Home Appliances as Home Controllers: Concepts and Set-Top Box Implementation

Milan Z. Bjelica, Nikola Teslic, Zoran Jovanovic and Zoran Marceta

***Abstract* — In this paper we present a novel software-based home control platform suitable as an extension to digital home appliances that are equipped with a CPU (set-top boxes, home theatre systems, TV sets, gaming consoles, etc). By using an appliance they are already accustomed to, users become able to control lights, appliances and media playback in their homes. Intelligence and awareness are achieved with a support for execution of recipes – pre-prepared scripts that define timely actions and respond to triggers obtained from sensors. Software abstraction layer facilitates integration of any desired communication protocol. In our prototype, we supported Zigbee and DMX for light control, X10 for light/appliances control over power line, as well as Ethernet-based optical cameras as motion / presence sensors and UPnP / DLNA based equipment for distributed media playback.**

***Keywords* — domotics, smart home, set-top box, Zigbee, DMX, home automation**

I. Introduction

ATTEMPTS to make everyday living more convenient and in-house computing ubiquitous are not new. Achieving true, context-aware and user-aware, intelligent ambience has been traditionally assuming a complex installation of various equipment. Integrating sensors, lighting and multimedia seamlessly into one system has always been a daunting task. Such systems have been hardly accessible for an average home in terms of cost and installation levity. Instead, users are offered solutions to a number of heterogeneous home applications [1]. Making the home intelligent or autonomous requires the user to acquire, mount and install different pieces of hardware, often including lots of wiring and non-uniform standards. An appealing alternative would be to utilize home appliances users are accustomed to, such as set-top boxes (STBs) or gaming consoles as home/ambience controllers.

Most modern home appliances run control software upon a POSIX compliant operating system (e.g. different Linux distributions). This control software mostly runs on a 32-bit processing unit that is often under-utilized. Moreover, network-ready devices, such as STBs, are equipped with Ethernet and USB connectors. For example, the way to turn a STB into a home controller without any hardware changes is to connect a USB stick with home control software on it, and to activate home control graphical user interface (GUI) by pressing a remote controller button. In this paper we present such a solution.

Our solution responds to several key challenges. Portability to different home appliance hardware is guaranteed by the use of open standards (POSIX/C). Concept of recipes enables the use of the controller for various home setups, consisting of heterogeneous devices, making the solution scalable. Existence of an abstraction layer provides extendibility, when support for new devices/protocols equals to writing additional plug-ins. Ability to integrate virtual sensorial plugins that provide behavioral estimation, imparts the creation of an awareness system. Abstraction layer helps to bridge confronted approaches in home automation nowadays that complicate standardization efforts (new wires or existing wires versus wireless). Customizability is achieved by a software model [2] that provides separation of core functions from UI, so the appearance of the software can be tailored to accommodate different interaction requirements. Finally, the solution can be used to intelligently control lights and appliances, with the goal to reduce their energy consumption, contributing to household energy savings.

In the next section we give an overview of current situation in the home automation, and present works related to this paper. In Section 3 we describe the architecture of our solution, while in Section 4 we provide details of the demonstration setup and our prototype, together with a use case that underlines immersive experience for users inside a living room. Section 5 presents experimental results that show how our controller can help save energy in an average household.

II. Related Work

Commercial influence on standards in home automation nowadays is overwhelming. Markets across the globe, dominantly in North America, Europe and Japan are introducing new, yet mutually incompatible wiring/wireless technologies and standards. Potpourri of available protocols renders the situation very complex on a global scale when achieving interoperability gets harder each day. Nonetheless, three main trends are clearly distinguishable: (1) dedicated wired networks; (2) existing wires reuse and (3) wireless connectivity.

The provision of a dedicated wired network is a most frequent approach. In this regard proprietary solutions are available (e.g. LonWorks [3]), but more general technologies are increasingly popular (e.g. Ethernet),

Milan Z. Bjelica, Zoran Jovanovic and Zoran Marceta are with the Faculty of Techical Sciences, University of Novi Sad, Trg Dositeja Obradovica 6, 21000 Novi Sad, Serbia (phone: 381-21-4801202; e-mail: milan.bjelica@rt-rk.com).

