming a wireless network and it also provides an interface for application development.

As mentioned in Chap. 5, although Zigbee is designed for low power sensor communication and enables interoperability, the power consumption of Zigbee is still relatively high for BSN applications. Following the release of the Bluetooth Low Energy (Bluetooth SMART) and the standardisation of IEEE 802.15.6, it is envisaged that new BSN platforms will increasingly rely on these new communication protocols, whereas WSN applications will remain to use 802.15.4 and Zigbee protocols.

A.2.3 Memory

Since limited *Random Access Memory* (RAM) is provided by MCUs, most WSN platforms are designed with an external flash memory or *Electrically Erasable Programmable Read-Only Memory* (EEPROM). Due to the non-volatile nature of the EEPROM, it is used in most embedded systems for storing configuration information because it does not require power to retain the stored data. It is also used as an immediate storage for sensor readings. For instance, in order to perform feature extraction or filtering of the sampled data, the EEPROM can be used as a processing buffer for these algorithms. Another use of the EEPROM is for storing program images. In addition to EEPROM, some WSN platforms have designed with a micro-SD card interface for data logging. Although micro-SD card consumes more power than EEPROM, the large storage capacity is ideal for long-term data logging.

A.2.4 Sensor Interface

To enable practical application development, most WSN platforms offer analogue and digital sensor interfaces.

A.2.4.1 Analogue Interface

Sensors such as simple photo resistors and thermistors, or more complex gyroscope and condenser microphones, generally provide analogue readings. Most WSN or wearable BSN platforms are equipped with ADC interfaces for data sampling and acquisition. For instance, the Atmel Atmega128L MCU has an eight-channel 10-bit ADC that can sample at a rate up to 15.4 ksps (*kilo-samples per second*), whereas the TI MSP430 microcontroller has a 12-bit ADC, which provides a higher precision than that of the Atmel processor. In addition to ADCs, some platforms are equipped with *Digital-to-Analogue Converter* (DAC) for controlling sensors or actuators.

A.2.4.2 Digital Interface

Since analogue readings are prone to voltage drift caused by the depletion of the battery power, sensors such as the 9-axis inertial motion unit MPU-9150 [9] provide direct digital readings. As sensor data is relatively small in size, serial communication is mainly used for interfacing with digital sensors, and the three most commonly used serial communication protocols are I^2C, SPI and UART.

A.2.4.3 Integrated Sensors

To ease application development, many WSN or wearable BSN platforms have built-in sensors such as humidity, temperature, inertial, magnetic and photo sensors. With integrated sensor board design, the hardware platform can be made more compact and immune to noise induced by cables and connectors. However, integrating sensors on the hardware platforms can limit the general use of the platforms as different applications may have varying sensor requirements.

A.2.5 Power Supply

Currently, power supply is the main determining factor for the size and lifetime of the WSN or BSN hardware. Similar to mobile phones, the battery or alternative power source is often the largest single component of these sensor nodes. To miniaturise the sensor nodes, a number of alternatives have been proposed. With recent advances in power harvesting technologies, it has been demonstrated that WSN sensor nodes can be powered by motion, temperature gradient, inductive coupling and other alternatives. However, batteries still remain the main source of power for current WSN or wearable BSN platforms. Among different battery technologies, Lithium-ion or Lithium Polymer batteries are the most popular choice

for sensor hardware because of their high power density. Although zinc-air batteries have a higher energy capacity than that of Lithium batteries, the high rate of power drains from the current radio transceivers limits the direct use of zinc-air battery for BSN applications. To simplify sensor deployment, most WSN platforms have integrated batteries. As discussed in earlier chapters of this book, the development of new power scavenging techniques coupled with ultra-low power BSN designs could provide significant improvements in BSN design in future years.

A.3 Conclusions

In this chapter, we have outlined the common WSN and wearable BSN development platforms that have emerged in recent years. Although many of the WSN hardware platforms can be adapted for BSN applications, due to the specific requirements and constraints imposed by BSNs, a number of dedicated platforms have been proposed for BSN research and development. As the BSN platform technologies mature, recent research has been increasingly focused on the development of optimised BSN platforms for both wearable and implantable applications. Many of them are entering into routine clinical use, thus realising the original goal of the BSN in supporting pervasive health monitoring, early detection and personalised treatment of diseases.

