Int. j. inf. tecnol. (March 2018) 10(1):49–58
https://doi.org/10.1007/s41870-017-0044-5

ORIGINAL RESEARCH

Operating systems for low-end smart devices: a survey and a proposed solution framework

Jasleen Kaur[1] · **S. R. N. Reddy**[1]

Received: 5 May 2017 / Accepted: 26 September 2017 / Published online: 24 October 2017

Abstract Everything around us is getting smarter with whole new world of technology called smart device that has changed the way we interact in our daily lives. Smart Devices comprise of both high-end and low-end devices with respect to software and hardware platform. High-End smart devices can run on the conventional OS like Linux though for low-end smart devices it is impossible to run as such because of stringent resource constraints. In this paper, detail survey and comparative analysis of various operating systems available for low-end smart devices is carried out to bring light on their features and potential pitfalls. Furthermore, survey in contour of feedback at various technical institutes is done to gather the research problems faced in context of smart device operating systems. This survey resulted in finding the essential requirements primarily an operating system must meet to run on low-end smart devices. The findings contribute to the proposal of a generic framework for automatically generating application specific lightweight operating system for low-end smart devices.

Keywords Smart device · Smart OS Framework · Low-end devices · Lightweight operating system · Generic OS framework · Developing smart device

✉ Jasleen Kaur
Jasleenkochar89@gmail.com

S. R. N. Reddy
srnreddy@igdtuw.ac.in

[1] Department of Computer Science, Indira Gandhi Delhi Technical University for Women, New Delhi, Delhi, India

1 Introduction

The Smart environment is portrayed by the ubiety of intelligence in common objects [1]. It can incredibly affect the way we experience our lives, with numerous ranges where there are the potential outcomes of connecting the devices and sensors. Some smart technology integration and innovations out of the throng are smart light bulb [2], smart water saving system [3], smart health cards [4], smart industrial automation [5] and smart home automation [6]. What makes these devices "Smart" is their ability to access web services, thereby enabling us to interact with data regardless of the location, type and number of devices that may be used. One example is smart health monitoring [4] that has turned into the piece of our lives for over 10 years, where the devices can be controlled, connected or monitored remotely.

As show in Fig. 1, about (47%) of the smart devices are in Healthcare, trailed by security (34%), energy and utilities (30%) and manufacturing (28%). Biotechnology, safety, retail and transportation each positioned in the center at about 20%, with construction, entertainment, and education positioned lowest [7]. Keeping in mind the end goal to model the low-end smart device, one must comprehend what the smart device is composed of and what are the different classes for low-end devices.

1.1 Components of smart device

Basic Components of a Smart Device as shown in Fig. 2 comprises of computing platform, operating system, communication module, application specific sensors, memory interface, database, power component and input/output units.

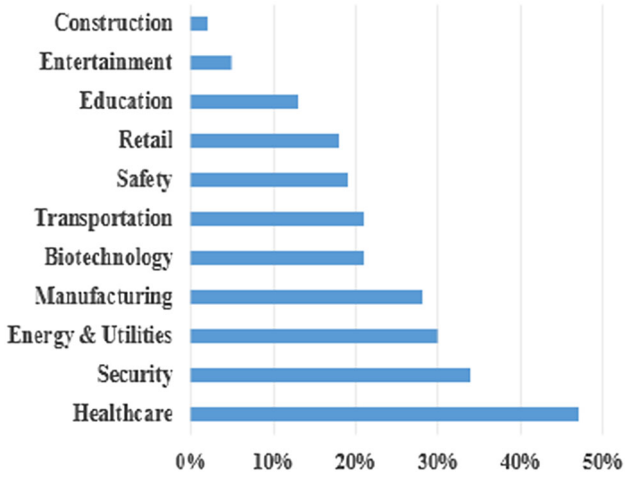

Fig. 1 Smart device area distribution

Computing platform is a hardware component required for software/OS executions and quicker execution of operations as the agility in smart devices expands. Therefore, requirement for the satisfactory computing platform is obvious. *Input/output* units represent those components responsible for entry, storage or processing of data/information. Smart device needs an Input/output unit to interact with a user. *Application specific sensors* take input from the smart environment and uses built in resources to execute the predefined job upon the detection of particular input. These sensors are used for real time monitoring and control mechanism in related environment and hence should give more accurate and automated collection sensed data with limited erroneous noise during the precisely recorded information. *Database* component consists of the compilation of information in a systemized manner so that it can be freely accessed, managed, and updated [8]. A Smart device required to be connected to the database component in order to enhance the performance and facilities. *Memory is* a tangible device adept of storing the information transiently or permanently [9]. All the smart devices consist of internal memory to store and process the certain tasks. Therefore memory is needed in the smart device for the better performance, multitasking and to perform complex operations. *Power Component* referred to as a source of power being catered to a smart device. Source of power can be catered in a different means, such as battery, power bank, solar panel or main power supply. Smart device needs a power component in order for its other peripherals to operate accurately with acceptable framework. *Communication Module* is one of the essential module in building a smart device in light of the fact that if a smart device ought to connect with other devices in its smart environment then it must give a mode of communication to these other devices. *Operating System* is a software that manages the resources and allows smart device to run applications and programs. Smart device need to be good enough to receive, compute, store and process the data by performing certain tasks. Hence, operating system component is desired to support connectivity, communications and security standards.

