Centralized Device Independent Software Architecture for the Internet of Things

Luis Ruiz Linares
Master of Engineering in Computer Engineering
Yahya M. Masalmah, PhD.
Electrical and Computer Engineering and Computer Science Department
Polytechnic University of Puerto Rico

Abstract *— The Internet of Things (IoT) is growing stronger every year, as developers we need to get ready for the challenges ahead of us. It is getting hard for software developers to have a starting point for implementing new embedded systems and test them on existing IoT platforms. A software platform which is easy to understand, with scalability and security in mind is needed. This paper provides a centralized device independent software platform as a possible solution for developers that need a starting point that has a simple to understand architecture. Since security is also a key factor in IoT, this platform is secured by implementing public key encryption using Secure Socket Layer (SSL). This proposed software platform has very familiar components to start developing and testing communication with custom or vendor specific embedded devices for the IoT.*

Key Terms *— Centralized Architecture, Device Independent, IoT, SSL.*

INTRODUCTION & MOTIVATION

The Internet of Things have been rapidly advancing in the past few years, it can be defined as a movement of global devices connected to a network to exchange data, monitor and control environment status. Cisco states that there will be more than 50 Billion devices connected to the internet by 2020 [1] shown in figure 1.

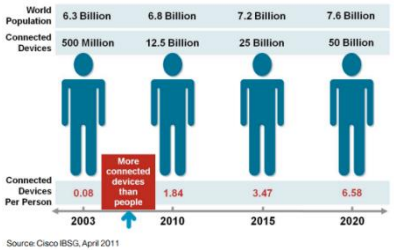

Figure 1
Cisco The Internet of Things was "Born" Between 2008 and 2009

A recent publication [2], states that IoT will grow to 26 Billion devices installed by 2020 and that will generate incremental revenues by $300 billion mostly in services and sales. Many devices are being developed either for future revenues or to solve a specific problem in the world such as environment changes monitoring, reducing traffic and CO2 emissions by finding the closest parking to a driver, helping monitor patients pacemakers and send data alerts to hospitals and reduce mortality incidents in medical patients. As these, there are many other motivations to be involved in developing software applications and embedded hardware devices for the IoT. But how can we contribute since the first question is Where do I start? What can I use to develop my application? How do I integrate a device to a system platform?

Developers are in need of a framework which is open source, with ease of scalability and highly customizable. This IoT framework is built for developers interested in learning ways to integrate a variety of devices to the internet. It provides a secure way to transmit data to their local servers without a third party services. The developer can start designing drivers for integrating new things without investing time and money in backend components such as database design, SSL server, message and command processing. Nevertheless, the developer should have basic knowledge of Java and Mobile application development at least.

LITERATURE REVIEW

There are many areas that are currently in need of connecting devices (figure 2) to the internet today such as [3]:

- Wearable Electronics
- Connected Homes
- Connected Cars
- Connected Cities

- Industrial Internet
- Transportation
- Healthcare
- Oil & Gas Industry

These are few examples of areas where IoT can be integrated using software frameworks today. This paper examines also existing solutions that may be applied to these areas to solve a problem.

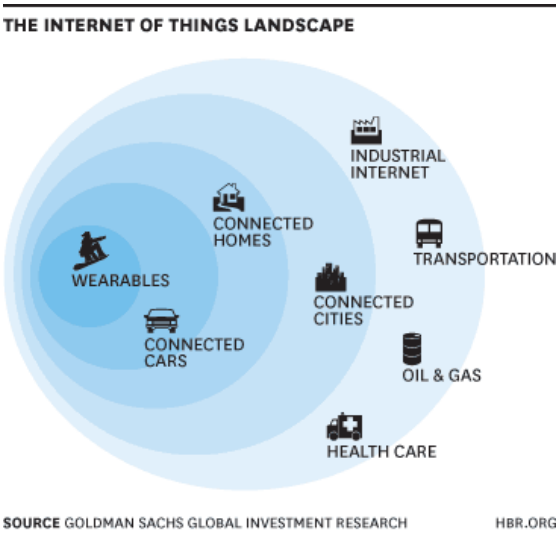

Figure 2
Goldman Sachs The Internet of Things Landscape

IoT Software Frameworks

IoT frameworks provide tools for software and hardware integration such as communication standards, data security, data storage and visualization. The framework should be scalable and performance is a big concern because of simultaneous device connections to the main infrastructure. There are many software frameworks being used for this purpose today such as:

- Thread Group [4] – provides a framework that makes use of IETF and IEEE wireless standards.
- Eclipse IoT Project [5] – provides open source implementations for IoT communication protocols such as MQTT CoAP, OMA-DM and OMA LWM2M.
- IoTivity [6] – an open source development framework based on Linux. Has Bluetooth integration, supports Belkin WeMo devices and Philips Hue Bulb. Also provides android mobile application support.
- Thingworx [7] – is a closed source IoT framework for enterprises seeking to develope scalable IoT infrastructures with secure communication standards and dashboards to analyze and display information.