References

1. (2004). *WsLAN summary: WP1000 – state of the art*. Available: http://www.sintef.no/units/informatics/projects/wslan/WP1000-summary.pdf
2. M. Weiser, “Hot topics: ubiquitous computing,” *Computer,* vol. 26, pp. 71–72, Oct 1993 1993.
3. J. W. Ng, B. P. Lo, O. Wells, M. Sloman, N. Peters, A. Darzi, C. Toumazou, and G.-Z. Yang, “Ubiquitous monitoring environment for wearable and implantable sensors (UbiMon),” in *International Conference on Ubiquitous Computing (Ubicomp)*, 2004.
4. B. Lo, S. Thiemjarus, R. King, and G.-Z. Yang, “Body sensor network-a wireless sensor platform for pervasive healthcare monitoring,” in *The 3rd International Conference on Pervasive Computing*, 2005, pp. 77–80.
5. V. Shnayder, B.-r. Chen, K. Lorincz, T. R. F. Jones, and M. Welsh, “Sensor networks for medical care,” in *Conference On Embedded Networked Sensor Systems: Proceedings of the 3 rd international conference on Embedded networked sensor systems*, 2005, pp. 314–314.
6. B. Kuris and T. Dishongh. (2006, SHIMMER - Sensing Health with Intelligence, Modularity, Mobility, and Experimental Reusability Available: http://www.snm.ethz.ch/snmwiki/pub/uploads/Projects/SHIMMER_HWGuide_REV1P3.pdf
7. V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M. Welsh, “Simulating the power consumption of large-scale sensor network applications,” in *the Second ACM Conference on Embedded Networked Sensor Systems*, Baltimore, MD, 2004, pp. 188–200.
8. M. J. Ammer, M. Sheets, T. Karalar, M. Kuulusa, and J. Rabaey, “A low-energy chip-set for wireless intercom,” in *the Annual ACM IEEE Design Automation Conference*, Anaheim, CA, 2003, pp. 916–919.

9. (2012). *Nine-Axis (Gyro + Accelerometer + Compass) MEMS MotionTracking™ Devices*. Available: http://www.invensense.com/mems/gyro/nineaxis.html
10. K. Mechitov, W. Kim, G. Agha, and T. Nagayama, "High-frequency distributed sensing for structure monitoring," in *Proc. First Intl. Workshop on Networked Sensing Systems (INSS 04)*, 2004.
11. J. Jeong, S. Kim, and A. Broad, "Network reprogramming," *University of California at Berkeley, Berkeley, CA, USA,* 2003.
12. G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh, "Deploying a wireless sensor network on an active volcano," *Internet Computing, IEEE,* vol. 10, pp. 18–25, 2006.
13. J. Beutel, "Fast-prototyping Using the BTnode Platform," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, 2006, pp. 1–6.
14. U. Maurer, A. Rowe, A. Smailagic, and D. P. Siewiorek, "eWatch: a wearable sensor and notification platform," in *Wearable and Implantable Body Sensor Networks, 2006. BSN 2006. International Workshop on*, 2006, pp. 4 pp.-145.
15. M. Laibowitz, J. Gips, R. AyIward, A. Pentland, and J. A. Paradiso, "A sensor network for social dynamics," in *Information Processing in Sensor Networks, 2006. IPSN 2006. The Fifth International Conference on*, 2006, pp. 483–491.
16. A. Elsaify, P. Padhy, K. Martinez, and G. Zou, "GWMAC- A TDMA Based MAC Protocol for a Glacial Sensor Network," presented at the 4th ACM PE-WASUN 2007, 2007.
17. D. Lymberopoulos, N. B. Priyantha, and F. Zhao, "mPlatform: a reconfigurable architecture and efficient data sharing mechanism for modular sensor nodes," in *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, 2007, pp. 128–137.
18. R. B. Smith, "SPOTWorld and the Sun SPOT," in *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, 2007, pp. 565–566.
19. J. Andersen, B. Lo, and Y. Guang-Zhong, "Experimental platform for usability testing of secure medical sensor network protocols," in *Medical Devices and Biosensors, 2008. ISSS-MDBS 2008. 5th International Summer School and Symposium on*, 2008, pp. 179–182.
20. A. T. Barth, M. A. Hanson, H. C. Powell, and J. Lach, "TEMPO 3.1: A Body Area Sensor Network Platform for Continuous Movement Assessment," in *Wearable and Implantable Body Sensor Networks, 2009. BSN 2009. Sixth International Workshop on*, 2009, pp. 71–76.