1.2 Low-end smart devices

Low-end smart devices are basically resource constrained in terms of CPU, energy and memory. Some of the well-known examples of low-end smart devices comprises of Arduino Due, Atmel Sam R21, Econotag, IoT-LAB M3 nodes, OpenMote nodes, Telosb motes and Zolertia Z1. As

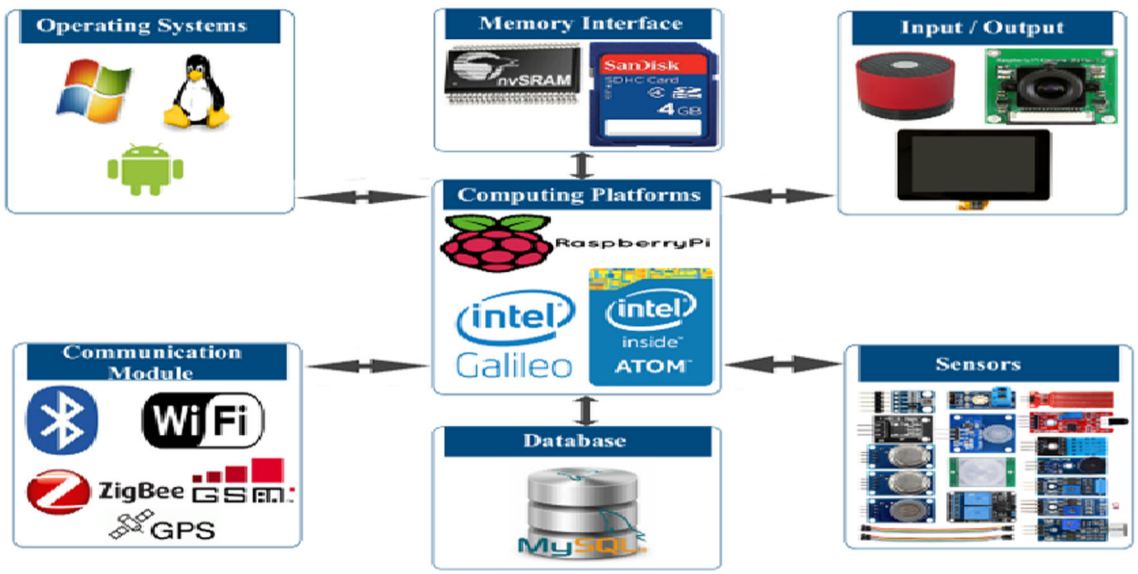

Fig. 2 Basic components of a smart device

reported by Internet Engineering Task Force (IETF), these low-end devices are standardized into three subcategories [10] in view of memory limit- class 0, class 1 and class 2.

- Class 0 (with low level resources)—devices with $\ll$ 10 kB data size and $\ll$ 100 kB code size.
- Class 1 (with medium level resources)—devices with $\sim$ 10 kB data size and $\sim$ 100 kB code size.
- Class 2 (with high level resources)—devices with $\sim$ 50 kB data size and $\sim$ 250 kB code size.

Class 0 devices are extreme constrained nodes. Larger devices such as gateways, servers, or proxies are mandatory for class 0 devices to communicate directly with the internet in a secure aspect. Therefore, software running on such devices need to be application and hardware specific. Class 0 devices for the most part comprises of MCU based on 8-bit architecture whereas class 1 and class 2 devices can go from 8 to 32-bit architectures. The shared factor of all the low-end devices are that their resources are too restricted to be able to run a resource demanding operating system like Linux [11]. It is expected that smart devices will develop in the direction of smaller size, low production cost, cheaper and more energy efficient [11]. Hence, lightweight operating system is enforced that can keep running on hardware platform with restricted resources. Different lightweight operating systems (open source and closed source) have been developed for low-end smart devices which are detailed in Sect. 3.1. Architecture of these operating systems may vary but they all offer a technique for interfacing the devices to the internet.

This paper highlights the essential traits of a smart device operating system and proposes a generic framework that meets the user expectations and helps them to create lightweight operating system for the particular use case. The remaining paper is organized as follows. Section 2 exemplifies the design aspects of an operating system for a low-end smart device. Section 3 illustrate the research workflow used in this paper i.e. literature survey, asserting the research problem, and result analysis. Section 4 proposes the new generic framework for a low-end smart device. Section 5 outlines the conclusions followed by the future work.

2 Design aspects of an OS for smart device

The entire unification of the various smart devices create the smart environment [2]. This unification is attainable over the software communication at a dynamic level [2] which is achievable through the operating system running within each smart device.

The operating system grown for a smart device requires less kilobytes of memory and small power utilization. The kernel of the operating system for a smart device can be either erected in a layered approach, microkernel or monolithic architecture [12]. The programming model of these operating systems should be such that experts can smoothly use the device. High level language is appropriate for the smooth development although assembly language is the finest substitute to interface with the hardware [12, 13]. Scheduler of these operating systems should be real time and support task priorities [13, 14]. Internet connectivity is an intrinsic requirement of a smart device. Enormous communication requirements of smart environment are not accomplished by the WSN protocols such as Bluetooth, Z-Wave, ZigBee etc. whereas distinctive requirement of a smart device can be achieved through these WSN protocols [12, 13]. Tools such as RPL, CoAP, 6LoWPAN etc. [12] are designed for low power systems. Operating system for a smart device should be integrated with the application specific communication requirement. These operating systems ought to be efficiently portable to various hardware platforms and support huge range of hardware architectures. An operating system for a smart device should be open source so as to customize it as per the specific needs of application [12, 14].