Nikola Teslic is with the RT-RK Computer Based Systems LLC, Fruskogorska 11, 21000 Novi Sad, Serbia (phone: 381-21-4801106, e-mail: nikola.teslic@rt-rk.com).

preferably for networks of general purpose devices (computers) and high bandwidth requirements. Based on these physical protocols, various data transport protocols are implemented (X10 [4], CEBus [5], HBS [6]) while neither one of them is yet globally dominant. Global interoperability in wired connectivity for home automation is pursued by a European initiative, called Konnex open standard [7], which involves EIB, EHS and BatiBus protocols (earlier developed independently).

Making use of the existing wirings is also a very interesting approach, given the relief to the users in terms of installation levity. For example, using a power line for carrying data is a basis for EHS, X10, or Homeplug [8]. The main problems in this area revolve around higher noise levels than with dedicated wirings and safety concern with simple power line protocols (such as X10).

Lastly, wireless technologies are also very attractive, where infrared, or radio link technologies (e.g. IEEE802.11b) are most frequently utilized. A radio link is usually adopted for connectivity at higher distances.

Nevertheless, with the availability of complex controllers and processors at lower costs, technologies that are well established for common PCs are being revisited for numerous embedded devices. It is not unusual therefore, that simple computer systems are being used to bridge devices interconnection gap, having the cheap PC act as a home/ambient controller. On top of PC-based architectures, user interfacing is often provided by a simple web server, when PC is exposed to the global network and can be accessed from any internet hot spot. Also, GSM or GPRS networks can be used with a standard mobile provider, to allow control of the system via short message service (SMS) or a phone call. PC-based software packages that provide management of supported devices with the goal of home automation are scarce, at least when it comes to providing ambient intelligence and awareness (ETS [9], commercial; EIBcontrol [10], open source). Dedicated home automation controllers are also deprived of high level ambient functions. For example, the OSGi Alliance [11] has already proposed a rather complete and complex specification for a residential gateway (and alike). Here, protocol and integration issues, as well as intelligence related behaviors, are demanded to third-party "bundles", that can be dynamically plugged into the OSGi framework at run-time. An additional effort is therefore required to enhance the system with a compact yet general approach to intelligent and automatic interoperability among the involved devices. Connecting devices together and controlling them from a unique spot is a subject that is actively pursued by another open source community, under the project Open Remote [12].

Generalized home control is also the subject of additional, independent research. Coyle et al. in [13] presented an idea for home control middleware, trying to integrate components, sensors and different applications. Similarly, Casimiro et al. presented their work on the architectural framework for smart components in [14]. Pellegrino et al. in [15] proposed a layered software concept for applications in domotics. Application of sensor-actuator networks in home automation, with accent to lighting was given by Gauger et al. in [16]. There have also been works regarding GUIs for home control. Koskela and Vaananen evaluated PC, TV and a cell phone as GUI devices for home control [17]. Using smart phones to control appliances, with a notion of Personal Universal Controller, was proposed by Nichols and Myers [18]. Home networking and control based on the popular UPnP/DLNA stack of protocols was a foundation for a prototype implemented by Yiquin, Fang and Wei [19].

It can be noted, to the best of our knowledge, that there has been no research which tried to extend any existing home appliance with home control functions. In this paper, we have implemented software that can easily fit home appliances of different architectures, possibly by a B2B model with a manufacturer. This proposal is commercially exciting because it complies with users' habits, trying to introduce new functions under the helmet of already proven and widely accepted devices and applications.

I. Home Controller Prototype

The software is divided into two components: the *core* with *core drivers* and the *UI*. Software architecture is presented in Fig. 1.