IoT Hardware Components

Another important aspect for the internet of things is hardware components. There are many hardware solutions in the market that provide IoT gateways, temperature sensors, power switches, security cameras and RF sensors, smart watches, washer/drier machines, etc. Developers of the IoT that want to start developing using vendor specific hardware can do it using an existing device. But for those that are interested in creating IoT devices from bottom up, there are many alternatives such as:

- Raspberry Pi [8] – Embedded development credit card size hardware solution with a variety of features that provide alternatives such as GPIOs, camera ports usb ports, low power consumption, dual or quad core ARM processors. Many operating system supports such as Raspian, Arch Linux, OpenElec, Pidora, OpenWrt, Kali Linux and Windows 10 Embedded.
- Beaglebone [9] – Embedded development credit card size hardware solution with variety of features such as pin headers, dual core ARM processor, ethernet, micro sd card storage support and 1 x USB port.
- Arduino Yun [10] – a microcontroller credit card size hardware solution powered by Linux and Atheros based processor. It has Ethernet and WiFi support as well as USB-A port micro sd card slot and 20 digital input/output pins.

IoT Software Architectures

There are some proposed solutions to overcome the interoperability between devices, software and communication systems. Well-designed system architecture is key for the success of these interactions. A recent research proposed a

Distributed Internet-like Architecture (DIAT) [11]. It proposed a smart control and actuation of devices that will overcome most obstacles in the process of large expansion of the IoT. DIAT is a layered and distributed architecture supporting Automation, Intelligence, Dynamicity and zero configurations.

Bosch IoT Suite [12] provides a centralized device independent closed source software platform where the end user interacts with the platform by means of mobile, web or desktop applications (figure 3). Devices are controlled from the central application services at the servers. Application developers have different tools to interact with their platform such as API's to interact with the services and end devices, built-in security and high degree of modularization.

Figure 3
Bosch IoT Suite Software Platform

Bosch's Monaco 3.0 is a "project where they tackled challenges such as city dense population and narrow road network, which led to traffic jams and frustrating delays" as in [12]. Their main focus was to minimize the traffic jams and reduce CO2 emissions by using the city's parking garages as a network and an App provides the driver the nearest parking space.

Just as these software platforms there are many others being currently in execution for year these are a few examples of how developers can contribute to the internet of things. But since most of these software platforms are protected by proprietary software structures and patents, developers won't be able to modify or tinker with the platform even if they can afford purchasing their services. Therefore the following paper sections describe our open source centralized device independent software architecture for developers in the internet of things.

METHODOLOGY

The approach to a possible solution the system architecture will be described for each of the main architectural components. The main components are the Mobile Application, the Message Processing Servers and the Device Integration. Each section is easy to customize and where implemented with scalability in mind. Each system architecture component is described in the following sections.

System Architecture

The main components of this proposed architecture are the mobile application Layer, the Message Processing servers Layer and the Device integration layer. Figure 4 is a pictographic description of the system architecture main components. The end user will interact with the mobile application. In this case we developed this component using an Android smartphone. The user will be able to select a device of choice to send device specific commands already implemented into the application. The command message is sent over an encrypted channel to the main connection broker server. The Message broker then processes the request, in case of the message be a device command it forwards the command for processing to the command broker. At this point the command is routed to the device and the command broker should receive an acknowledgement from the device after a successful command execution.

The command broker determines which communication protocol to use and where and who is the recipient device in the network to send the command for execution. If no acknowledgement is sent back from the device after a few seconds the command broker then sends back a negative command status. The raspberry pi plays the role of providing device drivers for communication with the vendor specific devices or to provide custom device communication with GPIO headers, Bluetooth or other communication technologies.

The raspberry pi runs a python UDP server which provides the command/device execution routing at the device integration layer. It is assumed that the device will be in a secure network behind a firewall or secure subnet. By means of an API called Ouimeaux developed in python [13], we are able to communicate command messages to be executed by Belkin Wemo devices such as Light Switch and Insight Switch [14]. The Wemo Server component provides an interface with Wemo devices using an Ouimeaux python class for command execution.