3 Method

As shown in Fig. 3, the research design used in this paper includes various steps. In this section results are formalized based upon the literature survey and questionnaire based user feedback survey. Brief literature survey of various existing operating systems for smart devices is done with the main objective being to find their features and flaws. Essential requirements adopted by majority of smart device operating systems are formalized based upon the literature survey. Furthermore, questionnaire based user feedback survey is done to know the perception of users regarding smart device development and requirement of an operating system for the particular use-cases. Participants in this survey included 90 from two different technical institutes.

3.1 Literature survey

The literature survey of the few existing operating systems targeted for a low-end smart device is carried out to boost

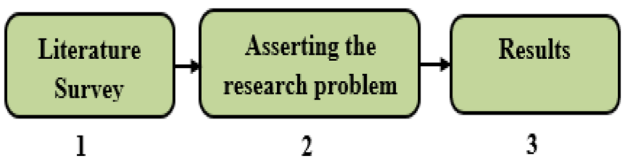

Fig. 3 Research design

the knowledge about the problem domain. Furthermore, literature survey provides the theoretical background and knowledge regarding what others have done within the area of operating systems for smart devices.

Contiki [15] is an open source operating system designed for tiny sensor nodes with a few kilobytes of accessible memory. It supports pre-emptive multithreading and build around an event driven kernel [15, 16] Contiki is executed in the C language and supports ample range of tiny platforms such as 8051-powered SoC through MSP430, Atmel AVR and number of ARM devices. It includes complete IP network stack with UDP and TCP support, in addition to some low power standards such as 6lowpan, RPL and CoAP. Contiki supports RIME which is a lightweight stack for network based communication. It give security solution for wireless sensor networks as ContikiSec [17]. Network simulator named Cooja, is supported by Contiki for providing the simulation environment to the applications.

TinyOS [18]. is an open source operating system designed for low powered wireless devices used in sensor networks, smart buildings, smart meters, pervasive computing etc. The base TinyOS framework is under 400 bytes. It supports multithreading and threads are called as TOS threads. TinyOS is executed in nesC language a dialect of C and supports wide range of hardware platforms such as Atmel AT90L-series, Atmel AT mega-series, and Texas Instruments MSP-Series processors. TinyOS has no support for ARM and Intel 8051 but work is in progress. The standard TinyOS task scheduler uses FIFO scheduling policy which have no support for real time applications. Support for Earliest Deadline First (EDF) scheduling algorithm has been added to simplify real time applications. It provides support for two multi-hop protocols: dissemination and TYMO, 6lowpan [19], and IPV6 networking layer. It gives database bolster as TinyDB [20] and communication security solution as TinySec [21]. TOSSIM is supported by TinyOS for providing the simulation environment to the applications [22].

Mantis [23] is an open source operating system for wireless micros sensor systems such as BTNodes [24] project, Europe Smart-Its [25] project etc. It is lightweight operating system with a footprint of 500 bytes. Mantis supports pre-emptive multithreading. It is executed in C language and supports extensive range of hardware platforms such as Mica2, MicaZ and Telos. It provides the cross platform support as it can be tested on PC's or PDA before porting to any hardware platform [23]. AVRORA [26] is supported by Mantis for providing the simulation environment to the applications.

VxWorks [27] is a proprietary software, developed initially in 1987 by Wind River. VxWorks supports numerous hardware architectures such as Intel, MIPS, Power PC, SH-4 and ARM architectures. Vxworks has support for IPv6 and other Smart Device features but lack support for 6LoWPAN stack.

QNX [28] is closed source OS, developed by Quantum Software Systems in 1982 which was renamed as QNX software systems. It was acquired by Blackberry in 2010. QNX was one of the first commercially effective microkernel based operating system and has been utilized in many smartphones and cars. The latest version, ONX Neutrino, is supported by many hardware platforms such as PowerPC, X86, MIPS, ARM and SH-4.

Android [29] developed by google is a variation of Linux, focusing largely on touchscreen mobile devices like smartphones and tablets, yet has been additionally utilized as a smart cars, smart watches, smart TV, digital cameras, smart notebooks, game consoles etc. The idea of mobile applications, available through online stores where clients can buy and download applications, helped the advancement of smartphones. The primarily hardware platform support for Android is ARM. It was initially developed by Android Inc. and bought by Google in 2005. The Android source code is publicized by the Google under open source licences, yet many of the device driver and hardware support is proprietary. In 2015, Brillo [30] was announced by the Google, which is a reduced version of Android. Brillo is designed to be utilized for low power and memory constrained smart devices. It is supported across many hardware platforms such as ARM, Intelx86 and MIPS based hardware.

EmbOS [31] is a real time and priority controlled multitasking operating system developed by Segger MCU Systems. It has been optimized for basic memory utilization in both RAM and ROM usage i.e. only the required functions are linked into the application, keeping the ROM size small. EmbOS is composed in ANSI C dialect, highlighting a priority based, tickles, pre-emptive scheduler and focusing on different compelled 8-bit, 16-bit, and 32-bit MCUs. A network stack including ZigBee, USB support, a GUI, and a file system are available as separate add-on products.

