Zigbee-to-IP Application Development
Remote Light Control with Policy

Afonso Falardo Garcia
Aalto University
School of Science
afonso.falardoromeiragarcia@aalto.fi

Jani Lillkåll
Aalto University
School of Science
jani.lollkaal@aalto.fi

I. Introduction

Nowadays, many companies and individuals are implementing measures to reduce energy consumption. This can be motivated by either cost issues or by environmental beliefs. One of the commonly targeted areas is lighting systems. Keeping a light off when it's not needed is one of the most effective ways to cut energy consumption.

Technology can provide a way to help users achieving this goal by implementing automation systems that control the lights. These systems can vary from remote controlling the lights with a smartphone to controlling the lights automatically as the user moves around. However, they may not achieve their goal since they may be turning on lights in rooms that are illuminated by sunlight or other natural light sources.

To avoid this situation, light sensors can be integrated in the system. This way, the system will be able to know how much light is available in the room and decide to turn on a light based on this information.

In this report, we present the implementation of a prototype for a system of this type. Our prototype is able to control a single light using an Android smartphone. In section II we present the overall architecture of the developed system. Section III presents the network technologies used. Section IV explains how the Arduino prototypes were setup and assembled. Section V presents the communication between each subsystem and the configurations of the ZigBee network. Section VI explains the decisions made during the implementation of the prototype. Section VII describes two possible extensions to the prototype. Finally, section VIII discusses the prototype and concludes the report.

II. System architecture

The developed system is composed by 4 subsystems, interconnected by an IP network supported by a router and an ad-hoc Zig-bee network. The architecture of the system is demonstrated in *Fig. 1*.

The first subsystem is the *Sensor Arduino*. This system is an Arduino that has a LDR (Light Dependent Resistor), a LED (Light Emitting Diode) and a Zig-bee module for communication. In order to provide the needed functionality, it replies to the requests defined in *Appendix A*.

The second subsystem is the *Base Arduino*. This Arduino performs the role of a base station in a sensor network. It

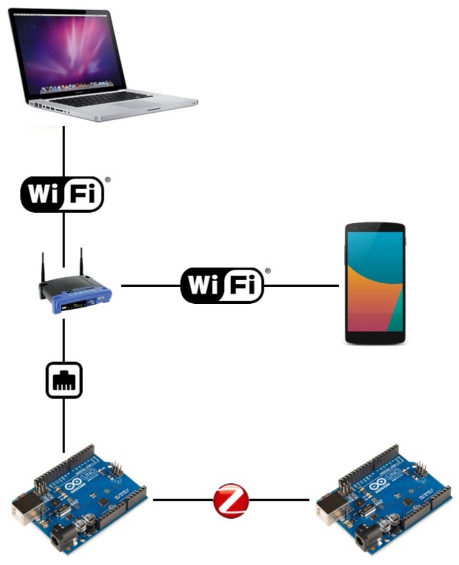

Fig. 1. System Architecture

is equipped with a Zig-bee module for communication with the *Sensor Arduino*. This system is responsible for issuing the requests defined in *Appendix A*. It is also equipped with an Ethernet shield, to provide communication via IP to other elements of the system, using the protocol speficied in *Appendix B*.

The third subsystem is the *Server*. This system is responsible for receiving the requests to turn on or to turn off the light and to adequately reply to them. In order to to this, it implements a simple logic that specifies that the light cannot be turned on if the LDR reading is below a certain threshold. Using the protocol defined in *Appendix B*, this system communicates with the *Base Arduino* to read the LDR and to operate the LED. It also provides an implementation of the protocol defined in *Appendix C*.

The final subsystem is the *Android Smartphone*. This system runs an app that emulates when an user arrives to a building by sending a request to turn on the lights and when a user leaves the building by sending a request to turn off the lights, using the protocol specified in *Appendix C*.