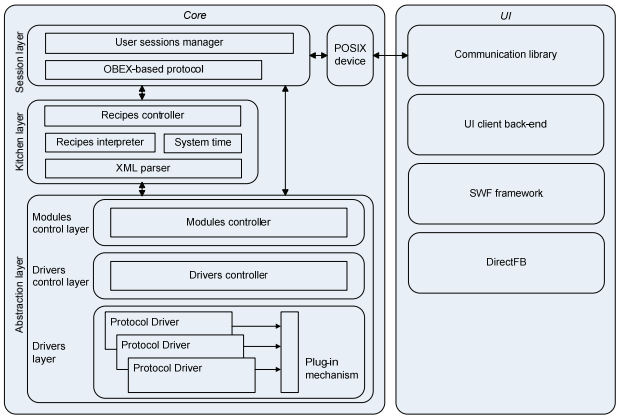

Fig. 1. Layered home controller software architecture.

A. Core component

Core component is "hidden" within a standard POSIX device, and acts as a home control server providing all desired functions for client UI applications (listing home devices, playing/stopping recipes, issuing a command etc). Communication to this device is done with standard read/write calls, by using an open communication protocol based on OBEX [20]. This way it is possible to access *core* software even from a remote platform in charge of user interaction, leveraging on the total processor load. For example, while core home control runs on an appliance, user can invoke functions from his cell phone or via web.

Core component is developed as an atomic, monolithic software module written purely in C language, avoiding operations that rely on concepts of stack, endianess or threading. This module contains API that is wrapped in a POSIX device and rolled into Linux kernel as a *.ko* object. Core software comprises several needed software layers.

Communication with a device (Zigbee, UPnP, Ethernet camera, X10 etc) is provided by *drivers layer*, which can be extended by plugins to support new protocols. A driver is in charge of passing on commands to physical devices and reporting events that occurred on a device. Drivers layer provides a simple C-based interface that defines structures holding pointers to functions with a fixed API, that all plugins must be conformant to. Software also supports a concept of a virtual driver that can have behavioral modeling capabilities, therefore providing awareness. These virtual awareness drivers are by no means different in interfacing to the core component from regular, physical drivers. Behavioral modeling and classification results (e.g. detected user activity) are regarded as raw data by the upper layers. In this case, virtual drivers must handle physical communication to all sensors they are dependable upon. Behavioral estimation concepts and virtual drivers inside the system described in this paper are presented in separate researches [21], [22].

A unified view to all drivers, their initialization and storage is provided by *drivers control layer*. This layer loads all driver plugins, enumerates them and provides mapping of symbolic names coming from upper layers, to a driver/device pair of identifiers (and its corresponding structure holding API functions). This layer provides a mechanism for *automatic profiling*, by measuring busyness of each driver (average time needed for execution of its functions). Based on this profiling, drivers control layer sets priorities to each function and orders a priority-based delta list. This way the real-time constraints of a home automation system are respected (timely reaction to sensor triggers, synchronization of light effects etc).

Since the household consists of modules (lights, appliances, sensors) as control objects, *modules control layer* is introduced. This layer enables issuing commands to modules (e.g. toggle light, play video file, set color of a DMX led lamp), as well as registering for different events that modules may generate (e.g. presence detection, light level change). The basic purpose of this layer is to provide abstraction to the above scripting, eliminating the need of knowing physical details of addressed devices. Primitives provided by the modules control layer can be invoked indirectly, from the upper layers that interpret recipes, or directly, by a user command from the UI.

Drivers layer, drivers control layer and modules control layer together form an *abstraction layer*, hiding all details on devices, protocols, behavior modeling and storage from the layers above.