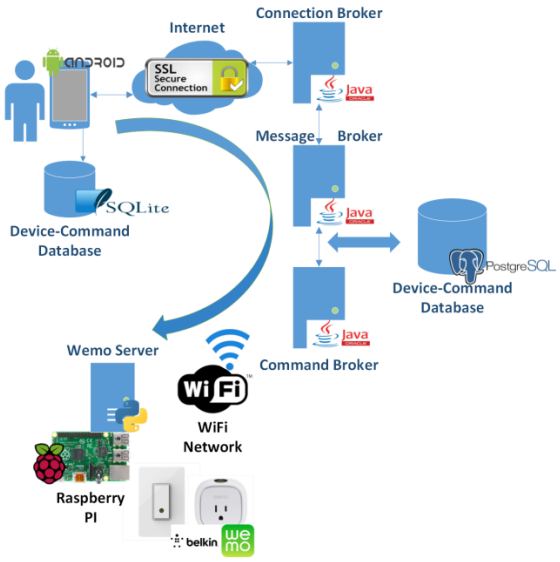

Figure 4
IOT Framework System Architecture

Mobile Application

The mobile application was developed using android-studio installed in Ubuntu Linux 14.04.3 LTS and a Samsung Note Edge smartphone device with Android Lollipop 5.0.1. The application has a primary activity screen (figure 5) which provides the device list using a ListView design. At startup the device list is collected from a SQLite database. The secondary activity screen is displayed after a device from the primary screen is selected by the user (figure 6). Then the user may be able to select the desired command to send to the server for the selected device (figure 7). The command message is sent to the server by means of Java SSL client developed for the android application. The SSL client establishes a secure link between the application and server by means of public key cryptography. The SSL client uses java ssl socket programming API integrated with java keystores (JKS) for storing the server public key located at the android device. The server public key was created using OpenSSL, an Open Source toolkit for implementing Transport Layer Security and Secure Socket Layer (SSL) protocols. This toolkit has a terminal command interface to create private and public keys for public key cryptography.

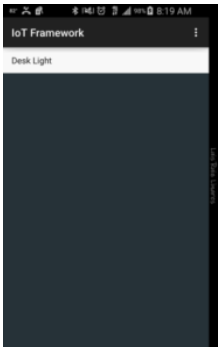

Figure 5
IoT Framework Mobile Application Device Selection

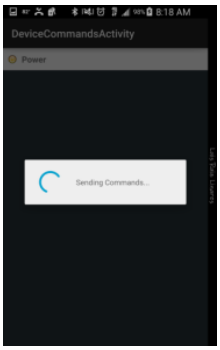

Figure 6
IoT Framework Mobile Application Command Selection

Figure 7
IoT Framework Mobile Application Command Selection Power On

Connection Broker

The Connection Broker was implemented using Java Runnable Thread package. The Connection Broker has the role of establishing a secure communication channel between the mobile application and the devices' local network. This feature is provided by means of a Secure Socket Layer Java language server. The mobile application must have a valid public key installed generated from the servers private key. The server then extracts the message from the communication channel and creates a Message object. The message object is then added to the Message Broker input Queue and passes the client stream channel to the message broker. At this point a secure communication channel is already established and the server can receive commands and send responses to the mobile application client. A fragment of the SSL server socket developed in android is shown in figure 8 below.

```
KeyStore ks = KeyStore.getInstance("JKS");
ks.load(new FileInputStream(keystore),keystorepass);
KeyManagerFactory kmf =
    KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
kmf.init(ks, keypassword);

SSLContext sslcontext =
    SSLContext.getInstance("TLS");

sslcontext.init(kmf.getKeyManagers(), null, null);

ServerSocketFactory ssf =
    sslcontext.getServerSocketFactory();


serverSocket = (SSLServerSocket)
ssf.createServerSocket(socket);
```

Figure 8
Connection Broker SSL Android Socket Code Fragment

Message Broker

The Message Broker server has the role of identifying the message sent from the mobile application. The main requests are defined as follows:

1. Device List – sends back a string containing the list devices to the mobile application.
2. Command List- sends back a string containing the list commands to the mobile application.
3. Device Command Status List – sends a list of a relation between the devices and commands associated with the device and the status of that command.
4. Device Status Command – processes and execute a command to a device and sets the required status.

This broker is constantly listening to the queue for arrival of messages from the mobile application. It filters the requests using the above specified request commands, responses to the mobile application are sent using the client ssl socket buffer output stream. It may also query the Postgresql database for the desired data requested by the mobile application. If the message is in fact a device command, a Command object is added to the Command Broker Queue for processing. Communication console output sample is shown below on figure 9.

```
MESSAGE_BROKER::DEV_STAT:Desk Light;Power;true
COMMAND_BROKER::Desk Light:Power:true
SENT UDP MSG ::Desk Light;Power;true
Waiting for UDPServer...
FROM UDPSERVER:Desk Light;Power;true
Client Disconnected!
Waiting for Client...
```