The *Base Arduino*, *Server* and *Android Smartphone* subsystems are connected with an IP network. The *Base Arduino* is connected to the router using Ethernet, while the *Android Smartphone* and the *Server* use WiFi to connect to it. However, the communication between the *Base Arduino* and the *Sensor Arduino* is done using Zig-bee, which means that the *Base Arduino* works as a relay, receiving packets from the *Server* using the IP network and forwarding the requests to the *Sensor Arduino* using Zig-bee.

III. Network technologies used

To implement the described system, we used Ethernet (IEEE 802.3) and Wi-Fi (IEEE 802.11) technologies to communicate within the IP network. These technologies are mature technologies that are widely deployed for the communication scenarios used in our system.

To communicate between the Arduinos, we use ZigBee. Part of ZigBee is standardised in IEEE 802.15.4. This is a standard defined by the IEEE 802.15 working group, which is responsible for wireless personal area networks (WPAN). Another standard defined by this workgroup is Bluetooth (IEEE 802.15.1). Although ZigBee and Bluetooth are WPAN standards, their use cases differ. In table I both technologies are compared and it is shown what type of use cases they are suited for and why.

Feature	ZigBee	Bluetooth
Channel width	0.3 and 0.6 MHz	1 MHz
Range	Up to 70 m	10 m
Transfer rates	250 Kbps	1 Mbps
Data types	Small data packets	Text, multimedia
Application types	Remote controls, sensors	Mouse, Keyboard, Headsets

TABLE I. Comparison of ZigBee and Bluetooth [1]

Bluetooth has a higher transfer rate which makes it more suitable for applications that require large packets to be transmitted. On the other hand, ZigBee is better suited for remote controlling objects and sensor networks, since it uses less energy and, in normal usage, batteries powering ZigBee devices can last several years. It is up to 10 times faster at switching the radio from on to off, and vice versa, than Bluetooth, making it better in use cases that require frequent exchange of small messages.

IV. Arduino setup

The described sensor network was implemented with the Arduino platform. In the following sections, this report describes the components used and how they were assembled. The required components are presented in the list below.

- 1 Arduino Ethernet
- 1 Arduino Uno
- 2 Arduino Wireless Proto Shield
- 2 Digi International XBee S1 modules
- 1 Tinkerkit Sensor Shield
- 1 Tinkerkit LED
- 1 Tinkerkit LDR

A. Base Arduino

The *Base Arduino* was assembled with 1 Arduino Ethernet, 1 Arduino Wireless Proto Shield and 1 Digi International XBee S1 module. Figure 3 shows the assembled *Base Arduino*.

The assembled device takes advantage of the stackable shields to connect the Wireless Proto Shield to the Arduino. The XBee module is connected to the Wireless Proto Shield.

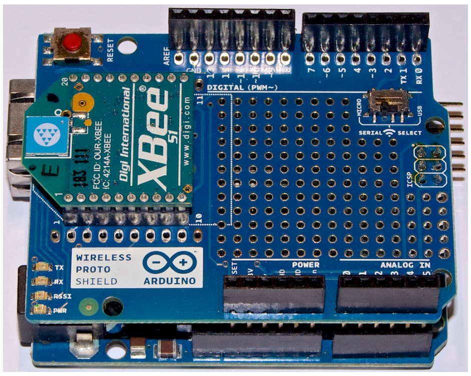

Fig. 3. Assembled Base Arduino

B. Sensor Arduino

The *Sensor Arduino* was assembled with 1 Arduino Uno, 1 Arduino Wireless Proto Shield, 1 Digi International XBee S1 module, 1 Tinkerkit Sensor Shield, 1 Tinkerkit LED and 1 Tinkerkit LDR. Figure 4 shows the assembled *Sensor Arduino* without the sensors connected to it.

As with the *Base Arduino*, the *Sensor Arduino* also takes advantage of the stackable shields. This Arduino is assembled the same way as the *Base Arduino*, but adds the Sensor Shield on top of the Wireless Proto Shield.

The LDR and the LED are connected to the Sensor Shield using the inputs and outputs provided by it. The LDR is an input device and, as such, is connected to the inputs of the shield. To work with our code, it is connected to input 0, which maps to pin A0 in the Arduino. The LED is an output device and, as such, is connected to the outputs of the shield. To work with our code, it is connected to output 0, which maps to pin 11 in the Arduino.

