Wireless Pers Commun
DOI 10.1007/s11277-012-0582-x

A Context-Aware User Interface for Wireless Personal-Area Network Assistive Environments

Bastien Paul · Séverin Marcombes · Alexandre David · Lotte N. S. Andreasen Struijk · Yannick Le Moullec

Abstract The daily life of people with severe motor system impairments is challenging and thus often subordinated to extensive external help; increasing their level of self-support is thus highly desirable. Recent advances in wireless communications, in particular in wireless personal-area networks, serve as technological enablers well suited for implementing smart and convenient assistive environments which can increase self-support. This paper presents the design and prototyping of a versatile interface for such wireless assistive environments. We propose a modular framework that can accommodate several wireless personal-area network standards. The interface is built upon this framework and is designed in such a way that it can be controlled by various types of input devices such as a touch screen or a tongue-control unit. The interface can automatically discover consumer appliances (e.g. Zigbee and Bluetooth enabled lights and computers) in the user's environment and display the services supported by these devices on a user-friendly graphical user interface. A demonstrator is prototyped and experimental results show that the proposed interface is context-aware, i.e. it successfully detects available appliances, adapts itself to the changes that occur in the user's environment, and automatically informs the user about these changes. The results also show that the proposed interface is versatile and easy to use, i.e. the user can easily control multiple devices by means of a browser menu. Hence, the proposed work illustrates how assistive technology based on wireless personal-area networks can contribute to improving the quality of life of motor system impaired persons.

B. Paul · S. Marcombes · A. David
Department of Computer Science, Aalborg University, Aalborg, Denmark

L. N. S. Andreasen Struijk
Department of Health Science and Technology, Aalborg University, Aalborg, Denmark

Y. Le Moullec (✉)
Department of Electronic Systems, Technology Platforms Section, Aalborg University,
Fr.Bajers Vej 7, A3-216, 9220, Aalborg Ø, Denmark
e-mail: ylm@es.aau.dk

Published online: 27 March 2012

Keywords Context-aware control interface · Wireless assistive environments · Service discovery · Motor system impairments · Tongue-controlled interface

1 Introduction

Spinal Cord Injuries (resulting in quadriplegia), brain injury and other sources of motor system impairments raise severe barriers for individuals who are subject to these. Very often, their daily lives are challenging and activities that we take for granted can be or can become out of scope for these individuals. These activities include, among others, attending school, having a job, and socializing. In such situations, life is subordinated to extensive or continuous external help; however, there are cases where technology makes it possible to increase the level of self-support of these individuals, and in turn can improve their Quality of Life (QOL).

Over the last two decades, the amount of work carried out in the multidisciplinary domains of Assistive Technology (AT) and e-health [1] has been increasing, encouraged by the evolution of available technologies such as sub-micron VLSI digital circuits, digital signal processing platforms, and, more recently, advances in wireless communications, in particular in terms of Wireless Personal-Area Networks (WPAN) based on e.g. Zigbee and Bluetooth. The combination of these technologies pave the way for advanced systems which can improve the QOL of people with motor system impairments by enabling them to control their environments (e.g. in smart homes) and to access modern communication channels (e.g. email, web, audio/video calls). This is a domain that has witnessed many research and development efforts [2]. Indeed, even when living with severe impairments such as quadriplegia, people are often still able to move body parts such as jaws, eyes, head, and tongue. Research efforts have resulted in several types of adapted control systems that enable disabled people to control their environment, e.g. their wheelchair, lights, TVs, computers. Although many promising concepts have been demonstrated, only a limited fraction of these environmental control systems have become widely accepted by the users; price, ease of control, aesthetics, and social cost are essential issues that can determine the adoption and success of such systems [3,4].

Home medical equipment is expensive and not always affordable without health insurance compensation. Any added feature that increases the overall price too much has reduced chances to become widespread. Typically, AT relies on three major parts [2]. The first one, the "access pathway" is composed of the physical sensors or input devices actuated by the user; their outputs are then converted into electrical signals which are further analyzed by means of digital signal processing techniques. In the context of severe motor system impairment, access pathway examples include control systems based on the eye, head, brain, or tongue actuation. The second part is the actual "user interface" which analyzes the digitally processed input signals and converts them into output signals. These output signals are then used to enable the third part, i.e. "functional activities" such as controlling appliances in the user's environment.

Ideally, the user interface should be versatile so that it enables the user to interact with as many types of devices located in his/her environment as possible, as transparently as possible. Moreover, aesthetics is too often underestimated; in many cases it is highly desirable to minimize the feeling of being "different" so that the control system can be accepted and adopted by the users [5]. Two of the above mentioned issues, namely aesthetics and access pathway, have been investigated at Aalborg University and have resulted in a fully integrated wireless inductive tongue control system [6,7]. The work presented in the current paper deals

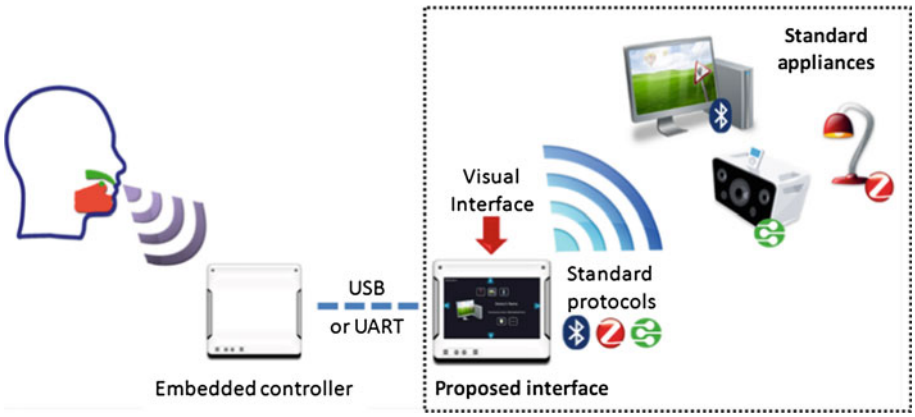

Fig. 1 Illustration of the proposed interface for controlling appliances in a wireless assistive environment. The work presented in this paper is delimited by the *dashed box*

with the design and prototyping of a user interface, i.e. a versatile interface which could be used together with (but is not limited to) the above mentioned tongue control system. The proposed interface exploits the fact that wireless features are increasingly found in daily appliances and that inexpensive and versatile embedded systems (e.g. microcontroller-based Linux platforms) are more and more widespread. This interface automatically discovers wireless-enabled appliances (e.g. Zigbee-enabled lights, Bluetooth-enabled computers), displays their services on the visual interface and lets the user control them by means of an access pathways (input) device, which could be, for example, a touch screen (part of the proposed interface) or a tongue control system (e.g. [6,7]). The AT scenario considered for this work is illustrated in Fig. 1, where the work presented in the current paper is delimited by the dashed box.

The remainder of this paper is organized as follows. Section 2 briefly reviews related works in terms of (i) assistive technologies for motor system impaired persons and (ii) discovery, multi-protocol compatibility and context awareness. It also summarizes our contributions. Section 3 presents the design of the proposed user interface and the underlying discovery and connectivity mechanisms. Section 4 presents the prototyping of the proposed interface and Sect. 5 presents the experimental setup and results. Finally, Sect. 6 discusses the results and suggests directions for future work.