Appendix B: BSN Software and Development Tools

Joshua Ellul, Benny Lo, and Guang-Zhong Yang

B.1 Introduction

Practical applications of Body Sensor Networks (BSN) give rise to programming challenges that are not present in traditional computing systems. The typical memory resources available on sensor nodes are extremely limited, typically equipped with tens of kilobytes of program space and 10 KB of volatile memory. Therefore, algorithm developers must ensure that memory is not used imprudently. More challengingly, typical applications impose prolonged lifetimes, whilst sensor nodes are deployed with extremely limited power resources. Therefore, software developers must ensure that algorithms are highly efficient and that the processor and any peripheral hardware are put into sleep modes as much as possible. Sensor nodes are required to communicate with other nodes in order to relay sensed information or to send updates or configuration messages into the network. Therefore, nodes usually cannot sleep indefinitely and are required to follow wakeup schedules so that they can communicate in predetermined communication windows. Further programming complexity is increased due to internal clock drift which sensor nodes are prone to, and therefore clocks must be corrected to accommodate for such drift typically done using time synchronisation techniques. The complexity does not end there; the wireless medium over which sensor nodes communicate only allows one node to communicate at a time (within the same transmission range). Therefore, Medium Access Control (MAC) protocols must be implemented to avoid wireless collisions. Also, sensor networks aim to maximise lifetime by optimising the route taken from sensor nodes to base stations to consume the least amount of energy. A plethora of MAC and routing protocols

J. Ellul • B. Lo • G.-Z. Yang, PhD (✉)
The Hamlyn Centre, Imperial College London, London SW7 2BY, UK
e-mail: g.z.yang@imperial.ac.uk

G.-Z. Yang (ed.), *Body Sensor Networks*, DOI 10.1007/978-1-4471-6374-9,

have been proposed that focus on different aspects including scalability, dynamicity, responsiveness and density amongst other attributes.

The challenges that face sensor node application programming also stem from the low level embedded programming expertise required. Wireless sensor nodes typically comprise of a microcontroller, a wireless transceiver, a number of sensors and other peripherals such as additional flash storage. Drivers must be implemented for each hardware peripheral which typically communicate over SPI or I^2C using a strict communication protocol often involving low level registers. Such hardware devices often utilise General Purpose Input/Output (GPIO) pins for status updates, which are in turn wired up to microcontroller interrupts. Moreover, substantial internal microcontroller peripherals are wired to interrupts. Thus, knowledge and programming experience of interrupt based systems is required to program such low level embedded systems.

Sensor node applications are predominantly developed in C or flavours of C such as nesC [1]. Therefore, development challenges also include those faced by traditional low-level systems developed in C, including allocation and deallocation of memory by the programmer, as well as no type safety mechanisms. The most popular sensor network operating system, TinyOS, allows developers to code applications in nesC which exposes an event based programming paradigm. The abstraction layers provided are very low, and, as noted by [2], "*it is often difficult to implement even simple programs*."

As described above, the development learning curve for sensor networks is steep. Higher-level languages providing higher abstractions can be used to lower the learning curve substantially. Higher-level abstractions can help by hiding the lower-level embedded systems and sensor networks specific requirements. Nodes can be put to sleep automatically by underlying drivers; default routing and MAC protocols can be used and swapped without the higher level developer having to change any code in relation to this; drivers for different hardware peripherals can be used and then exposed to the higher level language by abstracting the communication protocols, interrupts and register access. However, in using a higher-level language such as Java, the developer is also relieved of memory management since this will be taken care of by a garbage collector. Implicit type safety also ensures that the programmer will not incorrectly cast types. More so, the majority of the available workforce is already familiar with high level languages, such as Java. Therefore, the gradient of the learning curve can be drastically decreased be removing the requirement to learn a new language. It has been shown that higher level languages such as Java provide a more efficient development and maintenance environment compared to lower level languages such as C [3]. Therefore, by using a higher-level language, the development and maintenance costs of BSNs can be reduced, and thus increase the successful adoption of such technology.