Pre-scripted recipes are the most useful means of home control within the developed system. Home behavior is defined by XML-based recipes that are composed of sequence of module control operations. Recipes define a sequence of actions when a certain complex condition is met. The criteria based on which the condition is evaluated rely on the reception of events, coming from physical devices or virtual behavioral drivers. For example, user may want to trigger an ambient lightshow and a multimedia presentation on an LCD panel in the room, if there are more than four people in the room on Saturday evening. Although recipes can be scripted manually and provided directly by users, the intention is that these are generated automatically based on users' actions on the GUI. XML-based language used for scripting uses constructs common to high-level programming languages (loops, selections, declaring variables, basic arithmetic) but remains fast and easy to interpret (we adapted and used extremely light *McbXML* parser for this purpose). A basic concept for the creation of the interpreter is reused from our previous works [23]. Example of a recipe for the aforementioned party ambient setting is given in Fig. 2. Interpreting recipes, storing them and controlling their playback are all functions provided by the *kitchen layer*.

```
<driver type=”signal” module=”time” stat=”reached” val=”20:00”>
<linked type=”signal” module=”date” stat=”is” val=”Sat”/>
<linked type=”signal” module=”presence” stat=”reached” val=”4”/>
      <print> Recipe for triggering party </print>
      <print>
         Description: When there are more than 4 people
         on Saturday evening, trigger lightshow and multimedia.
      </print>
      <print> Author: John Doe </print>
      <print> </print>
  <var name="counter" val="1"/>
  <for>
     <if>
        <case namea="counter" val="1" rel="=">
           <for loop=3>
              <instruct module=”rgb1” function=”show1” />
              <instruct module=”rgb2” function=”show2” />
              <instruct module=”video” function=”par.avi” />
           </for>
        </case>
        <case namea="counter" val="2" rel="=">
           <for loop=3>
              <instruct module=”rgb1” function=”show2” />
              <instruct module=”rgb2” function=”show1” />
           </for>
        </case>
        <default>
           <instruct module=”video” function=”fadeout” />
           <var name="counter" val="1"/>
        </default>
     </if>
     <var name="counter" val="++"/>
  </for>
</driver>
```

Fig. 2. Example recipe for party ambient creation.

Finally, there is a *user sessions layer*, in charge of communication with (G)UI layers by implementing an OBEX-based session protocol. Communication primitives are attached directly to the *read/write* calls of the home control POSIX device that represents core to the user. Therefore, to use the functions of the core, user needs to establish the connection first. Each connection is uniquely identified, and a separate session is created for each user. Interaction to the core based on provided communication protocol facilitates integration of different remote UI units (e.g. mobile phone, PDA computer) without the need to implement additional protocols. The only concern when using the core for such UI applications is how to provide adequate data flow from the communication socket used for connecting remote UI device to the POSIX home control device. For this purpose, a switching daemon application is used. This application listens to different communication interfaces (Bluetooth, Ethernet, RS232) for data. When data is received, it is read to a buffer (POSIX *read*) and written to the home control POSIX device (POSIX *write*). Daemon also listens to home control POSIX device and writes data back to the originating interface. Therefore, daemon serves only as a connection point for data, while the core component

handles all communication logic. POSIX home control device is also targeted from an application residing locally on the appliance. This is illustrated in the diagram in Fig. 3.

The main characteristic of core component is its lightness, with about 400Kb in size of sources (20 pure C source files, without core drivers) and ~70kb the footprint of the kernel object at runtime. So far, core component has been tested on 32-bit big-endian MIPS processor (with uCLinux [24]), as well as on 32-bit and 64-bit PC processors, running XUbuntu [25].