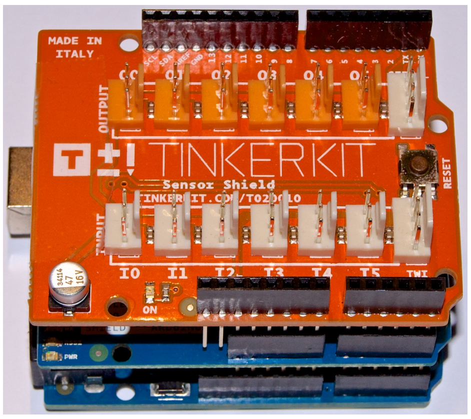

Fig. 4. Assembled Sensor Arduino

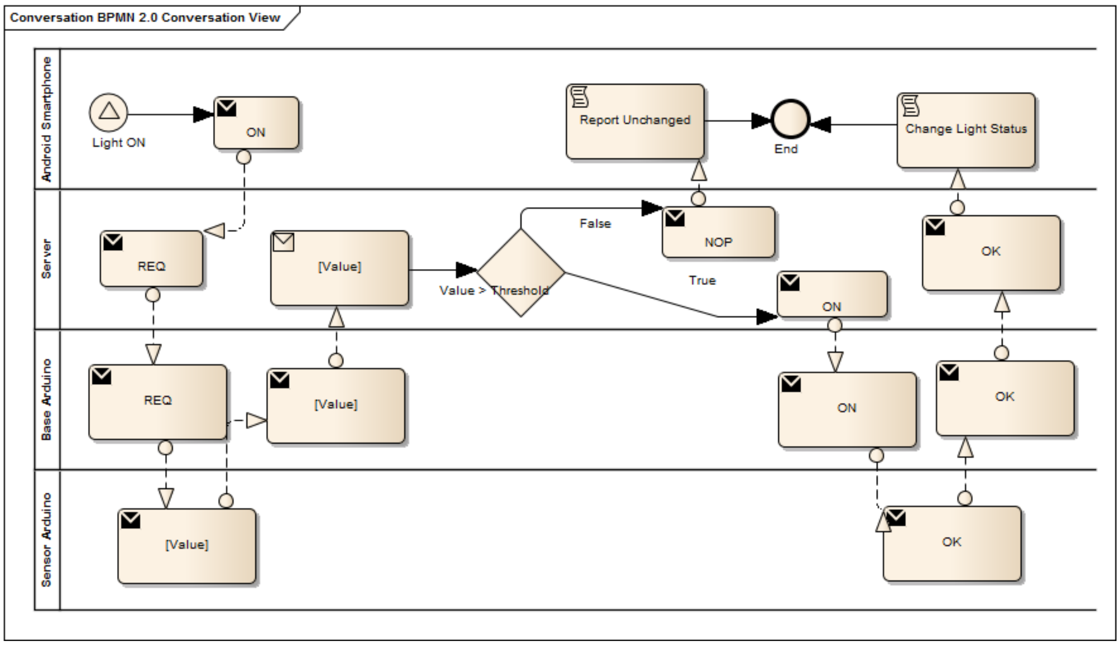

Fig. 2. Messages exchanged to turn on the light

V. Communication between subsystems

On our system, the *Android Smartphone* acts as the controller for the simulated light bulb on the *Sensor Arduino*. Yet, they are not directly connected and must, therefore, communicate using other subsystems. This is achieved by passing messages between the subsystems.

In *Fig. 2*, the messages that are exchanged when the *Android Smartphone* requests to turn on the light. It also shows the logic implemented in the *Server* to decide if the light should be turned on or not.

Fig. 5 shows the message exchange for an off request issued by the *Android Smartphone*. The process is simpler because there is no logic associated to this process and the *Server* acts as a relay, transmitting the messages to the *Base Arduino*.