2 Related Work and Contributions

2.1 Related Work

Earlier works in terms of assistive technologies for motor system impaired persons include, among others, [8] that proposes a palatal tongue controller consisting of touch-sensitive pads mounted on the surface of the dental plate. A transmitter embedded in the dental plate transmits switch activations remotely to external devices fitted with receivers. Computers in the user's environment can be controlled via a mouse emulator whereas other appliances are controlled via infrared signals; however, its users may have to face challenges due to the limitations of that specific technology (e.g. line-of-sight and code learning). [9] describes

a Hall effect-based approach for controlling appliances through tongue movement. Besides being rather complex and physically large, this system requires cables going from the mouthpiece unit to the processing unit. Moreover, no description of the interfacing/control methods to the devices in the user's environment is provided. More recently, a ZigBee-based wireless intra-oral control system for quadriplegic patients has been proposed in [10]. Their system is composed of a wireless intra-oral module, a wireless coordinator, and distributed ZigBee wireless controllers. The intra-oral module communicates wirelessly with the coordinator which itself communicates wirelessly with the Zigbee controllers to activate external devices, depending on the requests made by the user via a GUI. Although their concept and ours share a few similarities, they differ significantly on the interfacing aspects: their interface for controlling external devices is quite un-versatile since it relies on a static GUI that runs on either a PC or a pocket PC and it can only communicate with Zigbee-enabled appliances. [11] reports on an external tongue controlled device that communicates with a PC through a proprietary 2.4- GHz wireless link. An application executing on the PC enables the user to control a powered wheelchair. That paper also suggests the opportunity to control other types of appliances by means of a Wi-Fi/Bluetooth enabled PDA, but this is not implemented nor discussed. Finally, [12] explores an EOG-based eye tracking technique combined with infrared and Bluetooth connectivity for controlling appliances in the user's environment. Although sufficiently compact for being mounted on a wheelchair, their interface is neither designed for tongue control nor versatile enough for accommodating extra wireless protocols.

Clearly, the above works have paved the way for improved self-support in terms of environment control; however, there is still room for advancing the opportunities made possible by technological advances in wireless communications and context-aware computing. For our application, the ideal interface should not only be wireless, but should also feature mechanisms such as service discovery, multi-protocol compatibility and context awareness. Works related to these features are found in e.g. [13], where service discovery for mobile network is surveyed, while distance-sensitive service discovery is addressed in [14]; service reconfiguration for ensuring service availability in assistive environments is considered in e.g. [15]. Multi-protocol compatibility is investigated in works such as [16–22]. For example, [16] proposes a layered middleware that enables multiple protocol to coexist and [17] considers Wi-Fi/Zigbee coexistence. [18] and [19] present translation strategies for KNX-Zigbee and infrared-Zigbee, whereas [20] proposes a more universal approach by means of a dynamic device integration manager that enables transparent service discovery. [21] describes a so-called adaptive-scenario-based reasoning system where adaptive history scenarios are used to collect and aggregate user habits in smart homes; similarly, [22] suggests a dynamic service composition system for coordinating Universal Plug and Play (UPnP) services in smart homes and identifying devices that can work together, taking user habits into account.

Finally, [23] suggests guidelines and recommendations for constructing robust context awareness applications; specific context-aware wireless network systems are discussed in e.g. [24] that describes CANE, a Context-Aware Network Equipment for multimedia applications that adapts dynamically to the user's environment by taking user's preferences, network status, as well as service requirements and policies into account.

2.2 Contributions

To overcome the limitations of [8–12] we exploit concepts similar to that of [13–24]. We present the design of a context-aware and versatile framework upon which the user interface is built. The paper details the mechanisms used so that the framework can (i) discover and

connect seamlessly to WPAN enabled appliances in the user's environment and (ii) give the user access to the control and communication services of these appliances. The novelty of the work is twofold. Firstly, in contrast to existing assistive user interfaces, the one that we propose is not limited to a single communication standard or a predefined, fixed set of standards. To do so, we propose a discovery and multiprotocol connectivity mechanism combined with a plug-in system that makes the framework versatile enough to accommodate a wide range of existing and future WPAN standards. Secondly, the features needed for such a versatile interface are implemented as an embedded system with hard constraints in terms of size, power, and price so that it could be mounted on e.g. a wheelchair. As opposed to desktop implementations or less constrained embedded implementations, the proposed user interface can be implemented by means of relatively inexpensive and physically small components with low computational capabilities. Furthermore, a specific power management system (PMS) is designed to reduce the power footprint of the user interface. A demonstrator is prototyped and used to experimentally test the proposed approach. The experiments show that the proposed interface is context-aware, i.e. it successfully detects available appliances, adapts itself to changes that occur in the user's environment, and automatically informs the user about these changes. Furthermore, the proposed interface is versatile and easy to use, i.e. the user can easily control multiple devices by means of a browser menu.

3 Design

3.1 Design Considerations

The purpose of the proposed interface is to provide the necessary features for (i) enabling the connection of various access pathway (input) types, such as e.g. a touch screen or a tongue control unit, with multiple WPAN-enabled appliances in the user's environment and (ii) enabling the user to interact with those appliances. To implement the above, the following design aspects have been considered:

- The interface must be versatile enough to accommodate a significant set of appliances; as the number of wireless standards and home automation products is expanding, the interface must be flexible and upgradable.
- Using appliances through the interface must be as easy as it would with their native interfaces. In particular, the user should not feel the need for having someone to help him or her when interacting with the appliances. Similarly, the interface must provide an "easy first-time installation" procedure.
- The interface must provide the user with the possibility to switch easily and quickly between the appliances, without disrupting their operation. Support for some form of multitasking is thus needed.
- Moreover, the interface must be aware of its environment (i.e. able to detect available appliances) and inform the user about it (i.e. maintain a list of available appliances and their respective services and present them visually to the user).
- Finally, as the interface is expected to be on mounted on e.g. wheelchairs, the size and power factors are critical and should be kept as low as possible.

Regarding the networking aspect, this work combines various standard protocol stack layers; this makes that interoperability with multiple protocols stacks only requires that WPAN enablers, such as Bluetooth or Zigbee, are already available or installed on the appliance. Protocols using Service Oriented Architectures (SOA) are used here since they enable ser-

Table 1 Examples of stack combinations for the proposed interface

#	Application stack	Network stack	Physical stack
1	Bluetooth profiles + SDP	Bluetooth	802.15.1 (BLUETOOTH)
2	Zigbee profiles + SDP	Zigbee	802.15.4
3	DLNA + UPnP	TCP/IPv4	802.11.X (Wi-Fi)
4	DLNA + UPnP	TCP/IPv6	802.11.X (Wi-Fi)
5	DLNA + UPnP	6LoWPAN (TCP/IPv6)	802.15.4
6	DLNA + UPnP	TCP/IP + Bluetooth	802.15.1 (Bluetooth)

vice discovery and service access. Examples of possible combinations of standard protocols providing both a physical layer and an application layer, as targeted in this work, are listed in Table 1. Although many SOA-based protocols implementing an application layer would be suitable for this work, the DLNA (Digital Living Network Alliance) specification is currently the only one broadly implemented in consumer appliances. Besides supporting UPnP, DLNA also enables the targeted users of the proposed interface to enjoy various audio and video media. The six combinations shown in Table 1 are the ones that have been considered. However, as the proposed interface is modular, this list can easily be extended.