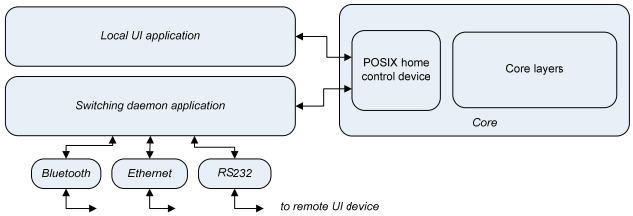

Fig. 3. A concept of interfacing the core component

B. Core drivers

To provide physical communication with home control devices, we developed/integrated several drivers, placed within the drivers layer. Given the extensible plugin mechanism, additional drivers can be written and added at any time. The following drivers were provided: (1) DMX light driver (for controlling DMX-based RGB lights connected over KWL DMX2USB adapter [26]; (2) Smart sockets driver (for controlling power on power outlets over Zigbee and UZBee USB2Zigbee dongle [27]); (3) Powerline light driver (for dimming lights using X10 protocol and Marmitek CM11 X10 transceiver [28]); (4) Ethernet camera driver (for communication with AXIS M1011 Ethernet camera [28]); (5) Multimedia driver (for controlling UPnP/DLNA playback on compatible devices attached to Ethernet) and (6) behavioral driver (gesture recognition using accelerometer – development is in progress). We believe that the given set of drivers is sufficient for most home applications (light effects control over DMX, light control over X10, multimedia control over UPnP, user presence and gestures detection with camera and accelerometer, mechanical actions – windows blinds control, garage door control etc. over X10).

DMX is a connectionless protocol with a simple addressing scheme (512 addresses per bus) where in practice each address corresponds to one light channel. For example, DMX Led lamp we used in development had four address channels (Red, Green, Blue and Mode, the fourth one defining light behavior, e.g. strobe). When the Mode channel is set to 189, solid colors are used without any effects. Developed driver provided the following functions: (1) Set color of a device to R, G, B; (2) Fade color of a device to R, G, B in time T and (3) start a lightshow with a given name. Parameters of each DMX light are its addresses and their purpose. Real-time constraints for fading effects were respected by implementing fading for all addressed DMX devices within a single thread, which sets lights towards the target value in steps different for each DMX light, depending on given T. Light shows were predefined in the driver to make the scripting easy for users that do not want highly customized effects.

Smart sockets driver was based on a system developed in research by Radin et. al in [30], with the addition of Zigbee protocol support [31]. This driver enabled users to set a specific socket (defined by its ID) to the given output power. Sockets itself are based on Zigbee communication module and TRIAC component for setting power. Therefore, sockets are intended to be used with simple devices, like lamps and bulbs. The driver provides the ability to read actual power value on each socket, making the commands reliable.

X10 driver implements communication with Marmitek X10 transceiver. This device is able to send dim/bright commands over a power line to counterpart receivers connected to sockets in the same power network within the household (LW12 [32], AM12 [33]). X10 driver provides the ability to increment/decrement current brightness value on a target module (and therefore the light connected over it), or to switch relay-based modules on or off (and therefore switch the target device on or off – garage door motors, window blinds, etc).

Ethernet camera driver was originally tested with a single camera, Axis M1011, but it may work well with any Ethernet camera supporting Motion JPEG [34] formats. By default, camera driver performs basic presence detection and people counting. These algorithms are extremely simple (given the constraints of target processing power), working for small resolution images (320x240 pixels) while the output is calculated by comparing the last received image with the reference background image taken periodically (this period can be adjusted). If the target appliance does not have enough processing power even for this operation, the driver can be set to listening mode, receiving event notifications on presence detection reported by the Axis camera (using one of the embedded functionalities of this camera model).

Multimedia driver is based on the AVSXLib library [35], acting as a control unit for UPnP/DLNA compatible devices. Therefore, serving multimedia content and its presentation is not performed within the core. This approach appears very convenient, given the number of UPnP/DLNA compatible servers/renderers on the market nowadays (and growing). User is able to start/stop/pause/resume multimedia playback on a target renderer device specified by its friendly name. Content provider (server) is also defined by stating URI (Uniform Resource Identifier) of the served content.

Behavioral driver is currently in development, and its intention is to detect hand gesture-based commands of users in the household, wearing accelerometer-equipped bracelets with RF. As a result, users can make meaningful gestures, setting currently used ambient profile in the household. For example, a recipe can be written to trigger change to the specific room ambient, when a user starts waving his hand.

C. User Interface

User interface construction is dependent on the

appliance used (since GUI applications are utmost processor and memory consumers). In our case, we used a STB based on 32-bit MIPS with 128MB of memory. Accordingly, *UI* layer in our prototype consisted of a shockwave flash based GUI, with a simple client back-end in charge of communication with the core. This approach provides portability and facilitates GUI development. For rendering *swf* file format and showing it on screen, *swfdec* Linux library was used [36], on top of *libgtk2* (*cairo*) [37] used for drawing operations and event system. Drawing board for the creative designers in this case was very convenient, since all the development could be done on their PC workstations. A socket connection is established to the back-end part of GUI application running locally that receives important GUI events from Flash ActionScript [38]. All requests made by users inside this flash application (start recipe playback, turn the light on/off), are packed into TCP/IP socket packets and sent to the back-end. Then, a communication primitive of core software is invoked, to activate actual home control operation.