B.2 BSN Requirements and Issues

BSNs consist of a number of sensor nodes that can sense the environment, process the sensed data and transmit (and receive) the processed data for an extended period of time. The requirements and challenges inherent in a BSN can directly influence

programmability. Therefore, the main requirements and issues of BSNs will be described here with a focus on how they affect ease of programming.

Body sensor nodes are most often equipped with a limited energy source and are usually expected to operate for an extended period of time. A primary requirement of BSNs lies in the fact that they are deployed in environments monitored in an unobtrusive manner. This relies on miniaturisation of nodes and therefore battery size, which in turn requires an energy efficient system. In order to meet the expected lifetime, and given the limited battery source, the system must be as energy-efficient as possible. This involves ensuring that the wireless transceiver, sensors and any other devices are only used and turned on when they are required. This becomes more complex when considering other aspects including MAC protocols, routing protocols and time synchronisation amongst other factors. Programmers are commonly required to explicitly turn different hardware components on and off, and even in to different sleep modes. Such fine-grained control of hardware is often intimidating to programmers since they are not familiar with such low level fine-grained control. Computational efficiency on the other hand is often considered to be of minor importance and therefore cheaper, more energy-efficient (and slower) processors than those used in larger platforms can be used. However, computational efficiency is often discarded without considering the impact it may have on quality of service and energy expenditure. Computational inefficiency can impact energy expenditure both directly and indirectly [4]. Therefore, programming environments should be both as energy and computationally efficient as possible without requiring extensive effort from the system developer.

Body sensor nodes communicate with each other over the same physical, wireless medium. Therefore, protocols and overall system implementation must be able to scale with the network size and limited bandwidth. Developers typically have to cater for such low-level intrinsic properties when really, they should concentrate their effort on application requirements. In addition to the work involved in deploying a BSN, one must also keep in mind that like other computing platforms, sensor nodes may be required to be updated from time to time due to various reasons including bug fixes, new application requirements and even complete retasking of a network. Therefore, software reconfiguration and reprogrammability is essential, however this is often left up to the developer to implement.

The underlying theme from the above is that the low level internals of BSNs is more often than not left up to application developers to implement, when really they should be focusing on the application specific requirements. It was for this reason that a new operating system was designed to facilitate ease of programming for BSNs.

B.3 Operating Systems for BSNs

The first task required to implement a BSN application is to choose an operating system. The choice of operating systems will also greatly affect many aspects of the development life cycle. Table B.1 provides a development-centric comparison of

Table B.1 Operating systems for BSNs

Operating system	Programming paradigm	Pros	Cons
BSNOS	Java Single-threaded	Easy to use High-level programming language Integrated development environment	Not suitable for complex networking protocol research or implementation Currently only supports the BSN platform
TinyOS	nesC Event driven, TOS thread support can be added	Supports many hardware platforms Extensive code available	Requires learning a new programming language Even simple applications can be hard to implement Does not support the BSN platform out-of-the-box
Contiki	C Event driven and protothread support	Supports many hardware platforms Extensive code available	Requires C programming experience Does not support the BSN platform out-of-the-box
Mantis OS	C Multi-threaded	Supports several platforms	Requires C Programming experience Does not support the BSN platform out-of-the-box
LiteOS	LiteC++ Events and multi-threaded	Supports several platforms Easy to use	Does not support the BSN platform out-of-the-box

popular operating systems typically used traditional for wireless sensor networks and a new operating system specifically proposed for ease of development of BSNs.

The programming language and Application Programming Interface (API) are prime factors that determine ease of developing applications. TinyOS [5] is a popular operating system for wireless sensor networks. Applications are written in nesC [1], an event-driven, component-based language. Although, the event-based paradigm does couple well with microcontroller based interrupts, event handling is non-trivial for those beginning programming. Also, the requirement to learn a new programming language for only BSN development is unfavourable. Contiki [6] and MANTIS OS [7] require applications to be programmed in C. Although C is the primary language of choice for embedded systems developers, it does not provide high level abstractions that would be sufficient for novice programmers. LiteOS [8] provides a UNIX-like operating system abstractions and requires applications to be programmed in LiteC++. The operating systems provide extensive support for different platforms, networking protocols and other low-level features which are useful to those conducting research in the low-level wireless sensor networks field, however the abstraction provided is still

too low for non-electronics and computer scientists. BSNOS [9] is an operating system targeted specifically to facilitate ease of programming for BSNs and is further described in the next section.