To make it versatile and expendable with future standards, the framework has been designed following a modular approach. Its constituting elements fall into three main categories: devices, protocols, and functionalities. Each of these categories includes classes, plug-ins and extensions, as described in the following sections.

3.2 Listing Controllable Appliances

Remote appliances are represented by means of the 'Device' class; each device has a name, a location, a unique ID, an icon, as well as a type (light, computer…); thus remote appliances are represented by means of objects derived from both the 'Device' and 'Type' classes.

3.3 Listing Accessible Functionalities

'Service' is the most generic class that can define a functionality, depending on the protocol it belongs to and its family. All devices are aggregated, through the 'Element' class, in the 'ConcurrentList' class that is used to add, access, or remove a device. Objects derived from the 'Service' class represent supported functionalities. Each 'Device' contains a list of the 'Services' (i.e. functionalities) it offers. Functionalities are referred to as 'services'. Each functionality is therefore represented by an object derived from the 'Service' class. A summary of the connections between these classes is shown in Fig. 2.

3.4 Awareness of and Adaptation to the User's Environment

When the user moves from place to place or when appliances are displaced, the appliances seen by the framework enter or leave its coverage range. For usability sake, device discovering must take place without any command from the user. This is supported by the 'Protocol' class. The 'Protocol' class defines all protocol-related attributes and methods the framework can use. Adding new protocols is rather easy since it only requires a new class derived from 'Protocol' to describe the given protocol; 'Protocol' plug-ins also have to define extensions to the 'Device' class. Another aspect is that communication activities should not prevent the

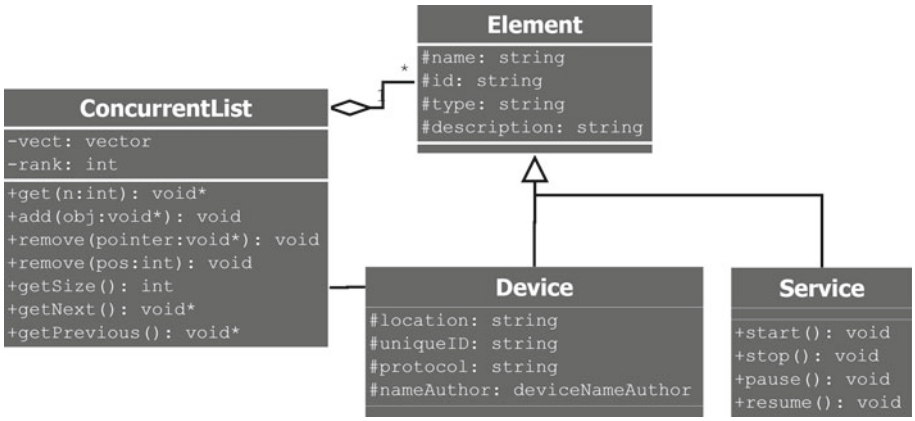

Fig. 2 The 'ConcurrentList' class aggregates the 'Device' and 'Service' classes through the 'Element' class

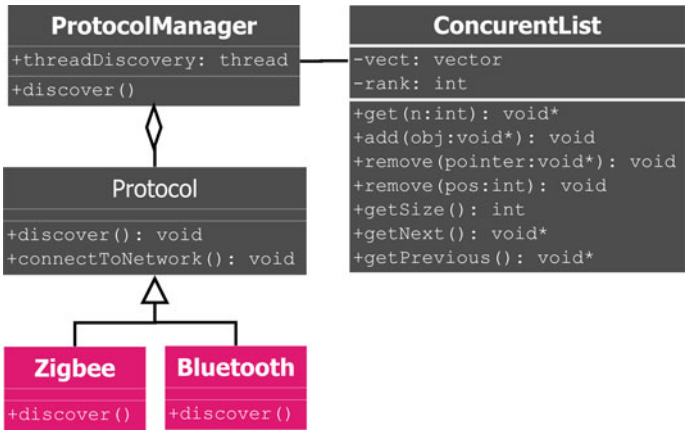

Fig. 3 'ProtocolManager' connects the 'Protocol' and 'ConcurrentList' classes by aggregating the protocols (e.g. Zigbee and Bluetooth)

system from being responsive to the user's actions. This is supported by means of threads: the communication activities are performed by one or several threads while the GUI is performed by another one, thus the framework is reactive and can continuously adapt to the user's environment. Adaptation is achieved by means of the discovery features of the considered protocols. The 'Protocol' class is responsible for discovering devices and for adding or removing them from the list. Moreover, the GUI has to have access to the list of 'Element' just like 'Protocol' that is implemented in another thread. To avoid any conflict, 'ConcurrentList' provides a mutex mechanism to protect the access to the list of devices. These elements are connected through the 'ProtocolManager' class, as shown in Fig. 3.

'ProtocolManager' aggregates all the protocols and 'ConcurrentList' is implemented in a thread where all the protocols are to be executed. 'ProtocolManager' manages a thread dedicated to device and service discovery. To represent a remote appliance, the 'Protocol' plug-in instantiates a subclass of the 'Device' class adapted to its needs in terms of e.g. data storage. These instances are then attached to the object that represents the appliance.

3.5 Switching Between Devices

Switching rapidly from one device to another is made possible by not terminating open connections when performing the switch. Instead, these connections are kept alive for a certain duration, or set in an idle mode that is less energy consuming, depending on the needs of the services and on the properties of the protocols. When a protocol does not support multiple connections, switching between devices may imply terminating an active connection: in this case, the termination is done in such a way that the current state of the appliance is not modified.

3.6 Modularity

Modularity is achieved by means of a plug-in mechanism which supports new technologies (protocol-dependent plug-ins) and new functionalities (functionality plug-ins). A plug-in is a library used by the framework and contains the definition of protocol-dependent and type-dependent devices as well as the definition of specific protocols. Figure 4 shows how the plug-in mechanism relates to the framework (in grey) and the dependencies between the plug-ins. 'Devices' (red) represent appliances such as computers and lights. 'Services' (orange) define the supported services and their associated GUI items. 'Protocols' (pink) bring connectivity support in conjunction with 'Device Definition' (purple) and 'Service Definition' (blue). Plug-ins are composed of three main classes and, optionally, a set of other classes providing the plug-in implementation: the 'Implementation' class that depends on the plug-in type, the 'Plug-in' class which creates an instantiation of the implementation class and contains the dependency information between plug-ins, and the 'Loader' class which registers the plug-in class in the framework. Finally, a plug-in manager loads all plug-ins.