Fig. 4. Flash-based GUI page rendered by Set-top box.

The appearance of one GUI page (recipe creation) on the LCD screen is given in Fig 4. Depending on the appliance hardware possibilities and target operating system, different GUI wrappers can be created. For example, usage of GUI based on GTK widgets only is also viable [39], [40].

II. Case Study

Implementation and demonstration was conducted for a STB based on a chipset for digital television from Trident Microsystems. The core of the chipset is the 32-bit MIPS processor with a real-time uCLinux operating system. Prototype setup consisted of the *core* and *UI* applications located on a USB stick connected to a USB hub and then to the available USB port on the STB. To USB hub we also connected Marmitek CM11 adapter for lights/appliances control over a power line, KWL USB2DMX adapter to control RGB led lamps over DMX bus, and UZBee USB2Zigbee dongle to control intelligent power sockets. We used Ethernet to connect STB with Axis M1011 Network Camera with embedded motion detection abilities and with UPnP software on another PC: Twonky Media Server as a multimedia content provider and Simple Center as a renderer application. Setup is presented in Fig. 5.

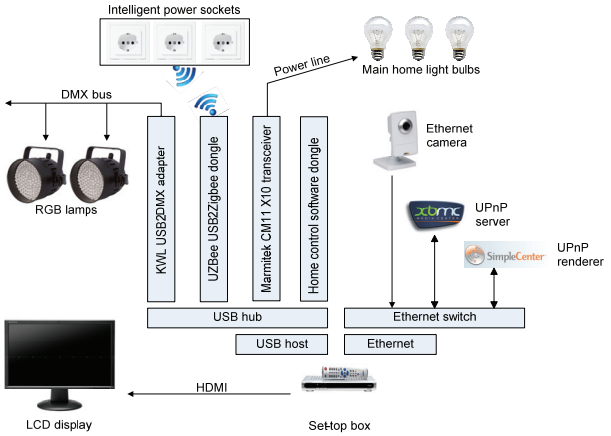

Fig. 5. System architecture used for demonstration.

By using a simple on screen menu users were able to choose one of the available recipes and set up ambient profiles for their home. Each mode consisted of a specific DMX lightshow and multimedia playback, tuned additionally depending on the number of people in the room. Ceiling bulbs were controlled over X10, while bedside lamps were connected to Zigbee power sockets (and one of them over AM12 appliance module with a relay, for demonstration purposes). Ambient was created by using two DMX EcoLed RGB lamps. When no people were present, ceiling lights and bedside lamps were dimmed or switched off to save energy.

No formal survey of users has been conducted, but their impressions were generally positive and the ambience has proven to be indeed immersive.

III. Experimental Results

Given the increasing trend of providing energy-saving applications, we tested our prototype to see if and how it can be used to help save energy in an average household. Energy saving was performed by utilizing two common approaches, both supported natively in our system (by an appropriate recipe): (1) if there is nobody present in the room for more than *n* minutes, turn all lights in that room off, and (2) decrease brightness of all lights in the household by 10% (what should be unnoticeable to users).

For our experiment, we installed the system to one of our co-worker's living room. We used Zigbee power sockets to control a bedside lamp and two ceiling bulbs in the living room, and Ethernet camera for presence detection. On the STB we had our software running a recipe that was detecting if a light was on to change its brightness by 10% (in small steps, so the change is smooth). Another recipe running in parallel was monitoring the state on the camera. When there were no people in the room for more than 2 minutes, all three lights would be switched off. The setup effort was marginal due to the scalability of our solution.