FreeRTOS (R. [32] is an open source RTOS which supports pre-emptive multithreading. It is easy to use and has smaller footprint. It is developed by Real Time Engineers Ltd. It supports the tickles mode for low power applications [32] and has been ported to numerous MCU. In spite of the fact that it doesn't give its own network stack, outsider network stacks can be utilized for Internet network.

Key features of the various operating systems surveyed in this section are evaluated and compared in Table 1. As discussed in Sect. 2, there are various design aspects/requirements of an operating system for a low-end smart device. Hence, support of these requirements by an existing

Table 1 Key features of an existing operating systems for low-end smart device

Features → Swart Device OS ↓	Design objectives	Licence	Architecture	Scheduling algorithm	Platforms	Language supported	Support in IDE
Contiki [15]	Commercial	BSD/Free for educational	Monolithic	Interrupts execute w.r.t priority	Tmote, AVR Series, ARM	C	Supported by Eclipse IDE
TinyOS [18]	Educational/ Research.	BSD/Free for educational	Monolithic	FIFO	Atmel AT90L-series, AtnieL ATmega-series, and TI,Tmote	nesC	No Support
Mantis [23]	Educational/ Research	BSD	Layered	Pre-emptive priority based scheduling	Mica2, MicaZ, Telos	C	No Support
VxWorks [27]	Commercial	Proprietary	Layered	Pre-emptive and Round Robin Scheduling	ARM. Intel 64, MIPS, PowerPC, SH-4: Strong ARM, xScale	C/C ++	Supported by Toolbox IDE
QNX Neutrino [28]	Commercial	Proprietary	Microkernel	Priority- Pre-emptive Scheduling	MIPS, PowerPC, 5H-4, ARM, xScale	C/C ++/ Java	Supported by Toolbox IDE
Android [29]	Commercial Educational/ Research	GPL/Free for educational purpose	Monolithic	Fair Scheduling	ARM,X36, MIPS	C/C ++/ Java	Eclipse
EmbOS [31]	Commercial	Proprietary	Modular	Priority based pre-emptive	8/16/32 bit processors, ARM cortex, ARM 7/9/11	C	No support
FreeRTOS [32]	Personal Interest based	Modified GNU GPL	Microkernel RTOS	Priority Based pre-emptive	ARM (ARM7, ARM9, Cortex-M3, Cortex-M4), Atmel AVR,AVR32, SuperH, Nios II.Cortex-R4	C	No support

operating systems surveyed in this section is summarized in Table 2. The closed focuses are epitomized in the Fig. 4.

Following points are concluded from the summarized data of Table 2 and Fig. 4.

- Priority based scheduling is adopted by majority of the operating systems.
- Majority of the operating systems support C language, which is the popular choice for the programming.
- Majority of the operating systems are open source/free based for research and educational purpose.
- Few operating systems support integrated development environment (IDE).
- Majority of the operating system have platform support for ARM processors.
- Very few operating systems have full support for real time requirements.

3.2 Asserting the research problem

There are large numbers of operating systems for a low-end smart device. Few out of the many are surveyed in this paper to formalize the essential requirements of an operating system for a low-end smart device. This variety in operating systems can lead to difficulties for developers as to which operating system is best suited for the particular use-case.

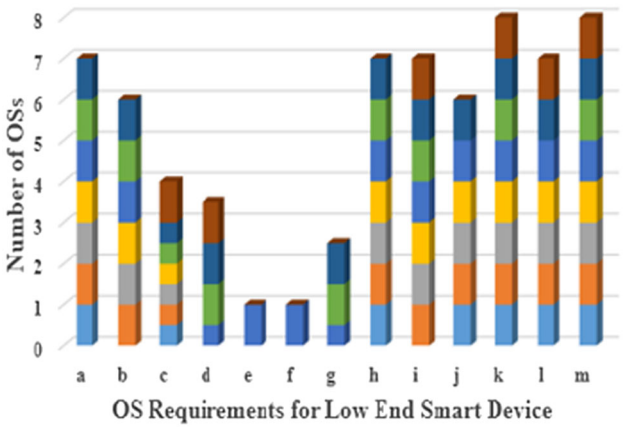

Fig. 4 Support of basic OS requirements

Table 2 Support of basic operating system requirements

OS requirements → / Smart device OS ↓	Open source (a)	Portability (b)	Real Time Requirement (c)	High Reliability (d)	Robustnes (e)	Failure Handling (f)	Security (g)	Scalability (h)	Multiasking (i)	IoT (j)	Scheduling (k)	Network Suport (l)	Memory management (m)
Coatiki [15]	↘	↘	*	×	×	×	×	↘	↘	↘	↘	↘	↘
Tiny OS [18]	↘	×	*	×	×	×	×	↘	×	↘	↘	↘	↘
Mantis OS [23]	↘	↘	*	×	×	×	×	↘	↘	↘	↘	↘	↘
VxWorks [27]	↘	↘	*	↘	×	×	↘	↘	↘	↘	↘	↘	↘
QNX Neutrino [28]	↘	↘	*	↘	×	×	↘	↘	↘	>	↘	×	↘
Android [29]	↘	↘	×	*	↘	↘	*	↘	↘	↘	↘	↘	↘
EmbOS [31]	×	×	↘	↘	×	×	×	×	↘	×	↘	↘	↘
FreeRTOS [32]	↘	↘	*	×	×	×	×	↘	↘	↘	↘	↘	↘

↙-full support, × - no support, * -partial support

Questionnaire based user feedback survey has been conducted in various workshops on "Design and Development of Smart Device" [33]. The questionnaire comprises of few questions solely regarding the operating system requirements for the design and development of a smart device. Following questions with multiple answers were designed which advances to this survey.