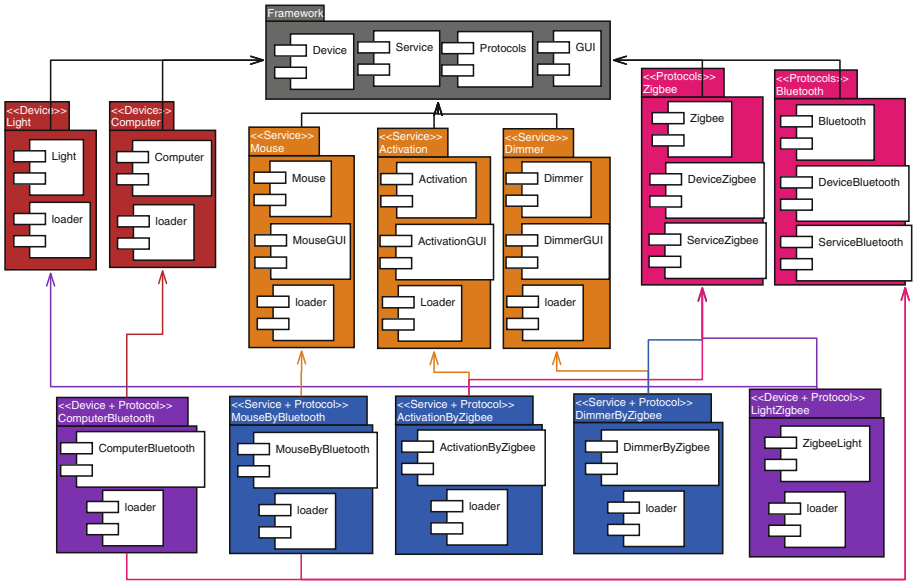

Fig. 4 *Plug-ins dependencies* Framework (*grey*). Devices (*red*). Services (*orange*). Protocols (*pink*). Device Definition (*purple*), and Service Definition (*blue*)

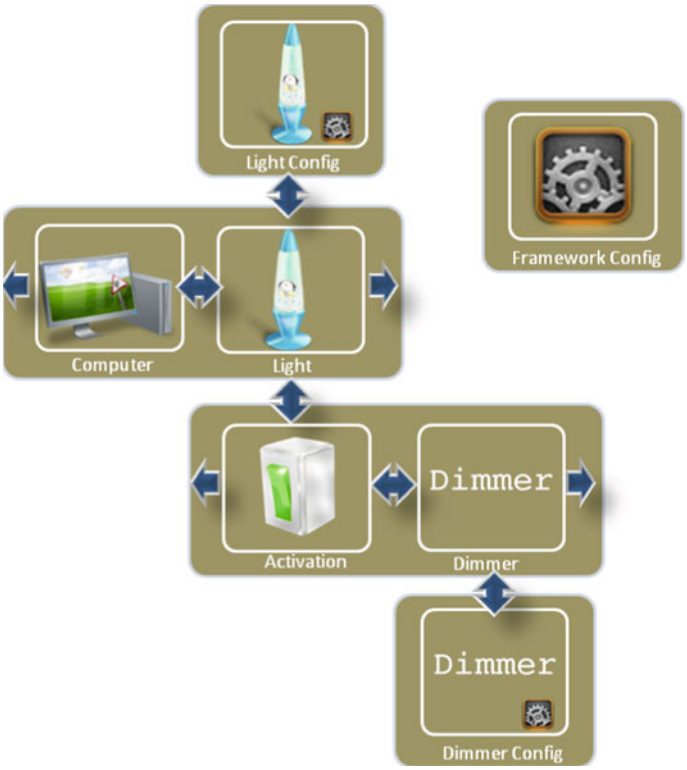

Fig. 5 Illustration of the GUI architecture. It is possible to select the appliance to be controlled (e.g. a light), to configure and activate it, and to access its features (e.g. turn on/off) by means of the *left, right, up, down, click* and *escape* commands available on e.g. the tongue-control unit and the touch-screen

When loaded, plug-ins register themselves automatically in the plug-in manager; then the manager checks the dependencies. The manager is also responsible for unloading plug-ins.

In the current AT scenario, it is expected that several types of access pathways (input) devices could be used for navigating through the GUI, for example a touch screen or the tongue-control system described in [1]. In order to avoid user's fatigue, the GUI has been designed so that the number of required movements and clicks is minimized. Figure 5 illustrates how it is possible to select the appliance to be controlled (e.g. a light), to configure and activate it, and to access its features (e.g. turn on/off) by means of the left, right, up, down, click, and the escape commands. The GUI is integrated with the framework as shown in Fig. 6. When the user switches to a device, a service or a configuration panel, the displayed object (e.g. device, service or configuration) is associated to the corresponding class. This class then reads the required information and displays them on the screen.

3.7 Managing the Power

A PMS has been designed and implemented. The role of the PMS is to minimize power by combining several techniques, both at the physical (HW) and framework (SW) levels. The following techniques are applied at the hardware level: (i) dynamically scaling the frequency and voltage of the CPU, (ii) turning off the touch panel when unused, (iii) diming or turning off the backlight when unused, (iv) using the RAM low power mode, (v) exploiting the low

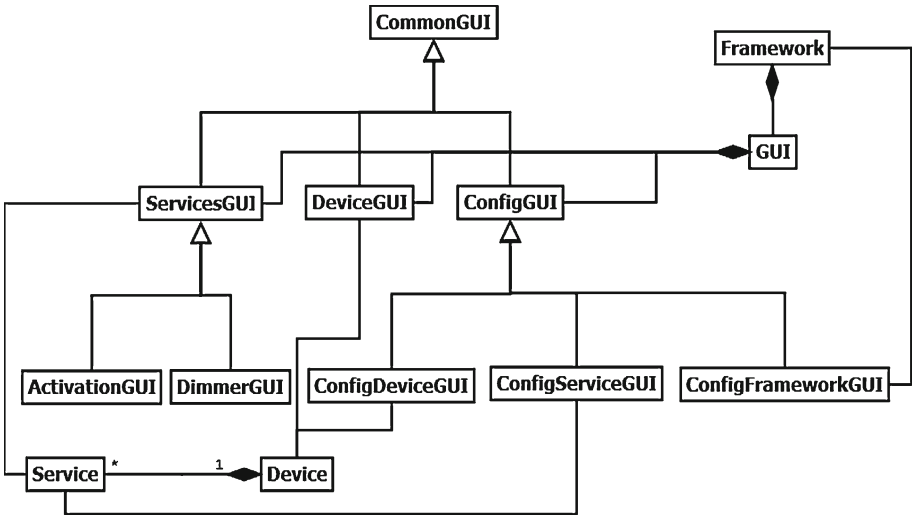

Fig. 6 *Integration of GUI with the framework* 'Service' is associated with 'ServicesGUI' and 'ConfigServiceGUI', 'Device' with 'DeviceGUI' and 'ConfigDeviceGui', and 'Framework' with 'ConfigFrameworkGUI'

power facilities provided by communication protocols, and (vi) turning off the various chips when unused. At the software level, flags and timers are used to determine how and when to apply the above techniques. This removes the need for monitoring the system usage and eases the detection of service requests. As seen in Fig. 7, the core of the PMS (i.e. for the CPU/memory) consists of four modes ('Run', 'Conservative', 'Sleep', 'Shutdown'). When the period of time a user is inactive reaches a first threshold value, the system switches from 'Run' to 'Conservative' where the voltage and frequency of the CPU are decreased. If the inactivity period exceeds a second, larger threshold value, the system switches to 'Sleep' where the memory is set to self-refresh and the wireless chips in stand-alone modes. Any activity while in the 'Conservative' or 'Sleep' modes makes the system to switch back to 'Run'. Finally, all the components are turned off when the system enters 'Shut down'. A flag is used so that when a service needs the CPU computational power, the system does not switch to the 'Conservative' (and thus 'Sleep') modes. Similarly (not shown in Fig. 7), the backlight is dimmed or turned off when inactivity thresholds are reached. A flag is used so that when a service needs to display visual feedback, the system does not switch to the dimmed and off modes.