To be able to provide comparison, we extended our original smart socket driver, so that it logs the following: (1) current power consumption on all three sockets (bulbs)

periodically (once every 10 seconds), (2) switch toggling moments (change in power on any socket - on to off and vice versa). We also extended the camera driver to log changes in presence detection. Then, we recorded energy consumption, switch toggling moments and presence changes for a day of operation in the living room without energy saving recipes activated, to be used as a control result. Using the same bulbs and equipment, the experiment was repeated the next day. The smart socket driver was previously modified to be able to play back switch toggling data, in order to simulate user actions and control bulbs accordingly. The camera driver was also modified, to play back presence changes and emit virtual events. In the repeated experiment, energy saving recipes were activated. This way we achieved the comparison of two experiments, in which people behavior was identical: one without and the other with energy saving functionality. Note that we assumed that automatic light control would not affect original people behavior in toggling switches. Turning light on when it was already on, or off when it was already off, are events that we disregarded in the second experiment. Results of the experiment are shown in Fig. 6 (consumption during the day) and in Table 1 (total daily consumption).

TABLE 1: ENERGY SAVINGS WITH HOME CONTROLLER.

Energy consumption (one day, 3x100W bulbs)	**STB-based HC control**	**Regular control**
	472 W	**1256 W**

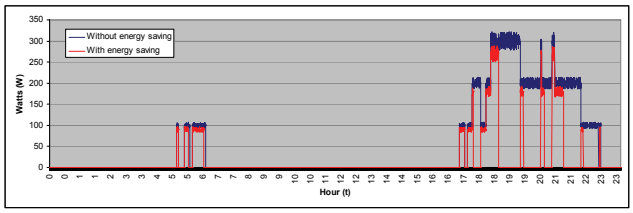

Fig. 6. Energy consumption during a day without energy saving (blue) and with energy saving (red).

IV. CONCLUSION

In this paper we presented concepts and a prototype system for home automation and ambient intelligence, that can fit into a home appliance, such as STB. One of the commercial benefits of the implemented prototype is the introduction of new technology to users seamlessly (through SW for their familiar device) therefore helping device manufacturers to enter new market with minimum risk. Apart from increasing users' ambient experience, the solution provides energy saving because the brightness of lights and their toggling are controlled by recipes, depending on the need. Future work would aim to achieve better ambient intelligence and context awareness, by implementing advanced behavioral models and user activity detection capabilities.

REFERENCES

[1] *Home Automation Index*, http://www.homeautomationindex.com

[2] M. Z. Bjelica, I. Papp, D. Samardzija Z. Jovanovic, "A Software Model with Remote User Interface for Embedded Systems," *in Proc. of conference TELFOR 2009*, pp. 1205-1208.

[3] *LonWorks,* www.echelon.com

[4] *X10 Knowledge Base,* http://kbase.x10.com/wiki/Main_Page

[5] *CEBus community forum,* http://www.cbusforums.com/forums/

[6] K. Sugihara, S. Kobatake, H. Shirai, H. Oowada, K. Yoshitomi, "HBS-standard-compatible home bus protocol controller," *IEEE Trans. on Consumer Electronics*, Vol. 35, No. 3, 1989, pp. 605-607.

[7] *KNX,* www.knx.org

[8] *HomePlug Powerline Alliance,* http://www.homeplug.org/home/

[9] *Eng. Tool Software (ETS),* http://www.knx.org/knx-tools/ets/description/

[10] *EIB Home Server,* http://sourceforge.net/projects/eibcontrol/

[11] *OSGi Alliance,* www.osgi.org

[12] *Open Remote: The Digital Home Operating System* http://www.openremote.org/display/HOME/OpenRemote

[13] L. Coyle, S. Neely, G. Stevenson, M. Sullivan, S. Dobson, P. Nixon, "Sensor Fusion-Based Middleware for Smart Homes," *International Journal of ARM*, Vol. 8, No. 2, June 2007, pp. 53-60.

[14] A. Casimiro, J. Kaiser, P. Verissimo, "An Architectural Framework and a Middleware for Cooperating Smart Components," in *Proc. of CF'04* Italy, ACM, 2004, pp. 28-39.