Q1. What operating system would you like to employ for your smart device?
Q2. Which programming language would you like to use to build your smart device solution?
Q3. Which communication protocol would you like to use for developing your smart device?
Q4. What are your top 3 concerns for developing smart device?
Q5. What memory footprint would you like to employ for your smart device solution?
Q6. What are the possible ways to build lightweight application specific operating system for your smart device?

3.3 Results

Participants were asked to complete survey questionnaire [34] as given in Sect. 3.2. Responses gathered from total 90 [35, 36] participants of two workshops are summarized in Fig. 5. Based upon the literature survey of an existing operating systems given in Sect. 3.1 and responses gathered from the participants, a list of essential traits for an operating system of a low-end smart device is formed as summarized below. A generic architecture for a low-end smart device is depicted in Fig. 6. Based upon the responses gathered, it has also been found that more that 65% of participants would like to have a generic framework for automatically generating user- directed application specific operating system.

(a) *Support for Heterogeneous Hardware* There are number of use cases within smart environment, hence smart device operating system must be able to support large variety of hardware architectures. Based upon literature survey, large number of operating systems are designed for ARM, Atmel AVR and Texas Instruments MSP Series of processors. Hence an operating system for low-end smart device must be capable of handling heterogeneous platforms in the heterogeneous environment.
(b) *Programming Model* There should be support for appropriate languages i.e. ANSI C and C ++ that allow memory efficient programming for low-end smart devices.

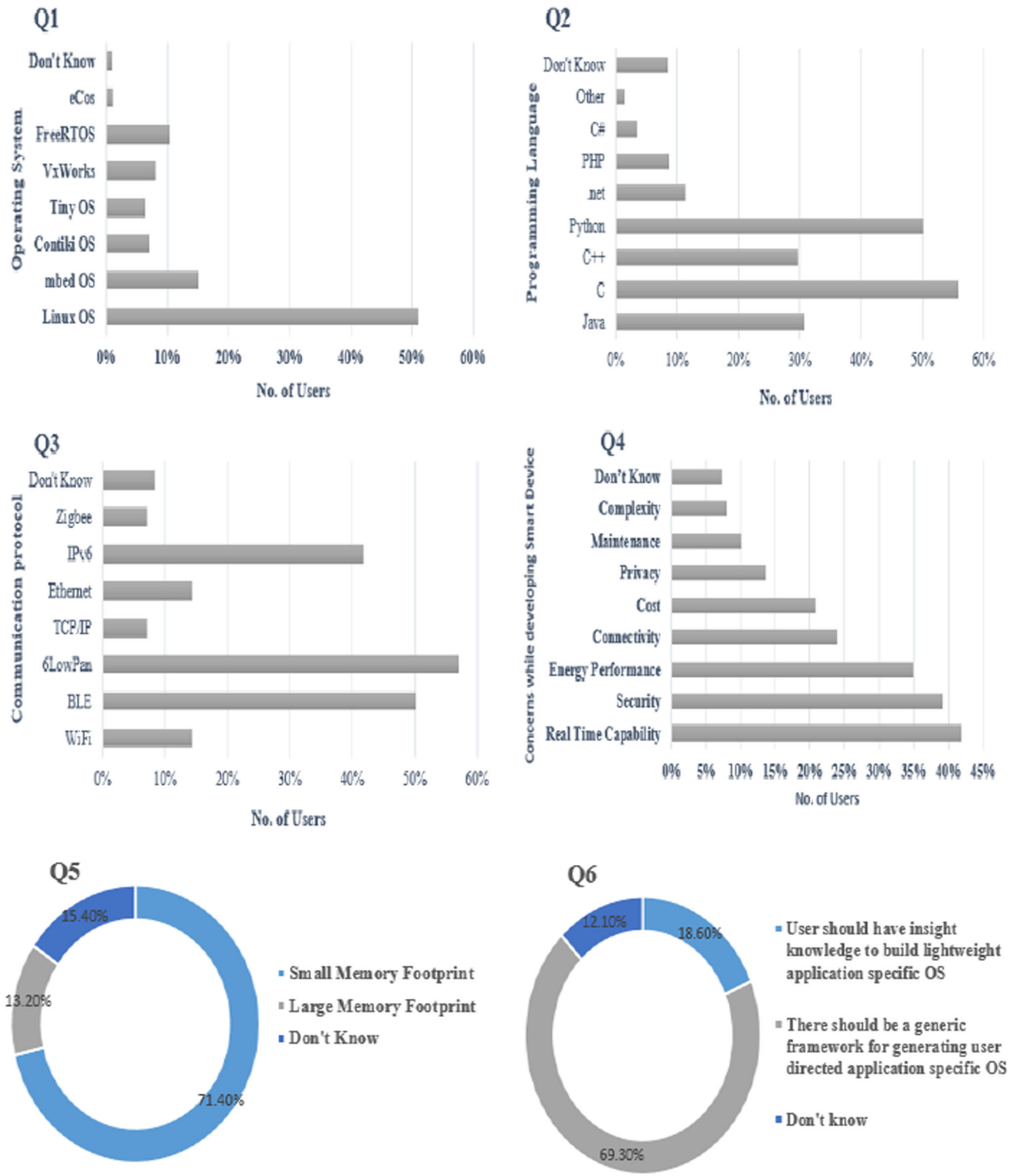

Fig. 5 Responses of participants gathered based upon survey questionnaire

(c) *Small Memory Footprint* Low-End smart devices are wonted to become smaller, cheaper and have limited resources. This device caters only few kilobytes of memory. To support these constrained devices, the device designer must be provided with set of optimized libraries i.e. the memory footprint of an operating system must be kept small.