Regarding the wireless modules, several strategies are used. These include limiting discoverability and connectivity, performing radio collision avoidance, communicating in low power modes whenever possible, switching to stand-by modes based on usage probabilities, and modulating the period and type of discovery according to the probability of finding devices around the user.

4 Prototyping

4.1 Prototyping Platform

The demonstrator has been prototyped on a platform composed of the following elements. The core of the platform is Atmel's AT91SAM9263-EK Evaluation Kit [25] featuring among

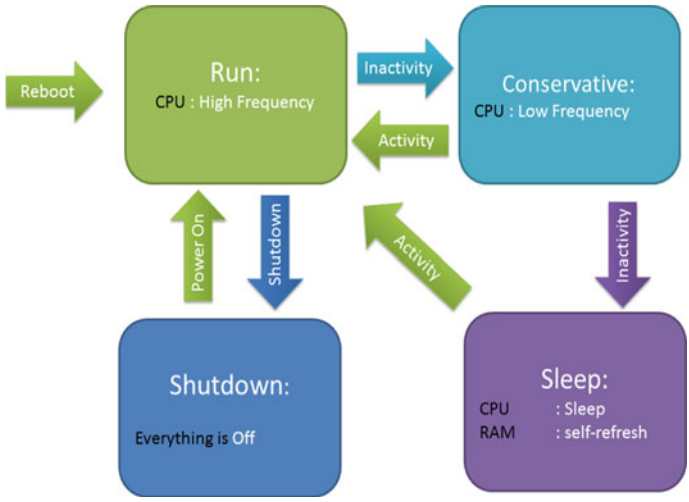

Fig. 7 The core of the power management system consists of four modes. As user inactivity reaches thresholds, the system is switched to the conservative and sleep modes; user activity makes the system return to the Run mode

others, an AT91SAM9263 micro-controller and a 3.5" 1/4 VGA TFT LCD module with touch screen and backlight. For demonstration purposes, two WPAN standards (Zigbee and Bluetooth) have been implemented on the interface using a Texas Instrument CC2530 Development Kit [26] and a generic Bluetooth dongle, respectively. The CC2530 Development Kit contains, among others, two CC2530EM evaluation modules, two SmartRF05EB Evaluation Boards (on which the CC2530EM evaluation modules are plugged), and one CC2531 Zigbee USB dongle. The CC2530EM evaluation modules and the CC2531 USB dongle are essentially composed of a 2.4- GHz IEEE 802.15.4/Zigbee RF transceiver, an 8051 microcontroller core, and 256 KB Flash/8 KB SRAM. The selected operating system executing on the AT91SAM9263 micro-controller is Ångström Linux [27]. Finally, the GUI is implemented by means of Qt for Embedded Linux [28], a compact, memory efficient windowing system for Linux.

4.2 Prototyping of the Framework

In order to keep the implementation modular, the framework has been implemented by means of three packages, namely 'Core', 'GUI' and 'Communication'. 'Core' is composed of three of the classes introduced in Sect. 3: 'ConcurrentList', 'Device' and 'Service'. 'Device' and 'Service' are stored in Vector containers supported by the Standard Template Library (STL) library. 'ConcurrentList' handles these containers by means of concurrency protection via mutex synchronization. 'GUI' implements the classes related to the graphical user interface. It makes use of the signal-and-slot mechanism supported by Qt: a signal is emitted when an action occurs (e.g. user click) and a slot (a method) is executed in response to the emitted signal. 'Communication' relates to the 'Protocol' class and its derived classes ('Zigbee' and 'Bluetooth' in the demonstrator). In the demonstrator, threads are implemented by means of the Pthread Library. Qt events are used as a messaging system between the threads, and mutexes are used to protect data and as a synchronization mechanism. Moreover, a notification

system exploits Qt's event system and the Pthread library so that when devices are discovered, the 'Protocol' classes can notify the GUI. These are added to 'ConcurrentList'. Subsequently a notification is sent to 'GUI' by 'ConcurrentList'. Then, the protocol lists the devices functionalities: it creates a service for each functionality and adds them to the corresponding devices. Finally, 'ConcurrentList' sends notification to 'GUI' so that the user can see on the screen which functionalities are available.

4.3 Zigbee Module

The Zigbee module implements a generic mechanism for Zigbee communication. A plug-in enables this module to support the Zigbee Home Automation Profile. The lower and upper layers of the Zigbee module are shown in Table 2. The hardware consists of the CC2531 Zigbee USB dongle connected to the AT91SAM9263-EK board. The firmware executing on the CC2531 is composed of Z-stack, a Zigbee stack provided by Texas Instruments, and a custom application developed specifically for the implementation of the demonstrator. Z-stack, based on a simple event-based OS, provides an OS Abstraction Layer (OSAL) Application Programming Interface (API) for interfacing with custom applications. Moreover, Z-stack provides a hardware abstraction layer API, so that multiple hardware components can be exploited generically.

The CC2531's Z-stack firmware searches for a Zigbee network to join, both upon power up and periodically once executing. Profile advertising is carried out upon joining a network: the CC2531 device describes itself to let the other Zigbee devices know which profiles it supports and the functions it provides. The first operation performed by the custom firmware application is to provide this information to Z-stack in order to advertise that the device is compatible with the Home Automation Profile and can act, in the demonstrator, as a remote light switch.

Service discovery is taken care of by the custom firmware application that requests Z-stack to search for a device providing a Zigbee cluster. In return, Z-stack sends a Zigbee "Descriptor Matching Request" to the network. This request is sent to a router or to the coordinator of the network, and is propagated to all routers of the network. If a device with a matching cluster is found on the network, the router or the coordinator to which the device is connected sends a response packet back. Upon reception of this packet, Z-stack calls the custom firmware

Table 2 The lower and upper layers of the Bluetooth and Zigbee modules implemented on the interface

	OSI layers	Bluetooth module	Zigbee module
Upper layers	Application	Blue-Z stack and custom application	Zstack stack, custom application, and OSAL API
	Presentation		
	Session		
	Transport		
Lower layers	Network	HCI	OSAL
	Data link (MAC)	802.15.1 Generic Bluetooth USB dongle and its firmware	802.15.4 CC2531 Zigbee USB dongle and its firmware
	Physical		

application by sending an OSAL event. The addresses of the Zigbee devices resulting from the search are then sent one by one to the interface framework software. Afterwards, the latter can request the custom firmware application to bind itself to the devices that have been discovered.

Connection management is taken care of by the custom firmware application that handles connections to other devices by sending, upon request, the list of available Zigbee networks and the channels that they use to the Zigbee communication module of the interface framework software. The interface framework software can also request the custom firmware application to connect to a certain network and, upon success, to search for compatible devices. It can also request the custom firmware application to disconnect from a certain network at any time.