[15] P. Pellegrino, D. Bonino, F. Corno, "Domotic House Gateway," in *Proc. of SAC'06* France, ACM, 2006, pp. 1915-1920.

[16] M. Gauger, D. Minder, P. J. Marron, A. Wacker, A. Lachenmann, "Prototyping Sensor-Actuator Networks for Home Automation," in *Proc. of REALWSN'08* UK, ACM, 2008, pp. 56-60.

[17] T. Koskela, K. Vaananen-Vainio-Matilla, "Evolution towards smart home environments: empirical evaluation of three user interfaces," *Personal Ubiquitous Computing*, Vol. 8, 2004, pp. 234-240.

[18] J. Nichols, B. A. Myers, "Controlling Home and Office Appliances with Smart Phones," *IEEE Pervasive Computing*, 2006, pp. 60-67.

[19] L. Yiquin, F. Fang, L. Wei, "Home Networking and Control Based on UPnP: An Implementation," in *Proc. of 2nd Int. Workshop on Comp. Sci. and Eng*, Vol. 2, 2009, pp. 385-389.

[20] P. Megowan, D. Suvak, D. Kogan, "*IrDA Object Exchange Protocol OBEX*^{TM}, Infrared Data Association, Version 1.2, 1999.

[21] M. Z. Bjelica, N. Teslic, "A concept of usability assessment for user-centered multimedia applications," in *Proc. of International Multiconf. on Comp. Sci. and Technology*, Poland, 2009, pp. 443-450.

[22] M. Z. Bjelica, N. Teslic, "Multi-purpose User Awareness Kit for Consumer Electronic Devices," in *Proc. of International Conf. on Consumer Electronics (ICCE)*, Las Vegas, USA, 2010, pp. 239-240.

[23] M. Z. Bjelica, M. Savic, V. Vujanovic, M. Temerinac, "Realization of a PC application for communication subsystem of an integrated circuit emulation," in *Proc. of conference TELFOR 2008*, pp. 759-762.

[24] *UCLinux OS,* http://www.uclinux.org/

[25] *XUbuntu – Linux for human beings,* http://www.xubuntu.org/

[26] *KWL DMX2USB,* www.feno.com/products/detail/fc-dmx-512u.html

[27] *UZBee USB2Zigbee dongle,* http://www.rfsolutions.co.uk/acatalog/UZBee_ZigBee_USB_Dongle.html

[28] *CM11 transceiver*, http://www.marmitek.com/en/manual/9647.pdf

[29] *Axis M1011 Ethernet Camera,* http://www.axis.com/products/cam_m1011/accessories.htm

[30] B. Radin, V. Nuhijevic, N. Pjevalica, "An Intelligent Power Supply Outlet as a Solution in Smart House," *TELFOR 2009*, pp. 815-818.

[31] *Zigbee protocol,* http://www.zigbee.org

[32] *LW12 Module*, http://www.marmitek.com/en/manual/9611.pdf

[33] *AM12 Module*, http://www.marmitek.com/en/manual/8915.pdf

[34] *Motion JPEG formats,* http://en.wikipedia.org/wiki/Motion_JPEG

[35] S. Maric, M. Z. Bjelica, B. Mrazovac, N. Teslic, "An implementation of scripted control for a distributed multimedia system," in *Proc. of conference ETRAN* 2010.

[36] *Swfdec Linux library,* http://swfdec.freedesktop.org/wiki/

[37] *GTK,* www.gtk.org

[38] *Flash ActionScript,* http://www.actionscript.org/

[39] M. Z. Bjelica, N. Teslic, "A concept and implementation of the Embeddable Home Controller," in *Proc. MIPRO* 2010, pp. 173-177.

[40] M. Z. Bjelica, N. Teslic, I. Papp, M. Savic, "A characterization to evaluate graphical user interface frameworks for television receivers," in *Proc. of conf. TELSIKS* 2009, pp. 285-288.