B.4 BSNOS – An Operating System for BSN

Operating systems developed for wireless sensor networks and bare metal coding have been popular choices for developing body sensor node applications. Expert knowledge of C and embedded programming has thus been required to create BSN applications, which even highly experienced programmers still find as a challenging task. Application deployment, and therefore data collection is usually therefore delayed due to few such expert programmers being available.

BSNs face different challenges and are prone to different requirements than that of traditional wireless sensor networks. WSN environments are typically much larger than that of the body. Also, the human body is geometrically the same to other body deployments (unlike environments where wireless sensor networks are used). Thus, BSNs require networks of smaller sizes with known locations for node placement. A BSN does not tend to change in size. A nodes placement on a body is unlikely to move, although the body is. Body sensor nodes do not require long-range communication and more often than not can communicate directly with all other nodes. Therefore, body sensor networks do not require the complex MAC and routing protocols, which are a primary requirement for most wireless sensor networks.

A smaller sized network implies that BSNs are relieved from scalability issues usually prone to larger networks. However, smaller sized networks also means that nodes cannot rely on neighbouring nodes for redundancy. Therefore, BSN applications are required to be more robust and resilient to noise and errors. More so, long sleep periods are a luxury that typical body sensor networks do not have since if an event is missed it could be vital. Short sleep times in turn affect the lifetime of the sensor node, which is already a challenge due to constrained battery sizes. That said, most body sensor network deployments do not require extended lifetimes since they are usually deployed temporarily to analyse or detect specific conditions or applications.

BSNOS [9] is an operating system specifically targeted to the challenges and requirements of BSNs with focus on ease of development for early stage programmers as well as scientists from other fields of research. Most wireless sensor networks programming tools require C programming as a means of development. BSNOS on the other hand, exposes a Java programming environment to lower the learning curve required to develop BSN applications. To facilitate an efficient execution platform, a run-time compiler similar to [10] is used in BSNOS. Figure B.1 presents an overview of the BSNOS main components and development tools.

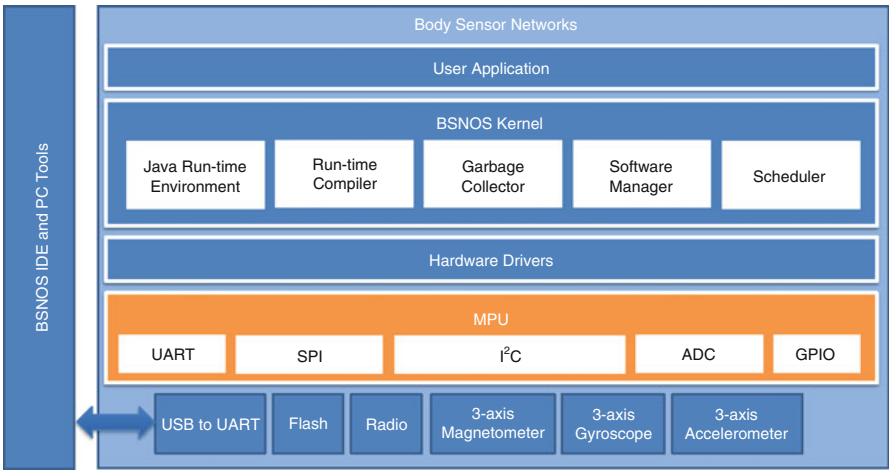

Fig. B.1 An overview of BSNOS and its main components

User applications sit at the top of the software stack. The applications are loaded on top of the BSNOS kernel, which is the main operating system component. The BSNOS kernel is comprised of five main subcomponents. At the core of the kernel lies the Scheduler. The Scheduler is responsible for scheduling execution of timers, threads and events. Body sensor networks may require updates and bug fixes occasionally. The Software Manager meets the requirement of software updates by exposing routines to dynamically install and remove classes and functions.

The run-time environment exposes a subset of Java functionality including object and array manipulation amongst other features. In order to provide an efficient Java execution platform, a run-time compiler has been implemented. One of Java's major benefits is that it relieves developers from memory management. When memory previously used by an application is no longer used, a check is required to release such memory so that it can be used by other parts of the application. This is the job of the Garbage Collector.