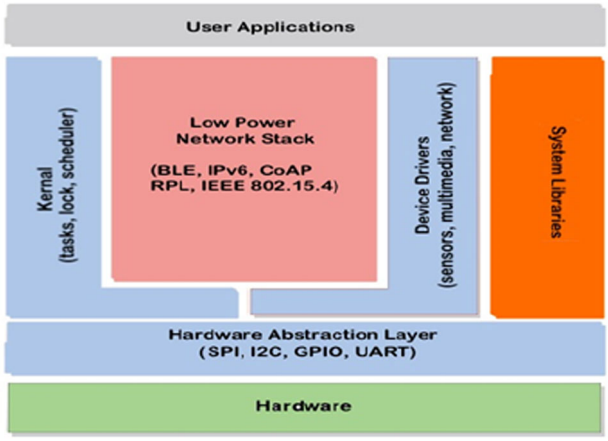

Fig. 6 Generic Architecture of an OS for low-end smart devices

(d) *Communication Protocols and communication Radio* Widely used communication protocol standards are vital for smart devices [11]. They should cater low power consumption and high level of security. The internet protocols residing in the network layer of OSI model such as IPv6 and 6LowPAN serve as useful for imminent smart devices [11, 37]. Operating system for low-end smart devices should support communication protocols based upon IEEE 802.15.4 radio technology. The Bluetooth Low Energy (BLE) is also targeted for low power networks.

e) *Real Time Efficiency* Accurate timing and convenient execution are critical in different smart devices such as in smart robots, smart health monitoring applications, smart vehicular networks etc. An operating system that processes the data as it comes, without any delay is known as real time operating system (Real time operating system, [38]. Hence, operating system for smart device should have real time efficiency.

f) *Energy Performance* As found large number of low-end smart devices are powered by batteries. For example smart industry/building automation are required to work for years within a single battery charge [39]. Therefore, energy performance becomes vital for these devices. Hence, an operating system designed for these low-end smart devices should cater low energy operations.