4.4 Bluetooth Module

The Bluetooth module supports the Bluetooth Human Interface Device (HID) Profile, enabling the proposed interface to emulate the mouse and keyboard of a Bluetooth-enabled computer. It also provides most of the generic features of the communication module, such as service discovery, multi-tasking and connection management. The lower and upper layers of the Bluetooth module are shown in Table 2. The hardware used in the demonstrator consists of a low-cost Bluetooth USB dongle. The implementation of the Bluetooth module is composed of several software components. The firmware of the Bluetooth USB Dongle manages the low-level part of the Bluetooth stack, while the framework software takes care of the high-level part. The two parts of the stack interact with each other by means of the Host Controller Interface (HCI) and the appropriate Linux drivers for Bluetooth USB dongles. The higher level of the Bluetooth stack is itself composed of several blocks.

The generic features of the Bluetooth stack are implemented with Blue-Z, the Linux De facto Bluetooth stack. The specific features that interact with Blue-Z to provide an application layer usable by the framework software are custom implemented. Our implementation makes use of the Blue-Z drivers for Bluetooth USB dongles, Blue-Z mechanisms for managing connections with other devices and the configuration of the local Bluetooth device, and the Blue-Z Service Discovery Protocol (SDP) server. These elements are accessed and used through the Blue-Z C/C++ library. The core components of the Bluetooth communication module are responsible for managing Bluetooth communications transparently for the interface framework by means of derivations of the ‘Protocol’, ‘Device’ and ‘Service’ classes.

Service advertisement is managed by the ‘SDPManager’ class together with the Blue-Z SDP server that listens continuously for service discovery requests from the other Bluetooth devices. The list of services itself is modified upon loading and unloading the plug-ins of the Bluetooth communication module. Service and device discovery are performed periodically by means of a threaded implementation of the ‘BluetoothServiceDiscoverer’ class. Discoveries are only performed if at least one service plug-in requested the class to be called upon discovery of a certain type of device or service, by registering a filter. Moreover, ‘BluetoothServiceDiscoverer’ can be configured to warn a service upon the availability of a certain device by means of the paging mechanisms.

The Bluetooth communication module manages its connections with the remote Bluetooth devices through the ‘BluetoothConnectionManager’ class. This class is used by the Bluetooth services to listen for incoming connections from specific devices on specific ports, and to establish connections with remote devices. It also keeps track of the active connections, in a centralized way, which enables the coordination of the service plug-ins.

4.5 Power Management

The 'PMS' class implements the various techniques and strategies presented in Sect. 3. The timers are based on an "alarm" mechanism by which a signal is sent to the relevant process on time-outs. The hardware modules are controlled by accessing their Linux drivers. This is achieved through the virtual file system interface maintained by the kernel. For example, the "/sys/class/backlight/atmel/" directory contains files for configuring the brightness of the backlight and the "/sys/devices/system/cpu/cpu0/cpufreq/" files for scaling the frequency and voltage of the CPU.

4.6 Reactivity Considerations

This subsection discusses possible sources of delays in the proposed framework and their impact on the reactivity of the interface. The selected operating system (Ångström Linux) as well as the selected GUI (Qt for Embedded Linux) are both lightweight and have been configured so as to minimize their respective overheads in terms of CPU usage and memory footprint. Doing so ensures that sufficient resources are available for the services that take care of the communication protocols, device management, and user input management. Regarding the interface reactivity, we consider the following two cases.

The first case is when a device (appliance) appears or reappears in the user's environment. The delay (latency) for a device to be discovered and to become available for the user depends on several factors. The first one, enumeration, is protocol dependent. For example, the theoretical enumeration latency for Bluetooth devices is 10.24 s and that of Zigbee is about 30 ms. Note that the actual time for a device to be discovered also depends on its distance to the interface, the physical obstacles that might attenuate the radio signal between the device and the interface, as well as the number of devices to be discovered 'simultaneously'. Furthermore, the discovery times are typically shorter (e.g. ca. 3 s instead of ca.10.24 s for the Bluetooth case) when a device has already been discovered in the past and its parameters have been stored in the interface.

Furthermore, as indicated in Sect. 3.7, the energy saving strategy that we have implemented makes use of the energy saving facilities supported by the protocols. These have an impact on the discovery and availability delays. In the current implementation it is possible to set the radio modules of the interface into sleep or idle modes, and to tune the discovery period.

When setting a radio module into sleep mode, one has to consider the wakeup times (ca. 3 s for Bluetooth and ca. 15 ms for Zigbee). A radio module is set to sleep only when it is not used in any active connection. Alternatively, a radio module can be set to idle (i.e. a low power mode) since the wake-up time when exiting the idle mode is lower as compared to when exiting the sleep mode. Regarding the discovery period, for the targeted users the environment is not expected to change very rapidly, so in order to save energy it is possible not to search for new devices continuously. In the current implementation the discovery period can be either automatically tuned according to several factors such as the number of recently discovered devices and how often they are used, or it can be manually set by the user.

The second case is when switching from one device to another one. In this case, the main source of possible delay is related to radio collision and the protocol/plugin manager. Radio collision can happen when several radio modules are operating next to each other, using the same frequencies. However, most protocols provide mechanisms for specifying which frequencies to use or to avoid. In the current implementation, the protocol plugin takes advantage of these mechanisms and provides a negotiation mechanism that reduces radio collision.

The negotiation phase introduces some delay, mostly depending on how many devices the user wants to switch among; however, negotiation should not take place very often as the number of devices in the user's environment does not change very rapidly for the application scenario. Also note that since the various functionalities of the interface are implemented as lightweight threads, switching between devices is easily handled by the selected microprocessor.

The various delays introduced by the operating system/GUI, the communication protocols, and the energy saving strategy are relatively small, and given the application scenario, we consider that these various delays are either negligible or not penalizing seen from the targeted user's point of view. This is confirmed by our observations reported in Sect. 5.

5 Experimental Results

5.1 Experimental Setup

The experimental setup is composed of the interface (i.e. the platform described in Sect. 4) and of several devices placed in the environment of the user. These devices are: (1) the two CC2530EM Zigbee evaluation modules fitted on the SmartRF05EB boards; their respective LEDs are used to emulate Zigbee-controlled lights; (2) a lamp connected to a Cleode Zplug (a ZigBee power plug), (3) a Linux-based laptop (Blue-Z Bluetooth stack); and (4) two Windows-based laptops with Widcomm and Microsoft Bluetooth stacks, respectively. A photograph of the setup (except the laptop running the Microsoft Bluetooth stack) can be seen in Fig. 8.

Please note that the following experiments have been conducted by abled bodied subjects, using the touch screen as the access pathway. Clinical trials with motor system impaired persons, using the tongue control unit as the access pathway, have yet to be conducted and are not described in this paper.

5.2 Experimental Results

The following experiments have been conducted to verify that the interface effectively allows the user to discover and control the above-mentioned devices (appliances).