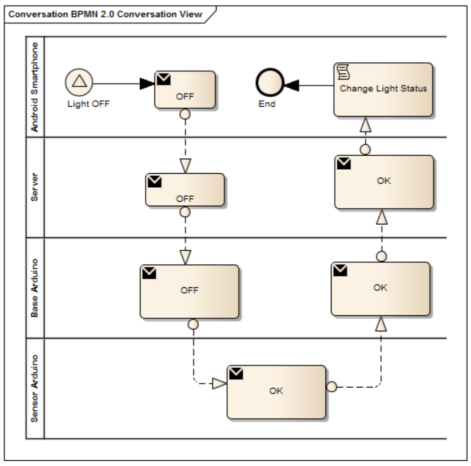

Fig. 5. Messages exchanged to turn off the light

A. *ZigBee network configuration*

In order to communicate with ZigBee, both the *Base Arduino* and the *Sensor Arduino* need to configure the XBee module with the correct parameters. In this section, the report describes the configuration performed during the setup phase of the Arduino sketches.

In the experiment we needed to assign addresses for the XBee modules. This is done by issuing AT commands to the module that configure it. The commands are issued using serial communication at a rate of 9600 Bd.

The first command issued is `ATDH`, which sets the serial number of the receiving module. This value is set to `0`, which represents a broadcast, because the selection will be done by the ID in the network. The next command issued is `ATDL`, which sets the ID of the receiving module. Finally, the `ATMY` command is issued, which sets the module ID in the network. Table II shows the values used in each Arduino.

Command	*Base Arduino*	*Sensor Arduino*
ATDH	0	0
ATDL	1235	1234
ATMY	1234	1235

TABLE II. Values of the AT commands issued

Before issuing the described commands, both Arduinos issue the command +++ to enter the configuration mode and `ATRE` to reset the memory of the module. After issuing the described commands, the command *ATWR* is issued to write the configuration to memory and *ATCN* is issued to exit configuration mode.

Other AT commands are available and allow the configuration of more complex systems. One example of such commands is `ATCH`, which allows to configure the channel on which the modules communicate.

VI. System implementation

To implement the described system, we have used the Arduino Language to program both Arduinos, Python to program the *Server* and Java to program the *Android Smartphone*. These decisions are justified in the sections below.

A. Arduino Language

The Arduino Language is similar in syntax to C, which makes it easy to learn and develop on. It supports the common control structures and operations used in other languages, even though the processing power of the Arduino is limited. Finally, it is the reference language for programming Arduinos and the language supported by the official Arduino IDE. Table III shows some of the used functions.

Function	Usage
pinMode(pin, mode)	Used to define the LED pin
digitalWrite(pin, value)	Used to set the value of the pin
analogRead(pin)	1234

TABLE III. Used functions

Together with this functions, we used the Serial and Ethernet libraries. The Serial library provides functions to interact with the XBee modules, such as `read`, `print` or `available`. The Ethernet library provides functions to communicate in the IP network and is similar in usage to the Serial library.

Other libraries for more sophisticated tasks are available, such as connecting to a GPRS network, connecting to a Wi-Fi network or drawing text and images on a TFT screen. These libraries are not needed for our system, but demonstrate the potential of the Arduino platform and possible extensions that can be implemented into our system.

B. Python

Python was used to implement the `Server` subsystem since it allows for rapid prototyping of the implementation. Python is less verbose than C and C like languages, which facilitates reading and maintaining the code. It also provides a mature set of libraries. Since it is an interpreted language, it provides code portability, allowing the server to run on any OS and architecture that supports Python.

We use the socket library to communicate in the IP network and implement the server and the client that communicates with the *Base Arduino*. The sys library is used to write a log of performed operations to a file.

The light threshold is provided when launching the *Server* in the command line as an argument, thus being configurable.

C. Java

Java was chosen to implement the *Android Smartphone* subsystem since it is the programming language used for Android. Android was chosen because it allows a graphical interaction with the system and provides a simulated remote control that is closer to what would be implemented in a real system.