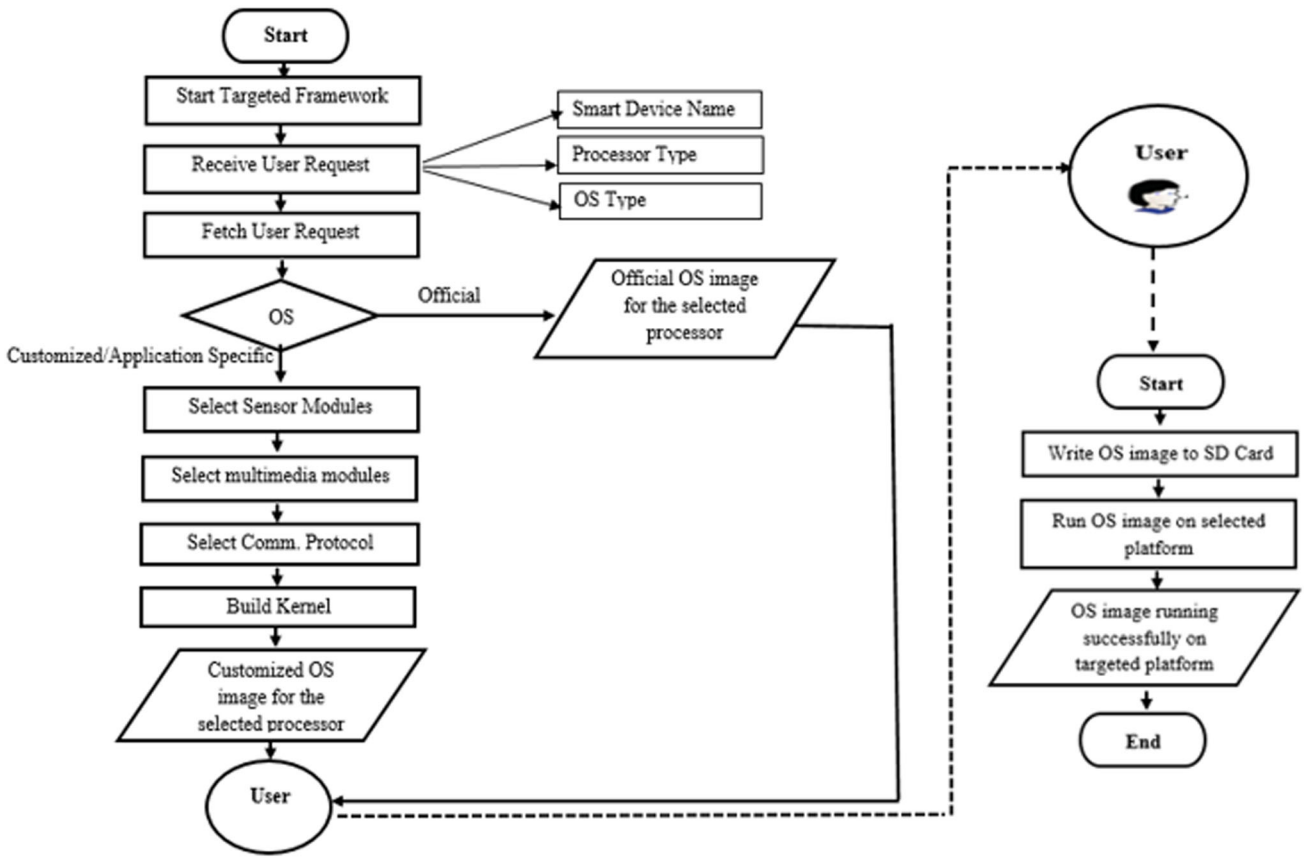

Fig. 7 Generic flow chart of the targeted framework

4 Proposed solution

How to reconfigure and customize general purpose operating system has attracted attention for application-specific low-end smart devices. Developers find it difficult to create application specific lightweight operating system for the particular use case due to various reasons that includes resource constraint or lack of knowledge.

A new generic framework may be proposed to encourage the user expectations and help developers to create lightweight application specific operating system for the particular use case. Targeted framework may incorporate an intelligent graphical user interface for the selection of different peripheral interfaces being computing platform, operating system, sensors and communication module for a particular application-specific smart device which may eliminate the use of official heavy weight operating system.

The basic requirements for smart device operating system are identified on the basis of literature survey and subjective experimentation as described in Sect. 3.3. Hence, the operating system generated from the targeted framework may comprises of all these essential basic traits in addition to the optimized application code, API & packages required for the selected sensors, communication modules and the device drivers. For an example, if user wants to build smart home automation system with ARM microcontroller and would like to integrate three sensors (light sensor, temperature sensor and ultrasonic sensor), one communication module (WiFi) and one multimedia module (Camera). User may find it difficult to write optimized application codes for different integrated peripherals and to create lightweight customize operating system for this particular use-case. In this case, user would like to have a generic framework which may automatically generate the lightweight customized operating system image for smart home automation project as per the requirements defined by the user. Thus, for building smart home automation project, targeted framework may generate lightweight customized operating system image for ARM computing platform that comprises of optimized codes and API packages for light sensor, temperature sensor, ultrasonic sensor, WiFi and camera module in addition to all the basic requirements as defined in Sect. 3.3.

Figure 7 shows the flow chart of targeted lightweight framework. In the first module, the framework will fetch the basic information from the user i.e. name of the smart device, processor type and operating system type. In the next module framework will request an input from the user with selection, to load an official image or customized lightweight image for the selected processor and operating system. In official image case, an official image for the selected processor and operating system will start downloading on the user's system. In customized image case, there will be user inputs for selecting various application specific peripherals for building smart device. Thereafter, a lightweight customized image will start downloading on the user's system which can be burn onto the SD card and can run successfully onto the selected processor.

5 Conclusion and future lines

In this paper, essential requirements for smart device operating system are formalized based upon the literature survey of existing operating systems and subjective experimentation. Based upon the user feedback analysis it has also been found that users would like to have a generic framework for automatically generating lightweight customize operating system as per the application specific requirements.

In today's era, smart device technology is increasing tremendously and the users are building smart devices in various domains at various places (schools, colleges, industries etc.). Hence, there is a need of such a generic framework which can provide ease to the users for selecting application specific peripherals and automatically generating lightweight customize operating system as per the application specific requirements. This framework should be an open source tool which can be easily accessed by all the users. It has been perceived that there is a need of such a generic framework which should be implemented in line with proposed solution.

Acknowledgements This survey and proposed solution framework is supported by Microsoft University Relations, Finland under the research grant of project Mobile Education Kit to Indira Gandhi Delhi Technical University for women, Delhi.

References

1. Davy A (2003) *Components of a smart device and smart device interactions* - M-Zones White Paper [online]. http://www.m-zones.org/deliverables/d234_1/papers/davy-components-of-a-smart-device.pdf. Accessed 20 Jan 2017
2. Lys I and Muller GG (2003) Smart light bulb. US Patent 6,528, 954, issued March 4, 2003
3. Popper S, Friedman R, et al. (2005) 'Smart device and system forimproved domestic use and saving of water. US Patent 6,895,985, issued May 24, 2005
4. Mohaideen AH (2009) System and method for providing health care services using smart health cards. US Patent 2,009, 025, 4363, issued Oct 8, 2009
5. Lawson DC, Reichard DJ et al. (2016) Smart device for industrial automation. US Patent 9,363, 336, issued June 7, 2016
6. Ehsani F, Witt Ehsani SM et al (2016) 'Smart home automation systems and methods', US Patent 9,230,560, issued Jan 5, 2016