5.2.1 Discovering and Controlling Individual Appliances

Controlling a Bluetooth-enabled computer: the goal of this experiment is to verify that it is possible to discover, connect to, and control a Bluetooth-enabled computer by using the Bluetooth HID Profile to emulate a Bluetooth mouse and keyboard. The HID Profile makes the Bluetooth HID peripherals to provide a service to the Bluetooth HID hosts; thus, the initial setup begins on the computer side. The three computer configurations listed in Sect. 5.1 have been used. For each configuration, the user activates the HID service displayed in the list of services of the target computer. If the computer has never been connected to before, the user sees a popup message requesting a security password (this password is displayed on the interface screen). Once authenticated, a 'virtual' keyboard or mousepad is displayed on the prototype's screen: the two devices are now connected and ready to use.

The user can control the mouse cursor or type some text. We observe that the three (four when authentication is carried out) phases of the experiment are successful on the three configurations. The interface is discovered by the three Bluetooth stacks and can connect to the computers. Using these stacks, the user can successfully control the corresponding computers.

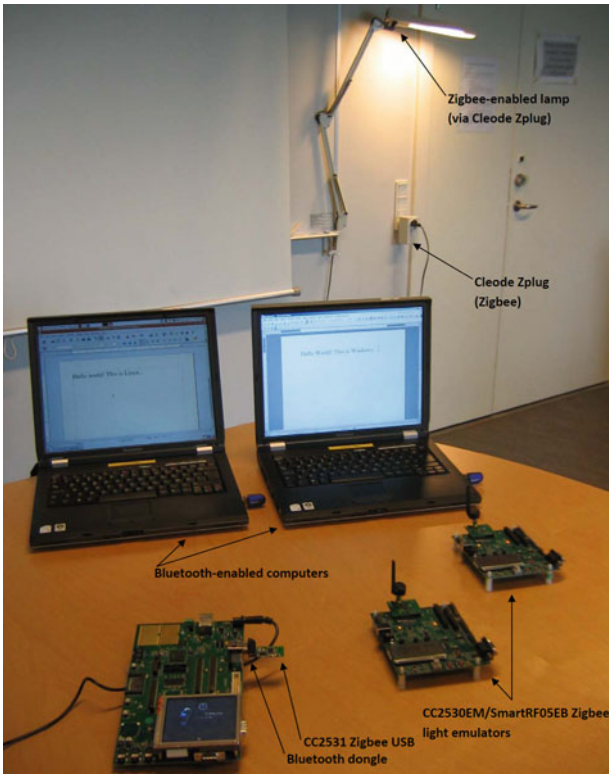

Fig. 8 Photograph of the setup. The CC2531 Zigbee USB dongle provides Zigbee connectivity to the interface prototype. The two CC2530EM Zigbee modules/SmartRF05EB boards are used as a Zigbee-enabled light emulators and a desk lamp is Zigbee-enabled by means of a Cleode Zplug. Bluetooth connectivity is provided by the Bluetooth dongles

We also observe that there is no visible delay between the commands being inputted on the interface and their effects on the computer; this is consistent with the discussion in Sect. 4.6.

The next experiment consists in turning Zigbee-controlled lights on and off: the goal is to verify that the proposed interface can be used to turn Zigbee-enabled lights on and off. The two CC2530EM/SmartRF05EBs are programmed with Texas Instruments' Zigbee light emulator (compatible with the Zigbee Home Automation profile). Moreover, a desk lamp is Zigbee enabled by means of a Cleode Zplug. Once the three lights have been detected and added to the list of available appliances, the lights are repetitively switched on and off at non-steady intervals. The lights respond smoothly to the commands entered by the user on the interface and no visible delay can be observed; this is also consistent with the discussion in Sect. 4.6.

5.2.2 Controlling Several Appliances Using the Same Protocol

The goal of this set of two experiments is to verify that the interface allows the user to control several appliances using the same protocol simultaneously, i.e. the user can switch rapidly between the controls of several appliances without affecting their respective states. The setup for the first experiment consists of one computer with Microsoft Windows

XP/Widcomm Bluetooth stack and one computer with Linux Ubuntu/Blue-Z Bluetooth stack. The setup for the second experiment consists of the three lights used in the previous experiment. In the first experiment, the user alternatively takes control of the two computers through the proposed interface as follows: connect to the first computer and use it, switch to the icon of the second one on the GUI, connect to the second one and use it, switch back to the first one, and so on and so forth. The experiment shows that the user can easily switch between the two Bluetooth-enabled computers and that there is no visible delay related to the switching operation; this is consistent with the discussion in Sect. 4.6. Similarly, in the second experiment the user alternatively takes control of the three Zigbee lights. Again, the user can easily switch between the appliances and there is no visible delay related to the switching operation; this is also consistent with the discussion in Sect. 4.6.

5.2.3 Controlling Several Appliances Using Different Protocols

With this experiment, we seek to verify that the interface can detect, connect to and control several appliances using different protocols simultaneously, without affecting their respective states. Moreover, the experiment illustrates how the interface abstracts, for the user, the various appliances and their services, irrespectively of their protocols. In this experiment, we use the two Bluetooth-enabled computers and the three Zigbee lights. The user alternatively takes control of the computers and of the lights by connecting to and using one of the computers, switching to one of the lights and turning it on and off, switching to the other computer, and so on and so forth. The experiment shows that the user can easily switch between all the appliances, and that there is no visible delay related to the switching operation; this is consistent with the discussion in Sect. 4.6.

5.2.4 Dynamic Adaptation to the User's Environment

Finally, the last set of experiments consists in verifying that the interface can dynamically adapt itself to the user's environment, i.e. update the list of appliances and services when e.g. the user moves from one location to another one, or when an appliance is removed by a third party from his/her vicinity. The experiments consist in turning the appliances on and off, as well as modifying the distance between the interface and the appliances without modifying their current states (e.g. on/off, service configurations). These are repeated at non-steady intervals: short (ca. 10 s), moderate (3 min) and long (10 min) ones. The experiments show that whenever one of the five appliances is turned on and is within reach, it is successfully detected by the interface and that a notification informs the user about availability of the appliance and of its services. Similarly, whenever one of the five appliances is turned off or becomes out of reach, it and its services are successfully removed from the list. Moreover, the user is informed by means of a non-blocking notification message. On the contrary, whenever an appliance is removed while it is being used by the user, a blocking message informs the user about the unavailability of the corresponding service(s).

5.2.5 Power Measurements

In order to evaluate the impact of the PMS we have measured the power on the main board in different modes. At boot-time, the power reaches 2,400 mW. Once the system has booted, and with the backlight turned on and maximum CPU frequency, the power is stable at 2,244 mW. Turning off the backlight or scaling to the minimum frequency brings the power down to

1,656 and 1,620 mW, respectively. Finally, combining the two above techniques brings the power down to 1,080 mW, i.e. 2.22 less than in the full mode. Note that these measures include components (e.g. audio) that are not used for the interface; hence a custom-made board would require less power.