To communicate with the *Server*, we use the `Thread` class to run the communication on a different thread and the `Socket` class to communicate in the IP network. The UI is built using the libraries available in the Android SDK.

D. Base Arduino

To simplify the implementation of the *Base Arduino* subsystem, we take advantage of the similarities between Appendix A and Appendix B protocols. Both protocols specify the same commands and replies and in the same format and their behaviour regarding errors and unknown commands is the same.

As such, our implementation of this subsystem receives a message in the IP network and relays it to the ZigBee networking, making the *Sensor Arduino* responsible for correctly implementing the protocol.

VII. Extensions to the system

The developed system provides a simplification of a platform that can remotely control lights and prevent the usage of the lights if there is enough light available in the room. This can be extended in multiple ways. This section presents two extensions to the system, one of which was implemented.

A. Multiple lights and sensors

The system described and implemented in this report could be extended to suit a real life scenario, where it could control a lighting system of a house. In a real life scenario, the system would have to be capable of controlling a number of different lights. The system could provide a way to adjust the light emmited in each room accordingly to the reading of the LDR, to maintain a constant light throughout the day.

A ZigBee network would be suitable for this purpose since it works well in scenarios that involve exchanging small amounts of data between the components of the network. It also has a small response time in the radio on to radio off shifting, and vice versa, and the communication can be done between a flexible number of nodes.

In this scenario, the ZigBee network could be fine tuned to allow the usage of different channels with the `ATCH` command. This would enable several nodes to transmit data simultaneously without disturbing the communication of the other nodes. This behaviour would make the system stabler and less prone to signal interference between its elements.

However, in a real life scenario, the network needs to be carefully planned to avoid interferences with Wi-Fi networks operating in the same area, since some Wi-Fi channels overlap some ZigBee channels. This can be avoided with careful planning the used channels in the Wi-Fi and ZigBee networks so that there is minimal interference between the networks.

To implement such a system, the protocols would have to be extended to support multiple lights. This could be done by associating each request with the ID of the light in the system. The *Server* logic would have to be extended to support multiple lights and to keep track of them. Finally, the *Android Smartphone* client would have to be redone to know in which part of the house the user is and to use the new functionalities provided by the *Server*. If the system is designed to maintain a fixed amount of light in each room, the *Android Smartphone* can be removed from the system since it doesn't need to be controlled.

B. *Guess the light! game*

The protocols designed to communicate with and within the sensor network allow its usage for different purposes than remote controlling lights. Our goal with this extension was to test its usage with a different application scenario. To do so, we have developed an extension to the Appendix C protocol, described in Appendix D, and implemented it on the *Server*.

The purpose of the *Guess the light!* game is to try to guess the value that is given by the LDR, truncated to the interval [0,999]. To do so, the user must provide to a client its guess, and the client will then communicate it to the server.

When the server is launched, it requests the LDR reading to the sensor network and sets the replied value as the value to be guessed. This value can be changed at any time by the client by requesting a new reading.

When the server receives the user guest, it compares each digit with the stored value. If a digit of the guess matches a digit of the stored value, it will instruct the sensor network to blink the light once. This is done for each correct digit. Blinking the light means turning it on and then off if the light is off and turning it off and then on if the light is on.

As such, if a user correctly guesses any 2 digits of the stored value, the lights will blink twice. If a user correctly guesses the stored value, the lights will blink three times. Note that the correctly guessed digit must match the digit in the stored value. This means that if the stored value is 234 and the user guesses 342, the lights won't blink, even though the user correctly guessed all three digits, because they aren't in the right order.

VIII. Conclusion

This report presents a prototype of a light control system that is capable of controlling a light and decide if that light should be turned on based on the reading of the amount of light available. It is able to correctly perform the tasks described in this report.

Some possible extensions to the system were discussed, including how to extend the prototype to a system capable of controlling more than one light.

References

[1] Engineersgarage.com, "Difference between bluetooth and zigbee — zigbee vs bluetooth," accessed on 04/12/2014. [Online]. Available: http://www.engineersgarage.com/contribution/zigbee-vs-bluetooth