7. Wipro and UBM Tech (2013) what smart systems can teach us—Wipro [online] http://www.wipro.com/documents/what-smart-systems-can-teach-us.pdf. Accessed 20 Jan 2017)
8. Rouse M (2006) Database-Techtarget [online] http://searchsqlserver.techtarget.com/definition/database. Accessed 20 Jan 2017
9. Beal V (2016) Memory–Webopedia [online] http://www.webopedia.com/TERM/M/memory.html. Accessed 20 Jan 2017
10. Bormann C, Ersue M, and Keranen A (2014) Terminology for constrained mode networks—Internet Eng. Task Force [online. https://tools.ietf.org/html/rfc7228. Accessed 26 Jan 2017
11. Hahm O, Baccelli E, Petersen H, Tsiftes N (2016) Operating systems for low-end devices in the internet of things—a survey. IEEE Internet Things J 3:720–734
12. Gaur P, Tahiliani MP (2015) Operating system for IoT Devices: a critical survey. In : IEEE Region 10 Symposium (TENSYMP), 2015, pp 33–36
13. Baccelli E, Hahm O et al (2013) RIOT OS: towards an OS for the Internet of Things. In: IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2013, pp 79–80
14. Chandra TB, Verma P, Dwivedi AK (2016) Operating systems for internet of things: a comparative study. In: ACM International Conference on information and communication technology for competitive Strategies (ICTCS'16), 2016, pp 1–6
15. Dunkels A, Gronvall B, Voigt T (2004) Contiki —a lightweight and flexible operating system for tiny networked sensors. In: IEEE International Conference on Local Computer Networks
16. Contiki (2017) [online] http://www.contiki-os.org/. Accessed 7 Feb 2017
17. Casado L and Tsigas P (2009) "ContikiSec: a secure network layer for wireless sensor networks under the contiki operating system. Springer Book—identity and privacy in the internet age, pp 133–147
18. 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, ambient intelligence book. Springer, Berlin
19. Montenegro G, Kushalnagar N, Hui J, Culler D (2007) Transmission of IPV6 packets over IEEE 802.15.4 Networks, RFC 4944. http://tools.ietf.org/html/rfc4944/. Accessed 10 Mar 2017
20. Madden SR, Franklin MJ, Hellerstein JM, Hong W (2005) TinyDB: an aquisitional query processing system for sensor networks. ACM Trans Database Syst 30:122–173
21. Karlof C, Sastry N, Wagner D (2004) TinySec: a link layer security for wireless sensor networks. In: Prooceedings of the 2nd ACM SenSys, Baltimore, MD, USA, 3–5 November 2004
22. Levis P, Lee N, Welsh M, Culler D (2003) TOSSIM: accurate and scalable simulation of Entire TinyOS Applications. In: Proceedings of the 1st ACM SenSys, Los Angeles, CA, USA, 5–7 November 2003
23. Bhatti S, Carlson J, Dai H, Deng J, Rose J, Sheth A, Shucker Gruenwarld BC, Torgerson A, Hen R (2005) MANTIS OS: an embedded multithreaded operating system for wireless micro sensor platforms, mobile networks and applications. Springer, Berlin, pp 563–579
24. Leopold M, Dydensborg MB, Bonnet P (2003) Bluetooth and sensor networks: a reality check. In: 1st ACM Conference on Sensor Systems (SenSys'03), Nov 2003
25. The Smart-Its Project (2017) [online]. http://www.smart-its.org/. Accessed 11 Mar 2017
26. Titzer BL, Lee DK, Palsberg J (2005) Avrora: Scalable Sensor Network Simulation with precise timing [online]. http://compilers.cs.ucla.edu/avrora/papers/avrora_ipsn2005.pdf Accessed 11 Mar 2017
27. Wind River System VxWorks (2005), [online] https://www.uio.no/studier/emner/matnat/fys/FYS4220/h11/undervisningsmateriale/laboppgaver-rt/VxWorks-6.2_Application_Programmers_Guide.pdf. Accessed 21 Mar 2017
28. QNX Neutrino RTOS (2014) [online]. http://support7.qnx.com/download/download/26183/QNX_Neutrino_RTOS_System_Architecture.pdf. Accessed 15 Mar 2017
29. Google (2014) Open Handset Alliance, Android Operating System" 2014 [online]. http://www.android.com/. Accessed 27 Feb 2017
30. Brillo (2016) [online]. http://events.linuxfoundation.org/sites/events/files/slides/Brillo%20and%20Weave%20-%20Introduction_v3_1.pdf. Accessed 15 Mar 2017
31. embOS Real time operating system (2015) [online] https://www.segger.com/admin/uploads/productDocs/UM01001_embOS_Generic.pdf. Accessed 21 Mar 2017
32. Barry R (2016) 'FreeRTOS, a free open source RTOS for small embedded real time systems', 2016 [online. http://www.freertos.org.Accessed 27 Feb 2017
33. Workshops on build your own smart device (2016) [Online]. http://mysmartphonekit.mobileeducationkit.net/index.php/blog/. Accessed 27 Feb 2017
34. Feedback form (2016) [Online. http://mysmartphonekit.mobileeducationkit.net/index.php/feedback-form/. Accessed 05 July 2017
35. Build Your Own Smart Device Workshop (2015) [Online]. http://mysmartphonekit.mobileeducationkit.net/index.php/nitttr/. Accessed 07 July 2017
36. Make My SmartPhone Workshop (2016) [Online]. http://mysmartphonekit.mobileeducationkit.net/index.php/msrit/. Accessed 07 July 2017
37. Mirani L (2014) Chip-makers are betting that Moore's Law won't matter in the internet of things—Quartz [online]. http://qz.com/218514/chip-makers-are-betting-that-moores-law-wont-matter-in-the-internet-of-things/. Accessed 26 Jan 2017
38. What is real time operating system (2013), [online] http://www.ni.com/white-paper/3938/en/.Accessed 25 Mar 2017
39. Min R, Bhardwaj M, Cho SH, Chandrakasan A (2002) Energy-centric enablingtechnologies for wireless sensor networks. IEEE Wireless Commun 9(4):28–39