6 Discussion and Outlook

The experiments show that the proposed interface effectively enables its user to remotely and easily control several types of appliances available in his/her environment through different types of WPANs. The interface enables the user to easily switch between appliances and their respective services, without interfering with each other. Moreover, the interface is context-aware, i.e. it successfully adapts itself to the user's environment by detecting remote appliances that enter or leave the interface coverage range and informs the user about those changes by means of the graphical user interface. The prototype, constructed as an embedded system, is relatively inexpensive, small, and power efficient. Suggestions for future work are listed in what follows. Firstly, other plug-ins (e.g. Wi-Fi, 6LowPan) could be added and experiments carried out to evaluate how the interface handles more than two protocols simultaneously. Secondly, experiments should be conducted with more appliances from multiple vendors to verify the robustness and compatibility of the proposed interface. Thirdly, although usability has been considered during the design of the interface, in-field clinical experiments with the targeted user group, using the mouth control unit as the access pathway, should be conducted. Finally, converting and merging the development kits into a custom board would result in a further optimized system in terms of size, power, and price.

References

1. Simunic, D., & Djurek, M. (2009). Transdisciplinarity of smart health care: Transmedical Evolution. *Wireless Personal Communications, 51*(4), 687–695.
2. Tai, K., Blain, S., & Chau, T. (2008). A review of emerging access technologies for individuals with severe motor impairments. *Assistive Technology, 20*(4), 204–219.
3. Craig, A., Tran, Y., Mcisaac, P., & Boord, P. (2004). The efficacy and benefits of environmental control systems for the severely disabled. *Medical Science Monitor, 11*(1), RA32–39.
4. Louise-Bender Pape, T., Kim, J., & Weiner, B. (2002). The shaping of individual meanings assigned to assistive technology: a review of personal factors. *Disability and Rehabilitation, 24*(1–3), 5–20.
5. Ville, I., Crost, M., Ravaud, J.F., & Tetrafigap, Group (2003). Disability and a sense of community belonging a study among tetraplegic spinal-cord-injured persons in France. *Social Science & Medicine, 56*(2), 321–332.
6. Andreassen Struijk, L. N. S. (2006). An inductive tongue computer interface for control of computers and assistive devices. *IEEE Transactions on, Biomedical Engineering, 53*(12), 2594–2597.
7. Lontis, R., & Andreassen Struijk, L. N. S. (2010). Design of inductive sensors for tongue control system for computers and assistive devices. *Disability and Rehabilitation: Assistive Technology, 5*(4), 266–271.
8. Clayton, C., Platts, R. G. S., Steinberg, M., & Hennequin, J. R. (1992). Palatal tongue controller. *Journal of Microcomputer Applications, 15*(2), 9–12.
9. Buchhold N. (1995). Apparatus for controlling peripheral devices through tongue movement, and method of processing control signals. United States Patent no. 5,460,186. 600/590; 340/4.11.
10. Peng, Q., & Budinger, T. (2007). ZigBee-based wireless intra-oral control system for quadriplegic patients. In *29th annual international conference of the IEEE engineering in medicine and biology society.* Lyon, August 23–26
11. Huo, X., & Ghovanloo, M. (2009). Using unconstrained tongue motion as an alternative control mechanism for wheeled mobility. *IEEE Transactions on Biomedical Engineering, 56*(6), 1719–1726.
12. Kirbis, M., & Kramberger, I. (2009). Mobile device for electronic eye gesture recognition. *IEEE Transactions on, Consumer Electronics, 55*(4), 2127–2133.

13. Ververidis, C. N., & Polyzos, G. C. (2008). Service discovery for mobile Ad Hoc networks: a survey of issues and techniques. *IEEE, Communications Surveys & Tutorials, 10*(3), 30–45.
14. Xu, L., Santoro, N., & Stojmenovic, I. (2009). Localized distance-sensitive service discovery in wireless sensor and actor networks. *IEEE Transactions on, Computers, 58*(9), 1275–1288.
15. Lankri, S., Berruet, P., & Philippe, J.-L. (2009) Multi-level reconfiguration in the DANAH assistive system. In *IEEE international conference on systems, man and cybernetics.* San Antonio, October 11–14.
16. Bronsted, J., Madsen, P. P., Skou, A., & Torbensen, R. (2010). The HomePort system. In *7th IEEE consumer communications and networking conference*. Las Vegas, January 9–12.
17. Gill, K., Shuang-Hua, Y., Fang, Y., & Xin, L. (2009). A zigbee-based home automation system. *Consumer Electronics, IEEE Transactions on, 55*(2), 422–430.
18. Woo Suk, L., & Seung Ho, H. (2009). Implementation of a KNX-ZigBee gateway for home automation. In *IEEE 13th international symposium on consumer electronics.* Braunschweig, May 25–28.
19. Young-Guk, H. (2009). Dynamic integration of zigbee home networks into home gateways using OSGI service registry. *IEEE Transactions on, Consumer Electronics, 55*(2), 470–476.
20. Guangming, S., Yaoxin, Z., Weijuan, Z., & Aiguo, S. (2008). A multi-interface gateway architecture for home automation networks. *IEEE Transactions on, Consumer Electronics, 54*(3), 1110–1113.
21. Cheng, S., & Wang, C. (2010). An adaptive scenario-based reasoning system across smart houses. *Wireless Personal Communications.*
22. Chou, J., Weng, S., & Chen, C. (2009). Action patterns probing for dynamic service composition in home network. *Wireless Personal Communications, 51*(1), 137–151.
23. Kulkarni, D., & Tripathi, A. (2010). A framework for programming robust context-aware applications. *IEEE Transactions on, Software Engineering, 36*(2), 184–197.
24. Mathieu, B. (2009). A context-aware network equipment for dynamic adaptation of multimedia services in wireless networks. *Wireless Personal Communications, 48*(1), 69–92.
25. Atmel (2009). AT91SAM9263-EK Evaluation Board User Guide.
26. Texas-Instruments (2009). CC2530 Development Kit User's Guide.
27. Luca M. (2010). Ångström Manual—Embedded Power-.
28. Nokia (2009). Qt for Embedded Linux White Paper.

Author Biographies

Bastien Paul earned his Engineer title in Electronics and Computer Science from ECE, Paris, France in 2010. He spent his final study year at Aalborg University, Denmark. His research interests include home automation, embedded system design and programming in addition to robotics.

Séverin Marcombes earned his Engineer title in Electronics and Computer Science from ECE, Paris, France in 2010. He spent his final study year at Aalborg University, Denmark. His research interests include wireless telecommunication (in particular Bluetooth), sensor networks, home automation, ubiquitous computing, and interfaces.

Alexandre David earned his Ph.D. degree in Computer Science from Uppsala University, Sweden, in 2003. He worked as a post-doc and then as Assistant Professor at Aalborg University before becoming Associate Professor in 2006. He is part of the distributed and embedded systems group and his research interests include formal verification of real-time systems.

Lotte N. S. Andreasen Struijk earned her Ph.D. degree in Biomedical Engineering from Aalborg University, Denmark, in 2002. She has previously worked as Research Assistant and Research Assistant Professor at SMI, Department of Health Science and Technology, Aalborg University, Denmark, where she is now Associate Professor. Her main interests are in rehabilitation technology, neural prostheses, and assistive devices.

Yannick Le Moullec earned his Ph.D. degree in Electrical Engineering from Université de Bretagne Sud, Lorient, France, in 2003. From 2003 to 2005 he was a Post-doc fellow, CISS, Denmark. From 2005 to 2008 he was Research Assistant, Department of Electronic Systems, Aalborg University, where he is now Associate Professor. His research interests include methods and tools for HW/SW co-design, embedded systems, and reconfigurable computing.