Figure 9
Message Broker Mobile Application Communication

WeMo Server

The Wemo Server is developed on python using UDP server socket to listen to command messages. A WeMo class was implemented using an API called Ouimeaux [13]. Ouimeaux is a python API for Belkin WeMo [14] devices that can be installed easily on any Linux OS distribution. The WeMo Server filters the message and sends the command to the correct WeMo device. In this case the WeMo devices used for this purpose have mostly two main commands either On or Off shown in the python class fragment code in figure 10. The server sends back a confirmation message to the Command server back to the mobile application. The mobile application should be able to update the status of the command for the respective device in the SQLite database. The WeMo server and the WeMo device class can be modified to integrate more WeMo devices easily. The developer only needs to have a python working knowledge and using the previous

command structures as example can easily add new functionalities to this python class.

```
import ouimeaux
from ouimeaux.environment import Environment

class WemoDeviceClass:

    def __init__(self):
        self.env = Environment()
        self.env.start()

    def On(self,device):

        self.switch = self.env.get_switch(device)
        self.env.wait(1)
        self.switch.on()

    def Off(self,device):

        self.switch = self.env.get_switch(device)
        self.env.wait(1)
        self.switch.off()
```

Figure 10
WeMo Server Python Class Code Fragment

Device Command Database

The Device Command Database component was developed in Postgresql engine. This is one of the best open source database engine with highest performance and with most support freely available for developers. The main role of the database is to store information of all the devices. Since commands may be the same across many devices, a command table is independent of the device table. This table contains unique device names, therefore naming convention is key for the system to work properly. The device/command table contains a relation between the device and command, this table relates the devices with their respective commands and their status. Each time the mobile application sends a new command the table is updated with the new status for the specific device.

CONCLUSIONS & FUTURE WORK

The framework has been proven to work with a single client application connected to the main Connection broker server. The system is capable of securely sending command messages to be executed by the WeMo server in this case, which in turn switches on or off an Insight switch of Light Switch depending of which is selected. The platform is also able to integrate new devices by adding more communication protocols at the command server. The developer may be able to code additional independent servers to send messages to custom embedded devices or vendor specific products.

This architecture outperforms the official Belkin WeMo application but has less features that the brand provided with the WeMo device and freely available from the android store. There are many IOT framework solutions already working as enterprise solutions and could be a disadvantage from the standpoint of cost, time and development resources. This architecture could have the potential of being a real product after more features are added and more devices are integrated from other brands.

The architecture should be scaled to accept more than one client, therefore it is needed a modification to convert the connection broker, message broker and command broker servers from single tenant to a multitenant architecture.

Also development of more API's or drivers is needed to support more IoT devices. Depending of the communication standards required for communication with each new device, the command broker will need a modification to include this client protocols in order to be able to send command messages to the end device.

REFERENCES

[1] D. Evans, "The Internet of Things How the Next Evolution of the Internet Is Changing Everything", *CISCO*, April 2011, pp.3.

[2] O. Vermesan, "Putting the Internet of Things Forward to the Next Level" in Internet of Things- Converging Technologies for Smart Environments and Integrated Ecosystems, River Publishers, 2013, ch. 2, pp. 20

[3] S. Jankowski, "The Internet of Things: Making sense of the next mega-trend", Goldman Sachs Group INC, 2014.

[4] Powerful Technology Designed for the Home. (2015). *Thread Group* [Online]. Available: www.threadgroup.org. [Accessed: September 17, 2015].

[5] Eclipse IoT Project. (2013). *IoT Eclipse Open Source for IoT* [Online]. Available: www.iot.eclipse.org. [Accessed: October 5, 2015].

[6] IoTivity. (2015). *Linux Foundation Collaborative Projects* [Online]. Available: www.iotivity.org. [Accessed: October 8, 2015].

[7] ThingWorx – Past and Present. (2015) *About ThingWorx A PTC Business* [Online]. Available: www.thingworx.com. [Accessed: October 8, 2015].

[8] Raspberry Pi. (2006). *What is a Raspberry Pi*? [Online]. Available: www.raspberrypi.org. [Accessed: October 9, 2015].

[9] Beaglebone. (2014). *What is a Beaglebone?* [Online]. Available: www.beaglebone.org. [Accessed: October 9, 2015].

[10] Arduino. (2015). *Arduino Products Overview* [Online] Available: www.arduino.cc. [Accessed: October 10, 2015].

[11] C. Sarkar, et al., "DIAT: A Scalable Distributed Architecture for IoT", *IEEE Internet of Things Journal*, vol. 2, no. 3, 2015.

[12] Bosch Software Innovations Corp., Bosch IoT Suite, unpublished.

[13] Ouimeaux. (2014). *Ouimeaux: Open Source WeMo Control* [Online]. Available: ouimeaux.readthedocs.org/en/latest /index.html.

[14] Belkin. (2015). *WeMo Home Automation* [Online]. Available: www.belkin.com/us/Products/home-automation/ c/wemo-home-automation.