The kernel interacts with hardware drivers which communicate directly with the underlying hardware including the microcontroller (Texas Instruments MSP430F1611) and its associated peripherals, 32 Mb Flash storage (Adesto AT45DB321D), a 2.4 GHz RF transceiver (Texas Instruments CC2420), a 3-Axis digital magnetometer/compass (Honeywell HMC5843), a 3-axis digital gyroscope (InvenSense ITG-3200) and a 3-axis accelerometer (Analog Devices ADXL330).

Since ease of use is one of the main driving factors for BSNOS an application programming interface (API) that provides a very high level abstraction of the underlying hardware is provided. A sample of the API made available is presented in Table B.2.

Table B.2 BSNOS sample API

Module	Function	Description
Accelerometer	void performAccelSample()	Instructs the Accelerometer driver to acquire an accelerometer reading
	short getAccelX()	Returns the X-axis reading for a previously acquired accelerometer reading
Radio	void appendShortToRadio (short s)	Appends a 16-bit value to the radio message buffer
	byte sendRadioMsg(short dest)	Send a radio message comprised of the content in the radio message buffer to the destination address specified in the parameter
LEDs	void toggleLed(byte ledNr)	Toggles the state of the LED indicated by the parameter
Timer	void waitMS(short ms)	Waits for the specified number of milliseconds

```
public static void main() {
    while ( true ) {
        BSN.performAccelSample();
        BSN.appendShortToRadio( (short) BSN.getAccelX() );
        BSN.appendShortToRadio( (short) BSN.getAccelY() );
        BSN.appendShortToRadio( (short) BSN.getAccelZ() );
        BSN.sendRadioMsg( (short) BSN.BROADCAST_ADDR );
        BSN.waitMS( (short) 20 );
    }
}
```

Fig. B.2 A simple sample and send program

The high-level programming interface enables for BSN programs to be written in not more than a few lines of code. A simple sample and send program is demonstrated in Fig. B.2.

To further provide a platform allowing for ease of development of BSN applications an Integrated Development Environment (IDE), BSNOS IDE, is provided along with BSNOS. The IDE is based on the Eclipse platform and bundled as a single executable. The IDE, besides providing Code Completion, Import/Export functionality and all other Eclipse based features, also provides an easy way to create BSN specific applications including Projects Wizards for both Body Sensor Nodes, base stations and PC based software; downloading software to sensor nodes by the touch of a button and classes to easily browse functionality available via BSNOS.

**More details on BSNOS and downloads are available from:* http://www.bsn-web.org/

References

1. Gay D, Levis P, von Behren R, Welsh M, Brewer E, Culler D, editors. The nesC language: A holistic approach to network embedded systems. The ACM SIGPLAN 2003 conference on Programming language design and implementation; 2003.
2. Sugihara R, Gupta RK. Programming models for sensor networks: A survey. ACM Transactions on Sensor Networks. 2008;4(2):8:1–8:29.
3. Butters AM. Total cost of ownership: A comparison of C/C++ and Java. 2007.
4. Ellul J. Run-time compilation techniques for wireless sensor networks: University of Southampton; 2012.
5. Levis P, Madden S, Polastre J, Szewczyk R, Woo A, Gay D, et al., editors. TinyOS: An operating system for sensor networks. Ambient Intelligence; 2004.
6. Dunkels A, Grönvall B, Voigt T. Contiki - a lightweight and flexible operating system for tiny networked sensors. Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks 2004. p. 455–62.
7. Bhatti S, Carlson J, Dai H, Deng J, Rose J, Sheth A, et al. MANTIS OS: an embedded multithreaded operating system for wireless micro sensor platforms. Mobile Networks and Applications. 2005;10(4):563–79.
8. Cao Q, Abdelzaher T, editors. liteOS: a lightweight operating system for C++ software development in sensor networks. Proceedings of the 4th international conference on Embedded networked sensor systems; 2006.
9. Ellul J, Lo B, Yang G-Z. The BSNOS platform: a body sensor networks targeted operating system and toolset. The 5th International Conference on Sensor Technologies and Applications 2011. p. 381–6.
10. Ellul J, Martinez K. Run-time compilation of bytecode in sensor networks. The 4th International Conference on Sensor Technologies and Applications 2010. p. 133–8.